Plugin-based Intervention for Secure Software Development

Hossain Shahriar[1], Kai Qian[1], Dan Lo[1], Mohammad Rahman[2], Fan Wu[3], Sheikh Ahamed[4], Emmanuel Agu[5]
[1]Kennesaw State University, Marietta, GA, USA
[2]Florida International University, Miami, FL, USA
[3]Tuskegee University, Tuskegee, AL, USA
[4]Marquettee University, Milwaukee, WI, USA
[5]Worcester Polytechnic Institute, Worcester, MA, USA

[1]{hshahria, kqian, dlo2}@kennesaw.edu, [2]marahman@fiu.edu
[3]fwu@tuskegee.edu, [4]sheikh.ahamed@marquette.edu, [5]emmanuel@wpi.edu

***Abstract*— This Innovative Practice Work in Progress presents a plugin tool named DroidPatrol. It can be integrated with the Android Studio to perform tainted data flow analysis of mobile applications. Most vulnerabilities should be addressed and fixed during the development phase. Computer users, managers, and developers agree that we need software and systems that are "more secure". Such efforts require support from both the educational institutions and learning communities to improve software assurance, particularly in writing secure code. Many open source static analysis tools help developers to maintain and clean up the code. However, they are not able to find potential security bugs. Our work is aimed to checking of security issues within Android applications during implementation. We provide an example hands-on lab based on DroidPatrol prototype and share the initial evaluation feedback from a classroom. The initial results show that the plugin based hands-on lab generates interests among learners and has the promise of acting as an intervention tool for secure software development.**

Keywords - Software Security; DroidPatrol, Static Analysis, Android Studio, SQL Injection, Open Learning Resources and Practices.

I. INTRODUCTION

With the increased demands of mobile applications in recent years, we have 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 consequences [20]. 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 as part of teaching computer science, information technology, and related courses. The rapid growth of mobile computing also results in a shortage of professionals for mobile software development, especially for Secure Mobile Software Development professionals and insufficient tool support to developing secure mobile applications [12, 13, 14].

Most vulnerability should be addressed and fixed in the software development phase [18]. If all the mobile applications are secure or have fewer 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 educational institute and learning communities to improve software assurance, particularly in writing secure code.

Many open source static analysis tools help developers to maintain and clean up the code through the analysis performed without actually executing the code (e.g., Eclipse IDE [15], IntelliJ IDE [16], FindBugs [17]). These tools focus on finding potential bugs such as inconsistencies, helping improve the code structure, conform source code to guidelines, and provide quick fixes. In general, these tools are used to ensure code quality from the very beginning and to make software development more productive. The security vulnerability checking is not their major task. Source code analysis tools could be also designed to identify security flaws within implemented code with a high confidence.

In this paper, we provide a hands-on lab using taint-based data flow analysis with DroidPatrol plugin for promoting secure software development practices. DroidPatrol allows developers to specify a list of sources and sinks and enable them to see the possible paths within the source code and suggestion of corresponding fixes. Our lab demonstrates fosters and support increased secure software development. It would allow learners to perform tainted data flow analysis for common security flaws before deploying applications.

II. RELATED WORK

Yuan and others [4] reviewed current efforts and resources in secure software development. Chi [8] built resources for secure coding practices of professionals. These learning modules provide the essential and fundamental skills to programmers and application developers in secure programming. These effort strengthens IAS Defensive Programming knowledge areas (KA) have been identified as topics/materials in the

ACM/IEEE Computer Science Curricula [5-6]. They successfully disseminated the mobile computing education, but did not emphasize the importance of secure mobile software development tools support integrated with the IDE.

Android has a complex communication system for sharing and sending data in both inter and intra-applications. Malicious applications may take advantage of built-in feature (e.g., Intent object broadcast by victim applications can be intercepted by a malware running on the same device) to avoid detection. Recently many tools are developed to perform taint-based static analysis checking, like Findbugs and DidFail [10]. They are not capable of detecting all known Android security bugs based on OWASP guidelines [7]. Detection of potential taint flows can be used to protect sensitive data, identify leaky apps, and identify malware.

A number of effort enhanced the secure software development. For example, Application Security IDE (ASIDE) plug-in for Eclipse can warn programmers of potential vulnerabilities in their code and assists them in addressing these vulnerabilities. ASIDE addresses input validation vulnerabilities, output encoding, authentication and authorization, and several race condition vulnerabilities [1-3]. However, it cannot identify Android specific security flaws. Further, ASIDE cannot be integrated with Android Studio.

FindSecurityBugs (FSB) is a plugin for the FindBugs (a plugin) that has been ported to IntelliJ IDE [19]. It only specializes in finding security issues in traditional Java code by searching for security instead of Android Specific security bugs (e.g., secure inter-process communication). Since it analyzes at the bytecode level to find defects and/or suspicious code, source code level warning not available for early fixing of security vulnerabilities. Further, it does not allow developers to customize the source of security flaws (*e.g.*, suspected API calls that should be flagged when reaching from one point to another).

