Recommended Impact Reduction Policy (In detect mode)

The below rules are the recommended impact reduction policy for mitigating the major vulnerability categories:

```
sqli:database:oracle;permit=query-provided:allow:warn

file:exec:*:allow:warn

path:traversal:relative:allow:warn

path:traversal:absolute:allow:warn

xss:html:*;source=database,httprequest,deserialization:allow:warn

deserial:harden:system:allow:warn

deserial:harden:dos:allow:warn

redirect:servlet:external:allow:warn
```

SQL Injection (CWE-89)

SQL Injection (SQLi) attacks constitute one of the main forms of vulnerability in modern software systems.

This rule category allows an administrator to specify that all input received via untrusted sources (such as HTTP requests, Database systems, RMI, etc.) will be checked for attempts to exploit SQLi vulnerabilities.

The above SQLi rule is configured in detect mode, meaning it will log to file but not prevent SQLi attempts. The rule can be configured to block SQLi attempts as follows:

```
sqli:database:oracle;permit=query-provided:deny:warn
```

Command Execution (CWE-78)

File operations, such as opening for reading or writing, modifying file attributes (such as last modified dates, etc.), can be controlled. The above rule can is configured to log to file any attempt by the protected process to fork operating system processes. The rule can be configured to block the forking of processes as follows:

```
file:exec:*:deny:warn
```

Path Traversal (CWE-23, CWE-36)

Path Traversal vulnerabilities are included in the SANS Top 25 Most Dangerous Software Errors. An application vulnerable to Path Traversal attacks uses unvalidated/unsanitized external input to construct a path that is intended to identify a file or directory which is located underneath a restricted parent directory. This causes the pathname to resolve to a location that is outside of the restricted directory.

Path Traversal attacks are also known as Directory Traversal attacks. There are two generic types of Path Traversal attacks:

- The Relative Path Traversal (CWE-23).
- The Absolute Path Traversal (CWE-36).

This rule category allows an administrator to enable protection for Path Traversal attacks. Two high-level descriptions of the path rule are:

- Block any file operation which uses a user-constructed path that allows a user to traverse back to the parent path.
- Block any file operation which uses a user-constructed path that allows a user to specify an absolute path to a file or a directory.

The above path traversal rules are configured to log to file any path traversal attempts. The rule can be configured in blocking mode as follows:

```
path:traversal:relative:deny:warn

path:traversal:absolute:deny:warn
```

Cross Site Scripting (XSS) (CWE-80)

Cross-site scripting (XSS) is one of the most dangerous and commonly found vulnerabilities in Web applications. This rule category allows an administrator to specify that all input received via untrusted sources (such as HTTP requests, Database systems, RMI, etc.) will be checked for attempts to exploit XSS vulnerabilities.

At a high-level, an XSS rule for protection stops the processing of the JSP/Servlet if untrusted input breaks the HTML context.

The above XSS rule logs XSS exploit attempts to file but does not block them. The rule can be configured in blocking mode as follows:

```
xss:html:*;source=database,httprequest,deserialization:deny:warn
```

Deserialization of Untrusted Data (CWE-502)

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 (CWE-502). Serialization is used in several components of the JVM as well as in numerous third-party frameworks and dependencies.

The Deserial security rule addresses the deserialization attacks by lowering the system privileges during deserialization (CWE-250). 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 (known as zero-day) that tries to access or change the state of the system fails.

The Deserial security rule is applicable to all Java applications that receive untrusted serialized objects and perform deserialization without proper validation or other security controls. This 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 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 the memory, by setting default limits to control the interaction frequency of the deserialized objects with the system resources (CWE-799). 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 (CWE-400).

The above Deserial rules are configured to log deserialization attempts but not prevent them. The rule can be configured in blocking mode as follows:

```
deserial:harden:system:deny:warn

deserial:harden:dos:deny:warn
```

Open Redirect (CWE-601)

Web applications that **redirect** the user to another location based on user-controlled input are vulnerable to Open Redirect attacks. In such attacks, the adversary 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. This rule category allows an administrator to enable protection for Open Redirect attacks.

The above rule is configured to log redirection attempts, but not prevent them. The rule can be configured in blocking mode as follows:

```
redirect:servlet:external:deny:warn
```

Full Policy in Protect Mode

```
sqli:database:oracle;permit=query-provided:deny:warn

file:exec:*:deny:warn

session:protect:regenerateSID::

path:traversal:relative:deny:warn

path:traversal:absolute:deny:warn

xss:html:*;source=database,httprequest,deserialization:deny:warn

deserial:harden:system:deny:warn

deserial:harden:dos:deny:warn

redirect:servlet:external:deny:warn
```

Phase Two Rules

The additional rules below can be included to provide targeted hardening of the protected application. As they require some additional configuration relative to the application in question, we typically deploy them in a second phase once the Default Impact Reduction rules have been migrated to production.

```
network:accept:*:allow:warn

network:bind:*:allow:warn

network:connect:*:allow:warn

network:serverbind:*:allow:warn

file:read:/etc/*:allow:warn

file:read:/tmp/*:allow:warn

file:write:/etc/*:allow:warn

file:write:/tmp/*:allow:warn

session:protect:regenerateSID::
```

Network I/O

Access control is supported for all network actions the Waratek Agent may perform or attempt to perform. One example is to prevent a ServerSocket binding to a port that needs to be free for another purpose. Another example configuration could be to log warnings when a ServerSocket connection is received from an IP address that is considered suspicious.

Addresses may be specified in the rules using IP addresses. Some high-level examples of rules are:

- Only accept socket connections from a defined set of remote hosts
- Only allow outbound connections to a defined set of remote hosts
- Restrict the Waratek Agent to specific inbound/outbound ports

File I/O

File operations, such as opening for reading or writing, modifying file attributes (such as last modified dates, etc.), can be controlled. Access control is supported, such as disallowing

modification of JAR files so that Trojan classes cannot be inserted into a JAR file unknowingly.

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, such as .rules and .xml
- Prohibit execution of external processes

Session Fixation* (CWE-384)

HTTP Session Fixation is an attack 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.

```
session:protect:regenerateSID::
```

***If the above rule is in place, Session Fixation will be applied. This rule does not have separate protect/detect capabilities.**