Fortify Security Report

Apr 11, 2013

sohr

Executive Summary

Issues Overview

On Apr 11, 2013, a source code review was performed over the src code base. 8 files, 15 LOC (Executable) were scanned and reviewed for defects that could lead to potential security vulnerabilities. A total of 3 reviewed findings were uncovered during the analysis.

Issues by Fortify Priority Order

High	2
Critical	1

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: C:/ThinkpadR60/Projekte/MobileApps/AndroidApps/src/ArrayAccess2/ArrayAccess2/src
Number of Files: 8
Lines of Code: 15
Build Label: <No Build Label>

Scan Information

Scan time: 00:33
SCA Engine version: 5.14.0.0034
Machine Name: TZI-SOHR-T510
Username running scan: sohr

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:
android.telephony.TelephonyManager.getDeviceId

Filter Set Summary

Current Enabled Filter Set:
Security Auditor 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:

Audit Guide Summary

File System Inputs

Hide issues involving file system inputs.
Depending on your system, inputs from files may or may not come from trusted users. AuditGuide can hide issues that are based on data coming from the file system if it is trusted.
Enable if you trust file system inputs.

Filters:
If taint contains file_system Then hide issue
If taint contains constantfile Then hide issue
If taint contains stream Then hide issue
If category is file access race condition Then hide issueTaint from Command-Line Arguments

Hide issues involving taint from command-line arguments.
Depending on your system, inputs from command-line arguments may or may not come from trusted users. AuditGuide can hide issues that are based on data coming from command-line arguments if they are trusted.
Enable if you trust command-line arguments.

Filters:
If taint contains args Then hide issueProperty File Inputs

Hide inputs from properties files.
Depending on your system, inputs from properties files may or may not come from trusted users. AuditGuide can hide issues that are based on data coming from properties files if they are trusted.
Enable if you trust inputs from properties files.

Filters:
If taint contains property Then hide issueEnvironment Variable Inputs

Hide issues involving environment variable inputs.
Depending on your system, inputs from environment variables may or may not come from trusted users. AuditGuide can hide issues that are based on data coming from environment variables if they are trusted.
Enable if you trust environment variable inputs.

Filters:
If taint contains environment Then hide issueJ2EE Bad Practices

Hide warnings about J2EE bad practices.
Depending on whether your application is a J2EE application, J2EE bad practice warnings may or may not apply. AuditGuide can hide J2EE bad practice warnings.
Enable if J2EE bad practice warnings do not apply to your application because it is not a J2EE application.

Filters:
If category contains j2ee Then hide issue
If category is race condition: static database connection Then hide issue

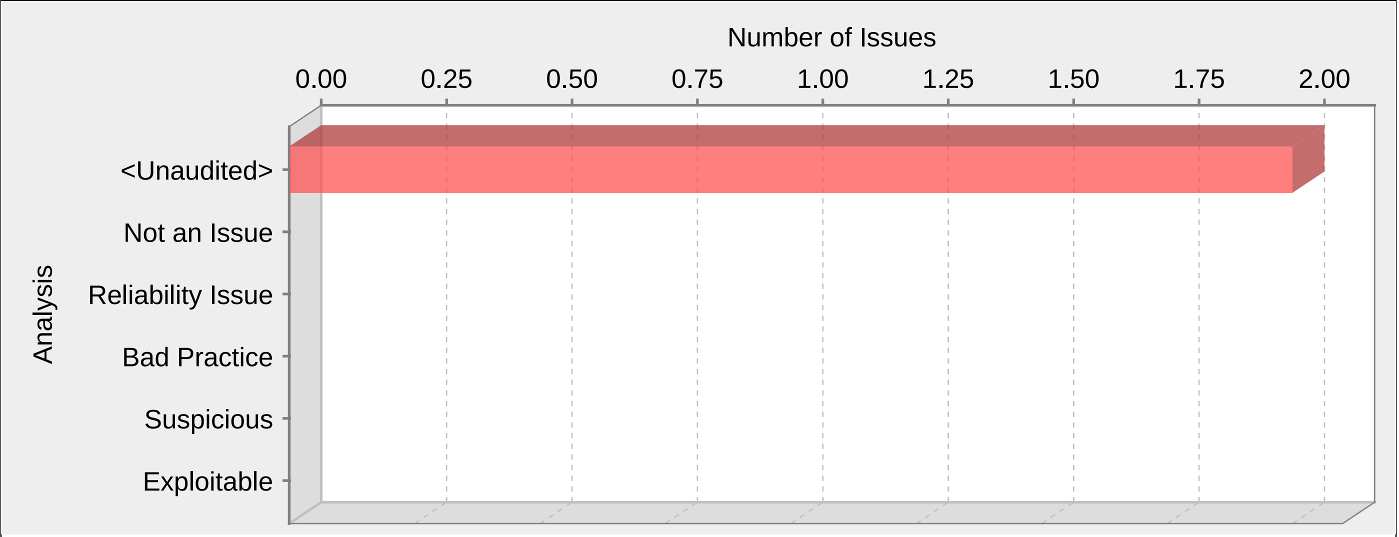
Results Outline

Overall number of results

The scan found 3 issues.

