Fall 11-28-2018

Static Analysis of Android Secure Application Development Process with FindSecurityBugs

Xianyong Meng

Static Analysis of Android Secure Application Development Process with FindSecurityBugs

A Thesis Presented to

The Faculty of the Computer Science Department

by

Xianyong Meng

In Partial Fulfillment

of Requirements for the Degree

Masters in Computer Science

Kennesaw State University

Fall 2018

Static code analysis of Android Secure Application Development Process with FindSecurityBugs

Approved by:

Professor Dr. Kai Qian, Advisor, Committee Chair
Department of Computer Science
Kennesaw State University

Professor Dr. Hossain Shahriar
Department of Computer Science
Kennesaw State University

Professor Dr. Xiaohua Xu
Department of Computer Science
Kennesaw State University

In presenting this thesis as a partial fulfillment of the requirements for an advanced degree from Kennesaw State University, I agree that the university library shall make it available for inspection and circulation in accordance with its regulations governing materials of this type. I agree that permission to copy from, or to publish, this thesis may be granted by the professor under whose direction it was written, or, in his absence, by the dean of the appropriate school when such copying or publication is solely for scholarly purposes and does not involve potential financial gain. It is understood that any copying from or publication of, this thesis which involves potential financial gain will not be allowed without written permission.

_____

Xianyong Meng

ABSTRACT

Mobile devices have been growing more and more powerful in recent decades, evolving from a simple device for SMS messages and phone calls to a smart device that can install third party apps. People are becoming more heavily reliant on their mobile devices. Due to this increase in usage, security threats to mobile applications are also growing explosively. Mobile app flaws and security defects can provide opportunities for hackers to break into them and access sensitive information. Defensive coding needs to be an integral part of coding practices to improve the security of our code.

We need to consider data protection earlier, to verify security early in the development lifecycle, rather than fixing the security holes after malicious attacks and data leaks take place. Early elimination of known security vulnerabilities will help us increase the security of our software, reduce the vulnerabilities in the programs, and mitigate the consequences and damage caused by potential malicious attacks.

However, many software developer professionals lack the necessary security knowledge and skills at the development stage, and secure mobile software development is not yet well represented in most schools' computing curriculum.

In this paper, we present a static security analysis approach with the FindSecurityBugs plugin for Android secure mobile software development based on OWASP mobile security recommendations to promote secure mobile software development education and meet the emerging industrial and educational needs.

PREFACE

The thesis is submitted in partial fulfillment of the requirements for the master's degree at the Kennesaw State University, Kennesaw.

The research project has been conducted under the supervision of Professor Dr. Kai Qian, Department of Computer Science, during the years 2016-2018. Financing for the work has been provided in the form of scholarship from the Kennesaw State University through the Graduate Research Assistantship.

ACKNOWLEDGEMENTS

I would like to thank Dr. Kai Qian of his support, encouragement, and motivation throughout this entire process.

I would also like to thank my thesis committee members, Dr. Hossain Shahriar and Dr. Xiaohua Xu for their insightful comments and valuable suggestions.

This research paper is made possible through the help and support from everyone, including my professors, parents, my wife, family and friends.

TABLE OF CONTENTS

CHAPTER I

Introduction

Nowadays, people access the Internet through portable devices, like cellular phones and tablets, as the client device rather than a PC or a laptop. Meanwhile, with new development technologies and increasing hardware performance, mobile devices can now easily implement functionalities that could only be implemented by PCs in the past. Mobile devices facilitate people to access information and resources through the Internet, wherever they are, whenever they need to, but this sort of lifestyle may cause potential security crisis. [1] Moreover, mobile application vulnerabilities usually come from unsecured code, which is one of the most significant reasons that causes security issues.

Accompany with growth of the mobile application development industry in this information age, hundreds of thousands of application developers dive into this field. But unfortunately, a lot of developers have not been adequately trained in writing secure code, which may cause flaws to exist inside applications. Also due to weakness may breach applications' confidentiality, integrity and availability from attacks. [3]

Another reason for the prevalence of unsecure programs is that sometimes coding errors are unavoidable, even for well-trained developers, especially in situations where managers or team leaders give priority to a multitude of other issues, such as deadlines, functional requirements, performance, etc. [2]

Detecting and mitigating security flaws as early as possible in the secure software development lifecycle (SDLC) is extremely important. One of the tools that can help quickly and automatically check for security flaws without executing the code is static code analysis (SCA). It helps developers to identify security vulnerabilities that can be hard to discover manually due to the lack of security threat awareness, knowledge, and skills. SCA performs code review with white-box testing and is carried out at the

implementation phase of a security development lifecycle (SDL) to provide developers with immediate feedback on possible security vulnerabilities. It can also significantly reduce the cost of 'test-patch-retest' [4, 5, 6].

