

Java Agent Security Features (22.x.x)

Waratek ARMR Concepts and Features

Exported on 04/09/2021

Table of Contents

1	The ARMR 2.0 Platform (22.x.x)	9
1.1	ARMR Language Syntax (22.x.x).....	10
1.1.1	Block	11
1.1.2	Statement.....	11
1.1.3	Types Of Values	11
1.1.4	Comments	12
1.1.5	Escape Character	12
1.1.6	Formatting.....	13
1.2	ARMR Mod (22.x.x).....	14
1.2.1	Requires.....	14
1.2.2	Version	14
1.2.3	ARMR Language Level	15
1.2.4	ARMR Mod Example	15
1.3	ARMR Rule (22.x.x).....	16
1.3.1	ARMR Rule Parts	16
1.3.2	Given (Conditions)	16
1.3.3	When (Event)	16
1.3.4	Then (Action)	16
1.3.5	Valid ARMR Example	17
1.3.6	Invalid ARMR Example	18
1.3.7	ARMR Rule Life-Cycle	18
1.3.8	Limiting ARMR rules to specific Operating Systems.....	19
2	ARMR Rules (22.x.x)	20
2.1	ARMR Patch Rule (22.x.x)	20
2.1.1	Given (Conditions)	20
2.1.2	When (Event)	20
2.1.3	Then (Action)	20
2.1.4	Runtime Notation	20
2.1.4.1	Java Types.....	21
2.1.4.2	More JVM Internal Form Examples.....	22
2.1.5	Function.....	23
2.1.6	Location-Specifier	23

2.1.6.1	ENTRY / EXIT / ERROR	23
2.1.6.2	INSTRUCTION / LINE	24
2.1.6.3	READ / READSITE / READRETURN	24
2.1.6.4	WRITE / WRITESITE / WRITEReturn	24
2.1.6.5	CALL / CALLSITE / CALLRETURN	25
2.1.7	Code.....	25
2.1.7.1	ARMR Patch Methods	26
2.1.7.2	ARMR Patch State.....	27
2.1.7.3	JavaFrame	27
2.1.7.4	JavaField / JavaMethod.....	27
2.1.8	Patch Rule Example	28
2.1.9	Occurrences.....	30
2.1.10	Patch Execution Ordering And Greedy Location Specifiers	34
2.1.11	Life-Cycle For ARMR Patch Rule.....	36
2.1.12	JavaFrame API.....	36
2.1.12.1	The 'this' Variable.....	36
2.1.12.2	Raising Exceptions	36
2.1.12.3	Returning Values	37
2.1.12.4	Load Variables	37
2.1.12.5	Store Variables	37
2.1.12.6	Load Operand.....	38
2.1.12.7	Store Operand.....	38
2.2	ARMR Deserial Rule (22.x.x)	39
2.2.1	Overview	39
2.2.2	Given(Condition)	40
2.2.3	When(Event)	40
2.2.4	Then(Action).....	40
2.2.5	Examples	40
2.2.5.1	Logging	41
2.2.6	Further Examples	42
2.2.6.1	Logging	42
2.2.7	Whitelist.....	43
2.2.7.1	Setup AllowDeserialPrivileges Flag.....	43
2.2.7.2	Examples	43
2.3	ARMR DNS Rule (22.x.x).....	43

2.3.1	Overview	43
2.3.2	Given (Condition)	44
2.3.3	Then (Action)	44
2.3.4	Examples	44
2.3.4.1	Logging	45
2.3.5	Further Examples	45
2.3.5.1	Logging	46
2.4	ARMR Library Rule (22.x.x)	46
2.4.1	Overview	46
2.4.2	Given (Condition)	46
2.4.3	When (Action)	47
2.4.4	Examples	48
2.4.5	Further Examples	48
2.4.5.1	Logging	49
2.5	ARMR Filesystem Rule (22.x.x)	51
2.5.1	Path Traversal Security Feature (22.x.x)	51
2.5.1.1	Overview	51
2.5.1.2	Given (Condition)	52
2.5.1.3	When (Event)	52
2.5.1.4	Then (Action)	53
2.5.1.5	Example	53
2.5.1.6	Further Examples	54
2.5.2	File I/O Security Feature (22.x.x)	57
2.5.2.1	Overview	57
2.5.2.2	When (Event)	57
2.5.2.3	Then (Action)	58
2.5.2.4	Examples	59
2.5.2.5	Further Examples	59
2.6	ARMR Process Rule (22.x.x)	64
2.6.1	Overview	64
2.6.2	When (Event)	64
2.6.3	Then (Action)	65
2.6.4	Examples	66
2.6.4.1	Logging	66
2.6.5	Further Examples	67


2.6.5.1	Logging	67
2.7	ARMR Sanitization Rule (22.x.x)	71
2.7.1	Overview	71
2.7.2	Given (Condition)	72
2.7.3	When (Event)	73
2.7.4	Then (Action)	74
2.7.5	Examples	74
2.7.5.1	Basic Single Rule Configuration	74
2.7.5.2	Advanced Multiple Rule Configuration	75
2.7.6	Logging	76
2.7.6.1	Protect Mode	76
2.7.6.2	Detect Mode	76
2.7.6.3	Safe Undetermined	77
2.8	ARMR Socket Rule (22.x.x)	77
2.8.1	Socket Control Security Feature (22.x.x)	77
2.8.1.1	Overview	77
2.8.1.2	Given (Condition)	78
2.8.1.3	Then (Action)	79
2.8.1.4	Examples	80
2.8.1.5	Further Examples	81
2.8.2	Secure Sockets (22.x.x)	82
2.8.2.1	Overview	82
2.8.2.2	When (Event)	83
2.8.2.3	Then (Action)	83
2.8.2.4	Examples	83
2.8.3	TLS upgrade (22.x.x)	84
2.8.3.1	Overview	84
2.8.3.2	When (Event)	85
2.8.3.3	Then (Action)	85
2.8.3.4	Examples	85
2.9	ARMR SQL Rule (22.x.x)	85
2.9.1	Overview	85
2.9.2	Given (Conditions)	86
2.9.3	When (Event)	87
2.9.4	Then (Action)	87

2.9.5	Example	88
2.9.5.1	Logging	89
2.9.6	Further Examples	89
2.9.6.1	Logging	89
2.10	ARRM HTTP rule (22.x.x).....	91
2.10.1	CSRF Security Feature (22.x.x).....	91
2.10.1.1	Overview	91
2.10.1.2	Given (Condition)	93
2.10.1.3	When (Event)	93
2.10.1.4	Then (Action)	94
2.10.1.5	Examples	95
2.10.1.6	Further Examples	96
2.10.2	HTTP Header Injection Security Feature (22.x.x).....	98
2.10.2.1	Overview	98
2.10.2.2	Given (Condition)	99
2.10.2.3	When (Event)	99
2.10.2.4	Then (Action)	100
2.10.2.5	Examples	100
2.10.3	Open Redirect Security Feature (22.x.x)	101
2.10.3.1	Overview	101
2.10.3.2	Given (Conditions)	101
2.10.3.3	When (Event)	102
2.10.3.4	Then (Action)	103
2.10.3.5	Examples	103
2.10.3.6	Further Examples	104
2.10.4	XSS Security Feature (22.x.x)	106
2.10.4.1	Overview	106
2.10.4.2	Given (Condition)	106
2.10.4.3	When (Event)	108
2.10.4.4	Then (Action)	109
2.10.4.5	Examples	109
2.10.4.6	Further Examples	109
2.10.5	Input Validation Security Feature (22.x.x)	111
2.10.5.1	Overview	111
2.10.5.2	Given (Condition)	111

2.10.5.3 When (Event)	112
2.10.5.4 Then (Action)	113
2.10.5.5 Examples	114
2.10.5.6 Further examples	114
2.10.6 HTTP/HTTPS Response Header Addition Feature (22.x.x)	117
2.10.6.1 Overview	117
2.10.6.2 Given (Condition)	118
2.10.6.3 Then (Action)	119
2.10.6.4 Examples	119
2.10.7 Session Fixation Security Feature (22.x.x)	123
2.10.7.1 Overview	123
2.10.7.2 Given (Condition)	124
2.10.7.3 When (Event)	124
2.10.7.4 Then (Action)	124
2.10.7.5 Example	124
2.10.8 HTTP Verb Tampering (22.x.x)	125
2.10.8.1 Overview	125
2.10.8.2 Given (Condition)	125
2.10.8.3 When (Event)	125
2.10.8.4 Then (Action)	126
2.10.8.5 Examples	126
2.10.8.6 Further Examples	127

The latest release of the Waratek Upgrade and Secure agent introduces the ARMR v2 platform. This replaces the legacy rule syntax in previous Waratek Secure and Upgrade versions.

The content below will introduce the ARMR platform, and core concepts for understanding the AMR syntax. Following this is a chapter on each specific rule, with examples and log output etc.

 Note : the version of ARMR in the examples below ranges from v2.0 to v2.2 - this will change in future versions of Waratek products so please consult Client Services for the appropriate ARMR version to use when deploying Waratek Secure and Upgrade.

1 The ARMR 2.0 Platform (22.x.x)

The ARMR 2.0 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	Abstract Rule Modelling Runtime.
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 is defined by Waratek and its partners and is supported by all of Waratek's ARMR Agents. 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 Waratek's ARMR Agents are designed to attach to the underlying runtime at bootup. Late (dynamic) attachment of Waratek ARMR Agents is not presently supported. Once a Waratek ARMR 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. Waratek also provides an optional Management Console appliance for managing ARMR Mod configuration and deployment via a web-based interface. Documentation for the Waratek Management Console is available separately.

1.1 ARMR Language Syntax (22.x.x)

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 Mods. ARMR Mods use a file extension of .armr; the content of the file is plain-text and can be written/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/2.0")

  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.

1.1.1 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
```

1.1.2 Statement

All statements must abide by the following rules,

- A statements is a declaration that is not open or closed.
- Notice that 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 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/2.0")
```

1.1.3 Types Of Values

Values can be of 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. example: `{"personal data form", id: 32, name: "John"}`.
- **list**: a collection of values of the same type, separated by commas, and encapsulated within square brackets. example: `[2, 3, 32, 100]`.
- A **dictionary** and a **list** value can contain other dictionaries and lists inside them.

1.1.4 Comments

It is possible to include comments in an ARMR File. There are two scenarios to mention,

- The hash symbol (also know 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 ARMR 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
```

1.1.5 Escape Character

Any character is legal as part of the string, even 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 behaviour.

Valid	Invalid
"hello\\ world\\"	"hello world\"
	"hello w"rld"
	"hello w\\"orld"

The following ARMR App name is valid,

```
app("App name with \\ and \"):
endapp
```

The following ARMR App name is invalid,

```
app("App name ending slash\"):
endapp
```

1.1.6 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 recommend that users do not write rules in this fashion as this will make it extremely difficult to read.

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

EnDapp
```

1.2 ARMR Mod (22.x.x)

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.

1.2.1 Requires

The `requires()` statement provides details about what is required for the ARMR Mod to run. It must be the first statement within the `app`, otherwise the rules 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.

1.2.2 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 version "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.

1.2.3 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.

1.2.4 ARMR Mod Example

The following example shows a well formatted ARMR Mod, so long as at least one ARMR Rule is contained within the ARMR Mod. Please consult the **ARMR Rule** section for more information on available rule types.

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

endapp
```

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

```
app("Security Policy"):

endapp
```

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

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

1.3 ARMR Rule (22.x.x)

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.

1.3.1 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.

1.3.2 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.

1.3.3 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.


1.3.4 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.

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).

1.3.5 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
```

1.3.6 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
```

1.3.7 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
--------	--

- i** 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 procid=52928 dvchost=localhost.localdomain rt=Mar 10
2020 14:51:34.493 +0000 appVersion=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 procid=52928 dvchost=localhost.localdomain rt=Mar 10
2020 14:51:34.494 +0000 appVersion=1
```

1.3.8 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-2019-2
933 :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
```

2 ARMR Rules (22.x.x)

This section contains a detailed description of each ARMR rule type and how to configure it.

2.1 ARMR Patch Rule (22.x.x)

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.

2.1.1 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.

2.1.2 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.

2.1.3 Then (Action)

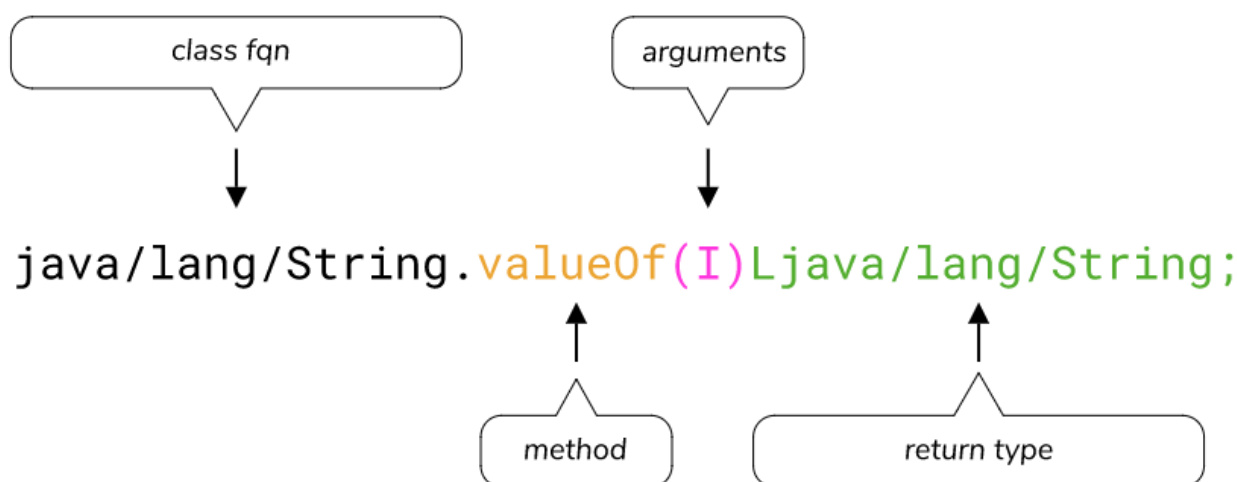
Unlike the other ARMR rules that have various declarative actions like `detect`, `protect`, `deny`, etc; a Patch rule must provide its intended action as a code block of supplied source-code. The code-block is specified by means of the `code` keyword and terminated with the `endcode` keyword. When the defined event conditions of a given Patch rule are triggered, the specified code statement will be compiled and executed at the specified patch location.

2.1.4 Runtime Notation

In order to target events, the `function` and **location-specifier** need to be declared using the internal signature notation used by the machine running the instructions. In the case of Java, this is the Java Virtual Machine (JVM). An event is relative to the function of a particular namespace. Using Java as an example, consider the following line of code.

```
String.valueOf(8);
```

The invocation of the method `valueOf()` on the `String` class would actually appear differently written in the JVMs internal form. The following example shows how it would appear.



Part Name	Part	Description
Class	<code>java/lang/String</code>	The fully qualified name FQN of the class that contains the targeted method.
Method	<code>valueOf</code>	The method name that needs to be targeted. Overloaded methods will have different arguments.
Arguments	<code>(I)</code>	It is important to target the specific method by specifying the correct arguments, in the order they are expected.
Return Type	<code>Ljava/lang/String;</code>	The return type is always declared at the end of the signature.
Descriptor	<code>(I)Ljava/lang/String;</code>	The descriptor is a combination of the arguments and the return type.

2.1.4.1 Java Types

Type	Internal	Example	Default Value	Size	Frame Slot Allocation
object	<code>L<type></code>	<code>Ljava/lang/String;</code>	<code>null</code>	16 bytes minimum	1
boolean	<code>Z</code>		<code>false</code>	1 bit	1

Type	Internal	Example	Default Value	Size	Frame Slot Allocation
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

2.1.4.2 More JVM Internal Form Examples

```

java/lang/String.toUpperCase()Ljava/lang/String;
java/lang/Class.forName(Ljava/lang/String;)Ljava/lang/Class;
java/io/File.setReadable(Z)Z
java/util/Hashtable.<init>(I)V
com/sun/crypto/provider/DESKey.getEncoded()[B
    
```

2.1.5 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")
```

2.1.6 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.