Vulnerability Examples by Category

Category: Privilege Management: Android Messaging (2 Issues)



Abstract:

The program asks to perform SMS operations.

Explanation:

Permissions to send and receive SMS must not be requested without cause, nor granted without consideration. Malicious software exploits these permissions to steal money and data from unwary users.

Example 1: In this case, the <uses-permission .../> element includes a messaging permission attribute.

<uses-permission android:name="android.permission.SEND_SMS"/>

Recommendations:

Do not request SMS permissions unless they are necessary for the core functionality of your program. A wallpaper program should not touch the messaging function of the device; neither should a media player.

AndroidManifest.xml, line 12 (Privilege Management: Android Messaging)

Fortify Priority:	High	**Folder**	High
Kingdom:	Security Features		

Abstract: The program requests permission to perform SMS operations on line 12 of AndroidManifest.xml.

Sink: AndroidManifest.xml:12 null()

```

        <uses-permission android:name="android.permission.READ_PHONE_STATE" />
        <uses-permission android:name="android.permission.SEND_SMS"/>

        <application
```

ArrayAccess2.java, line 22 (Privilege Management: Android Messaging)

Fortify Priority:	High	**Folder**	High
Kingdom:	Security Features		

Abstract: The program performs SMS operations on line 22 of ArrayAccess2.java.

Sink: ArrayAccess2.java:22 sendTextMessage()

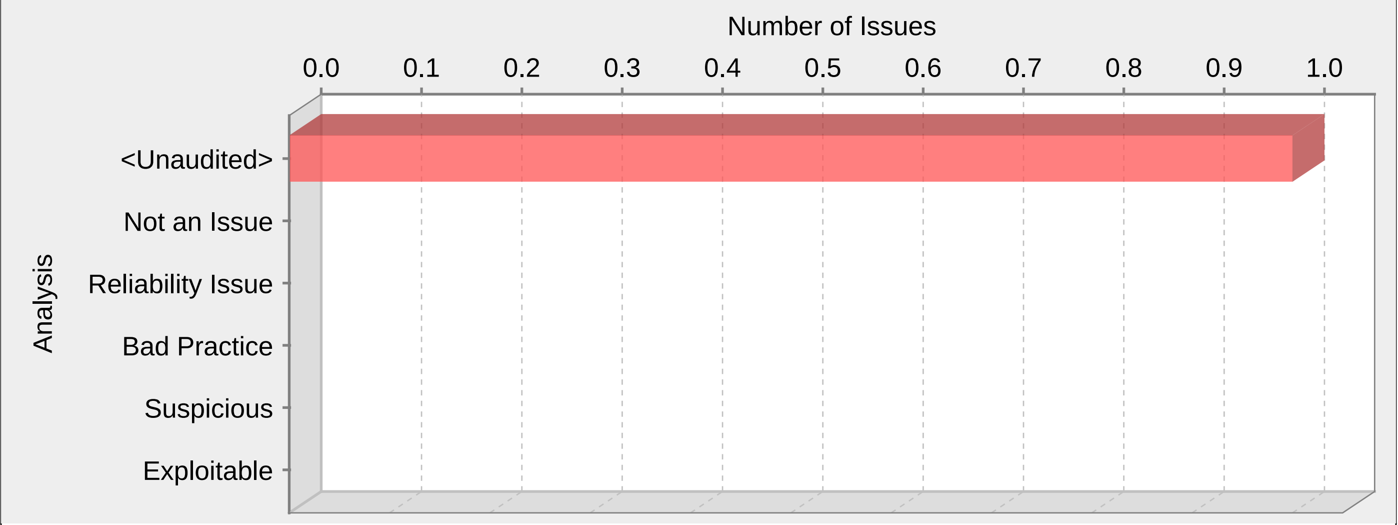
```

      SmsManager sm = SmsManager.getDefault();
      sm.sendTextMessage("+49 1234", null, array[calculateIndex()], null, null);

```

```
                        }
```

Category: Privacy Violation (1 Issues)



Abstract:

Mishandling private information, such as customer passwords or social security numbers, can compromise user privacy and is often illegal.

