Secure Mobile Software Development with Vulnerability Detectors in Static Code Analysis

Xianyong Meng, Kai Qian, Dan Lo
Computer Science Department
Kennesaw State University
Kennesaw, GA, USA
{xmeng5, kqian, dlo2}@kennesaw.edu

Prabir Bhattacharya
Computer Science Department
Morgan State University
Baltimore, MD, USA
prabir.bhattacharya@morgan.edu

Fan Wu
Computer Science Department
Tuskegee University
Tuskegee, AL, USA
fwu@tuskegee.edu

***Abstract*—The security threats to mobile application are growing explosively. Mobile app flaws and security defects could open doors for hackers to easily attack mobile apps. Secure software development must be addressed earlier in the development lifecycle rather than fixing the security holes after attacking. Early eliminating against possible security vulnerability will help us increase the security of our software, and militate the consequence of damages of data loss caused by potential malicious attacking. However, many software developer professionals lack the necessary security knowledge and skills at the development stage and Secure Mobile Software Development (SMSD) is not yet well represented in current computing curriculum. In this paper we present a static security analysis approach with open source FindSecurityBugs plugin for Android Studio IDE. We categorized the common mobile vulnerability for developers based on OWASP mobile security recommendations and developed detectors to meet the SMSD needs in industry and education.**

Keywords—Android vulnerability, secure software development, static analysis, FindSecurityBugs

I. INTRODUCTION

While the computing landscape is currently moving towards mobile computing, the security threats to mobile devices are also growing explosively. The mobile applications are becoming a major security target nowadays. Most of malicious mobile attacks take advantage of vulnerabilities in mobile applications, such as sensitive data leakage via inadvertent or side channel, unsecured sensitive data storage, data transmission, and many others. Most vulnerability should be addressed in the mobile software development phase; however, most development teams often have little to no time for security remediation, as they are usually tasked for the project deadlines. Even worse, many development professionals lack awareness of the importance of security vulnerability and the necessary secure knowledge and skills at the development stage. Security vulnerabilities open the doors to security threats and attacks which may be prevented at early stage. The combination of the mobile devices' prevalence and mobile threats' rapid growth has resulted in a shortage of mobile-security personnel. Education for secure mobile application development is in big demand in IT fields. With more schools developing teaching materials on mobile application development, more educational activities are needed to promote mobile security education and to meet the emerging industry and education needs. However, mobile security is a relatively weak area and is not well represented in most schools' computing curriculum. The secure mobile software development is an important and integral part of mobile application development instead of an add-on component. Moreover, Securing Mobile application has many special issues in addition to securing traditional software development such as security protection of SMS, GPS, sensors, cameras.

With increasing demands of mobile applications in recent years, the world has also witnessed numerous major cyber-attacks, resulting in stolen personal credit card numbers, leakage of classified information vital for national defense, industrial espionage resulting in major financial losses, and many more malicious consequences. Hackers have managed to make secure computing a more difficult task. This has resulted in the need for not only the concepts of cybersecurity, but also the secure software development in the education of computer science, information technology, and related fields. The rapid growth of mobile computing also results in a shortage of professionals for mobile software development, especially for Secure Mobile Software Development (SMSD) professionals. More and more higher education institutions have realized the importance for computer science and software engineering students to be exposed to the mobile software design and development. Unfortunately, despite the great need for mobile professionals and existing efforts in mobile software development education, mobile security is a relatively weak area and is not well represented in most schools' computing curriculum. The challenges in offering embedded system courses at these institutions include scarce dedicated staff and faculty in this field and the excessive time needed for developing course materials and projects.

The mobile application flaws and security defects could open doors for hackers to break into the app, access sensitive information, and conduct all kinds of malicious attacks. Most vulnerability should be addressed and fixed in the mobile software development phase. If all the mobile apps are secure or have less security flaws and vulnerabilities, the security threat risks will be greatly reduced. Computer users, managers, and developers agree that we need software and systems that are "more secure". Such efforts require support from both the education and training communities to improve software assurance, particularly in writing secure code. There are many open source static Java code analysis tool that helps developers to maintain and clean up the code through the analysis performed without actually executing the code such as Eclipse IDE, IntelliJ IDE, FindBugs Plugin. These tools

The work is partially supported by the U.S. National Science Foundation under awards: 1723586, 1723578, 1636995, 1623724, and U.S. Department of Homeland Security Scientific Leadership Award grant number: 2012-ST-062-000055.