2.1.6.1 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()	<code>error("java/io/IOException")</code>	Apply the patch to every exception which propagates from the targeted function.

2.1.6.2 INSTRUCTION / LINE

Location	Example	Description
<code>instruction()</code>	<code>instruction(391)</code>	Apply the patch immediately before the bytecode instruction at the specified instruction offset of the target function instruction stream.
<code>line()</code>	<code>line(12)</code>	Trigger the patch immediately before the instruction at the specified source code line number.

2.1.6.3 READ / READSITE / READRETURN

Location	Example	Description
<code>read()</code>	<code>read("java/io/File.path")</code>	Apply the patch in place of the memory read instruction for the specified memory field.
<code>readsite()</code>	<code>readsite("java/io/File.path")</code>	Apply the patch immediately before the memory read instruction for the specified memory field.
<code>readreturn()</code>	<code>readreturn("java/io/File.path")</code>	Apply the patch immediately after the memory read instruction for the specified memory field.

2.1.6.4 WRITE / WRITESITE / WRITERETURN

Location	Example	Description
<code>write()</code>	<code>write("java/io/File.path")</code>	Apply the patch in place of the memory write instruction for the specified memory field.
<code>writesite()</code>	<code>writesite("java/io/File.path")</code>	Apply the patch immediately before the memory write instruction for the specified memory field.

Location	Example	Description
<code>writereturn()</code>	<code>writereturn("java/io/ File.path")</code>	Apply the patch immediately after the memory write instruction for the specified memory field.

2.1.6.5 CALL / CALLSITE / CALLRETURN

Location	Example	Description
<code>call()</code>	<code>call("java/lang/String.valueOf(I) Ljava/lang/String;")</code>	Apply the patch in place of the invoke instruction for the specified method.
<code>callsite()</code>	<code>callsite("java/lang/String.valueOf(I) Ljava/lang/String;")</code>	Apply the patch immediately before the invoke instruction for the specified method.
<code>callreturn()</code>	<code>callreturn("java/lang/String.valueOf(I) Ljava/lang/String;")</code>	Apply the patch immediately after the invoke instruction for the specified method.

2.1.7 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.

2.1.7.1 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
<code>public void load();</code>	optional	The <code>load()</code> method will be invoked once the <code>link</code> life-cycle event is triggered. Since this is a once-off event, the <code>load</code> method is useful for the initialization of the state.
<code>public void patch(JavaFrame frame);</code>	mandatory	The <code>patch()</code> method will be invoked once the <code>execute</code> life-cycle event is triggered. This event can happen multiple times. Every patch must implement the <code>patch()</code> method.
<code>public void unload();</code>	optional	The <code>unload()</code> method will be invoked once the <code>unlink</code> life-cycle event is triggered. As the <code>load()</code> event, this is also a once-off event.

2.1.7.2 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
<code>saveValue(Object key, Object value)</code>	Store an object into the shared cache with a unique key.
<code>restoreValue(Object key)</code>	Retrieve an object stored into the shared cache by passing in the key.

2.1.7.3 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: <https://docs.oracle.com/javase/specs/jvms/se8/html/jvms-2.html>¹

2.1.7.4 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.

¹ <https://docs.oracle.com/javase/specs/jvms/se8/html/jvms-2.html#jvms-2.6>

```

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

```

2.1.8 Patch Rule Example

Consider the following Java source code.

```

package ie.example;

public class Utils {

    public byte[] createByteArray(int length) {
        return new byte[length];
    }

}

```

The Java bytecode for the method `createByteArray()` can be seen here.

```

public byte[] createByteArray(int);
descriptor: (I)[B
flags: ACC_PUBLIC
Code:
  stack=1, locals=2, args_size=2
    0: iload_1
    1: newarray      byte
    3: areturn
LineNumberTable:
  line 4: 0

```

Now consider the case where a source-code change was introduced to check whether the integer argument called `length` is a positive integer and that it does not exceed a size of 100 before creating the `byte[]`, throwing an `IllegalStateException` if either of these conditions are not met. Here is what the new source-code would look like for the `createByteArray()` method.

```

package ie.example;

public class Utils {

    public byte[] createByteArray(int length) {
        if (length < 0 || length > 100) {
            throw new IllegalStateException("Length must be a positive integer and cannot exceed a size
of 100");
        }
        return new byte[length];
    }
}

```

To apply the same effect with an ARMR Patch rule is trivial. To do so, we can create an ARMR Patch rule that targets the `createByteArray()` method, at the **entry** location, to be applied before instruction 0 is executed. At this location, the ARMR Patch will have access to the `length` argument from the local variable array, and can perform the same check conditions, raising an `IllegalStateException` if either of the conditions are not met. Below is an example ARMR Patch to provide this behavior.

- **Function:** "ie/example/Utils.createByteArray()[B"
- **Location Specifier:** entry()

```

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

  patch("Example Patch")
    function("ie/example/Util.createByteArray(I)[B")
      entry()

      code(language: java):
        public void patch(JavaFrame frame) {
          int length = frame.loadIntVariable(1);
          if (length < 0 || length > 100) {
            frame.raiseException(new IllegalStateException("Length must be a positive integer
and cannot exceed a size of 100"));
          }
        }
      endcode
    endpatch
  endapp

```

The above ARMR App declares a single ARMR Patch rule. The rule has the following statements.

- function
 - the signature of the method which contains the code to be patched
- location-specifier
 - the specific location within the function where the patch should be applied
- code
 - the code to be compiled into the target function at the specified location

2.1.9 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
<code>read()</code>	<code>read("java/io/File.path", occurrences: [2])</code>	Apply the patch by replacing only the 2nd occurrence of the <code>getfield</code> bytecode instruction of the <code>path</code> field.

Location	Example	Description
readsite()	readsite("java/io/File.path", occurrences: [3, 5])	Apply the patch immediately before the 3rd and 5th occurrence of the getfield bytecode instruction of the path field.
readreturn()	readreturn("java/io/File.path", occurrences: [4, 6])	Apply the patch immediately after the 4th and 6th occurrence of the getfield bytecode instruction of the path field.
write()	write("java/io/File.path", occurrences: [1, 7])	Apply the patch by replacing the 1st and 7th occurrence of the putfield bytecode instruction of the path field.
writesite()	writesite("java/io/File.path", occurrences: [2])	Apply the patch immediately before the 2nd occurrence of the putfield bytecode instruction of the path field.
writereturn())	writereturn("java/io/File.path", occurrences: [4, 6])	Apply the patch immediately after the 4th and 6th occurrence of the putfield bytecode instruction of the path field.
call()	call("java/lang/String.valueOf(I) Ljava/lang/String;", occurrences: [2])	Apply the patch by replacing only the 2nd occurrence of the invoke* bytecode instruction of the valueOf method.
callsite()	callsite("java/lang/String.valueOf(I) Ljava/lang/String;", occurrences: [3, 5])	Apply the patch immediately before the 3rd and 5th occurrence of the invoke* bytecode instruction of the valueOf method.

Location	Example	Description
callreturn()	<pre>callreturn("java/lang/ String.valueOf(I) Ljava/lang/String;", occurrences: [4, 6])</pre>	Apply the patch immediately after the 4th and 6th occurrence of the <code>invoke*</code> bytecode instruction of the <code>valueOf</code> method.

Consider the following example.

```
package com.example;

class Person {
    private int age;

    public Person(int age) {
        this.age = age;
    }

    @Override
    public String toString() {
        if (age == 0) {
            return "This person has no age.";
        }
        return "This person is " + age + " years old.";
    }
}
```

The following bytecode is for the `toString()` method as shown in the `Person` class.

```

public java.lang.String toString();
  descriptor: ()Ljava/lang/String;
  flags: ACC_PUBLIC
  Code:
    stack=2, locals=1, args_size=1
      0: aload_0
      1: getfield      #2 // Field age:I
      4: ifne         10
      7: ldc          #3 // String This person has no age.
      9: areturn
     10: new          #4 // class java/lang/StringBuilder
     13: dup
     14: invokespecial #5 // Method java/lang/StringBuilder."<init>":()V
     17: ldc          #6 // String This person is
     19: invokevirtual #7 // Method java/lang/StringBuilder.append:(Ljava/lang/String;)Ljava/lang/
StringBuilder;
     22: aload_0
     23: getfield      #2 // Field age:I
     26: invokevirtual #8 // Method java/lang/StringBuilder.append:(I)Ljava/lang/StringBuilder;
     29: ldc          #9 // String years old.
     31: invokevirtual #7 // Method java/lang/StringBuilder.append:(Ljava/lang/String;)Ljava/lang/
StringBuilder;
     34: invokevirtual #10 // Method java/lang/StringBuilder.toString:()Ljava/lang/String;
     37: areturn

```

As shown here, the age field is being read at two different locations. The `getfield` bytecode instruction is called at instruction 1 and 23. If the ARMR developer is interested in only targeting the second `getfield` instruction, then they can use one of the read location specifiers, and pass in an occurrence of 2.

```

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

  patch("Readsite patch")
    function("com/example/Person.toString()Ljava/lang/String")
      readsite("com/example/Person.age", occurrences: [2])

        code(language: java):
          public void patch(JavaFrame frame) {
            // patch code here
          }
        endcode
      endpatch
    endapp

```

Whenever an ARMR developer specifies an occurrence that does not exist, that specific occurrence is ignored but others will still trigger the rule. For example, "occurrences: [2]" and "occurrences: [2,3]" would produce the same effect in the above case. However, if all the occurrences specified are **out of bounds**, then the patch code cannot be applied. For such cases, a link error message is generated in the CEF log file specifying the maximum event count and the set of configured occurrences of the patch. Consider the following **readsite** specifier.

```
readsite("com/example/Person.age", occurrences: [3])
```

Since there is no third occurrence of the `getField` instruction for the `age` field, the patch cannot be applied. As a result, a log message will be printed to the CEF log file.

```
<14>1 2020-07-09T04:01:00.321Z win_system_1 java 18914 - - CEF:0|ARMR:example app|example app|2.2|rule 2|
Link Rule|Low|rt=Jul 09 2020 04:01:00.307 +0000 dvchost=win_system_1 procid=18914 outcome=failure
reason=occurrences for patch [3] exceed the maximum occurrence count 2
```

2.1.10 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 outcome=failure
appVersion=1
```

2.1.11 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 necessarily 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.

2.1.12 JavaFrame API

2.1.12.1 The 'this' Variable

Returns the `this` instance for non-static functions.

```
Object loadThisVariable()
```

2.1.12.2 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)
```

2.1.12.3 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)
```

2.1.12.4 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);
```

2.1.12.5 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);

```

2.1.12.6 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);

```

2.1.12.7 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);

```


2.2 ARMR Deserial Rule (22.x.x)


2.2.1 Overview

Deserializing untrusted data can cause the JVM to instantiate any class 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. Serialization is used in several components of the JVM as well as in numerous third-party frameworks and dependencies.

The Deserial rule addresses the deserialization attacks by lowering the system privileges during deserialization. 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. Please refer to another document "Deserialization White Paper" for more details on Waratek approach to deserial mitigation.

The Deserial rule 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 Deserial rule can be used safely and proactively 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

2.2.2 Given(Condition)

deserialize	<p>The keyword <code>deserialize</code> is a required component in the <code>marshal</code> rule. <code>java</code> and <code>dotnet</code> are the only parameters accepted.</p> <pre>deserialize(java, dotnet)</pre>
-------------	--

2.2.3 When(Event)

One of `rce` or `dos` must be declared in a `marshal` rule. Only one of these can exist in a `marshal` rule and neither accept any parameter.

rce	Remote Code Execution
dos	Denial of Service

2.2.4 Then(Action)

The action component is a required component in the `marshal` rule. There are two available actions: `protect` and `detect`. 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`, `Med`, `High` or `Very-High`(case insensitive). The default severity is unknown.

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. A log message is generated with details of all attempts to deserial.

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.

2.2.5 Examples

Protecting the Java application from the `dos` attack.

```

app("myapp"):
requires(version: ARMR/2.3)
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/2.3)
marshal("protect the app from remote-code-execution attack"):
  deserialize(dotnet)
  rce()
  protect(message: "the logging message", severity: Low)
endmarshal
endapp

```

2.2.5.1 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:Walter|Walter|2.3|MarshalRule|
Execute Rule|High|rt=Mar 24 2021 10:14:24.053 +0000 dvchost=jenkins-qa-slave-centos.aws.waratek.lan
procid=9699 appVersion=1 act=protect msg=Walter limit=100000 reason=CWE-400: Uncontrolled CPU
consumption via API abuse methodName=java.util.EnumMap.hashCode()

```

- rce

```

<10>1 2020-09-10T00:24:50.513Z userX_system java 29417 - - CEF:0|ARMR:Walter|Walter|2.2|MarshalRule|
Execute Rule|High|rt=Sep 10 2020 00:24:50.512 +0000 dvchost=jenkins-qa-slave-centos.aws.waratek.lan
procid=29417 act=protect msg=Walter limit=100000 reason=CWE-400: Uncontrolled CPU consumption via
API abuse methodName=java.util.Hashtable.hashCode()

```

```

<10>1 2021-03-22T12:24:53.327Z userX_system java 28013 - - CEF:0|ARMR:Walter|Walter|2.3|MarshalRule|
Execute Rule|High|rt=Mar 22 2021 12:24:53.326 +0000 dvchost=jenkins-qa-slave-centos.aws.waratek.lan
procid=28013 appVersion=1 act=protect msg=Walter methodName=java.lang.Runtime.exec()
httpRequestUri=/objectinputstream-deserial/examples/deserial-PM-59-test.jsp remoteIpAddress=127.0.0.
1 httpSessionId=D194C19D465595307BBD2F04F5F7B632

```



The second example above has extra CEF extensions for httpRequestUri, remoteIpAddress and httpSessionId.

2.2.6 Further Examples

Protecting the Java application from the dos attack with the stacktrace also logged.

```
app("Mod for Marshal dos Rule"):
  requires(version: ARMR/2.3)
  marshal("Marshal dos Rule"):
    deserialize(dotnet, java)
    dos()
    protect(message: "Testing Marshal dos Rule", severity: Very-High, stacktrace: "full")
  endmarshal
endapp
```

Protecting the Java application from rce attack with the stacktrace also logged.

```
app("Walter"):
  requires(version: ARMR/2.3)
  marshal("Marshal sys Rule"):
    deserialize(java)
    rce()
    protect(message: "Walter", severity: High, stacktrace: "full")
  endmarshal
endapp
```

2.2.6.1 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|2.3|Marshal dos Rule|Execute Rule|Very-High|rt=Mar 31 2021 15:38:48.278 +0100
  dvchost=ckang-XPS-15-9570 procid=104596 appVersion=1 act=protect msg=Testing Marshal dos Rule
  stacktrace=java.util.AbstractSet.hashCode(AbstractSet.java)
  \ndeserialjar.runners.OverwrittenReadObject.abstractSetHashCode(OverwrittenReadObject.java:434)\ndeserialja
  r.runners.OverwrittenReadObject.invokeMethod(OverwrittenReadObject.java:375)\ndeserialjar.runners.Overwritt
  enReadObject.readObject(OverwrittenReadObject.java:57)\nsun.reflect.NativeMethodAccessorImpl.invoke0(Native
  Method)\nsun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)\nsun.reflect.Delegat
  ingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)\njava.lang.reflect.Method.invoke(Method.
  java:498)\njava.io.ObjectStreamClass.invokeReadObject(ObjectStreamClass.java:1170)\njava.io.ObjectInputStre
  am.readSerialData(ObjectInputStream.java:2232)\njava.io.ObjectInputStream.readOrdinaryObject(ObjectInputStr
  eam.java:2123)\njava.io.ObjectInputStream.readObject0(ObjectInputStream.java:1624)\njava.io.ObjectInputStre
  am.readObject(ObjectInputStream.java:464)\njava.io.ObjectInputStream.readObject(ObjectInputStream.java:422)
  \ndeserialjar.runners.OverwrittenReadObjectRunner.run(OverwrittenReadObjectRunner.java:32)\nMain.main(Main.
  java:63) limit=100000 reason=CWE-400: Uncontrolled CPU consumption via API abuse
  methodName=java.util.AbstractSet.hashCode()
```

```
<10>1 2021-03-31T15:51:32.787+01:00 userX_system java 105393 - - CEF:0|ARMR:Walter|Walter|2.3|Marshal sys
Rule|Execute Rule|High|rt=Mar 31 2021 15:51:32.785 +0100 dvchost=ckang-XPS-15-9570 procid=105393
  appVersion=1 act=protect msg=Walter
  stacktrace=deserialjar.runners.OverwrittenReadObject.invokeMethod(OverwrittenReadObject.java:103)\ndeserial
jar.runners.OverwrittenReadObject.readObject(OverwrittenReadObject.java:57)\nsun.reflect.NativeMethodAccess
orImpl.invoke0(Native Method)\nsun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62
)\nsun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)\njava.lang.reflect
.Method.invoke(Method.java:498)\njava.io.ObjectStreamClass.invokeReadObject(ObjectStreamClass.java:1170)\nj
ava.io.ObjectInputStream.readSerialData(ObjectInputStream.java:2232)\njava.io.ObjectInputStream.readOrdinar
yObject(ObjectInputStream.java:2123)\njava.io.ObjectInputStream.readObject0(ObjectInputStream.java:1624)\nj
ava.io.ObjectInputStream.readObject(ObjectInputStream.java:464)\njava.io.ObjectInputStream.readObject(Objec
tInputStream.java:422)\ndeserialjar.runners.OverwrittenReadObjectRunner.run(OverwrittenReadObjectRunner.jav
a:32)\nMain.main(Main.java:63) methodName=java.lang.Runtime.exec()
```

2.2.7 Whitelist

In the rare case where the deserial rule must allow specific privileges in certain environments, an optional waratek property `com.waratek.AllowDeserialPrivileges` can be used to whitelist specific deserial privileges.

2.2.7.1 Setup AllowDeserialPrivileges Flag

1. Open the `<absolute path to waratek agent>/conf_*/waratek.properties` file.
2. Add the following flag and make an adjustment according to the real-world requirement.

