Plugin-based Tool for Teaching Secure Mobile Application Development

A B M Kamrul Riad
aislamri@students.kennesaw.edu

Md Saiful Islam
Mislam16@students.kennesaw.edu

Hossain Shahriar
hshahria@kennesaw.edu

Chi Zhang
czhang4@kennesaw.edu

Maria Valero
mvalero2@kennesaw.edu

Sweta Sneha
ssneha@kennesaw.edu

Kennesaw State University
Kennesaw, GA

Sheikh Ahamed
Sheikh.ahamed@marquette.edu
Milwaukee, WI

Abstract

Mobile device security has become increasingly important in mobile computing. Since the mobile devices and applications are growing rapidly, the security threats are intensified due to mobile app flaws and lack of security consideration in early stages of software development. The unsecure software development process creates a serious weak path that causes potential malicious attacks in mobile devices. To mitigate the mobile threats, it is essential for application developers to follow secure code development processes to alleviate data leakage or access control vulnerabilities. Secure Mobile Software Development needs to be emphasized and adopted for reducing security vulnerabilities. In this paper we present a development tool of secure code analysis for mobile application development. The tool is designed to find the security leakage of static code and implementation of plugins such as Droid Patrol. The proposed code analysis and design procedure in the early stage of application development can eliminate the weak security path in coding. Our experience of running the plugin in classrooms are discussed and student feedback are provided.

Keywords: Android, Secure software Development, SQL injection, IoT, Static analysis, data flow, secure coding.

1. INTRODUCTION

As mobile devices become ubiquitous, numerous major cyber-attacks, stolen sensitive information, unauthorized credit card transactions and security concerns have been reported (Meng et al., 2018). Android application is in the most vulnerable position in malware collection, where two or more malicious apps associate together for target attacking. With the conventional attack detection, each individual app may use the flexible inter-app communication infrastructural support, so called Inter-Component Communication (ICC). However, potential leak may not be able to be tracked by ICC detection. (Elish et al, 2018; Tian et al., 2018). Android devices has a big share of the global smart devices market. There are about 2.2 million apps in Google Play Store and around 1.5 million apps are free. These free applications may have a dark side because the application codes may not be built with consideration of security that may lead to the potential malicious data flows (Tian et al., 2018).

Therefore severe data breach are found in mobile devices including health monitors and trackers when these health devices communicate with the databases. The data security and privacy are serious concerns. The vulnerabilities are due to poor security code, firmware system in the software, and malicious code injected while devices are connected to the apps (Zhang et al., 2020). In 2017, a popular virtual keyboard app leaks 31 million user's personal data because its database was not protected with a password, and Android users around the world were affected (Whitaker, 2020). Also, the analysis of the recent cyber-attacks in financial and healthcare organizations indicates that secure software development is important to protect the widespread cyber-attacks.

There are not many security measurements and tools that application developers use to ensure the essential data protection. Various apps for keeping Coronavirus test and diagnoses have been available to be downloaded since COVID-19 pandemic started. EFF (Electronic Frontier Foundation) warns COVID-19 tracing apps pose security and privacy risks. Despite that Google and Apple have transparent security and privacy policies, industries stakeholders along with security scientists warn the potential security threats that developer must take higher technical measures and tools while developing software applications. Currently smart devices are unable to verify any Proximity Tracking System (PTS) that checks a public database of keys against Rolling Proximity Identifiers (RPIDs) on a user device (Davis, 2020).

Most mobile security vulnerability should be addressed and fixed in the software development phase. In general, the security threat and vulnerability can be reduced during the application development phase. But such an effort to develop secure code requires ground support and tools from both educational institutions and training communities (Shahriar et al., 2019). Four most prominent Integrated Development Environments (IDE): Eclipse, IntelliJ IDEA, Visual Studio and Netbeans, help developers check for security flaws and determine input-validation-related vulnerabilities in code. Android Studio provides FindSecurityBugs plugin which analyzes the static byte code to look for bugs in java code from within IntelliJ IDEA and Findbugs, a security detect detection tool for java code, is used for static analysis to look for more than 200 bugs patterns such as recursive loops, null pointer differences, bad uses of java libraries and deadlocks. Android Studio plugin specializes in finding the static code bugs and inconsistency of code structure to ensure the code quality from the application development stage (Baset & Denning, 2017; Pfeiler, 2020).

However, there is not a code analysis tool that can automatically identify all the security flaws in the source code for developers to analyze vulnerabilities and security bugs in the initial phase of the mobile software development. In this paper, we design and implement the DroidPatrol which is an integrated plugin with the Android Studio to perform tainted data flow-based static analysis. DroidPatrol is the build in plugin in Android Studio for Intellij IDEA that allows code developers to identify a list of source code and sinks so developers can see the possible leak path within the source code and manipulate the related bugs to fix (Talukder et al., 2019).