978-1-5386-3779-1/18/$31.00 ©2018 IEEE

focus on finding probable bugs such as inconsistencies, helping improve the code structure, conform your code to guidelines, and provide quick-fix. In general, SCA tools are used to ensure code quality from the very beginning and to make software development more productive. The security vulnerability checking are not their major task. Source code analysis tools, also referred to as Static Application Security Testing (SAST) Tools, are designed to analyze source code and to help to find security flaws with a high confidence that what's found is indeed a flaw. However, there is no tool that can just automatically finds all flaws and can guarantee all detecting are positive or never miss any potential flaws.

FindSecurityBugs (FSB) is a static code analysis plugin for the FindBugs(a plugin for IntelliJ API). It 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 on rather than finding vulnerabilities much later in the development cycle.

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

II. Related Works

Many efforts have been made to enhance the secure software development education in recent years. UNCC has designed and developed an Application Security IDE (ASIDE) plug-in for Eclipse that warns programmers of potential vulnerabilities in their code and assists them in addressing these vulnerabilities. The tool is designed to improve student awareness and understanding of security vulnerabilities in software and to increase utilization of secure programming techniques in assignments. 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 [1-3]. ASIDE only works in the Java Eclipse IDE and cannot support the Android IDE. Many computing instructors and professors have integrated mobile application developments in their curriculums. Yuan and others [4] reviewed current efforts and resources in secure software engineering education, and provided related programs, courses, learning modules, hands-on lab modules. Chi [8] 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 [5-6]. All these works mainly focus on the mobile application development. They successfully disseminated the mobile computing education but did not emphasize the importance of SMSD and in their teachings.

III. Secure Mobile Software Development with Vulnerability Detectors

A. *FindSecurityBugs Detectors for Secure Android Software Development*

To meet the ever-increasing demand for high quality information security professionals with expertise in SMSD, we built innovative Android vulnerability detectors with an open source FindSecurityBugs API plugin for the popular Android Studio IDE based on the on most current OWASP 2017 mobile top 10 mobile security risks [7] 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 built package can be 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. For example, it 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. Once the developer clicks 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.

For many Android security vulnerabilities and flaws on the top 10 mobile risks by OWASP and other new identified unlisted flaws we need to develop our own customized detectors. Fig. 1 shows the architecture of FindSecurityBugs framework. [9]

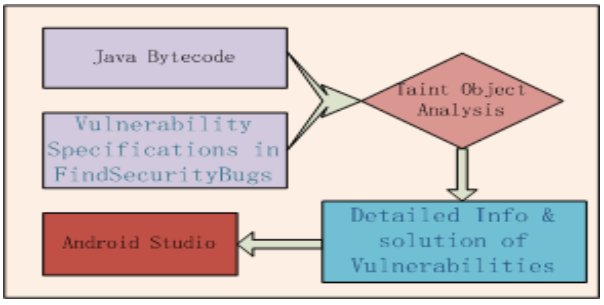

Fig. 1. Architecture of FindSecurityBugs' framework

Based on OWASP top 10 mobile risk reports, 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. The section A shows a SQL injection detector and section B shows unintended data leakage detector for clipboard cache memory data leakage.

A. SQL Injection Detectors

There are several forms 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 statement.

```
"SELECT * FROM users WHERE username= "+username+"'"
```

where 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:

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

Secure parameterized query:

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

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

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

Fig. 2 shows a waning alert and a solution hint in the Android Studio IDE after detector finds the SQL injection vulnerability:

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.

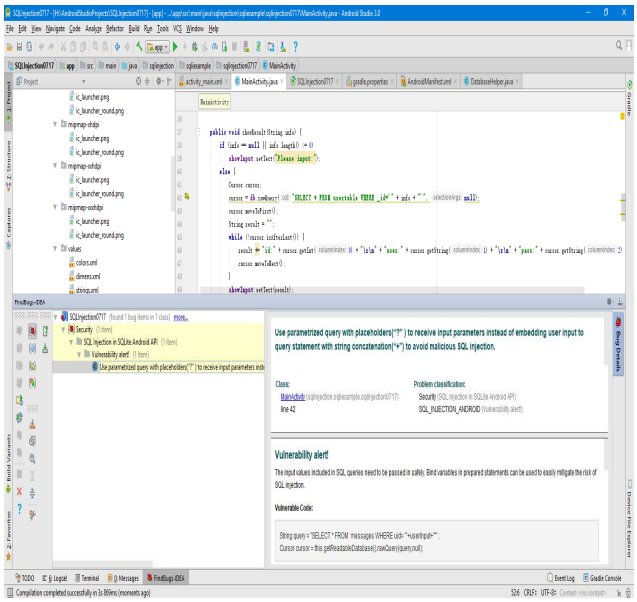

Fig. 2. Detector for Android SQL injection

B. Unintended Data Leakage Vulnerability Detectors

Unintended data leakage occurs when a mobile developer inadvertently leaves sensitive data in insecure place, such as cache memory and unprotected files on mobile devices that may be easily accessed by malware. Typically, these side-effects originate from the underlying mobile device's operating system. This is a common and prevalent mobile vulnerability in any mobile program developed by developers who lack the knowledge of the side-channel data leakage for data leakage.