```
com.waratek.AllowDeserialPrivileges="// comma-separated values."
```

2.2.7.2 Examples

Whitelist `java.lang.SecurityManager.<init>()`

```
com.waratek.AllowDeserialPrivileges="java.lang.SecurityManager.<init>()"
```

Whitelist `java.lang.SecurityManager.<init>()` and `java.lang.System.getenv()`

```
com.waratek.AllowDeserialPrivileges="java.lang.SecurityManager.<init>(), java.lang.System.getenv()"
```

2.3 ARMR DNS Rule (22.x.x)

2.3.1 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.

2.3.2 Given (Condition)

<code>lookup</code>	<p>The <code>lookup</code> takes a single parameter (string literal) where valid values are a quoted-hostname, a quoted-IPv4 address, or the constant <code>any</code> indicating any hostname or IPv4 address.</p> <div style="border: 1px solid #ccc; padding: 5px; margin: 10px 0;"> <pre>lookup("waratek.com") lookup("127.0.0.1") lookup(any)</pre> </div> <p>IPv6 addresses are not currently supported.</p>
---------------------	--

2.3.3 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.

<code>protect</code>	The DNS lookup is not allowed to proceed. If configured, a log message is generated with details of the event.
<code>detect</code>	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.
<code>allow</code>	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.

2.3.4 Examples

DNS rule with quoted-hostname.

```

app("DNS lookup mod"):
  requires(version: ARMR/2.3)
  dns("Blocking address resolution for waratek.com"):
    lookup("waratek.com")
    protect(message: "dns lookup occurred for waratek.com", severity: 8)
  enddns
endapp

```

DNS rule with quoted-IPv4 address.

```

app("DNS lookup mod"):
  requires(version: ARMR/2.3)
  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/2.3)
  dns("Detecting address resolution for any host/ip"):
    lookup(any)
    detect(message: "dns lookup event", severity: 4)
  enddns
endapp

```

2.3.4.1 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|Walter|2.3|DNS Test App
detect|Execute Rule|High|rt=Mar 22 2021 12:58:06.135 +0000 dvchost=jenkins-qa-slave-centos.aws.waratek.lan
procid=17522 appVersion=1 act=detect msg=Walter hostname=waratek.com

```

2.3.5 Further Examples

DNS rule with the stacktrace also logged.

```

app("DNS lookup mod"):
  requires(version: ARMR/2.3)
  dns("Detecting address resolution for localhost"):
    lookup("any")
    protect(message: "dns lookup event", severity: 9, stacktrace: "full")
  enddns
endapp

```

2.3.5.1 Logging

```


<10>1 2021-04-01T12:31:39.637+01:00 userX_system java 174476 - - CEF:0|ARMR:Walter|Walter|2.3|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 act=protect msg=dns lookup event
  stacktrace=walter.apps.DNSLookupApp.main(Container-1)(DNSLookupApp.java:94)\nsun.reflect.NativeMethodAccess
orImpl.invoke0(Native Method)\nsun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62
)\nsun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)\njava.lang.reflect
.Method.invoke(Method.java:498)\njava.lang.Thread.run(Container-1)(Thread.java:876)\njava.lang.Thread.begin
(Container-1)(Thread.java:897)\njava.lang.Thread.invokeRun(Container-1)(Thread.java:883)\njava.lang.Thread$
ThreadHandler.invokeRun(Container-1)(Thread.java:55) hostname=alto.aws.waratek.lan

```

2.4 ARMR Library Rule (22.x.x)


2.4.1 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 Waratek Upgrade.


2.4.2 Given (Condition)

To control native library loading using the ARMR library rule the user must specify the load declaration.

load	<p>A parameter must be supplied to the <code>load</code> declaration to determine the libraries to which the ARMR <code>library</code> rule will control loading.</p> <div style="border: 1px solid #ccc; padding: 5px; margin: 10px 0;"> <p> Both Unix and Windows filesystem paths are supported</p> </div> <p>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.</p> <p>Each string represented in the parameter can be:</p> <ul style="list-style-type: none"> • a single library name - the agent will control access to any library on the filesystem that matches the given name • an absolute path to a specific library <p>The wildcard character (*) is supported anywhere in the library name or path:</p> <ul style="list-style-type: none"> • only one wildcard character can be used with each path • the wildcard will only wildcard a single directory • the wildcard can be used to specify all libraries with a specific prefix • the wildcard character specified on its own represents all native libraries on the filesystem
------	---

2.4.3 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	<p>Monitoring mode: the application behaves as normal. Any attempt to load a native library specified by the ARMR <code>library</code> rule is allowed, and a log message is generated with details of the event.</p> <div style="border: 1px solid #ccc; padding: 5px; margin: 10px 0;"> <p> A log message must be specified with this action.</p> </div>
allow	Can be used to allow loading of specific libraries which are a subset of protected libraries covered by an ARMR <code>library</code> rule in <code>protect</code> 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.

2.4.4 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/2.3)
  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/2.3)
  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|2.3|
Prevent loading of all native libraries in specific directory|Execute Rule|High|rt=Mar 31 2021 10:52:42.102
+0100 dvchost=userX_system procid=6229 appVersion=1 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|2.3|
Prevent loading of all native libraries in specific directory|Execute Rule|High|rt=Mar 30 2021 16:56:46.512
+0100 dvchost=userX_system procid=4349 appVersion=1 act=protect msg=Blocked attempt to load library
path=C:\\Windows\\Counter.dll
```

2.4.5 Further Examples

As above, with the stacktrace also logged

Unix

```
app("Library mod - with stacktrace"):
  requires(version: ARMR/2.3)
  library("Prevent loading of all native libraries in specific directory"):
    load("/tmp/*")
    protect(message: "Blocked attempt to load library", severity: High, stacktrace: "full")
  endlibrary
endapp
```

Windows

```
app("Library mod - with stacktrace"):
  requires(version: ARMR/2.3)
  library("Prevent loading of all native libraries in specific directory"):
    load("C:\Windows\*")
    protect(message: "Blocked attempt to load library", severity: High, stacktrace: "full")
  endlibrary
endapp
```

2.4.5.1 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|2.3|Prevent loading of all native libraries in specific directory|Execute
Rule|High|rt=Apr 01 2021 12:10:21.282 +0100 dvchost=userX_system procid=27607 appVersion=1 act=protect
msg=Blocked attempt to load library
stacktrace=com.waratek.jvi.RuntimeSystemEnv.load0(RuntimeSystemEnv.java:175)\njava.lang.System.loadLibrary(
Container-1)(System.java)\nCounter.<clinit>(Container-1)(Counter.java:17)\nsun.reflect.NativeMethodAccessor
Impl.invoke0(Native Method)\nsun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)\
nsun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)\njava.lang.reflect.M
ethod.invoke(Method.java:498)\njava.lang.Thread.run(Container-1)(Thread.java:876)\njava.lang.Thread.begin(C
ontainer-1)(Thread.java:897)\njava.lang.Thread.invokeRun(Container-1)(Thread.java:883)\njava.lang.Thread$Th
readHandler.invokeRun(Container-1)(Thread.java:55) path=/tmp/libCounter.so
```

Windows

```
<10>1 2021-04-01T12:09:43.442+01:00 userX_system java 25465 - - CEF:0|ARMR:Library mod - with stacktrace|
Library mod - with stacktrace|2.3|Prevent loading of all native libraries in specific directory|Execute
Rule|High|rt=Apr 01 2021 12:09:43.442 +0100 dvchost=userX_system procid=25465 appVersion=1 act=protect
msg=Blocked attempt to load library
stacktrace=com.waratek.jvi.RuntimeSystemEnv.load0(RuntimeSystemEnv.java:175)\njava.lang.System.loadLibrary(
Container-1)(System.java)\nCounter.<clinit>(Container-1)(Counter.java:17)\nsun.reflect.NativeMethodAccessor
Impl.invoke0(Native Method)\nsun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)\
nsun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)\njava.lang.reflect.M
ethod.invoke(Method.java:498)\njava.lang.Thread.run(Container-1)(Thread.java:876)\njava.lang.Thread.begin(C
ontainer-1)(Thread.java:897)\njava.lang.Thread.invokeRun(Container-1)(Thread.java:883)\njava.lang.Thread$Th
readHandler.invokeRun(Container-1)(Thread.java:55) path=C:\\Windows\\Counter.dll
```

Prevent loading a specific native library

Unix

```

app("Library mod 2"):
  requires(version: ARMR/2.2)
  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/2.2)
  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/2.2)
  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/2.2)
  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/2.2)

  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/2.2)

  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

```

2.5 ARMR Filesystem Rule (22.x.x)


2.5.1 Path Traversal Security Feature (22.x.x)

2.5.1.1 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.

2.5.1.2 Given (Condition)

The Path Traversal security feature is enabled using the ARMR `filesystem` rule. With this rule the user can specify a single condition - `input`.

<code>input</code>	<p>This allows the user to specify the source of the untrusted data. The following three sources are supported:</p> <ul style="list-style-type: none"> • <code>http</code> data introduced via HTTP/HTTPS requests • database data introduced via JDBC connections • <code>deserialization</code> data introduced via Java or XML deserialization <p>The rule will trigger if the source of the untrusted data matches that specified in the rule.</p> <p>If no value is specified then a default value of <code>http</code> is used.</p> <p>An exception will be thrown if an unsupported value is provided.</p>
--------------------	--

 This rule provides protection **only** when user input is received via an API that is enabled in the `input` declaration of the rule.

2.5.1.3 When (Event)

<code>traversal</code>	<p>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:</p> <ul style="list-style-type: none"> • <code>relative</code> • <code>absolute</code> <p>Each rule may contain a single protection type.</p> <p>If no value is specified then by default protection will be enabled for both <code>relative</code> and <code>absolute</code> path traversal attacks.</p>
------------------------	--

2.5.1.4 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.

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.

2.5.1.5 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/2.3)
  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|2.3|Protect against relative and absolute path traversal attacks|Execute Rule|High|rt=Mar 30 2021 17:31:15.234 +0100 dvchost=userX_system procid=32008 appVersion=1 act=protect msg=Path Traversal attack blocked path=/tmp/tomcat/webapps/spiracle/pathTraversal/testFilesParent/testFilesChild/../../TestFile httpSessionId=3153E581A645E2A54D3C12D3928473BC taintSource=HTTP_SERVLET httpRequestUri=/spiracle/FileServlet01 httpCookies=JSESSIONID\=3153E581A645E2A54D3C12D3928473BC remoteIpAddress=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|2.3|Protect against relative and absolute path traversal attacks|Execute Rule|High|rt=Mar 30 2021 17:32:30.903 +0100 dvchost=userX_system procid=32008 appVersion=1 act=protect msg=Path Traversal attack blocked path=/tmp/somefile.txt httpSessionId=3153E581A645E2A54D3C12D3928473BC taintSource=HTTP_SERVLET httpRequestUri=/spiracle/FileServlet03 httpCookies=JSESSIONID\=3153E581A645E2A54D3C12D3928473BC remoteIpAddress=0:0:0:0:0:0:1
```

2.5.1.6 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/2.3)
  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|2.3|Protect against relative and absolute path
traversal attacks|Execute Rule|High|rt=Apr 01 2021 11:37:24.201 +0100 dvchost=userX_system
procid=25024 appVersion=1 act=protect msg=Path Traversal attack blocked
stacktrace=com.waratek.spiracle.path_traversal.FileServlet01.doPost(FileServlet01.java:71)\njavax.se
rvlet.http.HttpServlet.service(HttpServlet.java:650)\njavax.servlet.http.HttpServlet.service(HttpSer
vlet.java:731)\nsun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
\nsun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)\nsun.reflect.Delegat
ingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)\njava.lang.reflect.Method.invoke(
Method.java:498)\norg.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilter
Chain.java:303)\norg.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.jav
a:208)\norg.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.java:52)\nsun.reflect.NativeMe
thodAccessorImpl.invoke0(Native Method)
\nsun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)\nsun.reflect.Delegat
ingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)\njava.lang.reflect.Method.invoke(
Method.java:498)\norg.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilter
Chain.java:241)\norg.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.jav
a:208)\norg.apache.catalina.core.StandardWrapperValve.invoke(StandardWrapperValve.java:218)\norg.apa
che.catalina.core.StandardContextValve.invoke(StandardContextValve.java:122)\norg.apache.catalina.au
thenticator.AuthenticatorBase.invoke(AuthenticatorBase.java:505)\norg.apache.catalina.core.StandardH
ostValve.invoke(StandardHostValve.java:169)\norg.apache.catalina.valves.ErrorReportValve.invoke(Erro
rReportValve.java:103)\norg.apache.catalina.valves.AccessLogValve.invoke(AccessLogValve.java:956)\no
rg.apache.catalina.core.StandardEngineValve.invoke(StandardEngineValve.java:116)\norg.apache.catalin
a.connector.CoyoteAdapter.service(CoyoteAdapter.java:442)\norg.apache.coyote.http11.AbstractHttp11Pr
ocessor.process(AbstractHttp11Processor.java:1082)\norg.apache.coyote.AbstractProtocol$AbstractConne
ctionHandler.process(AbstractProtocol.java:623)\norg.apache.tomcat.util.net.JIoEndpoint$SocketProces
sor.run(JIoEndpoint.java:318)\njava.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecuto
r.java:1149)\njava.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)\no
rg.apache.tomcat.util.threads.TaskThread$WrappingRunnable.run(TaskThread.java:61)\njava.lang.Thread.run(Th
read.java:748) path=/tmp/tomcat/webapps/spiracle/pathTraversal/testFilesParent/testFilesChild/./
TestFile httpSessionId=2912BD6B199C8B891244E63DC7DBCDE3 taintSource=HTTP_SERVLET httpRequestUri=/
spiracle/FileServlet01 httpCookies=JSESSIONID\=2912BD6B199C8B891244E63DC7DBCDE3 remoteIpAddress=0:0:0:0:0:0:0:0