III. SQL Injection Data Leak hands-on lab Design

An example is shown with SQL Injection data leakage in Android applications. Below we discuss the prelab and hands-on lab parts.

A. *Prelab*

In the prelab section, learners are introduced with the overview of static analysis, tainted data flow analysis concept, and SQL Injection vulnerability. SQL injection is a common security vulnerability in mobile applications leading to data leakage [5]. It works by adding user supplied data to a query string which leads to the alteration of SQL queries leading hackers to access to unauthorized data and bypassing logins. SQL Injection is usually used to attack Web Views or a web service. However, it can also be used to attack Activities in Android applications.

Consider the code segment in Figure 1. Here, a SELECT query is formed with user id (*userid*) and password (*password*) variables in the method *isValidUser()*. The input is obtained from *username* and *password* text boxes (which we mark as source) in the *onCreate()* method. After the query runs, the output is the name and grade, which are displayed by setting as text to textboxes (data sink) in the *isValidUser()* method.

Since, the input is not filtered, an attacker can exploit the application by providing malicious inputs for userid and password variables.

```
public void onCreate(Bundle savedInstanceState){
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main);
        username = findViewById(R.id.textView1); // source
        password = findViewById(R.id.textView1); // source
}
public boolean isValidUser(){
   …
   userid = findViewById(R.id.editText1); //retrieving id
   password= findViewById(R.id.editText2); //retrieving password
   …
   String query = "select name, grade from users where user_id= '"
+  userid + "' and  password = '" + password +"'";
   SQLiteDatabase db;
    ... … …
   Cursor c = db.rawQuery (query, null );
   name.setText(c.getString(columindex1)); //sink
   grade.setText(c.getString(columnindex2)); //sink
   return c.getCount() != 0;
}
```

