SACH: A Tool for Assisting Secure Android Application Development

Aakiel Abernathy, Xiaohong Yuan, Edward Hill, Jinsheng Xu, Kelvin Bryant, Kenneth Williams
Department of Computer Science
North Carolina Agricultural and Technical State University
Greensboro, NC, USA
aabernat@aggies.ncat.edu

***Abstract*— To mitigate the risk of attacks to mobile applications, it is important for mobile application developers to develop secure mobile applications. There have been tools that statically analyze the mobile applications to determine whether there are data leakage or access control vulnerabilities The Software Engineering Institute at Carnegie Melon University published CERT Java secure coding rules applicable to developing android applications. This paper describes SACH (Secure Android Coding Helper) - a tool we implemented to help developers identify security vulnerabilities in Android application. The tool analyzes Android application source code to detect violations of CERT Java secure coding rules. This tool will help Android developers to write Android code that comply with CERT Java secure coding rules. It can also be used in the classroom to teach students about Android secure coding.**

Keywords—Android application development; secure coding; CERT Java secure coding rules

I. INTRODUCTION

As the demand for mobile devices keeps increasing, mobile devices have become a popular computing platform. Large amounts of data are now stored in mobile devices. Many organizations have adopted Bring Your Own Device (BYOD) policy to allow employees to access data and applications of the organization with their own devices [1-3]. However, the pervasive use of mobile devices has brought a variety of security risks including mobile malware, mobile Operating Systems vulnerabilities, insecure network connection, attacks to mobile applications, and device lost or stolen [4, 5]. Attacks to mobile devices may lead to sensitive data leakage, data loss, mobile device being accessed or controlled by unauthorized user, denial of service, and so on [6, 7].

One way to mitigate the risk of attacks to mobile applications is to develop secure mobile applications. Mobile application developers need to incorporate secure software development activities in mobile application development lifecycle to reduce security vulnerabilities in the mobile applications [8].

There have been tools that statically analyze the mobile applications to determine whether there are data leakage or access control vulnerabilities. For example, COPES [9] detects permission gaps where applications are given more permission than they need to functionally perform. Brox [10] detects whether an application requests location, deviceID, contact information and sends it via network or SMS. LeakMiner [11] detects whether an application requested location, deviceID, contact information, calendar or SMS and send these information to log files. Flowdroid [12] analyze applications statically using taint analysis to detect sensitive data leakage. TaintDroid [13] detects when sensitive data is leaving the system with the assistance of third party applications.

The Software Engineering Institute at Carnegie Melon University published CERT Java secure coding rules applicable to developing android applications [14]. Twenty-six (26) rules have been published, however, not all of them are completed. Each rule includes a brief description of the rule, a noncompliant example, compliant solutions, risk assessment, possibility of automated detection, related vulnerabilities, related guidelines, and bibliography. These rules help developers to write secure code for android applications.

This paper describes SACH (Secure Android Coding Helper) - a tool we implemented to help developers identify security vulnerabilities in Android application. The tool is based on CERT Java secure coding rules for Android applications. This tool scans the source code, manifest file, and the .properties file of the mobile application to detect violations or possible violations of the CERT Java secure coding rules. A report is generated to inform developers of possible security vulnerabilities. According to the report, developers can modify the code to fix the vulnerabilities discovered.

The rest of the paper is organized as follows: Section II describes the CERT Java secure coding rules based on which the tool is implemented. Section III describes the design and implementation of SACH. Section IV discusses the results of using SACH to scan 147 mobile applications. Section V discusses the use of SACH in secure coding education. Section VI concludes the paper.

II. THE CERT JAVA SECURE CODING RULES FOR ANDROID

The CERT Java secure coding rules that are complete are briefly described below [14].