SCA is expected to automatically find security flaws with a high certainty of confidence to make sure that what is found is indeed a flaw. However, it is difficult to 'prove' that an identified security flaw is an actual vulnerability. SCA can only find a small portion of security flaws because new security flaws are growing dynamically.

In addition, SCA may produce false positive results where it reports a possible security flaw that, in fact, is not a security flaw because of lacking application context in terms of input and output contexts.

SCA can also result in false negative results where a flaw exists, but the tool fails to report it. This may occur if a new flaw is not detected by the SCA tool.

Our research goal has two-folds.

- Design and develop more flaw detectors to increase the confidence of FindSecurityBugs SCA.
- Enhance the effectiveness of SCA [9] with prevention solution to the discovered flaw recommendation rather than reminding only.

Current tools that support secure programming either come from vendors or research communities. The main objective of these tools is to check for security flaws after the code is written. The general process is that developers need to install security analysis tools first, then run the tools and scan code files. This means the examination usually happens at the end of the development life circle. These tools and methods do help to improve the quality of program. Our idea about secure programming concerns are that to provide an approach which integrate unsecure code checking job during development process inside popular IDEs. In other words, it is similar to spell and grammar check, the whole scanning process could be executed under IDE environment.

In this paper, we present a static security analysis approach with FindSecurityBugs plugin for Android secure mobile software development based on OWASP mobile security recommendations to promote secure mobile software development education and meet the emerging industrial and educational needs.

CHAPTER II

Background

This chapter briefly introduces some basic theoretical conceptions, and it starts with an overview of Android and then briefly describes static analysis.

2.1 Android Overview

Android is an open-source, customizable mobile operating system created by Google for use on touchscreen devices. It is the most popular operating system in use today. While mainly installed on smartphones, it is also used on other smart devices, such as TVs and watches.

Android apps can be written using Kotlin, Java, and C++ languages and are compiled to a Dalvik bytecode using the Android Software Development Toolkit (SDK). Then the Android SDK tools build your code along with any data and resource files into an APK, an Android package, which is an archive file with an .apk suffix. One APK file contains all the contents of an Android app and is the file that Android-powered devices use to install the app. Each Android application acts in its own security sandbox, which isolates apps from each other and prevents them from accessing each other's private data.

Because each app runs in a process sandbox, apps must explicitly share resources and data by declaring the permissions they need to access shared resources and data outside the sandbox. However, Android does not completely separate apps from each other, because apps often need to share data. For example, assume a user wants to share a photo with her friends using a social networking app. This process requires data to be delivered across isolated (sandboxed) applications.

2.1.1 Intent

Android provides a sophisticated message dispatch system, in which Intents are used to contact applications. An Intent is a message that contain the information of a recipient and optionally includes other data; an Intent can be thought of as a self-contained object that specifies a remote procedure to invoke and includes the associated arguments [19].

Applications use Intents for both inter-application communication and intra-application communication. Additionally, Android OS sends Intents to applications as event notifications. Some of these event notifications are system-wide events that can only be sent by the operating system. We call these messages system broadcast Intents.

Intents can be used for explicit or implicit communication. An explicit Intent specifies that it should be delivered to a particular application specified by the Intent, whereas an implicit Intent requests delivery to any application that supports a desired operation. In other words, an explicit Intent identifies the intended recipient by name, whereas an implicit Intent leaves it up to the Android platform to determine which application(s) should receive the Intent. For example, consider an application that stores contact information. When the user clicks on a contact's street address, the contacts application needs to ask another application to display a map of that location. To achieve this, the contacts application could send an explicit Intent directly to Google Maps, or it could send an implicit Intent that would be delivered to any application that says it provides mapping functionality (e.g., Yahoo! Maps or Bing Maps). Using an explicit Intent guarantees that the Intent is delivered to the intended recipient, whereas implicit Intents allow for late runtime binding between different applications.

2.1.2 App components

Intents are delivered to application components, which are logical application building blocks. Android defines four types of components:

- Activities provide user interfaces. Activities are started with Intents, and they can return data to their invoking components upon completion. All visible portions of applications are Activities.
- Services run in the background and do not interact with the user. Downloading a file or decompressing an archive are examples of operations that may take place in a Service. Other components can bind to a Service, which lets the binder invoke methods that are declared in the target Service's interface. Intents are used to start and bind to Services.

- Broadcast Receivers receive Intents sent to multiple applications. Receivers are triggered by the receipt of an appropriate Intent and then run in the background to handle the event. Receivers are typically short-lived; they often relay messages to Activities or Services. There are three types of broadcast Intents: normal, sticky, and ordered. Normal broadcasts are sent to all registered Receivers at once, and then they disappear. Ordered broadcasts are delivered to one Receiver at a time; also, any Receiver in the delivery chain of an ordered broadcast can stop its propagation. Broadcast Receivers have the ability to set their priority level for receiving ordered broadcasts. Sticky broadcasts remain accessible after they have been delivered and are re-broadcast to future Receivers.
- Content Providers are databases addressable by their application-defined URIs. They are used for both persistent internal data storage and as a mechanism for sharing information between applications.