```

- 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|2.3|Protect against relative and absolute path
traversal attacks|Execute Rule|High|rt=Apr 01 2021 12:02:26.627 +0100 dvchost=userX_system
procid=25024 appVersion=1 act=protect msg=Path Traversal attack blocked
stacktrace=com.waratek.spiracle.path_traversal.FileServlet03.doPost(FileServlet03.java:68)\njavax.se
rvlet.http.HttpServlet.service(HttpServlet.java:650)\njavax.servlet.http.HttpServlet.service(HttpSer
vlet.java:731)\nsun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
\nsun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)\nsun.reflect.Delegat
ingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)\njava.lang.reflect.Method.invoke(
Method.java:498)\norg.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilter
Chain.java:303)\norg.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.jav
a:208)\norg.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.java:52)\nsun.reflect.NativeMe
thodAccessorImpl.invoke0(Native Method)
\nsun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)\nsun.reflect.Delegat
ingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)\njava.lang.reflect.Method.invoke(
Method.java:498)\norg.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilter
Chain.java:241)\norg.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.jav
a:208)\norg.apache.catalina.core.StandardWrapperValve.invoke(StandardWrapperValve.java:218)\norg.apa
che.catalina.core.StandardContextValve.invoke(StandardContextValve.java:122)\norg.apache.catalina.au
thenticator.AuthenticatorBase.invoke(AuthenticatorBase.java:505)\norg.apache.catalina.core.StandardH
ostValve.invoke(StandardHostValve.java:169)\norg.apache.catalina.valves.ErrorReportValve.invoke(Erro
rReportValve.java:103)\norg.apache.catalina.valves.AccessLogValve.invoke(AccessLogValve.java:956)\no
rg.apache.catalina.core.StandardEngineValve.invoke(StandardEngineValve.java:116)\norg.apache.catalin
a.connector.CoyoteAdapter.service(CoyoteAdapter.java:442)\norg.apache.coyote.http11.AbstractHttp11Pr
ocessor.process(AbstractHttp11Processor.java:1082)\norg.apache.coyote.AbstractProtocol$AbstractConne
ctionHandler.process(AbstractProtocol.java:623)\norg.apache.tomcat.util.net.JIoEndpoint$SocketProces
sor.run(JIoEndpoint.java:316)\njava.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecuto
r.java:1149)\njava.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)\no
rg.apache.tomcat.util.threads.TaskThread$WrappingRunnable.run(TaskThread.java:61)\njava.lang.Thread.run(Th
read.java:748) path=/tmp/somefile.txt httpSessionId=2912BD6B199C8B891244E63DC7DBCDE3
taintSource=HTTP_SERVLET httpRequestUri=/spiracle/FileServlet03
httpCookies=JSESSIONID=2912BD6B199C8B891244E63DC7DBCDE3 remoteIpAddress=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/2.2)
  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/2.2)
  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/2.2)
  filesystem("Protect against relative and absolute path traversal attacks"):
    input(deserialization, http, database)
    traversal()
    protect()
  endfilesystem
endapp

```

2.5.2 File I/O Security Feature (22.x.x)

2.5.2.1 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)


2.5.2.2 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.

read	The user must specify either the <code>read</code> or the <code>write</code> declaration.
write	<p>A parameter must be supplied to the <code>read</code> or <code>write</code> declaration to determine the files and / or directories that the ARMR <code>filesystem</code> rule will control access to.</p> <div data-bbox="352 443 1417 533" style="border: 1px solid #ccc; padding: 5px; margin: 10px 0;"> <p> Both Unix and Windows filesystem paths are supported</p> </div> <p>This parameter takes the form of a list of one or more quoted-strings indicating specifically targeted files/directories.</p> <p>Each string represented in the parameter can be:</p> <ul style="list-style-type: none"> • 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 <p>The wildcard character (*) is supported anywhere in the file name or path:</p> <ul style="list-style-type: none"> • only one wildcard character can be used with each path • the wildcard will only wildcard a single directory • the wildcard can be used to specify all files with a specific prefix • the wildcard character specified on its own represents all files and directories on the filesystem

2.5.2.3 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	<p>Monitoring mode: the application behaves as normal.</p> <p>A log message is generated with details of all attempts to read from or write to a protected file.</p> <div data-bbox="389 1621 1417 1711" style="border: 1px solid #ccc; padding: 5px; margin: 10px 0;"> <p> A log message must be specified with this action.</p> </div>
allow	Can be used to allow access to specific files or directories under a parent directory that is covered by an ARMR <code>filesystem</code> rule in <code>protect</code> 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.

2.5.2.4 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/2.3)
  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/2.3)
  filesystem("Protect read access in specific directory"):
    read("C:\\Windows\\*")
    protect(message: "Unauthorized file read blocked", severity: 8)
  endfilesystem
endapp
```

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|2.3|Protect read access in specific directory|Execute Rule|High|rt=Mar 29 2021 11:59:25.146
+0100 dvchost=userX_system procid=15891 appVersion=1 act=protect msg=Unauthorized file read blocked path=/
tmp/somefile.txt
```

```
<10>1 2021-03-29T11:57:23.337+01:00 userX_system java 14223 - - CEF:0|ARMR:File read protect mod|File read
protect mod|2.3|Protect read access in specific directory|Execute Rule|High|rt=Mar 29 2021 11:57:23.337
+0100 dvchost=userX_system procid=14223 appVersion=1 act=protect msg=Unauthorized file read blocked path=C:
\\Windows\\somefile.txt
```

2.5.2.5 Further Examples

As above, with the stacktrace also logged

Unix

```
app("File read protect mod - with stacktrace"):
  requires(version: ARMR/2.3)
  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/2.3)
  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|2.3|Protect read access in specific directory|Execute
Rule|High|rt=Mar 29 2021 12:05:25.019 +0100 dvchost=userX_system procid=15891 appVersion=1 act=protect
msg=Unauthorized file read blocked stacktrace=java.util.Scanner.<init>(Scanner.java:611)\ncom.waratek.spira
cle.file.FileServlet.readFile(FileServlet.java:109)\ncom.waratek.spiracle.file.FileServlet.read(FileServlet
.java:90)\ncom.waratek.spiracle.file.FileServlet.executeRequest(FileServlet.java:71)\ncom.waratek.spiracle.
file.FileServlet.doPost(FileServlet.java:60)\njavax.servlet.http.HttpServlet.service(HttpServlet.java:650)\
njavax.servlet.http.HttpServlet.service(HttpServlet.java:731)\nsun.reflect.GeneratedMethodAccessor32.invoke
(Unknown Source)\nsun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)\nja
va.lang.reflect.Method.invoke(Method.java:498)\norg.apache.catalina.core.ApplicationFilterChain.internalDoF
ilter(ApplicationFilterChain.java:303)\norg.apache.catalina.core.ApplicationFilterChain.doFilter( Applicatio
nFilterChain.java:208)\norg.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.java:52)\nsun.reflect
.GeneratedMethodAccessor46.invoke(Unknown Source)
\nsun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)\njava.lang.reflect.
Method.invoke(Method.java:498)\norg.apache.catalina.core.ApplicationFilterChain.internalDoFilter( Applicatio
nFilterChain.java:241)\norg.apache.catalina.core.ApplicationFilterChain.doFilter( ApplicationFilterChain. jav
a:208)\norg.apache.catalina.core.StandardWrapperValve.invoke(StandardWrapperValve.java:218)\norg.apache.cat
alina.core.StandardContextValve.invoke(StandardContextValve.java:122)\norg.apache.catalina.authenticator.Au
thenticatorBase.invoke(AuthenticatorBase.java:505)\norg.apache.catalina.core.StandardHostValve.invoke(Stand
ardHostValve.java:169)\norg.apache.catalina.valves.ErrorReportValve.invoke(ErrorReportValve.java:103)\norg.
apache.catalina.valves.AccessLogValve.invoke(AccessLogValve.java:956)\norg.apache.catalina.core.StandardEng
ineValve.invoke(StandardEngineValve.java:116)\norg.apache.catalina.connector.CoyoteAdapter.service(CoyoteAd
apter.java:442)\norg.apache.coyote.http11.AbstractHttp11Processor.process(AbstractHttp11Processor.java:1082
)\norg.apache.coyote.AbstractProtocol$AbstractConnectionHandler.process(AbstractProtocol.java:623)\norg.apa
che.tomcat.util.net.JIoEndpoint$SocketProcessor.run(JIoEndpoint.java:316)\njava.util.concurrent.ThreadPoolE
xecutor.runWorker(ThreadPoolExecutor.java:1149)\njava.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadP
oolExecutor.java:624)\norg.apache.tomcat.util.threads.TaskThread$WrappingRunnable.run(TaskThread.java:61)\n
java.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|2.3|Protect read access in specific directory|Execute
Rule|High|rt=Mar 29 2021 12:55:25.034 +0100 dvchost=userX_system procid=14222 appVersion=1 act=protect
msg=Unauthorized file read blocked stacktrace=java.util.Scanner.<init>(Scanner.java:611)\ncom.waratek.spira
cle.file.FileServlet.readFile(FileServlet.java:109)\ncom.waratek.spiracle.file.FileServlet.read(FileServlet
.java:90)\ncom.waratek.spiracle.file.FileServlet.executeRequest(FileServlet.java:71)\ncom.waratek.spiracle
.file.FileServlet.doPost(FileServlet.java:60)\njavax.servlet.http.HttpServlet.service(HttpServlet.java:650)\
njavax.servlet.http.HttpServlet.service(HttpServlet.java:731)\nsun.reflect.GeneratedMethodAccessor32.invoke
(Unknown Source)\nsun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)\nja
va.lang.reflect.Method.invoke(Method.java:498)\norg.apache.catalina.core.ApplicationFilterChain.internalDoF
ilter(ApplicationFilterChain.java:303)\norg.apache.catalina.core.ApplicationFilterChain.doFilter(Applicatio
nFilterChain.java:208)\norg.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.java:52)\nsun.reflect
.GeneratedMethodAccessor46.invoke(Unknown Source)
\nsun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)\njava.lang.reflect
.Method.invoke(Method.java:498)\norg.apache.catalina.core.ApplicationFilterChain.internalDoFilter(Applicatio
nFilterChain.java:241)\norg.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.jav
a:208)\norg.apache.catalina.core.StandardWrapperValve.invoke(StandardWrapperValve.java:218)\norg.apache.cat
alina.core.StandardContextValve.invoke(StandardContextValve.java:122)\norg.apache.catalina.authenticator.Au
thenticatorBase.invoke(AuthenticatorBase.java:505)\norg.apache.catalina.core.StandardHostValve.invoke(Stand
ardHostValve.java:169)\norg.apache.catalina.valves.ErrorReportValve.invoke(ErrorReportValve.java:103)\norg.
apache.catalina.valves.AccessLogValve.invoke(AccessLogValve.java:956)\norg.apache.catalina.core.StandardEng
ineValve.invoke(StandardEngineValve.java:116)\norg.apache.catalina.connector.CoyoteAdapter.service(CoyoteAd
apter.java:442)\norg.apache.coyote.http11.AbstractHttp11Processor.process(AbstractHttp11Processor.java:1082
)\norg.apache.coyote.AbstractProtocol$AbstractConnectionHandler.process(AbstractProtocol.java:623)\norg.apa
che.tomcat.util.net.JIoEndpoint$SocketProcessor.run(JIoEndpoint.java:316)\njava.util.concurrent.ThreadPoolE
xecutor.runWorker(ThreadPoolExecutor.java:1149)\njava.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadP
oolExecutor.java:624)\norg.apache.tomcat.util.threads.TaskThread$WrappingRunnable.run(TaskThread.java:61)\n
java.lang.Thread.run(Thread.java:748) path=C:\\Windows\\somefile.txt
```

Prevent reading any file

```
app("File read protect mod - wildcard all"):
  requires(version: ARMR/2.2)
  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/2.2)
  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/2.2)
  filesystem("Protect read access to specific files"):
    read(paths: ["/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/2.2)
  filesystem("Protect read access to specific files"):
    read(paths: ["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/2.2)
  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/2.2)
  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/2.2)
  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/2.2)
  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 under a particular directory

Unix

```

app("File read protect mod"):
  requires(version: ARMR/2.2)
  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/2.2)
  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/2.2)

  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/2.2)

  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

```


2.6 ARMR Process Rule (22.x.x)

2.6.1 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.


2.6.2 When (Event)

To control access to executables using the ARMR process rule the user must specify the execute declaration.

execute	<p>A parameter must be supplied to the execute declaration to determine the executable(s) that the ARMR process rule will control access to.</p> <div data-bbox="395 376 1420 472" style="border: 1px solid #ccc; padding: 5px; margin: 10px 0;"> <p> Both Unix and Windows filesystem paths are supported</p> </div> <p>This parameter takes the form of a list of one or more quoted-strings indicating specifically targeted executables.</p> <p>Each string represented in the parameter can be:</p> <ul style="list-style-type: none"> • a single executable or directory name - the agent will control access to any executable or directory on the filesystem that matches the given name • an absolute path to a specific executable or directory <p>The wildcard character (*) is supported anywhere in the executable name or path:</p> <ul style="list-style-type: none"> • only one wildcard character can be used with each path • the wildcard will only wildcard a single directory • the wildcard can be used to specify all executables with a specific prefix • the wildcard character specified on its own represents all executables and directories on the filesystem
---------	---

2.6.3 Then (Action)

There are three supported actions for the ARMR process rule: protect, detect and allow.

protect	<p>All attempts to fork a process are blocked. If configured, a log message is generated with details of the event.</p>
detect	<p>Monitoring mode: the application behaves as normal.</p> <p>A log message is generated with details of all attempts to fork a process.</p> <div data-bbox="387 1529 1420 1626" style="border: 1px solid #ccc; padding: 5px; margin: 10px 0;"> <p> A log message must be specified with this action.</p> </div>
allow	<p>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.</p>

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.

2.6.4 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/2.3)
  process("Protect executable in a specific directory"):
    execute("/tmp/*")
    protect(message: "denying attempt to execute processes inside specific directory", severity: 10)
  endprocess
endapp
```

Windows

```
app("Process forking mod"):
  requires(version: ARMR/2.3)
  process("Protect executable in a specific directory"):
    execute("C:\Windows\*")
    protect(message: "denying attempt to execute processes inside specific directory", severity: 10)
  endprocess
endapp
```

2.6.4.1 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|2.3|Protect executable in a specific directory|Execute Rule|Very-High|rt=Mar 29 2021 11:44:30.232 +0100 dvchost=userX_system procid=15891 appVersion=1 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|2.3|Protect executable in a specific directory|Execute Rule|Very-High|rt=Mar 29 2021 11:47:50.278 +0100 dvchost=userX_system procid=13286 appVersion=1 act=protect msg=denying attempt to execute processes inside specific directory path=C:\Windows\myscript.bat commandLine=myscript.bat scriptArg
```

2.6.5 Further Examples

As above, with the stacktrace also logged

Unix

```
app("Process forking mod - with stacktrace"):
  requires(version: ARMR/2.3)
  process("Protect executable in a specific directory"):
    execute("/tmp/*")
    protect(message: "denying attempt to execute processes inside specific directory", severity: 10,
stacktrace: "full")
  endprocess
endapp
```

Windows

```
app("Process forking mod - with stacktrace"):
  requires(version: ARMR/2.3)
  process("Protect executable in a specific directory"):
    execute("C:\Windows\*")
    protect(message: "denying attempt to execute processes inside specific directory", severity: 10,
stacktrace: "full")
  endprocess
endapp
```