Explanation:

Privacy violations occur when:

1. Private user information enters the program.

2. The data is written to an external location, such as the console, file system, or network.

Example: The following code contains a logging statement that tracks the contents of records added to a database by storing them in a log file. Among other values that are stored, the getPassword() function returns the user-supplied plaintext password associated with the account.

```
pass = getPassword();
...
dbmsLog.println(id+":"+pass+":"+type+":"+tstamp);
```

The code in the example above logs a plaintext password to the filesystem. Although many developers trust the filesystem as a safe storage location for data, it should not be trusted implicitly, particularly when privacy is a concern.

Private data can enter a program in a variety of ways:

- Directly from the user in the form of a password or personal information

- Accessed from a database or other data store by the application

- Indirectly from a partner or other third party

Sometimes data that is not labeled as private can have a privacy implication in a different context. For example, student identification numbers are usually not considered private because there is no explicit and publicly-available mapping to an individual student's personal information. However, if a school generates identification numbers based on student social security numbers, then the identification numbers should be considered private.

Security and privacy concerns often seem to compete with each other. From a security perspective, you should record all important operations so that any anomalous activity can later be identified. However, when private data is involved, this practice can in fact create risk.

Although there are many ways in which private data can be handled unsafely, a common risk stems from misplaced trust. Programmers often trust the operating environment in which a program runs, and therefore believe that it is acceptable to store private information on the file system, in the registry, or in other locally-controlled resources. However, even if access to certain resources is restricted, this does not guarantee that the individuals who do have access can be trusted. For example, in 2004, an unscrupulous employee at AOL sold approximately 92 million private customer e-mail addresses to a spammer marketing an offshore gambling web site [1].

In response to such high-profile exploits, the collection and management of private data is becoming increasingly regulated. Depending on its location, the type of business it conducts, and the nature of any private data it handles, an organization may be required to comply with one or more of the following federal and state regulations:

- Safe Harbor Privacy Framework [3]

- Gramm-Leach Bliley Act (GLBA) [4]

- Health Insurance Portability and Accountability Act (HIPAA) [5]

- California SB-1386 [6]

Despite these regulations, privacy violations continue to occur with alarming frequency.

Recommendations:

When security and privacy demands clash, privacy should usually be given the higher priority. To accomplish this and still maintain required security information, cleanse any private information before it exits the program.

To enforce good privacy management, develop and strictly adhere to internal privacy guidelines. The guidelines should specifically describe how an application should handle private data. If your organization is regulated by federal or state law, ensure that your privacy guidelines are sufficiently strenuous to meet the legal requirements. Even if your organization is not regulated, you must protect private information or risk losing customer confidence.

The best policy with respect to private data is to minimize its exposure. Applications, processes, and employees should not be granted access to any private data unless the access is required for the tasks that they are to perform. Just as the principle of least privilege dictates that no operation should be performed with more than the necessary privileges, access to private data should be restricted to the smallest possible group.

Tips:

1. As part of any thorough audit for privacy violations, ensure that custom rules have been written to identify all sources of private or otherwise sensitive information entering the program. Most sources of private data cannot be identified automatically. Without custom rules, your check for privacy violations is likely to be substantially incomplete.

2. The Fortify Java Annotations FortifyPassword, FortifyNotPassword, FortifyPrivate and FortifyNotPrivate can be used to indicate which fields and variables represent passwords and private data.

3. A number of modern web frameworks provide mechanisms for performing validation of user input. Struts and Struts 2 are among them. To highlight the unvalidated sources of input, the HP Fortify Secure Coding Rulepacks dynamically re-prioritize the issues reported by HP Fortify Static Code Analyzer by lowering their probability of exploit and providing pointers to the supporting evidence whenever the framework validation mechanism is in use. We refer to this feature as Context-Sensitive Ranking. To further assist the HP Fortify user with the auditing process, the Fortify Security Research Group makes available the Data Validation project template that groups the issues into folders based on the validation mechanism applied to their source of input.

4. Fortify RTA adds protection against this category.

ArrayAccess2.java, line 22 (Privacy Violation)

Fortify Priority:	Critical	**Folder**	Critical
Kingdom:	Security Features		

Abstract: The method onCreate() in ArrayAccess2.java mishandles confidential information, which can compromise user privacy and is often illegal.

Source: ArrayAccess2.java:18 android.telephony.TelephonyManager.getDeviceId()

```
16        String[] array = new String[10];
17        TelephonyManager telephonyManager =
      (TelephonyManager)getSystemService(Context.TELEPHONY_SERVICE);
18        array[5] = telephonyManager.getDeviceId();
19        array[4] = "no taint";
20
```

Sink: ArrayAccess2.java:22 android.telephony.SmsManager.sendTextMessage()

```
20
21        SmsManager sm = SmsManager.getDefault();
22        sm.sendTextMessage("+49 1234", null, array[calculateIndex()], null, null);
23
24      }
```

Issue Count by Category

Issues by Category	
Privilege Management: Android Messaging	2
Privacy Violation	1

Issue Breakdown by Analysis

Issues by Analysis