We organize our paper as follows. In Section 2, we provide background and relevant work, in Section 3, we analyze the mobile application architecture and threat, in Section 4 we provide DroidPatrol tools model overview that including DroidPatrol architecture, features, Data leak

detection test and analysis result, Section 5, we provide conclusion and future work.

2. BACKGROUND AND RELATED WORK

In recent years, several research for Android app analysis technologies have been proposed. In this section we consider background code analysis into two parts: i) static code analysis and ii) dynamic code analysis. Static code analysis generally conforms to coding standards without executing the program and dynamic code analysis provides a real or simulated environment where apps can be installed virtually (Talukder et al., 2019; Ashfaq et al., 2019).

Static code analysis generally conforms to coding standards without executing the program. The main advantage of the static analysis is the control-flow and data-flow analysis. Control flow helps identify the possible execution path of the target app and data flow analysis can specify the possible predicted values of variables at the location of execution of the target app (Fan et al., 2020). For example, StubDroid (Arzt & Bodden, 2016), is a method for automatically generating correct and precise models for android applications using precise and extendable inheritance capabilities. StubDroid approaches the inferring library specification from binary distribution that can handle callbacks, a library method invokes client code. FlowDroid is an open source Java based static analysis tool that can detect the potential data leakage in source code of an Android application. While FlowDroid tool can detect and analyze data flow in the full lifecycle of the application development phase, it is not a highly potential data security tool that can detect the common security bugs in Android applications such as intent leakage, SQL injection, output encoding for secure communication (Shahriar et al., 2019; Talukder et al., 2019).

DroidSafe (Mumtaz, & El-Alfy, 2017) detects Android capability leaks to uncover the malicious code using Control Flow Graph (CFG) and static taint analysis. CFG can track data flows from source to sink and helps security analysts to assess the effectiveness of information leakage. Compared to other tools such as FlowDroid and IccTA, DroidSafe can detect the significant number of malicious information flaws approximately 69 malicious whereas FlowDroid and IccTA can detect only six malicious flows. DroidSafe still suffers from imprecision due to unacceptable numbers, false positive alarm and silent mode that may leave errors uncovered.

TrustDroid (Zhao, & Osono, 2012) is a taint tracking static code analyzer that statically performs semantic analysis of a compiled Android application (APK file). It can determine the leakage of sensitive information in two modes: i) off-line mode while analysis of the static resources and the performance indicates no such problem ii) real-time mode, it is reliable in considering the performance of the algorithm in terms of speed and battery/resource consumption. TrustDroid analyzes the byte code by searching the entries that manipulate sensitive data information source code marked as tainted with taint tag so the data is manipulated by bytecode when this tag propagates. If tainted data flows out through a predefined taint sinks such as network interface, the flag is created and a function is called for the process of copying one variable to another variable or to another memory location.

TaintDroid (Enck et al., 2010) is an implementation of dynamic taint analysis for Android applications, an extension of Dalvik virtual machine (DVM) to optimize efficient storage and memory-mappable execution memory, battery life and performance. It also protects sensitive user information from untrusted code that shares the limitation of dynamic taint analysis. TaintDroid uses the concepts of taint sources from which sensitive information e.g, text message, IMEI, GPS data or picture and contact information from mobile devices are obtained. TaintDroid issues a potential warning to the users when tainted data reaches a taint sink. On the other hand, TaintDorid's performance overhead occurs due to application wait state and heavyweight operations (Beal 2020; Babil et al., 2013). To minimize the overhead performance, TaintDroid only tracks explicit data flow but does not control flaws (e.g., implicit flaws). Full traffic control flaw requires static analysis, a challenge for third-party applications. Only direct control flaws can be tracked dynamically if taint scope is determined. In addition, TaintDroid creates significant false positives if the tracked information contains configure identifiers.

Although static analysis is faster than dynamic analysis for comprehensive code coverage in analyzing the apps for exploring different execution paths, it is not effective on dynamic loading where dynamic analysis is useful for runtime behavior of java code. As TaintDroid cannot handle dynamic payloads to run the native code level, DroidTrace (Zheng et al., 2014) can monitor and detect the behaviors of dynamic payloads. In addition, DroidTrace can use Process

ISSN: 2473-4901
v6 n5352

Trace (Ptrace) to monitor the system calls of the target process while running the dynamic payloads. DroidTace is also compatible with different hardware platforms without restoring emulation.