978-1-5386-1539-3/17/$31.00 ©2017 IEEE

- (DRD01-J) Limit the accessibility of an app's sensitive content provider. Application can share data with other applications using content providers. To prevent unauthorized access to sensitive data, the export attribute in the AndroidManifest.xml should be set to "false", making the content provider private. Before API level 16, the content provider is set to public by default unless the export value specified "false".
- (DRD02-J) Do not allow WebView to access sensitive local resource through file scheme. Malware can also manipulate WebView to open malicious code (ex. maliciously crafted HTML) that is stored on the device. This can be done through setJavaScriptEnabled setPluginState and setAllowFileAccess methods within WebViews.
- (DRD03-J) Do not broadcast sensitive information using an implicit intent. Another high security risk is when developers send sensitive data implicitly throughout the system. When data is sent implicitly, any application including malware can read the data. A malware can receive and modify the data or stop it from being sent to any other receiver. It is suggested that LocalBroadcastManager.sendBroadcast() should be used so that information is received by the same application.
- (DRD04-J) Do not log sensitive information. Applications that log sensitive information leave the data vulnerable to be read. Other applications may have access to the logs (before Android 4.0) or the logs can be read if plugged into a PC. It is better not to write sensitive information to logs or to implement a custom log class so output is not automatically displayed.
- (DRD08-J) Always canonicalize a URL received by a content provider. When receiving a URL from a content provider from the ContentProvider.openFile() method the URL should always be canoncalized. Misuse of this method could lead to a directory traversal vulnerablility.
- (DRD09-J) Restrict access to sensitive activities. Restriction to the application needs to be considered when developing applications. Other applications can activate an activity for unintended use if the developer's application is accessible due to the exports value being set to true.
- (DRD10-J) Do not release apps that are debuggable. Android applications can also be vulnerable to be debugged, even without the source code. The debuggable attribute need to be set to false to avoid the application being understood by users.
- (DRD15-J) Consider privacy concerns when using Geolocation API. When using the webChromeClient#onGeolocationPermissionsShowPrompt() method it must ask for user consent before giving away their location. Misuse of the method could leak your location through other methods that have overwritten the onGeolocationPermissionshowPrompt.
- (DRD19-J) Properly verify server certificate on SSL/TLS. When a developer decides to override the default SSL and it is wrongly Implemented a user sensitive data could leak via vulnerable SSL communication channel. All certificates need to be properly verified.

III. The Design and Implementation of SACH

SACH implements algorithms to scan for violations of the CERT Java secure coding rules for Android. Algorithms for rules that are currently complete (described in section II) are implemented. SACH is developed in Java using NetBeans IDE 8.1. SACH includes a main class, the algorithms for identifying secure coding rule violations or possible violations, and a graphic user interface.

A. The Graphic User Interface

Fig. 1 shows the graphic user interface of SACH. After SACH is started, user first select from the menu "1. Analyze code", a pop-up submenu will display which allows the user to browse the directory and select the project.properties file, the AndroidManifest.xml file, and the java source code. After SACH finishes analyzing the application, the user can select "2. View Results" to see the report.

The scan results are presented in two forms: (1) a summary report in a table formatting showing the source code file name, the rule it violates, and the severity level of the vulnerability or potential vulnerability. Fig. 2 shows a screenshot of the summary report. (2) A detailed report as a text file which describes the rule violated, the file name, the line number that the vulnerability is found, explanation of the vulnerability, and suggestions to fix the code. Fig. 3 shows a screenshot of the detailed report of SACH scan results.

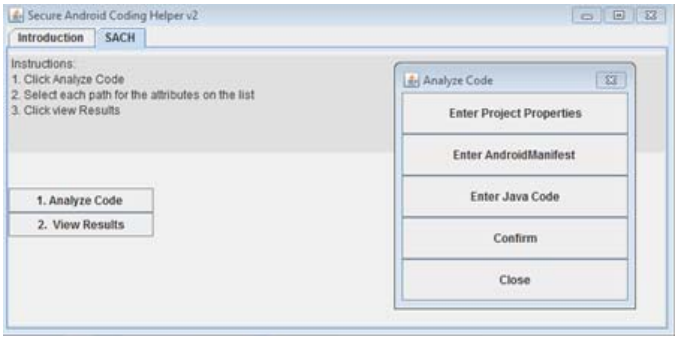

Fig. 1. The graphic user interface of SACH

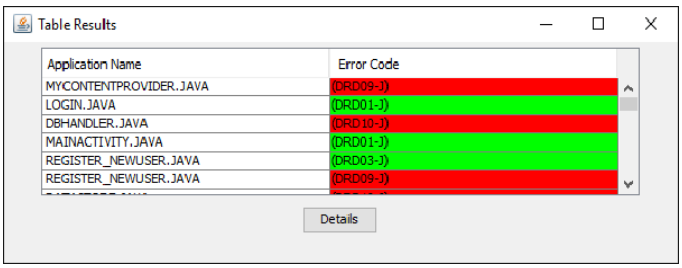

Fig. 2. The summary report of SACH scan results

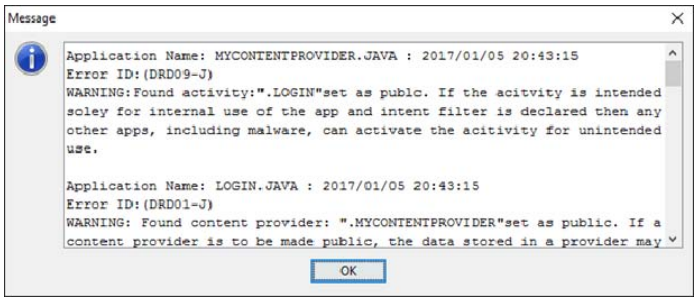

Fig. 3. The detailed report of SACH scan results

B. *Improving SACH Performance through Parallelism*

It has been noticed that when SACH scans a large mobile application with large number of Java source files, it causes long delay in the analysis and generating the results. To speed up the execution, SACH is re-designed to utilize multi-threading. The goal is that SACH could be run in cloud or big data environment to speed up the execution. The revised SACH program structure is shown in Fig. 4.

