Fortify Standalone Report Generator

Developer Workbook

akka-slf4j

Table of Contents

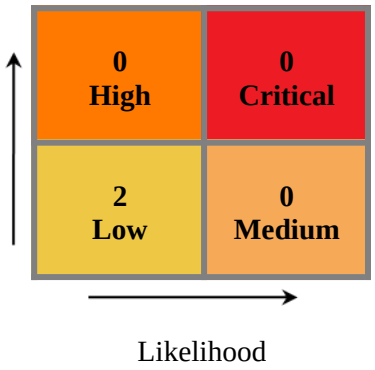
Executive Summary

This workbook is intended to provide all necessary details and information for a developer to understand and remediate the different issues discovered during the akka-slf4j project audit. The information contained in this workbook is targeted at project managers and developers.

This section provides an overview of the issues uncovered during analysis.

Project Name:	akka-slf4j
Project Version:	
SCA:	Results Present
WebInspect:	Results Not Present
WebInspect Agent:	Results Not Present
Other:	Results Not Present

Issues by Priority



Top Ten Critical Categories

This project does not contain any critical issues

Project Description

This section provides an overview of the Fortify scan engines used for this project, as well as the project meta-information.

SCA

Date of Last Analysis:	Jun 16, 2022, 11:46 AM	**Engine Version:**	21.1.1.0009
Host Name:	Jacks-Work-MBP.local	**Certification:**	VALID
Number of Files:	1	**Lines of Code:**	93

Rulepack Name	Rulepack Version
Fortify Secure Coding Rules, Extended, Java	2022.1.0.0007
Fortify Secure Coding Rules, Core, Scala	2022.1.0.0007
Fortify Secure Coding Rules, Extended, JSP	2022.1.0.0007
Fortify Secure Coding Rules, Core, Android	2022.1.0.0007
Fortify Secure Coding Rules, Extended, Content	2022.1.0.0007
Fortify Secure Coding Rules, Extended, Configuration	2022.1.0.0007
Fortify Secure Coding Rules, Core, Annotations	2022.1.0.0007
Fortify Secure Coding Rules, Community, Cloud	2022.1.0.0007
Fortify Secure Coding Rules, Core, Universal	2022.1.0.0007
Fortify Secure Coding Rules, Core, Java	2022.1.0.0007
Fortify Secure Coding Rules, Community, Universal	2022.1.0.0007

Issue Breakdown by Fortify Categories

The following table depicts a summary of all issues grouped vertically by Fortify Category. For each category, the total number of issues is shown by Fortify Priority Order, including information about the number of audited issues.

Category	Fortify Priority (audited/total)				Total Issues
	Critical	High	Medium	Low	
System Information Leak: Internal	0	0	0	0 / 2	0 / 2

Results Outline

System Information Leak: Internal (2 issues)

Abstract

Revealing system data or debugging information helps an adversary learn about the system and form a plan of attack.

Explanation

An internal information leak occurs when system data or debug information is sent to a local file, console, or screen via printing or logging. **Example 1:** The following code writes an exception to the standard error stream:

```
try {
    ...
} catch (Exception e) {
    e.printStackTrace();
}
```