Cuckoo Sandbox (Jamalpur et al., 2018), a widely used malware analysis tool based on dynamic analysis, runs applications under test in a control emulator, such as virtual platforms Virtual box, VM ware and KVM on Windows, Linux, and Mac. Cuckoo sandbox provides a flexible solution for malware detection while writing code in notepad and executing files in a virtual platform where the cuckoo agent acts as a communication medium between the cuckoo host (actual network) and cuckoo guest (operating system). It chooses the guest and uploads code samples when the host launches a new analysis and generates a complete report based on a series of tests made during execution of the malicious code sample.

In Secure Mobile Software development, many Android plugin tools emerge in recent years. For the Application Security IDE (ASIDE), Eclipse IDE extension and plugin help warn developers of potential vulnerabilities and helps detect potential bugs and fix the code quality issue during development. SonarLint (Vermeer, 2019) is an Eclipse IDE plugin that provides instant feedback for the most commonly used languages including Python, JavaScript and Java. The Snyk (Vermeer, 2019) plugin for Eclipse can scan the code dependencies with dependency trees and can check vulnerabilities with suggesting possible fixes. The most significant feature is an integrated view that provides the origin of vulnerabilities and how many layers deep. The plugin also provides the link to Snyk website when vulnerability is found and its severity that helps developers to make secure code for apps developing. However, Eclipse plugin tools do not support Android Development Studio.

3. ANALYSIS OF MOBILE APPLICATION THREATS

The main concern of mobile applications is vulnerability. Most of these applications have a client server architecture. The server side component is a web application that interacts with mobile clients through Application Programming Interface (API). Although the mobile OS has various security mechanisms, errors made by developers in designing and writing code for the mobile application caused loopholes in user data protection which may be exploited by attackers. The common attack scenario is malware infection that escalates the administrator privilege (root or jailbreak) when malware requests permission to access user data and sends data to the attackers if granted. Figure 1 shows how the client server interacts with app distribution platforms through mobile devices (Positive Technology, 2019).

Figure 1. Client–server interaction in a mobile application

The maximum risk level of vulnerabilities occurs in both client and server. 60% of vulnerabilities occur from client server; 89% of vulnerabilities are the exploited without physical access, and 56% of vulnerabilities are exploited without administrative privileges such jailbreak or root access (Positive Technology, 2019). In general, android applications contain more vulnerabilities than those applications are written for iOS (43% vs 38%) but the difference is not significant and the overall apps security level for both are roughly the same (Figure 2).

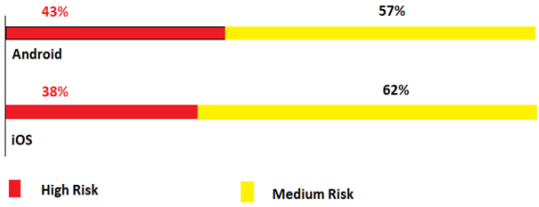


Figure 2: Maximum risk level of vulnerabilites (percentage of client-side components)

Figure 3 shows the statistical trends of the percentage of web applications that contain high risk vulnerability from 2015 to 2019. It shows the high risk vulnerabilities fall significantly by 20% compared to that in 2015 (Positive Technology, 2019).

ISSN: 2473-4901
v6 n5352

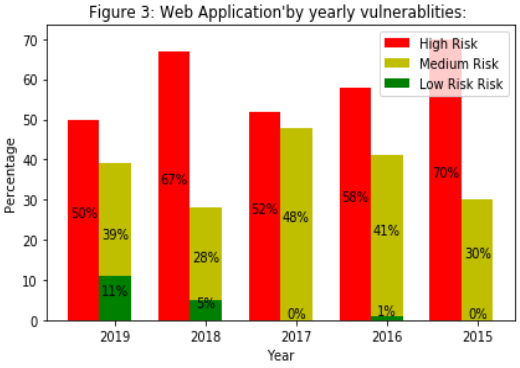


Figure 3. Website by maximum severity of vulnerabilities

This indicates that the percentage of sites containing server vulnerabilities gradually reduces, showing the consistency of improvement of web application security in the last five years. The security threat is approached on a regular basis in web applications that cause severe financial losses at various levels of Financial Institutes, IT, manufacturing, Telecom and Government. Many organizations from private to the government rely on web apps for their regular business transactions and customers' access of the relevant information. Such communication and payment activities are the target for cyber-attacks and many attempts to access the application server due to the poor code security patches configured in the application development phase. Figures 4 and 5 (extracted from (Statista, 2019) show the vulnerabilities in organizations and the most common causes of security threats and malicious attacks.