Intents can be passed between three of the four components: Activities, Services, and Broadcast Receivers. Intents can be used to start Activities; start, stop, and bind Services; and broadcast information to Broadcast Receivers. All of these forms of communication can be used with either explicit or implicit Intents. By default, a component receives only internal application Intents (and is therefore not externally invocable).

2.2 Related Work

Many works have been made to improve the secure software development education in recent years. UNCC has designed and developed an Application Security IDE (ASIDE) plug-in for Eclipse that notifies programmers of potential vulnerabilities in the code and assists them to handle those vulnerabilities. The program is designed to enhance student awareness and understanding of security vulnerabilities in processing of development and increasing utilization of secure programming techniques in programming. ASIDE is

used in a range of programming courses, from CS1 to advanced Web programming. ASIDE addresses input validation vulnerabilities, output encoding, authentication and authorization, and several race condition vulnerabilities [11-13]. ASIDE can only work in the Eclipse IDE for Java and cannot support the Android IDE. Many instructors and professors have integrated mobile application developments in their curriculums. Yuan and others [14] reviewed current efforts and resources in secure software engineering education, and provided related programs, courses, learning modules, hands-on lab modules. Chi [15] built learning modules for teaching secure coding practices to students. Those learning modules will provide the essential and fundamental skills to programmers and application developers in secure programming. The IAS Defensive Programming knowledge areas (KA) have been identified as topics/materials in the ACM/IEEE Computer Science Curricula 2013 that can be taught to beginning programmers in CS0/CS1 courses [16-17]. All these works mainly focus on the mobile application development. They successfully disseminated the mobile computing education but did not emphasize the importance of secure mobile software development in their teachings.

2.3 Static Analysis

Static analysis, also called static code analysis, is a method of examining the code without executing the program. It is usually performed as part of a code review (also known as white-box testing) and is carried out at the implementation phase of a Security Development Lifecycle (SDL). Static code analysis commonly refers to examining all possible execution paths in the program, and it attempts to highlight possible vulnerabilities within 'static' (non-running) source code by using techniques such as taint analysis and data flow analysis. However, static analysis can only provide useful results by approximating some facets of the actual execution of a program [10].

The process provides an understanding of the code structure and can help to ensure that the code adheres to industry standards. Automated tools can assist programmers and developers in carrying out static analysis. The process of scrutinizing code by visual

inspection alone (by looking at a printout, for example), without the assistance of automated tools, is sometimes called program understanding or program comprehension.

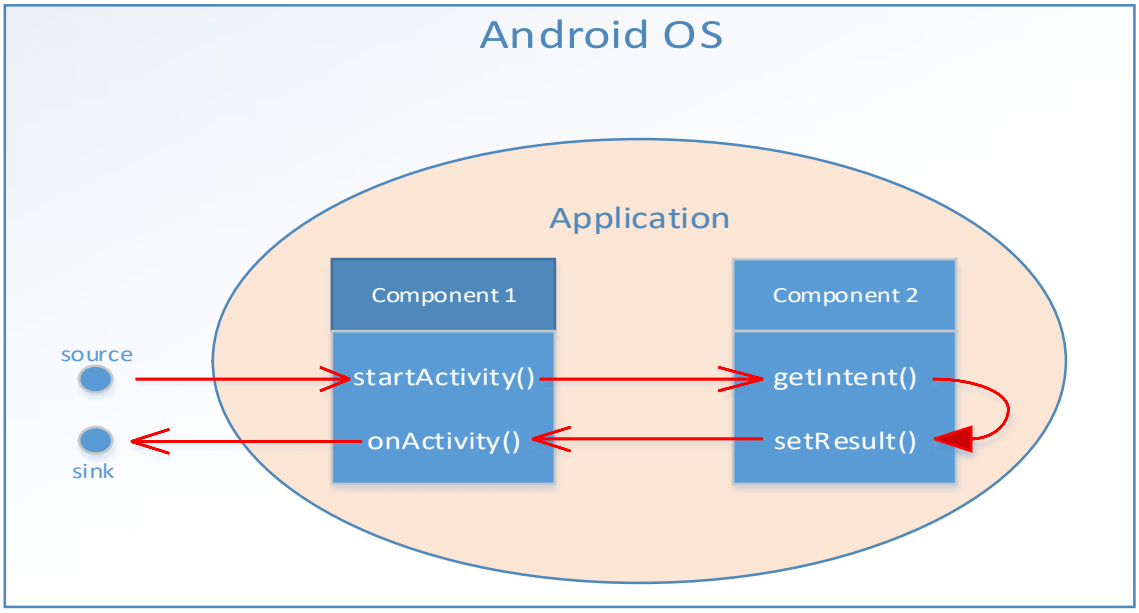

Figure 2.1: Interaction between components in the running example

One method to implement static analysis is utilizing and analyzing the data flow. Taint analysis is one sort of data-flow analysis that tracks data along the program execution process. In this method, sensitive data is marked with a taint at the source, and this taint is allowed to propagate further through all program execution processes. Receiving this taint at predefined sinks is used to create a flow which can be used to detect sensitive data leaks from source to sink.

