UNIVERSITY OF SÃO PAULO

School of Arts, Sciences and Humanities

Technical Report PPgSI-003/2017

Selecting Weaknesses for Security Assessment of Android Applications

Marcos Lordello Chaim
Tosin Daniel Oyetoyan

October - 2017

Technical Report Series

PPgSI-EACH-USP
Arlindo Béttio St. 1000 - Ermelino Matarazzo - 03828-000.
São Paulo, SP. Brazil.
TEL: 55 (11) 3091-8197 http://www.each.usp.br/ppgsi

Selecting Weaknesses for Security Assessment of Android Applications

Marcos Lordello Chaim[1] **, Tosin Daniel Oyetoyan**[2]

[1]Escola de Artes, Ciências e Humanidades – Universidade de São Paulo
São Paulo – SP, Brazil

chaim@usp.br

[2]SINTEF Digital
Trondheim, Norway

tosin.oyetoyan@sintef.no

Abstract. *Smartphones are prevalent today and store sensitive and private data. Malicious applications are constant threats to user data on smartphones as they could sniff or manipulate them by exploiting software weaknesses in legitimate mobile applications. Among the mobile devices, the prevalent operating system is Google's Android with 86.8% of marketshare as of the third quarter of 2016. We describe the selection of weaknesses for security assessment of Android applications. Our rationale is to select weaknesses related to the OWASP top 10 security risks and to the Common Weakness Enumeration (CWE)—a community-developed dictionary of software weaknesses. A preliminary report on the CWEs associated with Android applications is presented. These CWEs can be used to assess the ability of static tools to pinpoint weaknesses in Android applications.*

1. Introduction

Smartphone devices are very popular today. These devices aggregate personal data related to our lifestyle, relationships, finances, professions, locations, recordings, conversations, preferences, videos and photos [Mueller 2017]. These are very sensitive and private data. A breach as a result of vulnerabilities in the mobile software could have devastating impact on the user. Malicious mobile applications could sniff and manipulate sensitive user data [Chin et al. 2011] or even launch a denial-of-service attacks [Martin et al. 2004]. Despite these challenges, developers often do not code with a mindset of attackers because they care more about functionalities. As a result, common and inadvertent mistakes become exploitable vulnerabilities [Chin et al. 2011].

The OWASP[1] tallied the top security risks for mobile applications (app) in 2016. To highlight a few of these categories, the top risk category, *improper platform use*, concerns the misuse of a platform or failure to use platform securities controls. The second most important risk, *insecure data storage*, is related to insecure data storage and unintended data leakage. The risk category #7, *client code quality issues*, encompasses vulnerabilities such as buffer overflow, format string, use of insecure or wrong APIs, and insecure language constructs. As a final example, the *extraneous functionality* category relates to hidden backdoor functionalities or other internal development security controls that are not intended to be released into a production environment.

[1]OWASP – Open Web Application Security Project (https://www.owasp.org).

Static analysis of the application's source or object code has been advocated as a strategy to detect such weaknesses [Chess and McGraw 2004] during development. The goal is to detect part of the code that could become vulnerable. Static analysis tools (SATs) are utilized to support developers to identify security risks in their code.

To carry out comparisons among SATs a common ground is needed. Currently, the OWASP is sponsoring its "Benchmark Project" OWASP [2017b] which consists of a free and open test suite designed to evaluate speed, coverage, and accuracy of automated software vulnerability detection tools and services. The National Institute of Standards and Technology (NIST) Software Assurance Reference Dataset (SARD) project aggregates programs with a set of known flaws to be used in code-quality tools evaluation [NIST 2017]. It lists several open-source web applications but only one app with known vulnerabilities. The developers of the OWASP benchmark, though, caution that the tests are simpler than real applications and may have code patterns that do not occur frequently in real code [OWASP 2017b].

The goal of this research is to assess tools that detect security-related weaknesses in Android applications. We choose Android because of its open platform and market dominance. Data from the third quarter of 2016 show Android with 86.8% of marketshare followed by Apple's iOS with 12.5% and others (e.g., Windows phone, Simbian) with 0.7% [Corporation 2017]. In addition, other smartphone platforms have similar security model, however, Android is claimed to have the most sophisticated application communication system [Chin et al. 2011].

The first step to achieve our goal is to identify relevant weaknesses in Android application to assess SATs. In this work, we describe the selection of weaknesses for security assessment of Android applications. Our rationale is to select weaknesses related to the OWASP top 10 security risks and to the Common Weakness Enumeration (CWE)—a community-developed dictionary of software weaknesses—by MITRE [MITRE 2017]. The selected CWEs can be utilized to assess the effectiveness and efficiency of SATs in pinpointing vulnerable excerpts of code.