2.6.5.1 Logging

Unix

```

<9>1 2021-03-29T11:48:42.789+01:00 userX_system java 15891 - - CEF:0|ARMR:Process forking mod - with
stacktrace|Process forking mod - with stacktrace|2.3|Protect executable in a specific directory|Execute
Rule|Very-High|rt=Mar 29 2021 11:48:42.787 +0100 dvchost=userX_system procid=15891 appVersion=1 act=protect
msg=denying attempt to execute processes inside specific directory
stacktrace=com.waratek.spiracle.file.FileExecServlet.executeRequest(FileExecServlet.java:78)\ncom.waratek.s
piracle.file.FileExecServlet.doPost(FileExecServlet.java:70)\njavax.servlet.http.HttpServlet.service(HttpSe
rvlet.java:650)\njavax.servlet.http.HttpServlet.service(HttpServlet.java:731)\nsun.reflect.NativeMethodAcce
ssorImpl.invoke0(Native Method)
\nsun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)\nsun.reflect.DelegatingMeth
odAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)\njava.lang.reflect.Method.invoke(Method.java:
498)\norg.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:303)\nor
g.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:208)\norg.apache.tomcat.
websocket.server.WsFilter.doFilter(WsFilter.java:52)\nsun.reflect.NativeMethodAccessorImpl.invoke0(Native
Method)\nsun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)\nsun.reflect.Delegat
ingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)\njava.lang.reflect.Method.invoke(Method.
java:498)\norg.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:241
)\norg.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:208)\norg.apache.ca
talina.core.StandardWrapperValve.invoke(StandardWrapperValve.java:218)\norg.apache.catalina.core.StandardCo
ntextValve.invoke(StandardContextValve.java:122)\norg.apache.catalina.authenticator.AuthenticatorBase.invok
e(AuthenticatorBase.java:505)\norg.apache.catalina.core.StandardHostValve.invoke(StandardHostValve.java:169
)\norg.apache.catalina.valves.ErrorReportValve.invoke(ErrorReportValve.java:103)\norg.apache.catalina.valve
s.AccessLogValve.invoke(AccessLogValve.java:956)\norg.apache.catalina.core.StandardEngineValve.invoke(Stand
ardEngineValve.java:116)\norg.apache.catalina.connector.CoyoteAdapter.service(CoyoteAdapter.java:442)\norg.
apache.coyote.http11.AbstractHttp11Processor.process(AbstractHttp11Processor.java:1082)\norg.apache.coyote.
AbstractProtocol$AbstractConnectionHandler.process(AbstractProtocol.java:623)\norg.apache.tomcat.util.net.J
IoEndpoint$SocketProcessor.run(JIoEndpoint.java:316)\njava.util.concurrent.ThreadPoolExecutor.runWorker(Thr
eadPoolExecutor.java:1149)\njava.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)
\norg.apache.tomcat.util.threads.TaskThread$WrappingRunnable.run(TaskThread.java:61)\njava.lang.Thread.run(
Thread.java:748) path=/tmp/myscript.sh commandLine=/tmp/myscript.sh scriptArg

```

Windows

```
<9>1 2021-03-29T11:52:52.559+01:00 userX_system java 15844 - - CEF:0|ARMR:Process forking mod - with
stacktrace|Process forking mod - with stacktrace|2.3|Protect executable in a specific directory|Execute
Rule|Very-High|rt=Mar 29 2021 11:52:52.559 +0100 dvchost=userX_system procid=15844 appVersion=1 act=protect
msg=denying attempt to execute processes inside specific directory
stacktrace=com.waratek.spiracle.file.FileExecServlet.executeRequest(FileExecServlet.java:78)\ncom.waratek.s
piracle.file.FileExecServlet.doPost(FileExecServlet.java:70)\njavax.servlet.http.HttpServlet.service(HttpSe
rvlet.java:650)\njavax.servlet.http.HttpServlet.service(HttpServlet.java:731)\nsun.reflect.NativeMethodAcce
ssorImpl.invoke0(Native Method)
\nsun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)\nsun.reflect.DelegatingMeth
odAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)\njava.lang.reflect.Method.invoke(Method.java:
498)\norg.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:303)\nor
g.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:208)\norg.apache.tomcat.
websocket.server.WsFilter.doFilter(WsFilter.java:52)\nsun.reflect.NativeMethodAccessorImpl.invoke0(Native
Method)\nsun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)\nsun.reflect.Delegat
ingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)\njava.lang.reflect.Method.invoke(Method.
java:498)\norg.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:241
)\norg.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:208)\norg.apache.ca
talina.core.StandardWrapperValve.invoke(StandardWrapperValve.java:218)\norg.apache.catalina.core.StandardCo
ntextValve.invoke(StandardContextValve.java:122)\norg.apache.catalina.authenticator.AuthenticatorBase.invo
ke(AuthenticatorBase.java:505)\norg.apache.catalina.core.StandardHostValve.invoke(StandardHostValve.java:169
)\norg.apache.catalina.valves.ErrorReportValve.invoke(ErrorReportValve.java:103)\norg.apache.catalina.valve
s.AccessLogValve.invoke(AccessLogValve.java:956)\norg.apache.catalina.core.StandardEngineValve.invoke(Stand
ardEngineValve.java:116)\norg.apache.catalina.connector.CoyoteAdapter.service(CoyoteAdapter.java:442)\norg.
apache.coyote.http11.AbstractHttp11Processor.process(AbstractHttp11Processor.java:1082)\norg.apache.coyote.
AbstractProtocol$AbstractConnectionHandler.process(AbstractProtocol.java:623)\norg.apache.tomcat.util.net.J
IoEndpoint$SocketProcessor.run(JIoEndpoint.java:316)\njava.util.concurrent.ThreadPoolExecutor.runWorker(Thr
eadPoolExecutor.java:1149)\njava.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)
\norg.apache.tomcat.util.threads.TaskThread$WrappingRunnable.run(TaskThread.java:61)\njava.lang.Thread.run(
Thread.java:748) path=C:\\Windows\\myscript.bat commandLine=myscript.bat scriptArg
```

Prevent forking a specific process

Unix

```
app("Process forking mod 2"):
  requires(version: ARMR/2.2)
  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/2.2)
  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/2.2)
  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/2.2)
  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/2.2)

  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 executable", severity: 3)
  endprocess

endapp

```

Windows

```
app("Process forking mod 4"):
  requires(version: ARMR/2.2)

  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 executable", severity: 3)
  endprocess

endapp
```

2.7 ARMR Sanitization Rule (22.x.x)

2.7.1 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.

2.7.2 Given (Condition)

Directive	Attribute	Necessity	Description
request	paths	mandatory	<p>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)</p> <ul style="list-style-type: none"> • a quoted string • a list of one or more quoted-strings <p>If no value is specified then protection will be applied to all HTTP endpoints by default.</p> <p>If a string value is specified then it must:</p> <ul style="list-style-type: none"> • not be empty • be a valid relative URI <p>The paths can be configured similar to,</p> <ul style="list-style-type: none"> • <code>request(paths: ["/api/user", "/api/cart"])</code>
undetermined	values	mandatory	<p>If a payload cannot be cleanly identified as being safe or unsafe then the rule will consider these values as being <i>undetermined</i>. 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.</p> <p>The values can be configured only as,</p> <ul style="list-style-type: none"> • <code>undetermined(values: safe)</code> • <code>undetermined(values: unsafe)</code>

Directive	Attribute	Necessity	Description
ignore	payload	optional	<p>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.</p> <p>The payload can be configured similar to,</p> <ul style="list-style-type: none"> ignore(payload: ["abcd", "efgh", "1234"]) <p>The payload can be configured along with the attribute in the same ignore declaration.</p> <pre>ignore(payload: ["abcd", "efgh", "1234"], attribute: ["field1", "keyname2"])</pre>
	attribute	optional	<p>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.</p> <p>The attribute can be configured similar to,</p> <ul style="list-style-type: none"> ignore(attribute: ["field1", "keyname2"]) <p>The attribute can be configured along with the payload in the same ignore declaration.</p> <pre>ignore(attribute: ["field1", "keyname2"], payload: ["abcd", "efgh", "1234"])</pre>

2.7.3 When (Event)

The rule actively examines payloads coming from HTTP requests that use the `javax.servlet.ServletException` API and JSON/XML parsing within Spring Boot applications.

API
<pre>javax.servlet.ServletException.getParameter(Ljava/lang/String;)Ljava/lang/String;</pre>

API
<code>javax.servlet.ServletRequest.getParameterMap()</code> [Ljava/util/Map;
<code>javax.servlet.ServletRequest.getParameterValues()</code> [Ljava/lang/String;
<code>javax.servlet.ServletInputStream.readLine()</code> [B]I
Spring Boot - JSON to Object Conversion
Spring Boot - XML to Object Conversion

2.7.4 Then (Action)

protect	<p>Payloads that are marked for sanitization will be blocked by either throwing an exception, or replacing the malicious value with a <i>null</i> 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.</p>
detect	<p>Monitoring mode: the application behaves as normal.</p> <p>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.</p>

2.7.5 Examples

2.7.5.1 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/2.3")

  sanitization("SANITIZATION :01"):
    request()
    undetermined(values: safe)
    protect(message: "A payload has been marked for sanitization", severity: high)
  endsanitization

endapp

```

2.7.5.2 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/register"`, `"/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 *"sensitive API endpoint under attack"*.

```

app("SECURITY POLICY"):
  requires(version: "ARMR/2.3")

  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

```

2.7.6 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.

2.7.6.1 Protect Mode

```

<10>1 2021-02-19T20:06:56.939Z localhost java 19559 - - CEF:0|ARMR:SANITIZATION|SANITIZATION|2.3|
SANITIZATION :01|Execute Rule|High|rt=Feb 19 2021 20:06:56.939 +0000 dvchost=localhost procid=19559
  appVersion=1 act=protect msg=A payload has been classified as malicious reason=SQLI
  taintSource=HTTP_SERVLET httpRequestUri=/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

```

2.7.6.2 Detect Mode

```

<10>1 2021-02-19T20:15:25.002Z localhost java 19559 - - CEF:0|ARMR:SANITIZATION|SANITIZATION|2.3|
SANITIZATION :01|Execute Rule|High|rt=Feb 19 2021 20:15:25.002 +0000 dvchost=localhost procid=19559
  appVersion=1 act=detect msg=A payload has been classified as malicious reason=XSS taintSource=HTTP_SERVLET
  httpRequestUri=/api/forum/print/xml/requestbody/complex
  className=com.example.data.entity.xml.ForumEntityXml attribute=OBJECT_FIELD["message"] payload=<script>
  remoteIpAddress=127.0.0.1 localIpAddress=127.0.0.1 localPort=8080

```

2.7.6.3 Safe Undetermined

```
<10>1 2021-02-19T20:09:43.593Z localhost java 19559 - - CEF:0|ARMR:SANITIZATION|SANITIZATION|2.3|
SANITIZATION :01|Execute Rule|High|rt=Feb 19 2021 20:09:43.592 +0000 dvchost=localhost procid=19559
  appVersion=1 msg=A payload could not be classified reason=UNDETERMINED taintSource=HTTP_SERVLET
  httpRequestUri=/api/forum/add/json className=com.example.data.entity.ForumEntity
  attribute=OBJECT_FIELD["message"] payload=< > remoteIpAddress=127.0.0.1 localIpAddress=127.0.0.1
  localPort=8080
```

2.8 ARMR Socket Rule (22.x.x)

2.8.1 Socket Control Security Feature (22.x.x)

2.8.1.1 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 product version 19.2.0 onwards

2.8.1.2 Given (Condition)


bind	<p>The <code>bind</code> takes the following key-value pairs as parameters: <code>client</code> and <code>server</code>. They can be used simultaneously within <code>bind</code>. The value for both <code>client</code> and <code>server</code> keys within <code>bind</code> 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 <code>0.0.0.0</code>, and wildcard for port is specified by <code>0</code>.</p> <p>The following are examples of <code>bind</code> conditions specifying wildcarded IPv4 addresses and wildcarded port;</p> <pre>bind(client: "0.0.0.0:0") bind(server: "0.0.0.0:0") bind(server: "0.0.0.0:0", client: "0.0.0.0:0")</pre> <p>Specific IPv4 and/or port numbers may be specified, for example;</p> <pre>bind(client: "127.0.0.1:80") bind(server: "127.0.0.1:0") bind(client: "0.0.0.0:80")</pre> <p>Port ranges may be specified, for example;</p> <pre>bind(client: "0.0.0.0:80-90") bind(server: "0.0.0.0:8080-8090") bind(server: "127.0.0.1:8080-8090")</pre>
------	---

connect	accept and connect require only a single parameter which is the IPv4 address and port for accepting connections from and to a remote address, respectively.
accept	<p>Wildcard for IPv4 addresses is specified by 0.0.0.0, and wildcard for port is specified by 0.</p> <p>The following are examples of accept and connect conditions specifying wildcarded IPv4 addresses and wildcarded port;</p> <pre data-bbox="392 544 1418 647">accept("0.0.0.0:0") connect("0.0.0.0:0")</pre> <p>Specific IPv4 and/or port numbers may be specified, for example;</p> <pre data-bbox="392 741 1418 902">accept("127.0.0.1:5001") accept("0.0.0.0:5001") connect("127.0.0.1:8080") connect("127.0.0.1:0")</pre> <p>Port ranges may be specified, for example;</p> <pre data-bbox="392 996 1418 1099">accept("127.0.0.1:5000-5100") connect("0.0.0.0:8080-8100")</pre>

2.8.1.3 Then (Action)

If an empty message is passed to an action. The action will use a pre-defined logging format for the message. 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	<p>The protect action can have an extra key-value pair connection other than message and severity.</p> <p>Valid values for the connection key is the constant secure or upgrade-tls.</p> <div data-bbox="328 1574 1418 1697" style="border: 1px solid #ccc; padding: 5px;"> <p> When a specific protect action acting on connections is enforced, the accept event must be declared, and only a wildcarded IP address and wildcarded port is valid.</p> </div> <pre data-bbox="328 1720 1418 1798">protect(connection: secure, message: "sample message")</pre>
allow	allow and detectdo not take a connection parameter.

detect

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. The `stacktrace: "full"` action parameter is not a valid configuration if `connection: secure` or `connection: upgrade-tls` are specified.

2.8.1.4 Examples

Blocking client binds on all interfaces and all ports

```
app("Socket Client Bind Mod"):
  requires(version: ARMR/2.3)
  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/2.3)
  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/2.3)
  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/2.3)
  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/2.3)
  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)
  endssocket
endapp
```

Blocking server accepting connections on a specific interface, over a range of ports.

```
app("Socket Accept Mod"):
  requires(version: ARMR/2.3)
  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)
  endssocket
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|Walter|2.3|Socket rule protect|
Execute Rule|High|rt=Mar 22 2021 11:03:42.919 +0000 dvchost=jenkins-qa-slave-centos.aws.waratek.lan
procid=5989 appVersion=1 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|Walter|2.3|Socket rule protect|
Execute Rule|High|rt=Mar 22 2021 11:05:20.331 +0000 dvchost=jenkins-qa-slave-centos.aws.waratek.lan
procid=6442 appVersion=1 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|Walter|2.3|Socket rule protect|
Execute Rule|High|rt=Mar 22 2021 11:06:00.932 +0000 dvchost=jenkins-qa-slave-centos.aws.waratek.lan
procid=6591 appVersion=1 act=protect msg=Socket rule protect 127.0.0.1:0 remoteIpAddress=127.0.0.1
  remotePort=5001
```

2.8.1.5 Further Examples

Blocking server binds on all interfaces and all ports with stacktrace: "full" parameter.

```

app("Socket Server Bind Mod"):
  requires(version: ARMR/2.3)
  socket("Blocking server binds on all interfaces and all ports"):
    bind(server: "0.0.0.0")
    protect(message: "port binding blocked", severity: 8, stacktrace: "full")
  endssocket
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|2.3|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 act=protect msg=port binding blocked
stacktrace=java.net.ServerSocket.bind(ServerSocket.java)
\nNetworkServerSocket.main(NetworkServerSocket.java:19) localIpAddress=127.0.0.1 localPort=5001

```

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

```

app("Socket Connect Mod"):
  requires(version: ARMR/2.3)
  socket("Blocking client connections on all ports"):
    connect("0.0.0.0")
    protect(message: "connections blocked", severity: 8, stacktrace: "full")
  endssocket
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|2.3|Blocking client connections on all ports|Execute Rule|High|rt=Apr 01 2021 13:58:10.561
+0100 dvchost=hostnameX procid=23895 appVersion=1 act=protect msg=connections blocked
stacktrace=java.net.Socket.connect(Socket.java)
\nClientConnection.attemptServerConnection(ClientConnection.java:37)\nClientConnection.main(ClientConnectio
n.java:24) remoteIpAddress=127.0.0.1 remotePort=5001

```

2.8.2 Secure Sockets (22.x.x)

2.8.2.1 Overview

 This security feature is only available on Waratek Upgrade, is not supported on Waratek Secure

Creating plain TCP server sockets without data encryption allows attackers to intercept such communication channels and read/modify the transmitted data. To avoid such attacks the communication channel must be encrypted. To enforce this policy, the rule upgrades TCP server sockets to SSL/TLS server sockets. Upgrading TCP server sockets to SSL/TLS server sockets will significantly increase the difficulty of Man-in-The-Middle attacks and address known vulnerabilities such as CWE-319, CWE-311, and CWE-5 that are classified as "Sensitive Data Exposure" in OWASP's Top 10 list.

The upgrade is completely transparent to the application and behaves as if communication is occurring over an unencrypted channel. Additionally, because of the fact that the host could be a newer Java version than the guest, SSL/TLS server sockets are able to utilize the newer cipher suites available to the host JVM. This provides the advantage of stronger encryption via the use of the latest cryptographic algorithms for SSL/TLS communication.

In order for this rule to successfully upgrade TCP server sockets to SSL/TLS server sockets make sure that the following system properties are set, according to the desired SSL/TLS configuration. Note that the same system properties must be set on both the server and the client nodes.