CHAPTER III

FindSecurityBugs and Implementation

FindSecurityBugs (FSB) is a static code analysis plugin for the FindBugs (a plugin for IntelliJ API). It is also the medium of our approach to prompting and helping programmers to improve their practices. Our approaches based on some design considerations.

3.1 FindSecurityBugs in IDE

FSB specializes in finding security issues in Java code by searching for security. It can be used to scan Java applications, Android applications, and Scala applications. Since it analyzes at the bytecode level to find defects and/or suspicious code, source code is not mandatory for the analysis. It helps to prevent potential security flaws from released software. Moreover, it allows us to design our own custom flaw detectors to find and report the emerging security threats such that many flaws can be detected during the software development phase with immediate feedback to the developer, rather than finding vulnerabilities much later in the development cycle.

As security threats are constantly changing, vulnerability detections must also be continuously updated. FSB allows developers to design custom security vulnerability detectors. We have designed and developed a number of new security flaw detectors with the FindSecurityBugs plug-in for Android mobile software development based on most current OWASP 2017 top 10 mobile vulnerability to increase the security vulnerability check coverage.

As we know that the most effective way to prevent developers to write insecure code is to get start at the beginning of programming process or as early in the software development lifecycle as possible. This means that integrating secure programming process to programmers' IDE is one of best options, since programmers and developers can figure out vulnerabilities while coding.

Another consideration is the flexibility of FindSecurityBugs. Since potential vulnerabilities either come from human factors, such as less efficient knowledge or flaws hidden in APIs. Although developers spend considerable amounts of time and energy, bugs and logic mistakes cannot be avoided completely. With complex circumstances, situations get worse. In order to keeping an eye on new-found flaws or vulnerabilities, the expansibility is an important issue to consider. Equally, when a new release or new patch comes out, previous flaws of API may be fixed, so it should be handy to eliminate scan scope to save performance. Also, there is another situation: you introduced FindSecurityBugs to your team and have been running it as a part of your hourly/nightly build process. As the team becomes more familiar with the tool, you have decided that some of the bugs being detected are not important to your team, for whatever reason, or you want to pay attention to specific vulnerabilities, then the exclude and include filters would be useful.

Lastly, the reminder tool is an effective way that either helps developers to learn secure programming practices or reinforces developers' secure programming training. Meanwhile, the structure of FindSecurityBugs should keep the relationship among components low coupling since once existing information of vulnerabilities is changed or updated, the content should be easy to update and without the need to modify other files.

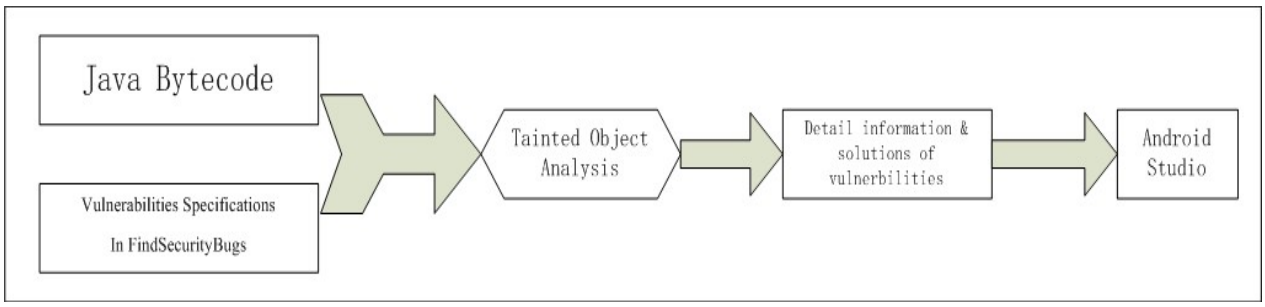

Figure 3.1 Architecture of FindSecurityBugs' framework

3.2 Flaws and Solution

FindSecurityBugs is a plugin for FindBugs that runs checks for most patterns that

relate to OWASP TOP 10 vulnerabilities, from different types of injection and XSS protection to sensitive data exposure and validation.