In the next section, we present the structure of Android apps highlighting the challenges it poses for security assessment. In Section 3, we present 8 CWEs associated to the OWASP top 10 security risks that SATs should be able to pinpoint in Android apps. In Section 4, we draw our conclusions and present our future work.

2. Structure of Android apps

Figure 1 describes the architecture of Google's Android operating system (OS). In Android, user-installed applications are sandboxed, each runs in a dedicated process, each has its own private data directory, and employs the least privilege principle [Elenkov 2014]. Android defines four types of components: **Activity** (user interface), **Service** that executes processes in the background, **Content Provider** for data sharing, and **Broadcast Receiver** that responds asynchronously to system-wide messages. Communication between applications are achieved through a message passing mechanism (Intent messages) (e.g., `sendBroadcast (Intent i)`, `startActivity(Intent i)`, `startService(Intent i)`). Configuration of application components are done in the mandatory manifest file (`AndroidManifest.xml`) which cannot be modified at run-time. In order to protect applications, Android defines four types of permissions:

Normal, Dangerous, Signature, and SignatureOrSystem.

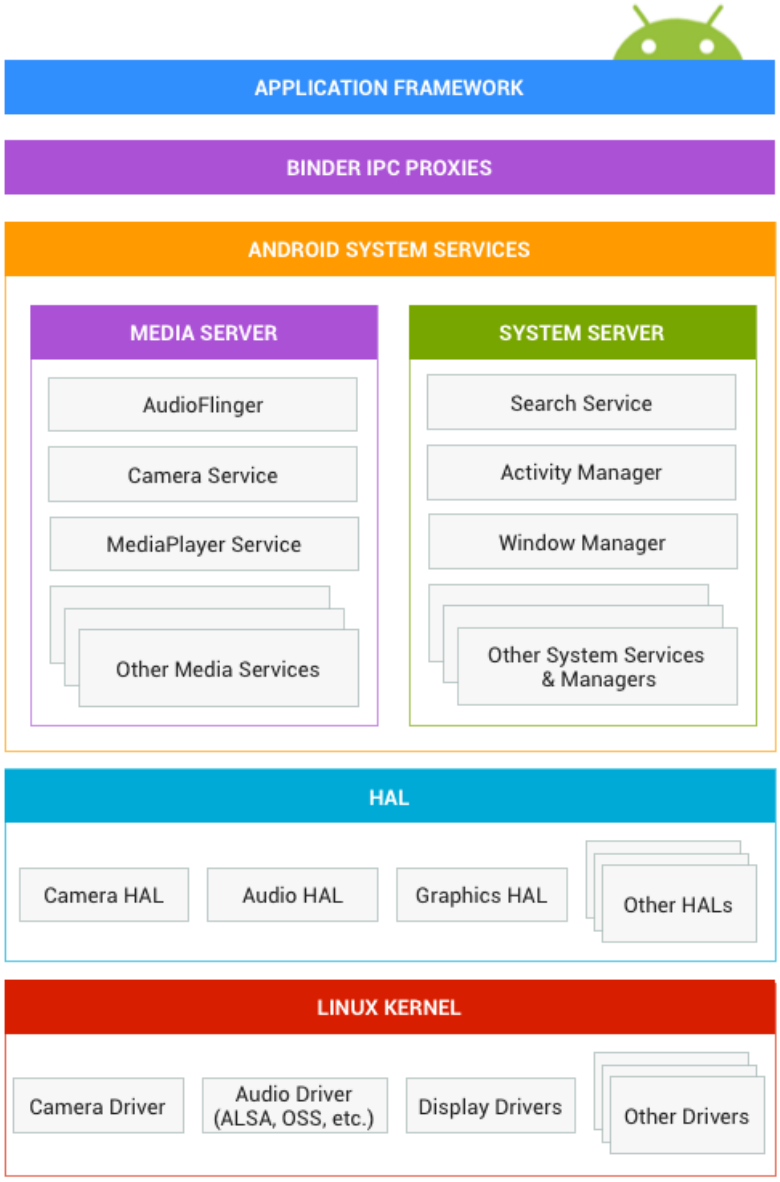

Figure 1. Android architecture

Specific challenges in Androids make static analysis different from regular Java applications [Li et al. 2016]. Android apps run in a special virtual machine named Dalvik that generate bytecodes differently from regular Java virtual machine. As a result, static analysis tools must be able to analyze the Dalvik bytecode when Java source code is not provided. Further, Android apps could have many entry (Main) points which make them different from regular Java applications. Additionally, in Android apps, different components have their own lifecycle. Because these lifecycle methods are not directly linked to the execution flow, they limit the soundness of some analysis scenarios.