Actually, it is easy to detect data leakage by inspecting all mobile device locations that are accessible to all applications and looking for the application's sensitive data.

The most common mobile unintended data leakage vulnerabilities are seen in the clipboard buffer, logging files, browser cookie, and any caching. On Android, the clipboard can be accessed by any application. It is best to avoid handling sensitive data with Copy/Paste if possible, however if it cannot be avoided, the developer should clear the buffer once the data is used and leave data there with limited time. Another option is for the developer to use the secured areas with cryptographic encryption for Copy and decryption for Paste.

Fig. 3 shows a screen shot of diagnosis result by unintended data leakage detector for clipboard cache memory data leakage. The alert message and suggested solutions are also displayed when such flaw is detected.

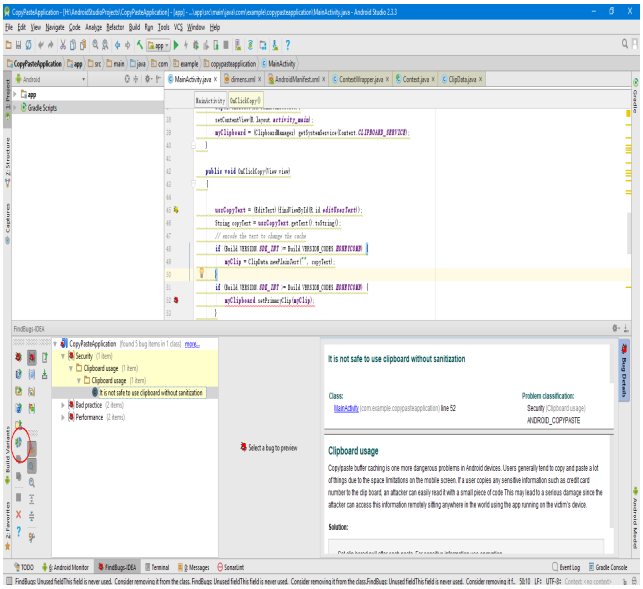

Fig. 3. Detecting Unintended Data Leakage for clipboard cache memory data

IV. DIAGNOSTIC ACCURACY

The flaw detecting and diagnosis accuracy is vital in the vulnerability SCA assessment because the primary requirement for a SCA is report all vulnerability accurately but it is not easy job. In some case SCA may miss to report a true flaw and in some other cases it may misleads developers to a fake flaw.

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 wrong message about security vulnerability which either misleads and confuses developers or result in a unnecessary development delay. A good SCA tool should void both of them but it is a challenge. First, the form of vulnerability are changing and evolving all the time. Second, for a specific flaw such as SQL injection, there are many different attack vectors. Third, for a specific API method invocation, some overloading calls are safe but some others may make flaws. Obviously you don't want either of FN and FP. Clearly it's important for a solution to find all real vulnerabilities. An inaccurate diagnosis can be more trouble than it's worth.

V. CONCLUSION AND FUTURE WORK

Our project focuses on developing teaching and learning materials on SMSD with Android based hands-on labs. This effort will overcome the shortage of hands-on learning materials of SMSD, which are essential in building capacity to secure mobile application development. Our project with SMSD 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 SMS in a real working environment.

ACKNOWLEDGEMENTS

The work is partially supported by the U.S. National Science Foundation under awards: NSF proposal 1723586, 1723578, 1636995, 1623724, and U.S. Department of Homeland Security Scientific Leadership Award grant number: 2012-ST-062-000055. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation and Department of Homeland Security.

REFERENCES

[1] 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)

[2] 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

[3] 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.

[4] 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

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

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

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

[8] Hongmei Chi, Teaching Secure Coding Practices to STEM Students, InfoSecCD '13 Proceedings of the 2013 on InfoSecCD '13: Information Security Curriculum Development Conference

[9] The FindBugs plugin for security audits of Java web applications, http://find-sec-bugs.github.io, 2017 accesed