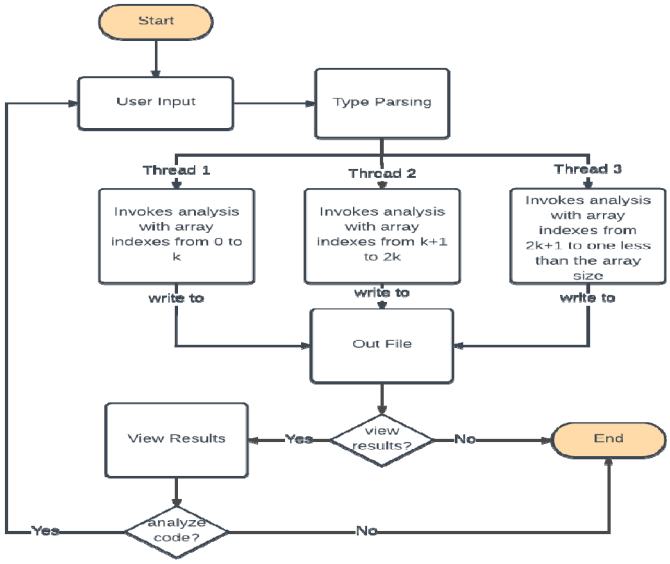

Fig. 4. The SACH program structure

IV. Results of Using SACH to Scan Mobile Applications

The algorithms implemented in SACH for identifying possible security vulnerabilities in Android applications are based on parsing the java source code and Android Manifest file to search for key words or function names that indicate possible violation of the Android secure coding rules. We used SACH to scan 174 Android applications with source code found from Google Play Store and GitHub. Table I shows the type of rules that are violated, or may have the possibility of being violated, and the number of times a warning is generated regarding the rule.

TABLE I. Results of Scanning 174 Android Applications

Rule ID	Rule Description	Number of Warnings
DRD01-J	Limit the accessibility of an app's sensitive content provider	11
DRD02-J	Do not allow WebView to access sensitive local resource through file scheme	3
DRD03-J	Do not broadcast sensitive information using an implicit intent	1
DRD04-J	Do not log sensitive information	5503
DRD09-J	Restrict access to sensitive activities	248
DRD10-J	Do not release apps that are debuggable	1

Table I shows a high number of warnings are generated for DRD04-J and DRD09-J types. However, this does not mean these two rules are violated these many times. For Rule DRD04-J, since SACH does analysis based on syntax of the source code, and not semantics, it will generate a warning when it finds a "log" function, however it cannot distinguish whether the logged information is sensitive or not. However, such warnings could prompt the developers to check what they are logging, and make sure no sensitive information has been logged. The case with DRD09-J is similar. The warnings generated for other rules show that vulnerabilities may exist for applications in the Google Play Store and Github.

V. Using SACH in Secure Coding Education

SACH has been used in the classroom for teaching Android secure coding. A course module on Android secure coding was created, which includes PowerPoint slides for a lecture on Android secure coding, and a hands-on assignment. The hands-on assignment asks students to use SACH to scan some Android applications, identify vulnerabilities, and modify the code to remove the vulnerabilities.

This course module was used in the course Programming Methodologies and Concept in the Department of Computer Science, North Carolina A&T State University in the Fall 2015 semester. The students were first introduced two modules on Android application development. Then the module of Android secure coding was introduced. After students were given a lecture on Android secure coding, and then the hands-on assignment. Twenty-nine (29) students were enrolled in the course. Nineteen (19) turned in their assignment. The average grade was 94.2. This module was taught again in the same course in the Spring 2016 semester. Sixteen (16) students enrolled in the course. Thirteen (13) students turned in the assignment.

A survey was given to the students after the course module was introduced. The survey results for both semesters are shown in Table II and Table III.

TABLE II. STUDENTS' AVERAGE RATING OF THEIR LEVEL OF KNOWLEDGE ON ANDROID SECURE CODING

Learning Objectives	Average Rating (Fall 15)	Average Rating (Spring 16)
Identify security vulnerabilitied in Android program according to CERT secure coding rules	3.06	3.1
Discuss methods to prevent security vulnerabilities in Android program	3	3.1

TABLE III. SURVEY RESULTS

Learning Objectives	Average Rating (Fall 15)	Average Rating (Spring 16)
How worthwhile was the Android Secure Coding sourse module material?	3.53	4.2
How organized was the Android Secure Coding course module material?	3.8	3.9
How motivated were you to learn about Android Secure Coding material?	3.33	4.2
How useful was the hands-on exercise with SACH in helping you understand security vulnerabilities in Android program?	3.21	4.1

The improvement of average ratings of the students may be due to the improvement of the tool SACH and teaching when the course module was taught for the second time. The above results show that SACH could be a useful tool for introducing Android secure coding to students.

