Fortify Security Report

Jan 28, 2020

psdravid

Executive Summary

Issues Overview

On Jan 28, 2020, a source code review was performed over the adminui code base. 157 files, 6,559 LOC (Executable) were scanned and reviewed for defects that could lead to potential security vulnerabilities. A total of 15 reviewed findings were uncovered during the analysis.

Issues by Fortify Priority Order	
High	15

Recommendations and Conclusions

The Issues Category section provides Fortify recommendations for addressing issues at a generic level. The recommendations for specific fixes can be extrapolated from those generic recommendations by the development group.

Project Summary

Code Base Summary

Code location:
Number of Files: 157
Lines of Code: 6559
Build Label: <No Build Label>

Scan Information

Scan time: 05:16
SCA Engine version: 19.1.0.2241
Machine Name: vclv99-200.hpc.ncsu.edu
Username running scan: psdravid

Results Certification

Results Certification Valid

Details:

Results Signature:

SCA Analysis Results has Valid signature

Rules Signature:

There were no custom rules used in this scan

Attack Surface

Attack Surface:
Private Information:
null.null.null

System Information:
null.null.null
null.null.getPatientId
null.null.lambda
null.null.restore
null.null.save

Filter Set Summary

Current Enabled Filter Set:
Quick View

Filter Set Details:

Folder Filters:
If [fortify priority order] contains critical Then set folder to Critical
If [fortify priority order] contains high Then set folder to High

If [fortify priority order] contains medium Then set folder to Medium
If [fortify priority order] contains low Then set folder to Low
Visibility Filters:
If impact is not in range [2.5, 5.0] Then hide issue
If likelihood is not in range (1.0, 5.0] Then hide issue

Audit Guide Summary

Audit guide not enabled

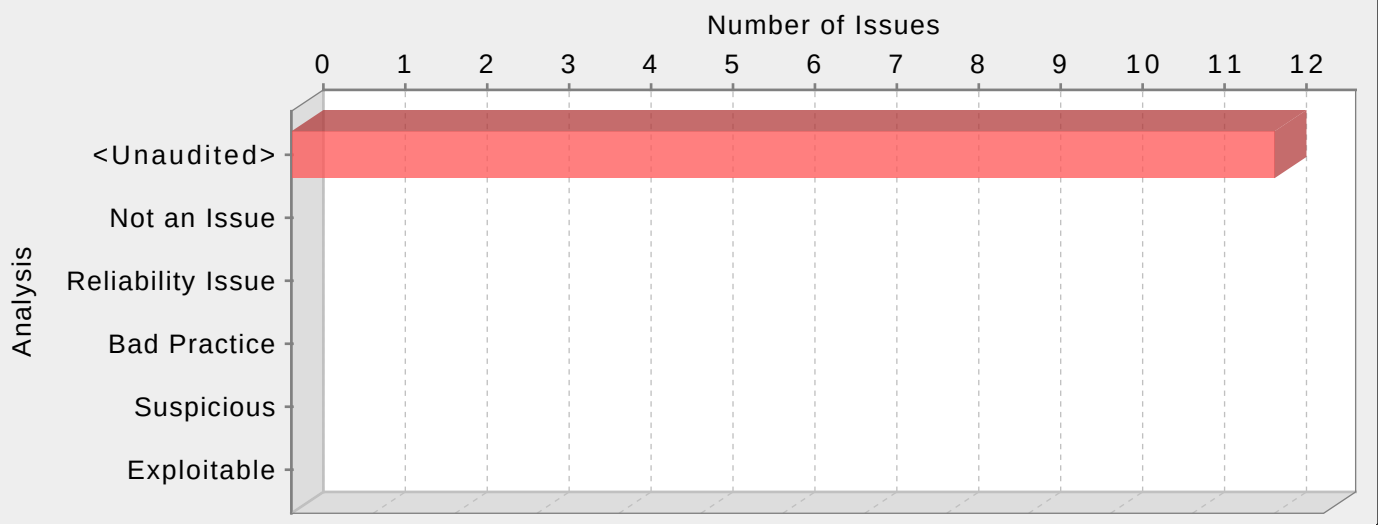
Results Outline

Overall number of results

The scan found 15 issues.

Vulnerability Examples by Category

Category: Password Management: Password in Configuration File (12 Issues)



Abstract:

Storing a plain text password in a configuration file may result in a system compromise.

Explanation:

Storing a plain text password in a configuration file allows anyone who can read the file access to the password-protected resource. Developers sometimes believe that they cannot defend the application from someone who has access to the configuration, but this attitude makes an attacker's job easier. Good password management guidelines require that a password never be stored in plain text.

Recommendations:

A password should never be stored in plain text. An administrator should be required to enter the password when the system starts. If that approach is impractical, a less secure but often adequate solution is to obfuscate the password and scatter the de-obfuscation material around the system so that an attacker has to obtain and correctly combine multiple system resources to decipher the password.

Some third-party products claim the ability to manage passwords in a more secure way. For example, WebSphere Application Server 4.x uses a simple XOR encryption algorithm for obfuscating values, but be skeptical about such facilities. WebSphere and other application servers offer outdated and relatively weak encryption mechanisms that are insufficient for security-sensitive environments. For a secure solution the only viable option is a proprietary one.