Figure 1: Example of vulnerable code

B. *Hands-on lab*

We first define the sources and the sinks (see Figure 2). Source means location where input data may be obtained from external inputs such as a user or database query. For example, in Figure 2, the source is defined as database Cursor object. This object allows a program to retrieve data. Data obtained from source can be transferred to a third party via SMS messaging. In Android, to send an SMS message, SmsManager object can be used which subsequently requires SEND_SMS permission to be listed in the manifest file. Figure 2 shows both SmsManager class and SEND_SMS permission listed in the sink list. A developer can include other possible sources and sinks based on secure programming practices and OWASP guidelines. This allows the flexibility to not only detecting new security bugs, but also reducing false positive warning.

```
<android.app.Activity: android.view.View findViewById(int) ->
_SOURCE_

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

Figure 2: Source and sink definition

The tool provides us a list of dataflow where information flow between sources and sinks are displayed in the log output of the Android IDE (Figure 3).

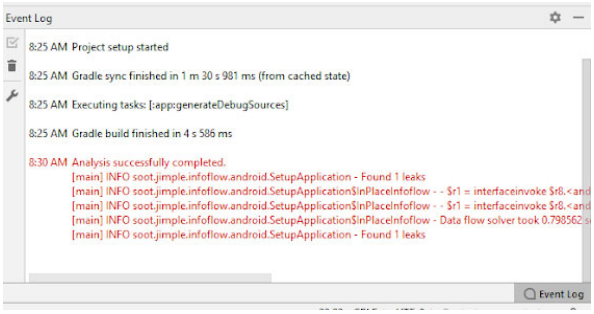

Figure 3: A sample result from DroidPatrol analysis

C. Postlab

Learners are asked to identify secure coding to prevent data leakages. Learners can identify several forms data leakage caused by SQL injection [11], 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.

IV. Preliminary Evaluation

We run our SQL injection module in Health Information Security &Privacy course (IT4533 and IT6533) during Spring 2019. The course had total 30 students. The students were asked to complete the hands-on lab after introducing mobile health app development module. A postlab survey questionnaire was conducted to receive the initial feedback on the effectiveness of the developed resources. Figure 4 shows the list of survey questionnaires. Each question answer was recorded in a scale of 1 (Strongly Disagree) to 5 (Strongly Agree).

Q1. I like being able to work with this hands-on DroidPatrol labware.
Q2. The real-world mobile security threat and attacks in the labs help me understand better on the importance of static analysis.
Q3. The hands-on labs help me gain authentic learning and working experience.
Q4. The online lab tutorials help me work on student add-on labs/assignments.

Figure 4: Postlab questionnaires ([1: Strongly Disagree, 2: Disagree, 3: Neutral, 4: Agree, 5: Strongly Agree])

We received feedback from 25 students. Below is a summary of the survey results (Figure 5), showing the mean and standard deviation. The results indicate the hands-on lab assisted learners identifying data leakage through taint analysis.

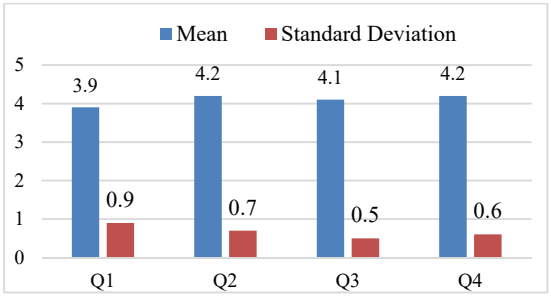

Figure 5: Postlab survey results

Below, we provide some comments received from the classroom students. The hands-on labware using the Droidpatrol generated interests among learners.

I really do enjoy performing real life exercises like this. It provides real life scenarios.

Overall the assignment was interesting and it was fun to see how you would actually do it rather than just reading a textbook or a journal about it

The material provided has good information

V. Conclusion and Future Work

Currently, there is no available learning resources relying on Android Studio plugins for hands-on learning of security concepts. In this paper, we developed a plugin tool-based SQL injection detection and mitigation lab. The labware has been applied to a classroom and the initial feedback shows interests among learners. Our developed plugin can perform tainted data flow analysis of application and intended to first hand demonstrate the active detection of various security bugs leading to privacy and data leaks based on OWASP guidelines. Our project can be used for both development in the industry and intervention-based learning resources in the classroom to promote secure coding practices. We plan to develop more hands-on labware using DroidPatrol and make them available for public use.

Acknowledgement

The work is partially supported by the National Science Foundation under award: NSF Award# 1723586, 1723578, 1723555, 1636995, and KSU OVPR Award 2018-19. 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.

References

[1] M. Whitney, H. Lipford, B. Chu, and J. Zhu, "Embedding Secure Coding Instruction into the IDE: A Field Study in an Advanced CS Course," *Proc. of the 46th ACM Technical Symposium on Computer Science Education (SIGCSE)*, Minneapolis, USA, 2015, pp. 60-65

[2] M. Whitney, H. Lipford, B. Chu and T. 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] J. Xie, H. Lipford, B. Chu, "Evaluating interactive support for secure programming," *Proceeding of SIGCHI Conference on Human Factors in Computing Systems*, Austin, TX, 2012, pp. 2707-2716.

[4] X. Yuan, K. Williams, D. Scott McCrickard, C. Hardnett, L. Lineberry, K. Bryant, J. Xu, A., Esterline, A. Liu, S. Mohanarajah, R. Rutledge, "Teaching mobile computing and mobile security," *Proc. of IEEE FIE*, 2016, pp. 1-6

[5] Computer Science Curricula 2013 - Association for Computing, https://www.acm.org/binaries/content/assets/education/cs2013_web_final.pdf

[6] K. Goseva-Popstojanovaa, A. 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] H. Chi, "Teaching Secure Coding Practices to STEM Students," *Proc. of the Information Security Curriculum Development Conference*, 2013.

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

[10] K. Dwivedi, H. Yin, P. Bagree, X. Tang, L. Flynn, W. Klieber, W. Snavely, *DidFail: Coverage and Precision Enhancement*, 2017, CMU/SEI-2017-TR-007.

[11] W. Halfond, A. Orso, and P. Manolios, "Wasp: Protecting web applications using positive tainting and syntax-aware evaluation," *IEEE Transaction of Software Engineering*, 34(1):65–81, 2008.

[12] H. Shahriar, K. Qian, M. Talukder, N. Patel and D. Lo, "Mobile Software Security Risk Assessment with Program Analysis," *Proc. of the 23rd IEEE Pacific Rim International Symposium on Dependable Computing*, Taipei, Taiwan, December 2018, 2 pp.

[13] K. Qian, D. Lo, H. Shahriar, L. Li, F. Wu, P. Bhattacharya, "Learning database security with hands-on mobile labs," *Proc. of IEEE Frontiers in Education Conference (FIE)*, Oct 2017, pp. 1-6.

[14] K. Qian, H. Shahriar, F. Wu, L. Tao, P. Bhattacharya, "Labware for Secure Mobile Software Development (SMSD) Education," *Proc. of the ACM Conference on Innovation and Technology in Computer Science Education*, March 2017, pp. 375-375.

[15] Eclipse IDE, https://www.eclipse.org/ide/

[16] IntelliJ IDEA, https://www.jetbrains.com/idea/

[17] FindBugs in Java Programs, http://findbugs.sourceforge.net/

[18] L. Tao, K. Qian, D. Lo, R. Parizi, F. Wu, B. Chu, "Enhancing Secure Software Development Education Through Relevant Active Learning," *Proc. of IEEE Southeast Conference*, 2018, St. Petersburgh, FL, USA, 2pp.

[19] Find security bugs, https://find-sec-bugs.github.io/

[20] I. Barsade, L. Davis, K. Dura, R. Ornelas, A. Smith, Prevention in the Cyber Domain, White paper, World House Student Fellows 2016-2017, https://global.upenn.edu/sites/default/files/perry-world-house/CyberPolicyProjectReport.pdf