```
-Djavax.net.ssl.trustStore
-Djavax.net.ssl.trustStorePassword
-Djavax.net.ssl.keyStore
-Djavax.net.ssl.keyStorePassword
```

2.8.2.2

When (Event)

accept	<p>IP address and port</p> <p>When a specific protect action acting on connections is enforced (e.g. forcing TCP connections to use TLS for connection by specifying <code>connection: secure</code> key-value), only wildcard IP and port are supported</p>
--------	--

2.8.2.3 Then (Action)

protect	<p>Upgrades TCP server sockets to SSL/TLS server sockets. If configured, a log message is generated with details of the event. The <code>stacktrace: "full"</code> action parameter is not a valid configuration for the Secure Sockets rule.</p>	
---------	---	--

2.8.2.4 Examples

Force TCP connections to use TLS for connections.

```

app("Socket Accept Forced TLS)
  requires(version: ARMR/2.3)
  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)
  endssocket
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|Walter|2.3|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.waratek.lan procid=27253 appVersion=1 act=protect msg=Forced TLS on every connection
localName=0.0.0.0 localPort=33547

```


2.8.3 TLS upgrade (22.x.x)


2.8.3.1 Overview

 This security feature is only available on Waratek Upgrade, is not supported on Waratek Secure

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.

2.8.3.2 When (Event)

accept	<p>IP address and port</p> <p>When 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</p>
--------	--

2.8.3.3 Then (Action)

protect	<p>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.</p>
---------	--

2.8.3.4 Examples

Upgrade TLS connections for connections.

```
app("myapp"):
  requires(version: ARMR/2.3)
  socket("Upgrade TLS connections for connections"):
    accept("0.0.0.0:0")
    protect(connection: upgrade-tls, message: "TLS connection upgraded", severity: 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:Walter|Walter|2.2|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 act=protect msg=Forced TLS on every connection dst=0 localPort=40071 localName=0.0.0
.0
```

2.9 ARMR SQL Rule (22.x.x)

2.9.1 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.

2.9.2 Given (Conditions)


The user can specify two conditions in the ARMR sql rule - `input` and `vendor`.

<p><code>input</code></p>	<p>This allows the user to specify the source of the untrusted data. The following three sources are supported:</p> <ul style="list-style-type: none"> • <code>http</code> data introduced via HTTP/HTTPS requests • database data introduced via JDBC connections • <code>deserialization</code> data introduced via Java or XML deserialization <p>The rule will trigger if the source of the untrusted data matches that specified in the rule.</p> <p>If no value is specified then a default value of <code>http</code> is used.</p> <p>An exception will be thrown if an unsupported value is provided.</p>
<p><code>vendor</code></p>	<p>This is an optional declaration that allows the user to specify the database type to be protected. The following databases are supported:</p> <ul style="list-style-type: none"> • <code>db2</code> • <code>mariadb</code> • <code>mssql</code> • <code>mysql</code> • <code>oracle</code> • <code>sybase</code> • <code>postgres</code> <p>In addition, a value of <code>any</code> may be specified which will enable the agent to automatically detect the database type used by the application.</p> <p>One of the listed database types, or the value <code>any</code>, must be specified if the <code>vendor</code> declaration is present.</p> <p>If no <code>vendor</code> declaration is specified then a default value of <code>any</code> is used.</p>

	options	<p>Depending on the database configuration, the following optional parameters are also supported to allow the agent to accurately detect SQL injection attacks:</p> <ul style="list-style-type: none"> • <code>ansi-quotes - mysql</code> and <code>mariadb</code>: corresponds to the <code>ANSI_QUOTES</code> server mode. • <code>no-backslash-escapes - mysql</code> and <code>mariadb</code>: corresponds to the <code>NO_BACKSLASH_ESCAPES</code> server mode. • <code>quoted-identifiers - mssql</code> and <code>sybase</code>: corresponds to the <code>QUOTED_IDENTIFIER</code> flag
--	---------	---

2.9.3 When (Event)

injection	<p>This condition allows the user to specify the type of injection:</p> <ul style="list-style-type: none"> • <code>successful-attempt</code> the rule will trigger upon detecting a valid SQLi payload that would have resulted in a successful SQLi attack, exploiting the underlying database. • <code>failed-attempt</code> 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. <p>If no value is specified then a default value of <code>successful-attempt</code> is used.</p> <p>In addition, the user may optionally specify the following parameter:</p> <ul style="list-style-type: none"> • <code>permit: query-provided</code> 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. <p>An exception will be thrown if an unsupported value is provided.</p>
-----------	--

 Multiple `sql` rules are allowed in the same ARMR mod providing they have different injection types.


2.9.4 Then (Action)

The action statement specifies the default action the agent takes whenever an attack is detected. There are two supported actions `protect` and `detect`:

	successful-attempt	failed-attempt
--	--------------------	----------------

protect	A valid SQL injection attack is not allowed to be processed by the database. A SQLException is thrown by the agent to indicate the SQL statement is invalid, letting the server handle the exception gracefully. If configured, a log message is generated with details of the event.	An invalid SQL statement is not allowed to return any sensitive information about the database. The HTTP connection, from which the malicious data that exploited the SQL statement originated, is disconnected. If configured, a log message is generated with details of the event.
detect	Monitoring mode: the application behaves as normal. A valid SQL injection attack is allowed to be processed by the database. If configured, a log message is generated detailing that the agent has detected an attempt to exploit the database using a valid SQL statement.	Monitoring mode: the application behaves as normal. An invalid SQL statement is allowed to be processed by the database. If configured, a log message is generated detailing that the agent has detected an attempt to exploit the database using an invalid SQL statement.

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 Waratek command-line option `com.waratek.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: `com.waratek.AllowSQLiPayloads=AND,OR`

2.9.5 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/2.3)
  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
```

2.9.5.1 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|2.3|Protect
MySQL database from SQL Injection attacks|Execute Rule|High|rt=Mar 30 2021 17:33:55.537 +0100
  dvchost=userX_system procid=32008 appVersion=1 act=protect msg=SQL injection attack detected and blocked
  databaseVendor=mysql httpSessionId=3153E581A645E2A54D3C12D3928473BC taintSource=HTTP_SERVLET
  httpRequestUri=/spiracle/Get_int httpCookies=JSESSIONID\=3153E581A645E2A54D3C12D3928473BC remoteIpAddress=0:0
```

2.9.6 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/2.3)
  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
```

2.9.6.1 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|2.3|Protect MySql database from SQL Injection attacks|Execute Rule|High|rt=Apr 01
2021 11:30:25.073 +0100 dvchost=userX_system procid=25024 appVersion=1 act=protect msg=SQL injection attack
detected and blocked stacktrace=com.waratek.spiracle.sql.util.SelectUtil.executeQuery(SelectUtil.java:67)\n
com.waratek.spiracle.sql.servlet.oracle.Get_int.executeRequest(Get_int.java:77)\ncom.waratek.spiracle.sql.s
ervlet.oracle.Get_int.doGet(Get_int.java:52)\njavax.servlet.http.HttpServlet.service(HttpServlet.java:624)\n
javax.servlet.http.HttpServlet.service(HttpServlet.java:731)\nsun.reflect.NativeMethodAccessorImpl.invoke@
(Native Method)\nsun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)\nsun.reflect
.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)\njava.lang.reflect.Method.invoke
(Method.java:498)\norg.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.
java:303)\norg.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:208)\norg.a
pache.tomcat.websocket.server.WSFilter.doFilter(WSFilter.java:52)\nsun.reflect.NativeMethodAccessorImpl.inv
oke@ (Native Method)\nsun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)\nsun.ref
lect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)\njava.lang.reflect.Method.in
voke(Method.java:498)\norg.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterCh
ain.java:241)\norg.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:208)\no
rg.apache.catalina.core.StandardWrapperValve.invoke(StandardWrapperValve.java:218)\norg.apache.catalina.cor
e.StandardContextValve.invoke(StandardContextValve.java:122)\norg.apache.catalina.authenticator.Authenticat
orBase.invoke(AuthenticatorBase.java:505)\norg.apache.catalina.core.StandardHostValve.invoke(StandardHostVa
lve.java:169)\norg.apache.catalina.valves.ErrorReportValve.invoke(ErrorReportValve.java:103)\norg.apache.ca
talina.valves.AccessLogValve.invoke(AccessLogValve.java:956)\norg.apache.catalina.core.StandardEngineValve.
invoke(StandardEngineValve.java:116)\norg.apache.catalina.connector.CoyoteAdapter.service(CoyoteAdapter.jav
a:442)\norg.apache.coyote.http11.AbstractHttp11Processor.process(AbstractHttp11Processor.java:1082)\norg.ap
ache.coyote.AbstractProtocol$AbstractConnectionHandler.process(AbstractProtocol.java:623)\norg.apache.tomca
t.util.net.JIoEndpoint$SocketProcessor.run(JIoEndpoint.java:316)\njava.util.concurrent.ThreadPoolExecutor.r
unWorker(ThreadPoolExecutor.java:1149)\njava.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecut
or.java:624)\norg.apache.tomcat.util.threads.TaskThread$WrappingRunnable.run(TaskThread.java:61)\njava.lang
.Thread.run(Thread.java:748) databaseVendor=mysql httpSessionId=Unknown taintSource=HTTP_SERVLET
httpRequestUri=/spiracle/Get_int httpCookies= remoteIpAddress=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/2.2)
  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/2.2)
  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/2.2)
  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/2.2)
  sql("Protect database from unsuccessful SQL Injection attacks from various sources"):
    input(http, database, deserialization)
    injection(failed-attempt)
    protect()
  endsql
endapp

```

2.10 ARMR HTTP rule (22.x.x)

2.10.1 CSRF Security Feature (22.x.x)

2.10.1.1 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.

Waratek 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.

2.10.1.2 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	<p>An optional key value pair can be supplied to this declaration where the key is <code>paths</code> and the value can be one of the following (indicating specifically targeted HTTP endpoints) :-</p> <ul style="list-style-type: none"> • a quoted string • a list of one or more quoted-strings <p>If no value is specified then protection will be applied to all HTTP endpoints by default.</p> <p>If a string value is specified then it must:</p> <ul style="list-style-type: none"> • not be empty • be a valid relative URI

2.10.1.3 When (Event)

csrf	<p>This declaration switches on the CSRF security feature and must be declared with one of the following values:</p> <ul style="list-style-type: none"> • <code>synchronized-tokens</code> enabling CSRF protection via STP • <code>same-origin</code> enabling CSRF protection via validation of origin headers
------	--

synchronized-tokens	<p>With this protection enabled, the following options may also be specified:</p> <ul style="list-style-type: none"> • exclude <ul style="list-style-type: none"> • 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 • method <ul style="list-style-type: none"> • specify the particular HTTP method(s) for which to enable protection (currently supported values are GET and POST) • if this option is not specified - the default value is POST • token-type <ul style="list-style-type: none"> • specify if a different token should be generated for each HTTP method type, or if a shared value is to be used for all HTTP method types (supported values are shared or unique) • if this option is not specified the default value is shared • token-name <ul style="list-style-type: none"> • specify the name of the token to be injected into the HTML • token names must be between 5 - 20 characters long, and each character of the token name must be URL safe • if this option is not specified the default value is <code>_X-CSRF-TOKEN</code> • ajax <ul style="list-style-type: none"> • specify whether the agent should validate AJAX requests (supported values are <code>validate</code> or <code>no-validate</code>) • if this option is not specified the default value is <code>validate</code>
same-origin	<p>With this protection enabled, the following options may also be specified:</p> <ul style="list-style-type: none"> • hosts <ul style="list-style-type: none"> • should the source origin not match the target origin, even for a non-malicious request, this option can be used to whitelist known safe origins • can specify a single string literal, or a non-empty array of one or more string literals • each string should comprise a host name and optional port number, separated by a colon

2.10.1.4 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.	Monitoring mode: the application behaves as normal. If a CSRF attack is identified then the agent will allow the request, along with all of its HTTP parameters and cookies, to be processed by the application. If configured, a log message is generated with details of the event.

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.

2.10.1.5 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/2.3)
  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/2.3)
  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|2.3|
CSRF STP|Execute Rule|Very-High|rt=Mar 29 2021 11:53:05.341 +0100 dvchost=userX_system procid=15891
  appVersion=1 act=protect msg=CSRF STP validation failed httpRequestUri=/spiracle/CSRFServlet
httpSessionId=E654F722AAFA3BF44F0D0BD4FB91134C httpCookies=JSESSIONID\=E654F722AAFA3BF44F0D0BD4FB91134C
remoteIpAddress=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|2.3|CSRF Same Origin|Execute Rule|High|rt=Mar 29 2021 10:03:16.832 +0100 dvchost=userX_system
procid=2402 appVersion=1 act=protect msg=CSRF Same Origin validation failed reason=Missing source origin
httpRequestUri=/spiracle/CSRFServlet remoteIpAddress=127.0.0.1
httpSessionId=8944B619DD9B0ADB37CA663F8337AFD httpCookies=JSESSIONID\=8944B619DD9B0ADB37CA663F8337AFD
```