FindSecurityBugs first needs to detect for potential vulnerabilities in the source code, and then provide programmers with information and possible solutions for the vulnerabilities. Thus, to be effective, FindSecurityBugs needs to have a certain level of precision in identifying vulnerable source code to provide proper assistance to address it. Based on prototype implementation, a SQL-Injection vulnerability can be easily eliminated if the corresponding vulnerable code is correctly identified. Therefore, a key measure of the effectiveness of FindSecurityBugs is the measure of its ability to find vulnerable code. The effectiveness of the interactive techniques to address those vulnerabilities heavily depend on how the programmer responds to the warning and how much detailed information can be provided.

3.3 FindSecurityBugs Detectors for Secure Android Software Development

To suit the increasing demand for high quality information security professionals with expertise in development process, we built innovative Android vulnerability FindSecurityBugs detectors for the popular Android Studio IDE based on the on most current OWASP 2017 mobile top 10 mobile security risks [18] in the category of SQL injection, unintended data leakage, insecure communication, insecure data storage vulnerability detectors. For example, the built detectors can recognize a vulnerability of SQL injection and data leakage in an Android mobile application program, which may face the threat of potential malicious code injection, and then issue a warning on the code line. Following the provided options, students or developers can enforce a new secure statement to replace the unsecure statement.

The detectors can be built with MAVEN then loaded into the Android Studio IDE, parse Android java source code, identify specific API calls, warn the potential vulnerabilities, recommend security solutions, and replace code statements. Once

developers execute the analysis and click on the warning icon, secure coding prevention and protection options will be shown in Android Studio. Following the provided options, students or developers can enforce a new secure statement to replace the unsecure statement.

To add a customized detector, usually there are five steps.

1. Creating vulnerable samples

First we need to create a sample class that contains the weakness which going to be illustrated. Vulnerable sample are placed in find-sec bugs/plugin/src/test/java/testcode. The code sample doesn't need to be a working application. We can write the minimum code to trigger the rule. For example, this class is created to illustrate copy/paste buffer caching with CopyPasteLeakageDetector class.

```
1  public class CopyPasteLeakage extends AppCompatActivity {
2      @Override
3      protected void onCreate(Bundle savedInstanceState) {
4          super.onCreate(savedInstanceState);
5      }
6      public void onClickCopy(View view) {
7          ClipboardManager myClipboard = new ClipboardManager();
8          myClipboard.setPrimaryClip(null);
9          Toast.makeText(null, "Text Copied",0);
10     }
11 }
```

Figure 3.2 CopyPasteLeakageDetector class sample

2. Writing a test case

We can copy-paste the basic test structure from another test class. For example, here is a test case which validates CopyPasteLeakage source code will trigger the "ANDOID_COPYPASTE" vulnerability caution at line 25 (Actual number base on the line number showed on your IDE).

```java
public class CopyPasteLeakageDetectorTest extends BaseDetectorTest {
    @Test
    public void detectCopyPasteDecoder() throws Exception {
        //Locate test code
        String[] files = {              getClassFilePath("testcode/android/CopyPasteLeakage")
        };
        //Run the analysis
        EasyBugReporter reporter = spy(new SecurityReporter());
        analyze(files, reporter);
        verify(reporter).doReportBug(
                bugDefinition()
                        .bugType("ANDROID_COPYPASTE")
                        .inClass("CopyPasteLeakage")
                        .inMethod("onClickCopy")
                        .atLine(35)
                        .build()
        );
        //The count make sure no other bug are detect
        verify(reporter).doReportBug(bugDefinition().bugType("ANDROID_COPYPASTE").build());
    }
}
```

Figure 3.3 CopyPasteLeakageDetector test class

3. Configuration of the new detector

   For this step, we need to register the detector and bug pattern in findbugs.xml file. The class reference will soon be created.

```
<Detector class="com.h3xstream.findsecbugs.android.CopyPasteLeakageDetector" reports="ANDROID_COPYPASTE"/>
[...]
<BugPattern type="ANDROID_COPYPASTE" abbrev="CP" category="SECURITY"/>
```

Figure 3.4 Register detector and bug pattern

4. Description

We need to create the appropriate description in the messages.xml. You can leave the Details tag node empty at this point and fill it later.

5. New detector class

There are many possibilities in order to implements a detector.