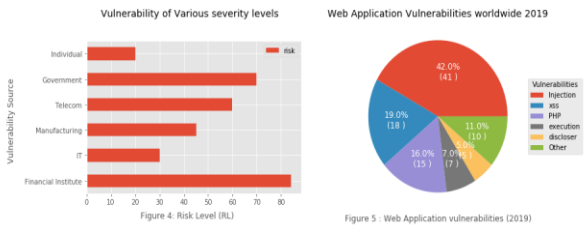


Figure 4 shows the vulnerability risk levels faced by organizations. For example, financial institutes are at high risk for cyber-attack at above 80%, and government institutes are in high-risk vulnerability too (70%). Figure 5 shows the web application vulnerabilities in which SQL injection is the major security threat globally. 42% of the threats are carried out through SQL injection, 19% are caused by cross-site scripting, and 16% by PHP vulnerabilities (Statista, 2019). Since Android has a complex system in both inter and intra application for sending and sharing data, the static analysis usually is limited to detect the malicious application due to build in application (e.g., Intent object broadcast which can be intercepted by malware running on the same device). Informed by the prior studies, we propose an Android Application tool, DroidPatrol, which offers more features to analyze static code for detecting the known Android security bugs based on OWASP guidelines.

4. DESIGN OF DROIDPATROL

We divide the DroidPatrol model into four parts: i) design of DroidPatrols, ii) features of the plugin iii) test Data leak detection: SQL injection, and iv) results from analysis. The basic code analyses focus on the possible malicious injection. The main idea is that DroidPatrol first uses static analysis to discover functions of dynamic loading behavior. For user apps and detection technique, there exist four steps:

DroidPatrol is an open source plugin for Android applications which can detect resource leakage during the application development phase. It analyzes two apk bases: source and sinks by the developer. It generates a call graph between the source and sinks that produces the output of leakage data. Since the Android application is based on Java, we use the static analysis library APIS which basically is Soot as a static analyzer for java-based applications. DroidPatrol requires two dependency libraries for jar files i) an android jar ii) an analysis-jar.

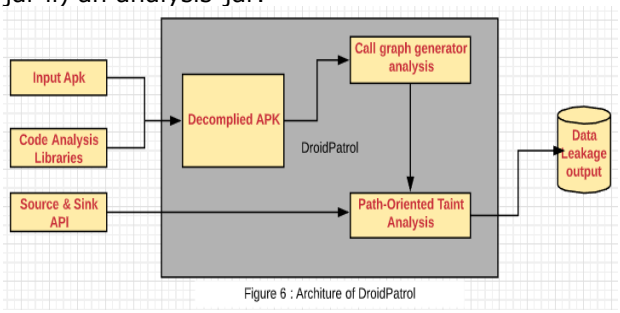


Figure 6 shows the basic architecture of Droid patrol that depicts the workflow on an apk file. The input apk is basically app-debug.apk file. DroidPatrol decomposes the apk and Code Analysis Libraries (DroidPatrol _Aanlyzer.jar and DroidPatrol_Android.jar) files, source and sink API declarations in text files. Then DroidPatrol decompiles the apk and generates a call graph and path. Finally, it generates a list of tainted data leakage output for users.

DroidPlatrol plugin applies tainted data flow analysis of android application with Tainted data

ISSN: 2473-4901
v6 n5352

flow analyzer. It intends to identify Android application security bugs based on Open Web Application Security Project (OWASP) to allow developers and security teams to use the resources they need for developing secure mobile applications. The application developers need to understand the security risks faced by the mobile apps globally. OWASP provides ten guidelines for developers to build secure applications and incorporates essential coding practices (Android Studio, 2020; Basatwar 2020). Figure 7 highlights the top 10 security risks that needs to practice by developers for application development phase.

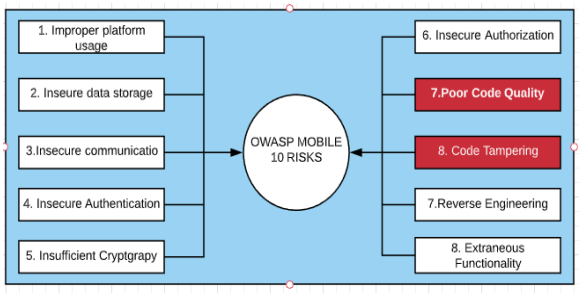
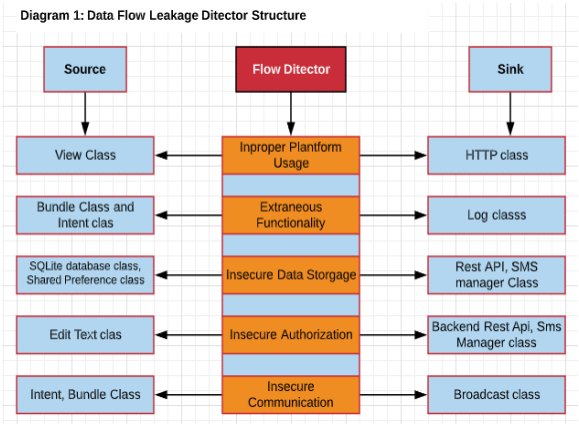


Figure 7: 10 Risks for mobile application development

The DroidPatrol plugin minimizes the mobile application security risks for SQL injection, unintended data leakage, and insecure data storage vulnerability. Diagram 1 shows how the data flow leakage from the source and sinks for extraneous functionality, improper platform usage, extraneous functionality, insecure data storage, insecure authorization and insecure communication.