2.10.1.6 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/2.3)
  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/2.3)
  http("CSRF Same Origin"):
    request()
    csrf(same-origin)
    protect(message: "CSRF Same Origin validation failed", severity: High, stacktrace: "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|2.3|CSRF STP|Execute Rule|Very-High|rt=Mar 29 2021 10:10:18.286 +0100
  dvchost=userX_system procid=8189 appVersion=1 act=protect msg=CSRF STP validation failed
stacktrace=org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:241
)\norg.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:208)\norg.apache.ca
talina.core.StandardWrapperValve.invoke(StandardWrapperValve.java:218)\norg.apache.catalina.core.StandardCo
ntextValve.invoke(StandardContextValve.java:122)\norg.apache.catalina.authenticator.AuthenticatorBase.invoke
(AuthenticatorBase.java:505)\norg.apache.catalina.core.StandardHostValve.invoke(StandardHostValve.java:169
)\norg.apache.catalina.valves.ErrorReportValve.invoke(ErrorReportValve.java:103)\norg.apache.catalina.valve
s.AccessLogValve.invoke(AccessLogValve.java:956)\norg.apache.catalina.core.StandardEngineValve.invoke(Stand
ardEngineValve.java:116)\norg.apache.catalina.connector.CoyoteAdapter.service(CoyoteAdapter.java:442)\norg.
apache.coyote.http11.AbstractHttp11Processor.process(AbstractHttp11Processor.java:1082)\norg.apache.coyote.
AbstractProtocol$AbstractConnectionHandler.process(AbstractProtocol.java:623)\norg.apache.tomcat.util.net.J
IoEndpoint$SocketProcessor.run(JIoEndpoint.java:318)\njava.util.concurrent.ThreadPoolExecutor.runWorker(Thr
eadPoolExecutor.java:1149)\njava.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)
\norg.apache.tomcat.util.threads.TaskThread$WrappingRunnable.run(TaskThread.java:61)\njava.lang.Thread.run(
Thread.java:748) httprequestUri=/spiracle/CSRFServlet httpsessionId=5D7CE07F605C3A6ABCADB35D065A95E5
httpcookies=JSESSIONID\=5D7CE07F605C3A6ABCADB35D065A95E5 remoteIpAddress=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|2.3|CSRF Same Origin|Execute Rule|High|rt=Mar 30 2021 10:0
5:09.119 +0100 dvchost=userX_system procid=2402 appVersion=1 act=protect msg=CSRF Same Origin validation
failed
stacktrace=org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:241
)\norg.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:208)\norg.apache.ca
talina.core.StandardWrapperValve.invoke(StandardWrapperValve.java:218)\norg.apache.catalina.core.StandardCo
ntextValve.invoke(StandardContextValve.java:122)\norg.apache.catalina.authenticator.AuthenticatorBase.invoke
(AuthenticatorBase.java:505)\norg.apache.catalina.core.StandardHostValve.invoke(StandardHostValve.java:169
)\norg.apache.catalina.valves.ErrorReportValve.invoke(ErrorReportValve.java:103)\norg.apache.catalina.valve
s.AccessLogValve.invoke(AccessLogValve.java:956)\norg.apache.catalina.core.StandardEngineValve.invoke(Stand
ardEngineValve.java:116)\norg.apache.catalina.connector.CoyoteAdapter.service(CoyoteAdapter.java:442)\norg.
apache.coyote.http11.AbstractHttp11Processor.process(AbstractHttp11Processor.java:1082)\norg.apache.coyote.
AbstractProtocol$AbstractConnectionHandler.process(AbstractProtocol.java:623)\norg.apache.tomcat.util.net.J
IoEndpoint$SocketProcessor.run(JIoEndpoint.java:316)\njava.util.concurrent.ThreadPoolExecutor.runWorker(Thr
eadPoolExecutor.java:1149)\njava.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)
\norg.apache.tomcat.util.threads.TaskThread$WrappingRunnable.run(TaskThread.java:61)\njava.lang.Thread.run(
Thread.java:748) reason=Missing source origin httprequestUri=/spiracle/CSRFServlet remoteIpAddress=127.0.0.
1 httpsessionId=8944B619DD9B0ADBF37CA663F8337AFD httpcookies=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/2.2)
  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/2.2)
  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 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/2.2)
  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

```

2.10.2 HTTP Header Injection Security Feature (22.x.x)

2.10.2.1 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.

i 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

i 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.

2.10.2.2 Given (Condition)

To enable the HTTP Header Injection security feature using the ARMR `http` rule the user specifies the response declaration.


response	<p>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 <code>paths</code> and the value can be one of the following (indicating specifically targeted HTTP endpoints) :-</p> <ul style="list-style-type: none"> • a quoted string • a list of one or more quoted-strings <p>If no value is specified then protection will be applied to all HTTP endpoints by default.</p> <p>If a string value is specified then it must:</p> <ul style="list-style-type: none"> • not be empty • be a valid relative URI <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> <p>i Only one ARMR <code>http</code> rule for HTTP Header Injection protection is allowed to be defined for a given HTTP endpoint.</p> </div>
----------	---

2.10.2.3 When (Event)

The header injection rule supports one event - `injection`

injection	<p>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:</p> <ul style="list-style-type: none"> • headers - protect against injection into HTTP response headers • cookies - protect against injection into HTTP response cookies
-----------	---

2.10.2.4 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	<p>Monitoring mode: the application behaves as normal.</p> <p>If an HTTP response header or cookie contains user-controlled newline characters then a log message is generated with details of the event.</p> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> <p> A log message must be specified with this action.</p> </div>

2.10.2.5 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/2.2)
  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/2.2)
  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/2.2)
  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/2.2)
  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
```

2.10.3 Open Redirect Security Feature (22.x.x)

2.10.3.1 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.

2.10.3.2 Given (Conditions)

The user can specify two conditions in the ARMR http rule to enable the ARMR Redirect security feature - input and response.

input	<p>This allows the user to specify the source of the untrusted data. The following three sources are supported:</p> <ul style="list-style-type: none"> • <code>http</code> data introduced via HTTP/HTTPS requests • database data introduced via JDBC connections • <code>deserialization</code> data introduced via Java or XML deserialization <p>The rule will trigger if the source of the untrusted data matches that specified in the rule.</p> <p>If no value is specified then a default value of <code>http</code> is used.</p> <p>An exception will be thrown if an unsupported value is provided.</p>
response	<p>This allows the user to specify that protection is required for an HTTP/HTTPS response.</p>

2.10.3.3 When (Event)

open-redirect	<p>This condition allows the user to specify that protection against open redirect attacks is required.</p> <p>This can be declared empty, without any parameters, indicating that protection against open redirects is required for all external domains or IP addresses.</p> <p>Alternatively, the user may specify the following options as a parameter:</p> <ul style="list-style-type: none"> • <code>options: {exclude: subdomains}</code> <p>This option is useful for applications that require open redirects to sub-domains of the same root domain to be allowed. Specifying the <code>exclude: subdomains</code> option allows all HTTP server-side redirects to URLs as long as the parent sub-domain or root domain is the same as the application's domain. For example:</p> <ul style="list-style-type: none"> • if the domain of the application is <code>foo.com</code>, then it may be necessary to allow open redirects to sub-domains such as: <ul style="list-style-type: none"> • <code>bar.foo.com</code> • <code>example.foo.com</code> • if the domain of the application is <code>something.foo.com</code> then it may be necessary to allow open redirects to another domain that has the same parent domain, such as: <ul style="list-style-type: none"> • <code>somethingElse.foo.com</code>
---------------	--

2.10.3.4

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.

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.

2.10.3.5

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/2.3)

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 Redirect mod|Open Redirect
mod|2.3|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 act=protect msg=Protect external redirects.
redirectLocation=http://www.waratek.com localIpAddress=0:0:0:0:0:0:1 localName=ip6-localhost
serverName=localhost httpSessionId=5D7CE07F605C3A6ABCFDB35D065A95E5 taintSource=HTTP_SERVLET
httpRequestUri=/spiraclle/SendRedirect httpCookies=JSESSIONID\=5D7CE07F605C3A6ABCFDB35D065A95E5
remoteIpAddress=0:0:0:0:0:0:1
```

2.10.3.6 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/2.3)

http("Protect against open redirect attacks"):
open-redirect()
response()
protect(message: "Protect external redirects.", severity: Very-High, stacktrace: "full")
endhttp

endapp
```

Logging

When the above ARMR http rule is triggered a log entry similar to the following is generated:

```
<9>1 2021-03-29T09:57:10.760+01:00 userX_system java 8189 - - CEF:0|ARMR:Open Redirect mod - with
stacktrace|Open Redirect mod - with stacktrace|2.3|Protect against open redirect attacks|Execute Rule|Very-
High|rt=Mar 29 2021 09:57:10.759 +0100 dvchost=userX_system procid=8189 appVersion=1 act=protect
msg=Protect external redirects.
stacktrace=com.waratek.spiracle.misc.SendRedirect.executeRequest(SendRedirect.java:36)\ncom.waratek.spiracle
e.misc.SendRedirect.executeRequest(SendRedirect.java:28)\ncom.waratek.spiracle.misc.SendRedirect.doGet(Send
Redirect.java:20)\njavax.servlet.http.HttpServlet.service(HttpServlet.java:624)\njavax.servlet.http.HttpSer
vlet.service(HttpServlet.java:731)\nsun.reflect.GeneratedMethodAccessor39.invoke(Unknown Source)
\nsun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)\njava.lang.reflect.
Method.invoke(Method.java:498)\norg.apache.catalina.core.ApplicationFilterChain.internalDoFilter(Applicatio
nFilterChain.java:303)\norg.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.jav
a:208)\norg.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.java:52)\nsun.reflect.NativeMethodAcc
essorImpl.invoke0(Native Method)
\nsun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)\nsun.reflect.DelegatingMeth
odAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)\njava.lang.reflect.Method.invoke(Method.java:
498)\norg.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:241)\nor
g.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:208)\norg.apache.catalin
a.core.StandardWrapperValve.invoke(StandardWrapperValve.java:218)\norg.apache.catalina.core.StandardContext
Valve.invoke(StandardContextValve.java:122)\norg.apache.catalina.authenticator.AuthenticatorBase.invoke(Aut
henticatorBase.java:505)\norg.apache.catalina.core.StandardHostValve.invoke(StandardHostValve.java:169)\nor
g.apache.catalina.valves.ErrorReportValve.invoke(ErrorReportValve.java:103)\norg.apache.catalina.valves.Acc
essLogValve.invoke(AccessLogValve.java:956)\norg.apache.catalina.core.StandardEngineValve.invoke(StandardEn
gineValve.java:116)\norg.apache.catalina.connector.CoyoteAdapter.service(CoyoteAdapter.java:442)\norg.apach
e.coyote.http11.AbstractHttp11Processor.process(AbstractHttp11Processor.java:1082)\norg.apache.coyote.Abstr
actProtocol$AbstractConnectionHandler.process(AbstractProtocol.java:623)\norg.apache.tomcat.util.net.JIoEnd
point$SocketProcessor.run(JIoEndpoint.java:316)\njava.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPo
olExecutor.java:1149)\njava.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)\norg
.apache.tomcat.util.threads.TaskThread$WrappingRunnable.run(TaskThread.java:61)\njava.lang.Thread.run(Threa
d.java:748) redirectLocation=http://www.waratek.com localIpAddress=0:0:0:0:0:0:1 localName=ip6-localhost
serverName=localhost httpSessionId=5D7CE07F605C3A6ABCFDB35D065A95E5 taintSource=HTTP_SERVLET
httpRequestUri=/spiracle/SendRedirect httpCookies=JSESSIONID\=5D7CE07F605C3A6ABCFDB35D065A95E5
remoteIpAddress=0:0:0:0:0:0:1
```

The following mod detects open redirect attacks that originate from an HTTP/HTTPS request:

```
app("Open Redirect mod 2"):
  requires(version: ARMR/2.2)

  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/2.2)

  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 sub-domain or root domain of the redirect URL is the different to the application's domain.

```
app("Open Redirect mod 4"):
  requires(version: ARMR/2.2)


  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
```


2.10.4 XSS Security Feature (22.x.x)

2.10.4.1 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.

2.10.4.2 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	<p>This allows the user to specify the source of the untrusted data. The following three sources are supported:</p> <ul style="list-style-type: none"> • <code>http</code> data introduced via HTTP/HTTPS requests • database data introduced via JDBC connections • <code>deserialization</code> data introduced via Java or XML deserialization <p>The rule will trigger if the source of the untrusted data matches that specified in the rule.</p> <p>If no value is specified then a default value of <code>http</code> is used.</p> <p>An exception will be thrown if an unsupported value is provided.</p>
-------	--

response	<p>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) :-</p> <ul style="list-style-type: none">• a quoted string• a list of one or more quoted-strings <p>If no value is specified then protection will be applied to all HTTP endpoints by default.</p> <p>If a string value is specified then it must:</p> <ul style="list-style-type: none">• not be empty• be a valid relative URI
----------	---

2.10.4.3 When (Event)

xss	<p>This declaration switches on the XSS security feature and must be declared with the mandatory parameter <code>html</code>.</p> <p>The following options may also be specified:</p> <ul style="list-style-type: none"> • <code>policy</code> <ul style="list-style-type: none"> • this allows the user to determine how conservative to configure the XSS security feature • this can be set to either: <ul style="list-style-type: none"> • <code>loose</code> - enable protection for user controlled changes to the HTML response that actively exploit the application • <code>strict</code> - enable protection for injection of untrusted data into HTML response • if this option is not specified the default value is set to <code>loose</code>. 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 4.4², 4.5³ and 4.9⁴ 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 3.2.5⁵ 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. <ul style="list-style-type: none"> • <code>async</code> • <code>autocomplete</code> • <code>autoplay</code> • <code>crossorigin</code> • <code>href</code> • <code>integrity</code> • <code>src</code> • <code>srcset</code> • <code>target</code> • <code>text</code> • <code>type</code>
-----	--

² <https://www.w3.org/TR/html5/grouping-content.html#grouping-content>

³ <https://www.w3.org/TR/html5/textlevel-antics.html#textlevel-antics>

⁴ <https://www.w3.org/TR/html5/tabular-data.html#tabular-data>

⁵ <https://www.w3.org/TR/html52/dom.html#global-attributes>

2.10.4.4 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.

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.

2.10.4.5 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/2.3)
  http("XSS"):
    response()
    xss(html)
    protect(message: "XSS attacked identified and blocked", severity: Very-High)
  endhttp
endapp
```

Logging

A log entry similar to the following is generated when above `http` rules identify an XSS attack:

```
<9>1 2021-03-29T11:54:42.717+01:00 userX_system java 15891 - - CEF:0|ARMR:XSS Mod|XSS Mod|2.3|XSS|Execute
Rule|Very-High|rt=Mar 29 2021 11:54:42.717 +0100 dvchost=userX_system procid=15891 appVersion=1 act=protect
msg=XSS attacked identified and blocked payload=<script>alert(1)</script>
httpSessionId=E654F722AAFA3BF44F0D0BD4FB91134C taintSource=HTTP_SERVLET httpRequestUri=/spiracle/xss.jsp
httpCookies=JSESSIONID\=E654F722AAFA3BF44F0D0BD4FB91134C remoteIpAddress=0:0:0:0:0:0:1
```