Tips:

1. Fortify Static Code Analyzer searches configuration files for common names used for password properties. Audit these issues by verifying that the flagged entry is used as a password and that the password entry contains plain text.

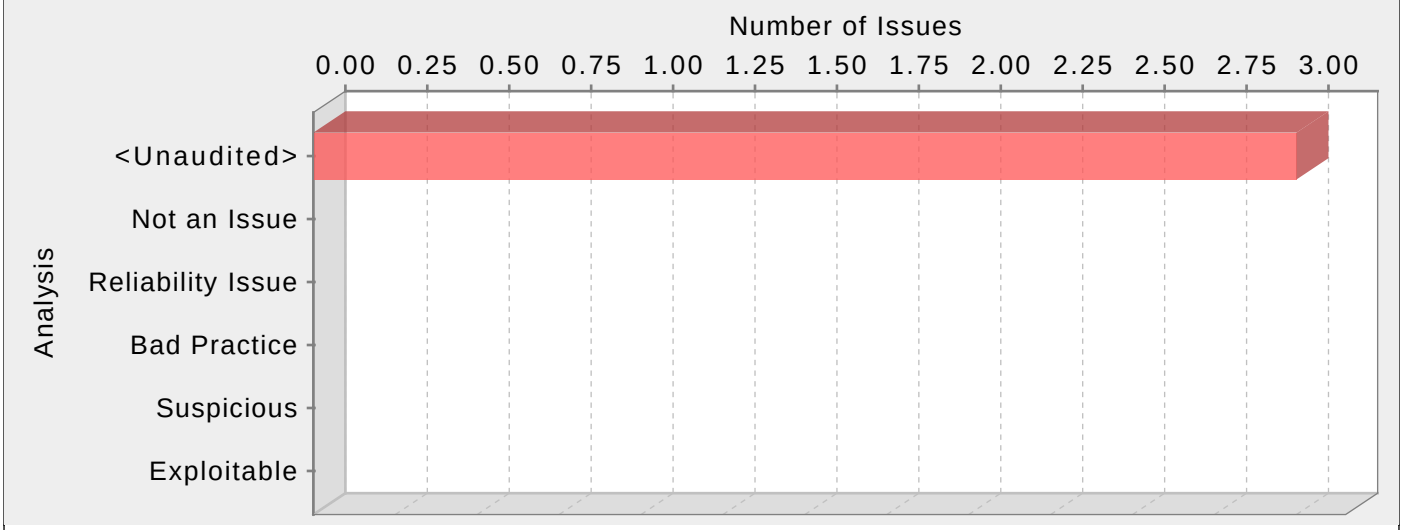
2. If the entry in the configuration file is a default password, require that it be changed in addition to requiring that it be obfuscated in the configuration file.

messages.properties, line 75 (Password Management: Password in Configuration File)

Fortify Priority:	High	**Folder**	High
Kingdom:	Environment		
Abstract:	Storing a plain text password in a configuration file may result in a system compromise.		
Sink:	messages.properties:75 adminui.myAccount.password.label()		

```
73  adminui.myAccount=My Account
74  adminui.myAccount.myLanguages.title=My Languages
75  adminui.myAccount.password.label=Password
76  adminui.myAccount.changeSecretQuestion.label=Secret Question
77  adminui.myAccount.defaults.label=User Defaults
```

Category: Password Management: Hardcoded Password (3 Issues)



Abstract:

Hardcoded passwords may compromise system security in a way that cannot be easily remedied.

Explanation:

It is never a good idea to hardcode a password. Not only does hardcoding a password allow all of the project's developers to view the password, it also makes fixing the problem extremely difficult. After the code is in production, the password cannot be changed without patching the software. If the account protected by the password is compromised, the owners of the system must choose between security and availability.

Example: The following code uses a hardcoded password to connect to an application and retrieve address book entries:

...
obj = new XMLHttpRequest();
obj.open('GET','/fetchusers.jsp?id='+form.id.value,'true','scott','tiger');
...

This code will run successfully, but anyone who accesses the containing web page will be able to view the password.

Recommendations:

Passwords should never be hardcoded and should generally be obfuscated and managed in an external source. Storing passwords in plain text anywhere on the web site allows anyone with sufficient permissions to read and potentially misuse the password. For JavaScript calls that require passwords, it is better to prompt the user for the password at connection time.

Tips:

1. Avoid hardcoding passwords in source code and avoid using default passwords. If a hardcoded password is the default, require that it be changed and remove it from the source code.

2. To identify null, empty, or hardcoded passwords, default rules only consider fields and variables that contain the word password. However, the Fortify Custom Rules Editor provides the Password Management wizard that makes it easy to create rules for detecting password management issues on custom-named fields and variables.

userDetails.js, line 139 (Password Management: Hardcoded Password)

Fortify Priority:	High	**Folder**	High
Kingdom:	Security Features		
Abstract:	Hardcoded passwords may compromise system security in a way that cannot be easily remedied.		

Sink: userDetails.js:139 FieldAccess: forcePassword()

```
137                     var uProperties = {};
138                     if(modelUser.userProperties.forcePassword){
139                         uProperties.forcePassword =******
140                     }
141                     angular.forEach(modelUser.userProperties, function(value, key) {
```

Issue Count by Category	
Issues by Category	
Password Management: Password in Configuration File	12
Password Management: Hardcoded Password	3

Issue Breakdown by Analysis

Issues by Analysis