- OpcodeStackDetector : Look for a specific method call
- ConfiguredBasicInjectionDetector : To implement an injection-like detector
- Detector : To analyze the complete class context

Here is a simple CopyPasteLeakageDetector example.

```java
package com.h3xstream.findsecbugs.android;

import com.sun.org.apache.bcel.internal.Constants;
import edu.umd.cs.findbugs.BugInstance;
import edu.umd.cs.findbugs.BugReporter;
import edu.umd.cs.findbugs.Priorities;
import edu.umd.cs.findbugs.bcel.OpcodeStackDetector;

public class CopyPasteLeakageDetector extends OpcodeStackDetector {

    private static final String ANDROID_COPYPASTE = "ANDROID_COPYPASTE";
    //private static InvokeMatcherBuilder XML_DECODER_CONSTRUCTOR = invokeInstruction().atClass("java/beans/XMLDecoder").atMethod("<init>");

    private BugReporter bugReporter;

    public CopyPasteLeakageDetector(BugReporter bugReporter) {
        this.bugReporter = bugReporter;
    }

    @Override
    public void sawOpcode(int seen) {
        if (seen == Constants.INVOKEVIRTUAL && getClassConstantOperand().equals("android/content/ClipboardManager")&&
                getNameConstantOperand().equals("setPrimaryClip")) {
            bugReporter.reportBug(new BugInstance(this, ANDROID_COPYPASTE, Priorities.HIGH_PRIORITY) //
                    .addClass(this).addMethod(this).addSourceLine(this));
        }
    }
}

```

Figure 3.5 CopyPasteLeakageDetector

Taint checks usually highlight specific security risks primarily associated with web sites which are attacked using techniques, such as SQL injection or buffer overflow attack approaches. For "injection-based" detectors, the system has a configurable list of inputs which it considers producing "tainted" data. The system also has a configurable list of "sinks" which should not be allowed to receive any tainted data. Sink file format is composed of those info separated by the characters:

•Method signature

•Injectable parameter indexes in the reverse order (Order on the stack before the method call)

3.4 SQL Injection Detectors

There are several types of SQL injection, consisting of direct insertion code to user input variables and then concatenated with SQL statements to be executed or other less direct code insertion technique. Some of them are listed as below:

1. Incorrectly filtered escape characters

This form occurs if user input is passed to SQL statement without filtering escape characters. Implementation is illustrated by following example.

“SELECT * FROM users WHERE username= ‘”+username+”’” in which the variable username can be crafted by attackers, either by inputting an always true clause or by commenting out the rest of query statement. What’s more, by inserting semicolon, attackers are able to execute separate SQL statement in this case.

2. Incorrect type handling

This form of injection takes place when no appropriate type checking is performed. The implementation of this form can be same as previous one. There are still many forms to perform injection, the point that an injection works is by prematurely terminating a text string and appending a new command.

Defense strategy for SQL injection

In addition to input validation to eliminate the malicious injection we also should use secure parameterized query statements with placeholders to receive parameters instead of embedding user input to query statement. Parameterized queries force the developer to first define all the SQL code, and then pass in each parameter to the query later. This coding style allows the database to distinguish between code and data, regardless of what user input is supplied. This strategy will also solve the problem if there is not a type handling mechanism, because a placeholder can only receive value of the given type.

Vulnerable code fragment:

```
1 cursor = db.rawQuery("SELECT * FROM usertable WHERE _id='" + info + "'", null);
```

Figure 3.6 Vulnerable code fragment

Secure parameterized query:

```
1 String s1 = input.getText().toString();
2 cursor = db.rawQuery("SELECT * FROM usertable WHERE _id= ? ", s1);
```

Figure 3.7 Secure parameterized query pattern

Here is a sample vulnerable SQL injection code detected by the SQL injection detector after scanning an Android app with the following snippet

```
1 EditText input;
2 String  info = input.getText().toString()
3 cursor = db.rawQuery("SELECT * FROM usertable WHERE _id='" + info + "'", null);
```

Figure 3.8 Vulnerable SQL injection code fragment

A warning alert and solution hint is shown as follows:

Vulnerability alert! Use parametrized query with placeholders (“?”) to receive input parameters instead of embedding user input to query statement with string concatenation (“+”) to avoid malicious SQL injection.

