Fortify Standalone Report Generator

Developer Workbook

akka-osgi

Table of Contents

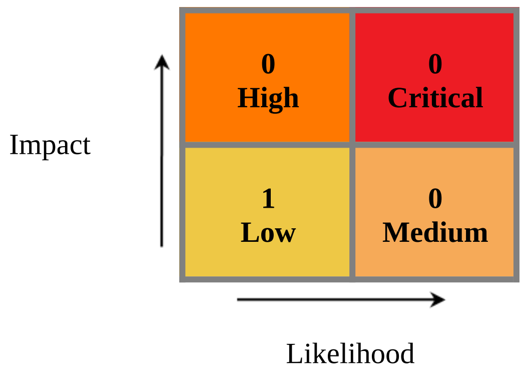
Executive Summary

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-osgi 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-osgi
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, 11:30 AM	**Engine Version:**	21.1.1.0009
Host Name:	Jacks-Work-MBP.local	**Certification:**	VALID
Number of Files:	4	**Lines of Code:**	118

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	
Code Correctness: Constructor Invokes Overridable Function	0	0	0	0 / 1	0 / 1

Results Outline

Code Correctness: Constructor Invokes Overridable Function (1 issue)

Abstract

A constructor of the class calls a function that can be overridden.

Explanation

When a constructor calls an overridable function, it may allow an attacker to access the `this` reference prior to the object being fully initialized, which can in turn lead to a vulnerability. **Example 1:** The following calls a method that can be overridden.

```
...
class User {
  private String username;
  private boolean valid;
  public User(String username, String password){
    this.username = username;
    this.valid = validateUser(username, password);
  }
  public boolean validateUser(String username, String password){
    //validate user is real and can authenticate
    ...
  }
  public final boolean isValid(){
    return valid;
  }
}
```

Since the function `validateUser` and the class are not `final`, it means that they can be overridden, and then initializing a variable to the subclass that overrides this function would allow bypassing of the `validateUser` functionality. For example:

```
...
class Attacker extends User{
  public Attacker(String username, String password){
    super(username, password);
  }
  public boolean validateUser(String username, String password){
    return true;
  }
}
...
class MainClass{
  public static void main(String[] args){
    User hacker = new Attacker("Evil", "Hacker");
    if (hacker.isValid()){
      System.out.println("Attack successful!");
    }else{
      System.out.println("Attack failed");
    }
  }
}
```

The code in `Example 1` prints "Attack successful!", since the `Attacker` class overrides the `validateUser()` function that is called from the constructor of the superclass `User`, and Java will first look in the subclass for functions called from the constructor.

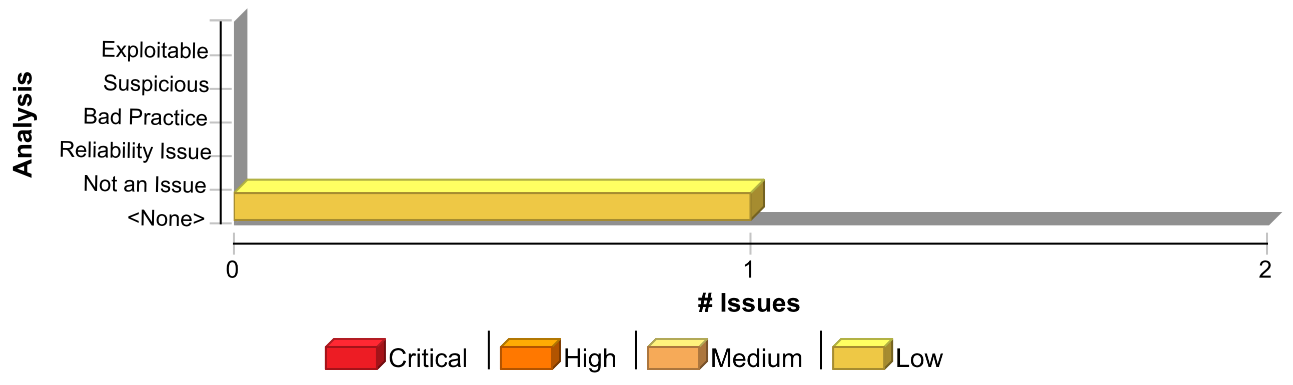
Recommendation

Constructors should not call functions that can be overridden, either by specifying them as `final`, or specifying the class as `final`. Alternatively if this code is only ever needed in the constructor, the `private` access specifier can be used, or the logic could be placed directly into the constructor of the superclass. **Example 2:** The following makes the class `final` to prevent the function from being overridden elsewhere.

```
...
final class User {
  private String username;
  private boolean valid;
  public User(String username, String password){
    this.username = username;
    this.valid = validateUser(username, password);
  }
  private boolean validateUser(String username, String password){
    //validate user is real and can authenticate
    ...
  }
  public final boolean isValid(){
    return valid;
  }
}
```

This example specifies the class as `final`, so that it cannot be subclassed, and changes the `validateUser()` function to `private`, since it is not needed elsewhere in this application. This is programming defensively, since at a later date it may be decided that the `User` class needs to be subclassed, which would result in this vulnerability reappearing if the `validateUser()` function was not set to `private`.

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
Code Correctness: Constructor Invokes Overridable Function	1	0	0	1
Total	**1**	**0**	**0**	**1**

Code Correctness: Constructor Invokes Overridable Function	Low

Package: akka.osgi

BundleDelegatingClassLoader.scala, line 42 (Code Correctness: Constructor Invokes Overridable Function)	**Low**

Issue Details

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.osgi

BundleDelegatingClassLoader.scala, line 42 (Code Correctness: Constructor Invokes Overridable Function) — Low

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: findTransitiveBundles
Enclosing Method: BundleDelegatingClassLoader()
File: BundleDelegatingClassLoader.scala:42
Taint Flags:

```
39 class BundleDelegatingClassLoader(bundle: Bundle, fallBackClassLoader: ClassLoader)
40  extends ClassLoader(fallBackClassLoader) {
41
42  private val bundles = findTransitiveBundles(bundle).toList
43
44  override def findClass(name: String): Class[_] = {
45  @tailrec def find(remaining: List[Bundle]): Class[_] = {
```