DroidPatrol can manage the SQL injection and data leakage vulnerability in mobile applications that are under security threat from cyber criminals who pass the potential malicious injection. DroidPatrol can create the flag warning to the developers in the code line. Application developers can maintain the secure code for the development by following OWASP guidelines. A build package can also be loaded into the Android Studio IDE, which results in parsing Android java source code to identify specific API calls and guide the code to replace what causes the potential vulnerability in the application development phase. A build package can also be tested into the Android Studio IDE, which will result in parsing Android java source code with notifying the potential code vulnerabilities, identifying the specific API call and suggesting the secure code replacement.

5. EXAMPLE MODULE USING DROIDPATROL

In this section we analyze the web application vulnerabilities worldwide in 2019. It shows that the SQL injection is the major security vulnerability that leads to many data leakage from the user end. SQL injection is a code insertion technique in which is used to attack data driven applications. The malicious code is inserted by cyber-hacker to normal SQL statements to dump content from the database. The SQL injection exploits security vulnerabilities of the mobile application such as taking use of user input to embed to malicious code to a hard code SQL statement. The method of SQL injection takes into many forms that consists of i) Incorrectly filtered escape characters, ii) Incorrect type handling. The DroidPatrol tools that we developed can be found at:
1.https://sites.google.com/site/droidpatrolproject/sql-injection/pre-lab?authuser=0
2. https://github.com/saiful-sdsl/ResearchProjects/tree/master/DroidPatrol

Incorrectly filtered escape character form occurs if user input is passed to a SQL statement without filtering escape character. The following is the example showing how this type of SQL injection takes place.

```
$statement = "SELECT * FROM users WHERE username = '$user' AND password
                '$password' ";
```

This type of SQL statement is passed to a function which in turn sends the string to the connect data where it is parsed, executed and returns the results:

```
#Define POST variables
uname = request.POST['username']
passwd = request.POST['password']

#SOL query vulnerableto SQLi
sql = "SELECT id FROM users WHERE username='"" + uname + "' AND password='" + "'"

#Execute the SQL Statement
database.execute(sql)
```

If input is not sanitized properly but the application, the attacker can easily insert crafted

ISSN: 2473-4901
v6 n5352

value as input as following SQL statement possible to be injected:

```
$statement = "SELECT * FROM users WHERE username='wanqing' OR '1'='1' '--'
              AND password = 'wanquing'";
```

The attacker input contains two parts:

1. OR '1' = '1' is a ***condition*** which will be always ***true*** and it is accepted as a valid input by application
2. "- -" (Double hyphen) instructs the SQL parser that the rest of the line is a comment and it should be executed.

When the query is executed, the SQL injection removes the password verification, so the injection bypasses user authentication resulting in the whole database returning as the invalid input always returns true. In this way, the consequence becomes a successful SQL injection attack (Choudary, 2020; Droidpatrol, 2020).

Incorrect type handling injection is the same type of implementation of incorrectly filtered escape character, but injection takes place without appropriate type checking. There are many other forms of SQL injection, in which an injection is executed by prematurely terminating a text string and appending a new command.

The DroidPatroltool we developed tests data flow analyses to determine the tainted data flow from every possible point of access. As we defined the sources and sinks respectively where source means the location to get the data from external input such as user database query. Obtaining data from source can be transferred to a third party via SMS messaging. Figure 8 shows the sources as database Cursor object which allows to retrieve data. SmsManager is used to require SEND_SMS permission which is the sink list. Figure 8 shows the Source and sink process.

<android.database.Cursor: java.lang.String getString(int)> -> _SOURCE_

<android.telephony.SmsManager: void sendTextMessage(java.lang.String,java.lang.String,java.lang.String,android.app.PendingIntent,android.app.PendingIntent)> android.permission.SEND_SMS -> _SINK_

Figure 8 : Source and Sink process

The DroidPatrol tool provides a data flows list where information flows between source and sinks. We ran the analysis to build the apk first from the menu in the top right corner where it shows the plugin named Droid Patrol. Under the DroidPatrol the button is a command called Eye which is the code vulnerabilities analyzer.