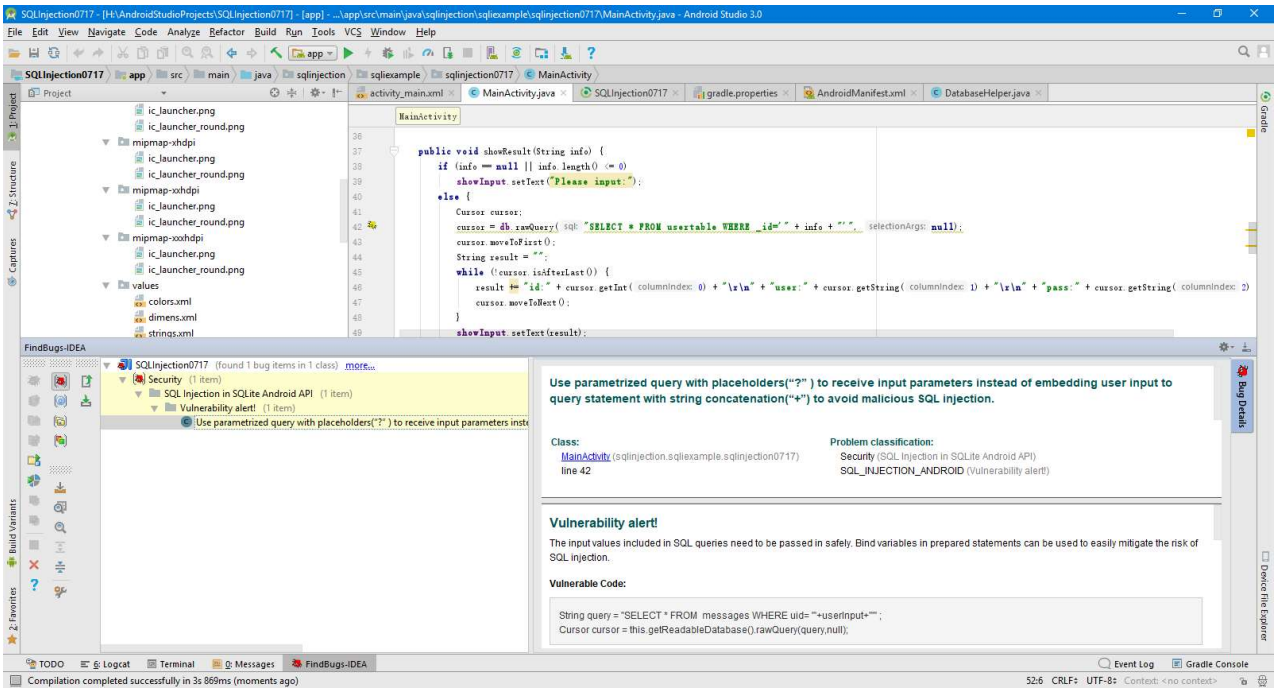

Figure 3.9 Detector for Android SQL injection

3.5 Other Android Vulnerability Detectors

Based on OWASP top 10 mobile risk report, we have developed more vulnerability detectors with FindSecurityBugs for secure Android software development in the categories of unintended data leakage, intent interception and spoofing, content provider data sharing, input validation and output encoding. Here is a screen shot of a diagnosis result by unintended data leakage detector for clipboard cache memory data leakage.

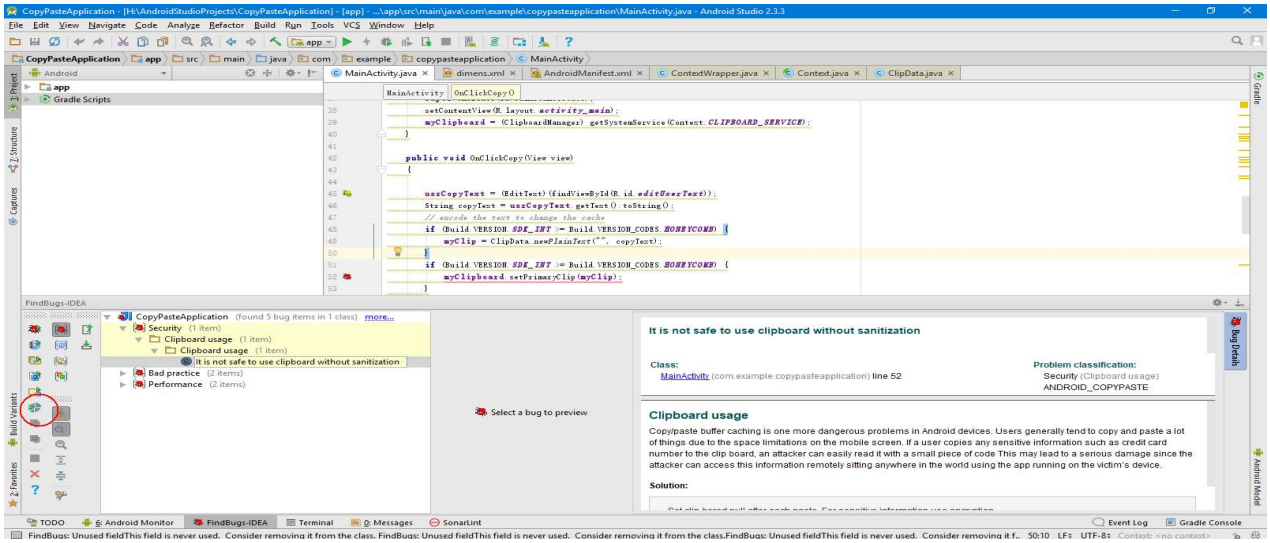

Figure 3.10 Unintended data leakage detector

CHAPTER IV

CONCLUSION AND FUTURE WORK

The accuracy is crucial in the process of assessment because the primary requirement for a SCA is accurate vulnerability Assessment testing.