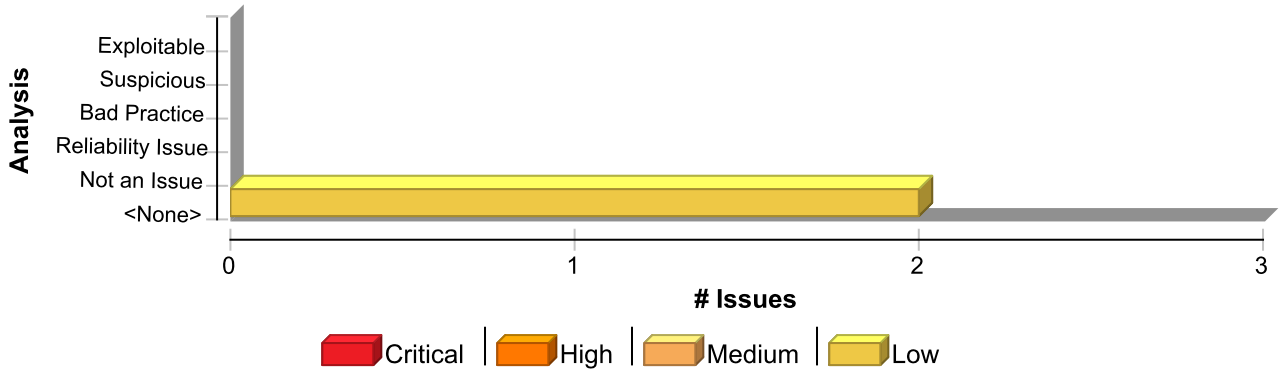
Depending upon the system configuration, this information can be dumped to a console, written to a log file, or exposed to a user. In some cases, the error message provides the attacker with the precise type of attack to which the system is vulnerable. For example, a database error message can reveal that the application is vulnerable to a SQL injection attack. Other error messages can reveal more oblique clues about the system. In `Example 1`, the leaked information could imply information about the type of operating system, the applications installed on the system, and the amount of care that the administrators have put into configuring the program. Information leaks are also a concern in a mobile computing environment. **Example 2:** The following code logs the stack trace of a caught exception on the Android platform.

```
...
try {
  ...
} catch (Exception e) {
    Log.e(TAG, Log.getStackTraceString(e));
}
...
```

Recommendation

Write error messages with security in mind. In production environments, turn off detailed error information in favor of brief messages. Restrict the generation and storage of detailed output that can help administrators and programmers diagnose problems. Debug traces can sometimes appear in non-obvious places (embedded in comments in the HTML for an error page, for example). Even brief error messages that do not reveal stack traces or database dumps can potentially aid an attacker. For example, an "Access Denied" message can reveal that a file or user exists on the system.

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
System Information Leak: Internal	2	0	0	2
Total	**2**	**0**	**0**	**2**

System Information Leak: Internal — Low

Package: <none>

Slf4jLogger.scala, line 76 (System Information Leak: Internal) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Data Flow)

Source Details

Source: java.lang.Throwable.getLocalizedMessage()
From: akka.event.slf4j.Slf4jLogger$$anonfun$receive$1$anonfun$applyOrElse$1.apply
File: Slf4jLogger.scala:78

```
75  case _ =>
76  Logger(logClass, logSource).error(
77  markerIfPresent(event),
78  if (message != null) message.toString else cause.getLocalizedMessage,
79  cause)
80  }
81  }
```

Sink Details

Sink: org.slf4j.Logger.error()
Enclosing Method: apply()
File: Slf4jLogger.scala:76
Taint Flags: EXCEPTIONINFO, SYSTEMINFO

```
73  case Error.NoCause | null =>
74  Logger(logClass, logSource).error(markerIfPresent(event), if (message != null) message.toString else null)
75  case _ =>
76  Logger(logClass, logSource).error(
77  markerIfPresent(event),
```

System Information Leak: Internal — Low

Package: <none>

Slf4jLogger.scala, line 76 (System Information Leak: Internal) — Low

```
78 if (message != null) message.toString else cause.getLocalizedMessage,
79 cause)
```

Slf4jLogger.scala, line 87 (System Information Leak: Internal) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Data Flow)

Source Details

Source: java.lang.Throwable.getLocalizedMessage()
From: akka.event.slf4j.Slf4jLogger$$anonfun$receive$1$anonfun$applyOrElse$2.apply
File: Slf4jLogger.scala:89

```
86 case e: LogEventWithCause =>
87 Logger(logClass, logSource).warn(
88 markerIfPresent(event),
89 if (message != null) message.toString else e.cause.getLocalizedMessage,
90 e.cause)
91 case _ =>
92 Logger(logClass, logSource).warn(markerIfPresent(event), if (message != null) message.toString else null)
```

Sink Details

Sink: org.slf4j.Logger.warn()
Enclosing Method: apply()
File: Slf4jLogger.scala:87
Taint Flags: EXCEPTIONINFO, SYSTEMINFO

```
84 withMdc(logSource, event) {
85 event match {
86 case e: LogEventWithCause =>
87 Logger(logClass, logSource).warn(
88 markerIfPresent(event),
89 if (message != null) message.toString else e.cause.getLocalizedMessage,
90 e.cause)
```