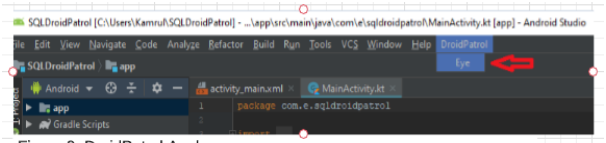


Figure 9: DroidPatrol Analyser

We prebuilt DroidPatrol "source" and "sink" files that require the process of code analyzer. The following steps are executed in the analysis process when the Eye analyzer starts in DroidPatrol. The pop-up window asks the Drive name for analyzer and android jar files. It then asks for the files that contains pre-build **SourcesAndSinks** txt file which creates the Android project folder. The text files are:

*<android.app.Activity: android.view.View findViewById(int)> -> _SOURCE_*
*<android.database.sqlite.SQLiteDatabase: android.database.Cursor rawQuery(java.lang.String,java.lang.String[])> -> _SINK_*

After analyzing the files, the DroidPatrol shows the result with 0 leaks. Therefore at the next step, we change the code in the source and sink files and the test run shows the following output: the application one data leak from input field to SQLite database query. Figure 10 shows the process of analysis by DroidPatrol.

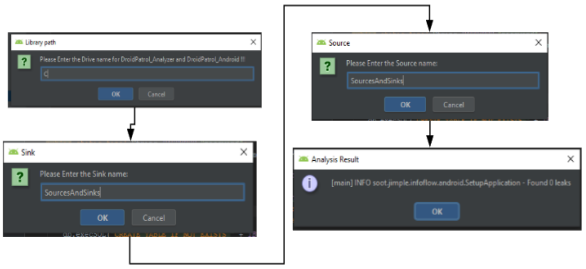
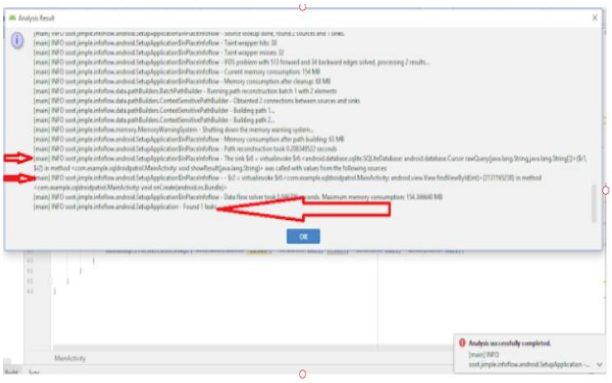


Fig 10 : DroidPatrol Analysing process

The screenshot below contains one data leak from the input field in the SQLite database query.



ISSN: 2473-4901
v6 n5352

6. STUDENT FEEDBACK

We integrated the DroidPatrol hands-on module in three courses in the summer 2020 term: IT 6513 (Electronic Health Record and App Development), IT 6843 (Ethical Hacking & Networking Security), and IT 3503 (Foundations of Health IT). To assess the effectiveness of the DroidPatrol materials and hands-on exercises, we collected student feedback. The key questions driving the survey are: What are students' knowledge levels of the specific technologies? Did the materials help students learn about the topics/technologies for analyzing application security? Did each new exercise help?

The survey was created in the University's Qualtrics system. Students were provided the link to the survey and they completed the survey online. Five questions to assess students' learning are included and the responses were collected using the Likert scale that uses a 5-point scale, 1 (Highly disagree) to 5 (Highly agree).

Q1. I like working with this hands-on labware.

Q2. The hands-on labware helped me understand SQL injection attack in mobile application and sources/sinks for SQL injection.

Q3. The real-world mobile security threats and attacks provided in the labs help me understand better the importance of static analysis.

Q4. The hands-on labs help me gain authentic learning experience to detect data flow via SQL injection and preventing it.

Q5. The online lab helped me set up the needed environment for monitoring mobile security detection.

The sample size of the survey was 65 for the three course sections. The results show that most students agreed that the DroidPatrol-based hands-on labware enabled them to learn SQL injection and detection by using statics analysis. The plugin tool also helped them prevent the data flow through SQL injection.