VI. CONCLUSION

This paper describes SACH – a tool we implemented to help Android developers to write secure android applications. The tool analyzes Android application source code to detect possible violations of CERT Java secure coding rules for Android. Currently, algorithms for scanning for possible violations for the following rules have been implemented: DRD01-J, DRD02-J, DRD03-J, DRD04-J, DRD08-J, DRD09-J, DRD10-J, DRD15-J, and DRD19-J. The algorithms implemented in SACH are based on parsing the Java source code and Android Manifest file to search for key words or function names that indicate possible violation of the Android secure coding rules. Since semantics is not considered in the algorithms, SACH could not determine whether some of the situations are truly vulnerabilities or not. Therefore, some or a good amount of the warnings generated by SACH can only serve as a reminder to the developer to be careful with implementing certain functions.

SACH has a simple and user-friendly graphical user interface. To speed up execution time, SACH is implemented using multi-threading for parallelism. This tool will help Android developers to write Android code that comply with CERT Java secure coding rules. It has also been demonstrated that SACH could be used in teaching students about Android secure coding.

Future work will include developing and implementing more algorithms as more CERT Java secure coding rules for Android are completed. We will run SACH on cloud environment and big data environment and measure the performance of scanning large Android applications. We will also investigate how to reduce "false positives", i.e., reduce warnings that are not true vulnerabilities.

ACKNOWLEDGMENT

This work is partially supported by National Science Foundation (NSF) under the award HRD-1332504. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF.

REFERENCES

[1] Miller, K.W., Voas, J. and Hurlburt, G.F. 2012. "BYOD: security and privacy considerations", IT Professional, Vol. 14 No. 5, pp. 53-55.

[2] Yun, H., Kettinger, W. and Lee, C. 2012. "A new open door: the smartphone's impact on work-to-life conflict, stress, and resistance", International Journal of Electronic Commerce, Vol. 16 No. 4, pp. 121-152.

[3] Citrix.com, Best practices to make BYOD, CYOD, and COPE simple and secure. Retrieved on July 19, 2015 from https://www.citrix.com/content/dam/citrix/en_us/docume/oth/byod-best-practices.pdf

[4] Dwivedi, Himanshu., Clark, Chris., & Thiel, David (2010). Mobile Application Security. McGraw Hill.

[5] OWASP. Projects/OWASP Mobile Security Project - Top Ten Mobile Risks. Retrieved January 7, 2017 from https://www.owasp.org/index.php/Projects/OWASP_Mobile_Security_Project_-_Top_Ten_Mobile_Risks

[6] A. Andronic, Z. Sahlu and S. Bajracharya. Summary of Top 10 existing Android mobiel attacks and vulnerabilities (2010-2013). Retrieved January 7, 2107 from https://docs.google.com/viewer?a=v&pid=sites&srcid=ZGVmYXVsdGRvbWFpbnxtb2JpbGVzZWN1cml0eWxhYndhcmV8Z3g6NzZiYzcwOGU3NTA0ZmFhZA

[7] Six, Jeff. Application Security for the Android Platform. 1st ed. Sabastopol: O'Reilly Media, 2012. Print.

[8] McGraw, Gary. Software Security: Building Security in. Upper Saddle River, NJ: Addison-Wesley, 2006.

[9] Bartel, A., Klein, J., Le Traon, Y., & Monperrus, M. (2012, September). Automatically securing permission-based software by reducing the attack surface: an application to android. In Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering, pp. 274-277.

[10] Siyuan Ma ; Zhushou Tang ; Qiuyu Xiao., and et al (2013). Detecting GPS Information Leakage in Android Applications. Global Communications Conference (GLOBECOM), 2013 IEEE, pp. 826 – 831.

[11] ZheMin Yang., Min Yang (2012). LeakMiner: Detect information leakage on Android with static taint analysis. Third World Congress on Software Engineering. Third World Congress on Software Engineering, pp. 101 – 104.

[12] Arzt, Steven, et al. "FlowDroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps." Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation. ACM, 2014

[13] Enck, William, et al. "TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones." OSDI. Vol. 10. 2010.

[14] Seacord Robert. The CERT Oracle Secure Coding for Java: Android (DRD) .Retrieved January 7, 2017 From https://www.securecoding.cert.org/confluence/pages/viewpage.action?pageId=1115