A False Negative (FN) assessment overlooks vulnerability and fails to identify or detect a serious security threat. SCA should be able to cover as much as security risks.

The False Positive (FP) (false alarm) assessment falsely identifies acceptable code as a security flaw and notifies developer with an incorrect message of a security vulnerability, which either misleads and confuses developers or results in an unnecessary development delay. A good SCA tool should avoid both of them, but it is a challenge. Firstly, the forms of vulnerabilities are changing and evolving all the time. Secondly, for a specific flaw such as SQL injection, there are many different attack vectors. Thirdly, for a specific API method invocation, some overloading calls are safe, but some others may make flaws.

Obviously you do not want any FN or FP. Clearly it is important for a solution to find all real vulnerabilities. An inaccurate diagnosis can be more trouble than its worth.

Our project focuses on developing teaching and learning materials on secure mobile software development with Android based hands-on labs. This effort will overcome the shortage of hands-on learning materials of secure mobile software development, which are essential in building capacity to secure mobile application development. Our project with secure mobile software development Android Studio FindSecurityBugs plugin will help students, faculty, and professionals to use and develop custom Android app vulnerability detectors to increase and enhance their knowledge and skills in SMSD in a real working environment.

REFERENCES

[1] https://www.wired.com/2015/02/smartphone-only-computer/

[2] D. E. Knuth. The errors of tex. Softw. Pract. Exper.,19:607–685, July 1989.

[3] VERACODE. State of Software Security Report Volume 1, 2, and 3, 2011. http://www.veracode.com/reports/index.html.

[4] A. J. Ko and B. A. Myers. A framework and methodology for studying the causes of software errors in programming systems. J. Vis. Lang. Comput.,16:41–84, February 2005.

[5] V. B. Livshits and M. S. Lam. Finding security errors in Java programs with static analysis. In Proceedings of the 14th Usenix Security Symposium, pages 271–286, August 2005.

[6] J. Xie, H. R. Lipford, and B. Chu. Why do programmers make security errors? In Proceedings of 2011 IEEE Symposium on Visual Languages and Human Centric Computing, pages 161–164, 2011.

[7] V. Benjamin Livshits and Monica S. Lam Finding Security Vulnerabilities in Java Applications with Static Analysis October 2005

[8] Static Analysis of Android Apps: A Systematic Literature Review Li Li, Tegawende F. Bissyande, Mike Papadakisa, Siegfried Rasthoferb, Alexandre Bartela, Damien Octeauc, Jacques Kleina, Yves Le Traon

[9] F. Software. Fortify SCA, 2011. https://www.fortify.com/products/fortify360/source-code-analyzer.html.

[10] Patrick Cousot and Radhia Cousot. Abstract Interpretation: A Uni_ed Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, pages 238{252. ACM, 1977

[11]Michael Whitney, Heather Richter Lipford, Bill Chu, and Jun Zhu. Embedding Secure Coding Instruction into the IDE: A Field Study in an Advanced CS Course. In Proceedings of the 46th ACM Technical Symposium on Computer Science Education (SIGCSE '15)

[12]Michael Whitney, Heather Richter Lipford, Bill Chu, and Tyler Thomas. "Embedding Secure Coding Instruction into the IDE: Complementing Early and Intermediate CS Courses with ESIDE" In press, Journal of Educational Computing Research, 2017

[13]Jun Zhu, Heather Richter Lipford, Bill Chu. Interactive Support for Secure Programming Education. In the Proceedings of SIGCSE 2013, the 44th Technical

Symposium on Computer Science Education, March 2013

[14] Xiaohong Yuan, Kenneth Williams, D. Scott McCrickard, Charles Hardnett, Litany H. Lineberry, Kelvin S. Bryant, Jinsheng Xu, Albert C. Esterline, Anyi Liu, Selvarajah Mohanarajah, Rachel Rutledge: Teaching mobile computing and mobile security. FIE 2016: 1-6

[15] Hongmei Chi, InfoSecCD '13 Proceedings of the 2013 on InfoSecCD '13: Information Security Curriculum Development Conference

[16]Computer Science Curricula 2013 - Association for Computing, https://www.acm.org/education/CS2013-final-report.pdf

[17] Katerina Goseva-Popstojanovaa, Andrei Perhinschib, On the capability of static code analysis to detect security vulnerabilities, community.wvu.edu/~kagoseva/Papers/IST-2015.pdf

[18]Projects/OWASP Mobile Security Project - Top Ten Mobile Risks, https://www.owasp.org/index.php/Projects/OWASP_Mobile_Security_Project_-_Top_Ten_Mobile_Risks

[19] Erika Chin, Adrienne Porter Felt, Kate Greenwood, David Wagner Analyzing Inter-Application Communication in Android https://people.eecs.berkeley.edu/~daw/papers/intents-mobisys11.pdf