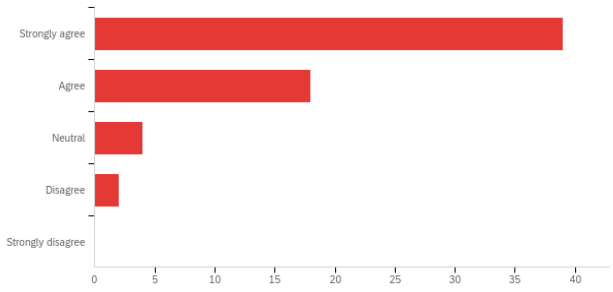


Figure 11: Survey results of Q1

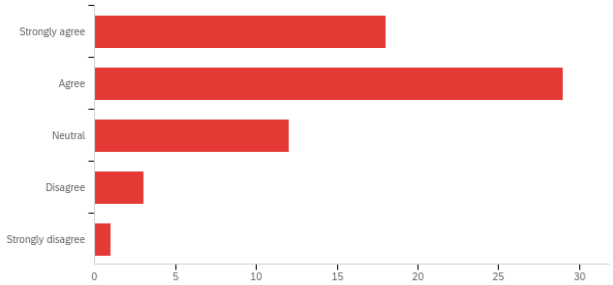


Figure 12: Survey results of Q2

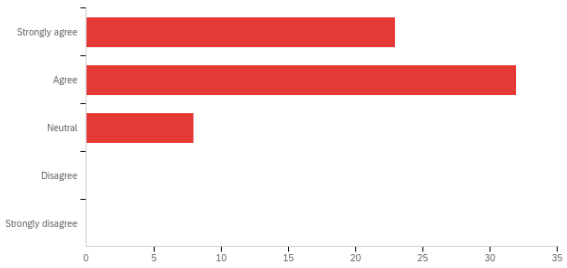


Figure 13: Survey results of Q3

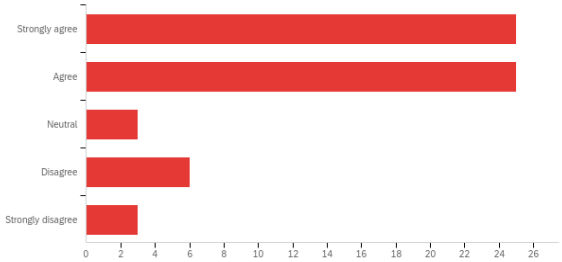


Figure 14: Survey results of Q4

Students also provided comments on their experience of using static analysis plugin tool in the hands-on labware.

- *That is a great start for software developer who can take care of security issues while developing code.*
- *The lab is great. We need more like this in the future.*
- *Lab is very good with all the necessary instructions.*
- *I am taking training on SQL this summer. This lab helped me in gaining more knowledge of it.*
- *Hands on is the best learning tool.*
- *I really liked doing this hands-on lab. I think it is easier to learn this way compared to by reading about how to set up an environment or prevent an SQL attack.*
- *I liked the variety of attacks in this lab.*

The survey shows that students are interested in learning by doing, and the plugin-based tool helps student learn developing secure mobile applications.

7. CONCLUSIONS

Currently, there is no available plugins for Android Development Studio that can be integrated for static data flow analysis. In this paper, we analyzed mobile application threats and applied a test method to find the data leakage through the DroidPatrol plugin that we developed, based on OWASP security risk analysis and guidelines. We plan to make the DroidPatrol as an open source plugin tool for application developers. The tool can perform tainted data flow analysis of applications that would help developers to detect various security bugs in static code currently leading to a number of privacy and data leaks. In addition, DroidPatrol helps developers to flag the code alarm that would be vulnerable for application.

ACKNOWLEDGEMENT

Our thanks to Affordable Learning Georgia Textbook Transformation Grants (Round 15, #484) and SunTrust Summer Faculty Fellowship for supporting the development of the lab materials.

8. REFERENCES

Ashfaq, Q., Khan, R., & Farooq, S. (2019). A Comparative Analysis of Static Code Analysis Tools that check Java Code Adherence to Java Coding Standards, Proc. of 2nd International Conference on Communication, Computing and Digital systems (C-CODE), pp. 98-103, doi: 10.1109/C-CODE.2019.8681007.

Arzt, S. & Bodden, E. (2016). StubDroid: Automatic Inference of Precise Data-Flow Summaries for the Android Framework, Proc. of 38th IEEE/ACM International Conference on Software Engineering (ICSE), Austin, TX, pp. 725-735, doi: 10.1145/2884781.2884816.

Babil, G., Mehani, O., Boreli, R. & Kaafar, M. (2013). On the effectiveness of dynamic taint analysis for protecting against private information leaks on Android-based devices, Proc. of International Conference on Security and Cryptography (SECRYPT), Reykjavik, Iceland, pp. 1-8.

Baset, A. & Denning, T. (2017). IDE Plugins for Detecting Input-Validation Vulnerabilities, 2017 IEEE Security and Privacy Workshops (SPW), San Jose, CA, pp. 143-146, doi: 10.1109/SPW.2017.37.

Basatwar, G. (2020). OWASP Mobile Top 10: A Comprehensive Guide For Mobile Developers To Counter Risks, https://www.appsealing.com/owasp-mobile-top-10-a-comprehensive-guide-for-mobile-developers-to-counter-risks/.

Beal, V. (2020). Dalvik, Available: https://www.webopedia.com/TERM/D/Dalvik.html.

Choudary, A. (2020). What Are SQL Injection Attacks And How To Prevent Them?, Available: https://www.edureka.co/blog/sql-injection-attack.