2.10.4.6 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/2.3)
  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|2.3|XSS|Execute Rule|Very-High|rt=Mar 29 2021 10:36:49.591 +0100 dvchost=userX_system
procid=12043 appVersion=1 act=protect msg=XSS attacked identified and blocked
stacktrace=org.apache.jsp.xss_jsp._jspx_meth_c_005fforEach_005f0(xss_jsp.java:305)\norg.apache.jsp.xss_jsp.
_jspService(xss_jsp.java:159)\norg.apache.jasper.runtime.HttpJspBase.service(HttpJspBase.java:70)\njavax.se
rvlet.http.HttpServlet.service(HttpServlet.java:731)\nsun.reflect.NativeMethodAccessorImpl.invoke0(Native
Method)\nsun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)\nsun.reflect.Delegat
ingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)\njava.lang.reflect.Method.invoke(Method.
java:498)\norg.apache.jasper.servlet.JspServletWrapper.service(JspServletWrapper.java:439)\norg.apache.jsp
er.servlet.JspServlet.serviceJspFile(JspServlet.java:395)\norg.apache.jasper.servlet.JspServlet.service(Jsp
Servlet.java:339)\njavax.servlet.http.HttpServlet.service(HttpServlet.java:731)\nsun.reflect.NativeMethodAc
cessorImpl.invoke0(Native Method)
\nsun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)\nsun.reflect.DelegatingMeth
odAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)\njava.lang.reflect.Method.invoke(Method.java:
498)\norg.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:303)\nor
g.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:208)\norg.apache.tomcat.
websocket.server.WsFilter.doFilter(WsFilter.java:52)\nsun.reflect.NativeMethodAccessorImpl.invoke0(Native
Method)\nsun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)\nsun.reflect.Delegat
ingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)\njava.lang.reflect.Method.invoke(Method.
java:498)\norg.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:241
)\norg.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:208)\norg.apache.ca
talina.core.StandardWrapperValve.invoke(StandardWrapperValve.java:218)\norg.apache.catalina.core.StandardCo
ntextValve.invoke(StandardContextValve.java:122)\norg.apache.catalina.authenticator.AuthenticatorBase.invo
ke(AuthenticatorBase.java:505)\norg.apache.catalina.core.StandardHostValve.invoke(StandardHostValve.java:169
)\norg.apache.catalina.valves.ErrorReportValve.invoke(ErrorReportValve.java:103)\norg.apache.catalina.valve
s.AccessLogValve.invoke(AccessLogValve.java:956)\norg.apache.catalina.core.StandardEngineValve.invoke(Stand
ardEngineValve.java:116)\norg.apache.catalina.connector.CoyoteAdapter.service(CoyoteAdapter.java:442)\norg.
apache.coyote.http11.AbstractHttp11Processor.process(AbstractHttp11Processor.java:1082)\norg.apache.coyote.
AbstractProtocol$AbstractConnectionHandler.process(AbstractProtocol.java:623)\norg.apache.tomcat.util.net.J
IoEndpoint$SocketProcessor.run(JIoEndpoint.java:316)\njava.util.concurrent.ThreadPoolExecutor.runWorker(Thr
eadPoolExecutor.java:1149)\njava.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)
\norg.apache.tomcat.util.threads.TaskThread$WrappingRunnable.run(TaskThread.java:61)\njava.lang.Thread.run(
Thread.java:748) payload=<script>alert(1)</script> httpSessionId=5D7CE07F605C3A6ABCADB35D065A95E5
taintSource=HTTP_SERVLET httpRequestUri=/spiracle/xss.jsp
httpCookies=JSESSIONID\=5D7CE07F605C3A6ABCADB35D065A95E5 remoteIpAddress=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/2.2)
  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/2.2)
  http("XSS"):
    xss(html, options: {policy: loose})
    response()
    input(http, database, deserialization)
    protect(message: "XSS attacked identified", severity: Medium)
  endhttp
endapp

```


2.10.5 Input Validation Security Feature (22.x.x)

2.10.5.1 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.

2.10.5.2 Given (Condition)

To enable the input validation security feature using the ARMR `http` rule the user specifies the `request` declaration.


request	<p>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 <code>paths</code> and the value can be one of the following (indicating specifically targeted HTTP endpoints) :-</p> <ul style="list-style-type: none"> • a quoted string • a list of one or more quoted-strings <p>If no value is specified then protection will be applied to all HTTP endpoints by default.</p> <p>If a string value is specified then it must:</p> <ul style="list-style-type: none"> • not be empty • be a valid relative URI
---------	--

2.10.5.3 When (Event)

validate	<p>Two separate key-value pairs are required for this declaration to switch on input validation protection. Valid values for the first key include:</p> <ul style="list-style-type: none"> • <code>parameters</code>, <code>cookies</code>, <code>headers</code> <p>Valid values for the second key include:</p> <ul style="list-style-type: none"> • <code>is</code>
headers	<ul style="list-style-type: none"> • The <code>headers</code> key is used to enable input validation of HTTP request headers. • The value of the <code>headers</code> key defines the names of one or more HTTP request headers whose values must be validated. • Empty header names are not allowed.
parameters	<ul style="list-style-type: none"> • The <code>parameters</code> key is used to enable input validation of HTTP request parameters. • The value of the <code>parameters</code> key defines the names of one or more HTTP request parameters whose values must be validated. • Empty parameter names are not allowed
cookies	<ul style="list-style-type: none"> • The <code>cookies</code> key is used to enable input validation of HTTP request cookies. • The value of the <code>cookies</code> key defines the names of one or more HTTP request cookies whose values must be validated. • Empty cookie names are not allowed

is	<ul style="list-style-type: none"> • The <code>is</code> 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 <code>is</code> key are: <ul style="list-style-type: none"> • <code>integer</code> • <code>integer-positive</code> • <code>integer-unsigned</code> • <code>alphanumeric</code> • <code>sql-no-single-quotes</code> • <code>sql-no-double-quotes</code> • <code>html-no-single-quotes</code> • <code>html-no-double-quotes</code> • <code>html-attribute-unquoted</code> • <code>html-text</code> • 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
----	--

2.10.5.4 Then (Action)

protect	<p>HTTP targets that fail validation are stripped from the request. If configured, a log message is generated with details of the event.</p>
detect	<p>Monitoring mode: the application behaves as normal.</p> <p>A log message is generated with details of the HTTP request target that fails validation.</p> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> <p> A log message must be specified with this action.</p> </div>
allow	<p>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 <code>http</code> rule for the same target in <code>protect</code> mode.</p>

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.

2.10.5.5 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/2.3)
  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 Input Validation mod|HTTP
Input Validation mod|2.3|HTTP single parameter validation|Execute Rule|Medium|rt=Mar 29 2021 11:55:47.243
+0100 dvchost=userX_system procid=15891 appVersion=1 act=protect msg=number parameter was not an integer
parameters=number validationRule=integer value=<script>alert(1)</script> httpRequestUri=/spiracle/xss.jsp
remoteIpAddress=0:0:0:0:0:0:1 httpSessionId=E654F722AAFA3BF44F0D0BD4FB91134C
httpCookies=JSESSIONID\=E654F722AAFA3BF44F0D0BD4FB91134C
```

2.10.5.6 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/2.3)
  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, stacktrace: "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 Input Validation mod - with
stacktrace|HTTP Input Validation mod - with stacktrace|2.3|HTTP single parameter validation|Execute Rule|
Medium|rt=Mar 29 2021 11:57:06.951 +0100 dvchost=userX_system procid=15891 appVersion=1 act=protect
msg=number parameter was not an integer stacktrace=org.apache.jsp.xss_jsp._jspService(xss_jsp.java:119)\nor
g.apache.jasper.runtime.HttpJspBase.service(HttpJspBase.java:70)\njavax.servlet.http.HttpServlet.service(Ht
tpServlet.java:731)\nsun.reflect.GeneratedMethodAccessor32.invoke(Unknown Source)
\nsun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)\njava.lang.reflect.
Method.invoke(Method.java:498)\norg.apache.jasper.servlet.JspServletWrapper.service(JspServletWrapper.java:
439)\norg.apache.jasper.servlet.JspServlet.serviceJspFile(JspServlet.java:395)\norg.apache.jasper.servlet.J
spServlet.service(JspServlet.java:339)\njavax.servlet.http.HttpServlet.service(HttpServlet.java:731)\nsun.r
eflect.GeneratedMethodAccessor32.invoke(Unknown Source)
\nsun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)\njava.lang.reflect.
Method.invoke(Method.java:498)\norg.apache.catalina.core.ApplicationFilterChain.internalDoFilter( Applicatio
nFilterChain.java:303)\norg.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.jav
a:208)\norg.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.java:52)\nsun.reflect.GeneratedMethod
Accessor46.invoke(Unknown Source)
\nsun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)\njava.lang.reflect.
Method.invoke(Method.java:498)\norg.apache.catalina.core.ApplicationFilterChain.internalDoFilter( Applicatio
nFilterChain.java:241)\norg.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.jav
a:208)\norg.apache.catalina.core.StandardWrapperValve.invoke(StandardWrapperValve.java:218)\norg.apache.cat
alina.core.StandardContextValve.invoke(StandardContextValve.java:122)\norg.apache.catalina.authenticator.Au
thenticatorBase.invoke(AuthenticatorBase.java:505)\norg.apache.catalina.core.StandardHostValve.invoke(Stand
ardHostValve.java:169)\norg.apache.catalina.valves.ErrorReportValve.invoke(ErrorReportValve.java:103)\norg.
apache.catalina.valves.AccessLogValve.invoke(AccessLogValve.java:956)\norg.apache.catalina.core.StandardEng
ineValve.invoke(StandardEngineValve.java:116)\norg.apache.catalina.connector.CoyoteAdapter.service(CoyoteAd
apter.java:442)\norg.apache.coyote.http11.AbstractHttp11Processor.process(AbstractHttp11Processor.java:1082
)\norg.apache.coyote.AbstractProtocol$AbstractConnectionHandler.process(AbstractProtocol.java:623)\norg.apa
che.tomcat.util.net.JIoEndpoint$SocketProcessor.run(JIoEndpoint.java:316)\njava.util.concurrent.ThreadPoolE
xecutor.runWorker(ThreadPoolExecutor.java:1149)\njava.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadP
oolExecutor.java:624)\norg.apache.tomcat.util.threads.TaskThread$WrappingRunnable.run(TaskThread.java:61)\n
java.lang.Thread.run(Thread.java:748) parameters=number validationRule=integer value=<script>alert(1)</
script> httpRequestUri=/spiracle/xss.jsp remoteIpAddress=0:0:0:0:0:1
httpSessionId=E654F722AAFA3BF44F0D0BD4FB91134C httpCookies=JSESSIONID\E654F722AAFA3BF44F0D0BD4FB91134C
```

The following mod ensures the HTTP request cookie named “loginId” is a positive integer. This applies to the “index.jsp” page of the application only.

```
app("HTTP Input Validation mod 2"):
  requires(version: ARMR/2.2)
  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/2.2)
  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/2.2)
  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/2.2)
  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/2.2)
  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/2.2)
  http("Monitoring mode - multiple parameters with multiple validation rules"):
    request()
    validate(parameters: ["items", "total"], is: [sql-no-single-quotes, sql-no-double-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/2.2)
  http("HTTP single parameter validation for all HTTP requests - default log message"):
    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/2.2)
  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

```

2.10.6 HTTP/HTTPS Response Header Addition Feature (22.x.x)

2.10.6.1 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](https://owasp.org/www-community/attacks/Clickjacking)⁶ attacks, also known as UI redressing.

⁶ <https://owasp.org/www-community/attacks/Clickjacking>

- **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.

2.10.6.2 Given (Condition)

The HTTP/HTTPS response header addition feature is enabled using the ARMR `http` rule. The following condition must be specified - response.

response	<p>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 <code>paths</code> and the value can be one of the following (indicating specifically targeted HTTP endpoints) :-</p> <ul style="list-style-type: none"> • a quoted string • a list of one or more quoted-strings <p>If no value is specified then custom headers will be applied to all HTTP endpoints by default.</p> <p>If a string value is specified then it must:</p> <ul style="list-style-type: none"> • not be empty • be a valid relative URI
----------	---

2.10.6.3 Then (Action)

protect	<p>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:</p> <ul style="list-style-type: none"> • <code>http-response: {set-header: {headerName: "headerValue"}}</code> <p>The <code>set-header</code> declaration can contain multiple headers providing each one has a unique header name. Each header is represented as a key-value pair where:</p> <ul style="list-style-type: none"> • the key is the header name • the value is the header value, which can be one of: <ul style="list-style-type: none"> • string literal • integer • float • boolean
---------	--

2.10.6.4 Examples

The following examples show, for each of the headers listed in the introduction, how the ARMR `httprule` 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 (XSS⁷) attacks:

```
app("Header response addition mod"):
requires(version: ARMR/2.2)

http("Add custom headers to HTTP/S response"):
response()
protect(http-response: {set-header: {X-XSS-Protection: 1}}, message: "Setting custom header.",
severity: High)
endhttp

endapp
```

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.

More information about the X-XSS-Protection response header: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-XSS-Protection>

⁷ <https://developer.mozilla.org/en-US/docs/Glossary/XSS>

- The **X-Content-Type-Options** response HTTP header is a marker used by the server to indicate that the **MIME types**⁸ advertised in the Content-Type headers should not be changed and be followed. This allows to opt-out of **MIME type sniffing**⁹.

```
app("Header response addition mod"):
requires(version: ARMR/2.2)

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
```

More information about the **X-Content-Type-Options** response header: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Content-Type-Options>

- The **X-Frame-Options** HTTP response header can be used to indicate whether or not 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**¹⁰ 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/2.2)

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/2.2)

http("Add custom headers to HTTP/S response"):
response()
protect(http-response: {set-header: {X-Frame-Options: "SAMEORIGIN"}}, message: "Setting custom
header.", severity: High)
endhttp

endapp
```

⁸ https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types

⁹ https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types#MIME_sniffing

¹⁰ <https://owasp.org/www-community/attacks/Clickjacking>

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/2.2)

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
```

More information about the `X-Frame-Options` response header: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Option>¹¹

More information on how to use HTTP response headers to protect against Clickjacking: https://cheatsheetseries.owasp.org/cheatsheets/Clickjacking_Defense_Cheat_Sheet.html

- 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`¹² 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](https://www.owasp.org/index.php/Clickjacking)¹³. 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/2.2)

http("Add custom headers to HTTP/S response"):
response()
protect(http-response: {set-header: {Content-Security-Policy: "frame-ancestors '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`.

¹¹ <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options>

¹² <https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/HTML5>

¹³ <https://www.owasp.org/index.php/Clickjacking>

```

app("Header response addition mod"):
requires(version: ARMR/2.2)

http("Add custom headers to HTTP/S response"):
response()
protect(http-response: {set-header: {Content-Security-Policy: "frame-ancestors '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 ¹⁴the other trusted URIs to frame pages of this site. This setting is recommended if the application allows specific third party applications or web sites to frame its pages.

```

app("Header response addition mod"):
requires(version: ARMR/2.2)

http("Add custom headers to HTTP/S response"):
response()
protect(http-response: {set-header: {Content-Security-Policy: "frame-ancestors 'self' *.somesite.com
https://trusted.site.com"}}, message: "Setting custom header.", severity: High)
endhttp

endapp

```

More information about the **Content-Security-Policy** response header: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy>

More information on how to use HTTP response headers to protect against Clickjacking: https://cheatsheetseries.owasp.org/cheatsheets/Clickjacking_Defense_Cheat_Sheet.html

- The HTTP **Strict-Transport-Security** response header (often abbreviated as **HSTS**¹⁵) 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/2.2)

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

```

More information about the Strict-Transport-Security response header: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-Security>

¹⁴ <http://somesite.com/>

¹⁵ <https://developer.mozilla.org/en-US/docs/Glossary/HSTS>

- The **Access-Control-Allow-Origin** response header indicates whether the response can be shared with requesting code from the given [origin](#)¹⁶.

```
app("Header response addition mod"):
  requires(version: ARMR/2.2)

  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
```

More information about the **Access-Control-Allow-Origin** response header: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Access-Control-Allow-Origin>

2.10.7 Session Fixation Security Feature (22.x.x)


2.10.7.1 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).
 - Waratek 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.

¹⁶ <https://developer.mozilla.org/en-US/docs/Glossary/origin>

2.10.7.2 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`.

<code>request</code>	This declaration allows the user to define an ARMR <code>http</code> rule that will act upon receiving a user request.
----------------------	--

2.10.7.3 When (Event)

<code>authenticate</code>	This condition allows the user to specify that the ARMR <code>http</code> rule should authenticate a user at login. The following parameter is supported: <ul style="list-style-type: none"> • <code>user</code>
---------------------------	---

2.10.7.4 Then (Action)

<code>protect</code>	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: <ul style="list-style-type: none"> • <code>http-session: regenerate-id</code>
----------------------	--

2.10.7.5 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/2.2)
  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.

2.10.8 HTTP Verb Tampering (22.x.x)

2.10.8.1 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.

2.10.8.2 Given (Condition)

To enable the HTTP Verb Tampering security feature using the ARMR `http` rule the user specifies the `request` declaration.


request	<p>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 <code>paths</code> and the value can be one of the following (indicating specifically targeted HTTP endpoints) :-</p> <ul style="list-style-type: none"> • a quoted string • a list of one or more quoted-strings <p>If no value is specified then protection will be applied to all HTTP endpoints by default.</p> <p>If a string value is specified then it must:</p> <ul style="list-style-type: none"> • not be empty • be a valid relative URI
---------	--

2.10.8.3 When (Event)

validate	<p>To enable HTTP verb tampering protection the user must provide the <code>method</code> parameter to this declaration.</p> <p>In addition, the key-value pair with key <code>is</code> must also be defined.</p>
----------	--

method	The method key signifies that HTTP verb (method) tampering protection is in use
is	<p>The is key indicates the permitted values of HTTP verbs for a given request.</p> <p>Possible values for the is key are:</p> <ul style="list-style-type: none"> • GET • POST • HEAD • PUT • DELETE • CONNECT • OPTIONS • TRACE • PATCH

2.10.8.4 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	<p>Monitoring mode: the application behaves as normal.</p> <p>A log message is generated with details of the HTTP request target that fails validation.</p> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> <p> A log message must be specified with this action.</p> </div>
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.

2.10.8.5 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/2.3)
  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|2.3|HTTP method tampering protection, all HTTP endpoints|Execute Rule|Very-High|rt=Mar 30 2021 17:43:54.537 +0100 dvchost=userX_system procid=32008 appVersion=1 act=protect msg=HTTP method/verb is not GET or POST validationRule=OneOf:[GET, POST] value=DELETE httpRequestUri=/webapp/index.jsp remoteIpAddress=127.0.0.1 httpSessionId=3153E581A645E2A54D3C12D3928473BC httpCookies=JSESSIONID\=3153E581A645E2A54D3C12D3928473BC

```

2.10.8.6 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/2.2)
  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/2.2)
  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

```