DroidPatrol. (2020). https://sites.google.com/site/droidpatrolproject/sql-injection/pre-lab?authuser=0.

Davis, J. (2020). EFF Warns COVID-19 Tracing Apps Pose Cybersecurity, Privacy Risks,Available:https://healthitsecurity.com/news/eff-warns-covid-19-tracing-apps-pose-cybersecurity-privacy-risks.

Elish, K., Cai, H., Barton, D., Yao, D., & Ryder, B. (2020). Identifying Mobile Inter-App Communication Risks, IEEE Transactions on Mobile Computing, vol. 19, no. 1, pp. 90-102.

Enck, W., Peter, G., Byung-Gon, C., Cox, L., & Jaeyeon, J., McDaniel, P., & Sheth, A. (2010). TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. Communications of the ACM. 57. 393-407. 10.1145/2494522.

Fan, W., Zhang, D., Chen, Y., Wu, F., & Liu, Y. (2020). EstiDroid: Estimate API Calls of

Android Applications Using Static Analysis Technology, IEEE Access, Vol. 8, pp. 105384-105398, 10.1109/ACCESS.2020.3000523.

Jamalpur, S., Navya, Y., Raja, P., Tagore, G., & Rao, G. (2018). Dynamic Malware Analysis Using Cuckoo Sandbox, Proc. of 2nd International Conference on Inventive Communication and Computational Technologies, Coimbatore, pp. 1056-1060, doi: 10.1109/ICICCT.2018.8473346.

Meng, X., Qian, K., Lo, D., Bhattacharya, P., & Wu, F. (2018). Secure Mobile Software Development with Vulnerability Detectors in Static Code Analysis, International Symposium on Networks, Computers and Communications (ISNCC), pp. 1-4.

Mumtaz, H. & El-Alfy, E. (2017). Critical review of static taint analysis of android applications for detecting information leakages, Proc. of 8th International Conference on Information Technology (ICIT), Amman, pp. 446-454, doi: 10.1109/ICITECH.2017.8080041.

Pfeiler, A. (2020). FindBugs-IDEA, Available:https://plugins.jetbrains.com/plugin/3847-findbugs-idea/

Shahriar, H., Riad, A., Talukder, A., Zhang, H., & Li, Z. (2019). Automatic Security Bug Detection with FindSecurityBugs Plugin. Conference on Cybersecurity Education, Research and Practice. http://par.nsf.gov/biblio/10156137

Statista. (2019). Global web application vulnerability taxonomy, Available: https://www.statista.com/statistics/806081/worldwide-application-vulnerability-taxonomy/.

Studio, Android. (2020). Report a bug, https://developer.android.com/studio/report-bugs.

Talukder, A., Shahriar, H., Qian, K., Lo, D., Ahamed, S., & Rahman, M. (2019). DroidPatrol: A Static Analysis Plugin For Secure Mobile Software Development, Proc. of 43rd IEEE Annual Computer Software and Applications Conference (COMPSAC), Milwaukee, WI, USA, pp. 565-569, doi: 10.1109/COMPSAC.2019.00087.

Technologies, Positive. (2019). Vulnerabilities and threats in mobile applications, 2019, Available:https://www.ptsecurity.com/ww-en/analytics/mobile-application-security-threats-and-vulnerabilities-2019/

Tian, C., Xia, C., Duan, Z. (2018). Android Inter-Component Communication Analysis with Intent Revision, IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion), Gothenburg, pp. 254-255.

Vermeer, B. (2019). 10 Eclipse plugins you shouldn't code without, Available: https://snyk.io/blog/10-eclipse-plugins-you-shouldnt-code-without/

Whittaker, Z. (2020). A popular virtual keyboard app leaks 31 million user's personal data, ZDNet. [Online]. Available at https://www.zdnet.com/article/popular-virtual-keyboard-leaks-31-million-user-data/

Zhang, C., Shahriar, H., & Riad, A. (2020). Security and Privacy Analysis of Wearable Health Device, Proc. of 44th IEEE Annual Computers, Software, and Applications Conference (COMPSAC), Madrid, Spain, pp. 1767-1772.

Zhao, Z. & Osono, F. (2012). TrustDroid: Preventing the use of SmartPhones for information leaking in corporate networks through the used of static analysis taint tracking, Proc. of 7th International Conference on Malicious and Unwanted Software, pp. 135-143, doi: 10.1109/MALWARE.2012.6461017.

Zheng, M., Sun, M., & Lui, J. (2014). DroidTrace: A ptrace based Android dynamic analysis system with forward execution capability, International Wireless Communications and Mobile Computing Conference (IWCMC), Nicosia, pp. 128-133, doi: 10.1109/IWCMC.2014.6906344.