3. Common Weaknesses in Android Applications and CWE Selection

Based on the OWASP top 10 2016, the most common security risks in mobile applications are: (1) Improper platform use, (2) Insecure data storage, (3) Insecure com-

munications, (4) Insecure authentication, (5) Insufficient cryptography, (6) Insecure authorization, (7) Client code quality issues, (8) Code tampering, (9) Reverse engineering, and (10) Extraneous functionality OWASP [2017a]. Many empirical studies have as well validated the existent of these risks in many real-world Android applications. (see Chin et al. [2011], Enck et al. [2011], Martin et al. [2004], Jiang and Xuxian [2013])

To show that it is possible to map CWEs to top OWASP risk categories and vice-versa, we report priliminarily 8 weaknesses MITRE [2017] categories to assess the selected static analysis tools. Three categories are specific to Android applications. The rest are general quality weaknesses applicable to all applications. The rationale behind this choice is to investigate how the tools could detect weaknesses in the different categories. Additionally, we mapped the selected CWEs to the OWASP's top security risk categories.

CWE-927: Use of Implicit Intent for Sensitive Communication (#3) An implicit intent can be used to transmit data without specifying the receiver. It is possible for any application to process the intent by using an Intent Filter for the intent.

CWE-926: Improper Export of Android Application Components (#1) Android application components (Activity, Service, or Content Provider) are exported through the manifest file. Exporting components without proper restriction as to which applications can launch or access the data could result into integrity, confidentiality and availability issues.

CWE-319: Unencrypted Socket (#3) The study by Enck et al. Enck et al. [2011] shows that certain Android applications include code that use the Socket class directly. Java sockets are potential attack surface as they represent an open interface to external services.

CWE-921: Storage of Sensititve Data in a Mechanism without Access Control (#2) This weakness occurs when applications store sensitive information in file systems or devices that are not protected. Examples include memory cards or USB devices.

CWE-359: Exposure of Private Information ('Privacy Violation') (#6) Accessing private data such as passwords or credit card numbers need explicit authorization. Privacy violation could occur when unauthorized entities have access to data.

CWE-478: Missing Default Case in Switch Statement (#7) This weakness occurs when code that uses switch statement omit the default case. Execution logic may be altered when the system encounters variable value not handled in the logic. Security issues may happen, if switch logic is used to handle security decision or is linked to other aspects of code where security decision happens.

CWE-611: Improper Restriction of XML External Entity Reference ('XXE') (#7) Applications that process XML documents could be vulnerable to XXE-attacks if proper validations and sanitations are not put in place. An example is the CVE-2016-6256 XML External Entity(XXE) attack in the SAP Business One Android Application.

Debug Mode Activated (DMA) (#10) There are cases where production code is shipped with developer's configuration. An example is when debug option is enabled which can lead to disclosure of confidential and senstitive data.

Table 1 presents the same information but in a different form. In this preliminary

Table 1. OWASP top security risks *versus* CWEs

#	Top OWASP security risk	CWE
1	Improper platform use	CWE-926
2	Insecure data storage	CWE-921
3	Insecure communications	CWE-927, CWE-319
4	Insecure authentication	—
5	Insufficient cryptography	—
6	Insecure authorization	CWE-359
7	Client code quality issues	CWE-478, CWE-611
8	Code tampering	—
9	Reverse engineering	—
10	Extraneous functionality	Debug Mode Activated (DMA)

selection, we do not identify CWEs that are related to security risks #4, #5, #8, and #9. Category #4 captures notions of authenticating the end user or bad session management. This can include: failing to identify the user at all when that should be required; failure to maintain the user's identity when it is required; and weaknesses in session management [OWASP 2017a]. Category #5 is related to insuficient application of cryptography to a sensitive information asset. Category #8 covers binary patching, local resource modification, method hooking, method swizzling, and dynamic memory modification [OWASP 2017a]. Risks in category #9 include analysis of the final core binary to determine its source code, libraries, algorithms, and other assets, indicating that parts of the code should be obfuscated to protect against reverse engineering.

These issues related to categories #4, #5, #8, and #9 seem not directly related to code issues–the target of our research. Category #4 suggests weaknesses that are related to the lack of requirements and category #5 of appropriate cryptography. Category #8 encompasses risks that could be mitigated by run-time safe guards; and category #9 those risks associated to the lack of obfuscation of the code. Nevertheless, we intend to conduct a thorough analysis of the CWE dictionary to identify issues related to these categories.

4. Conclusions

We presented a selection of common weaknesses enumerations (CWE) — a dictionary of known weaknesses type maintained by MITRE [MITRE 2017] — to assess static analysis tools (SAT). These CWEs were selected having as guidelines the top risk categories for mobile applications developed by OWASP. We focus on Android related security issues because it is the prevalent OS among smartphone users and is known for its complex architecture.

CWEs are a industrial standard serving as a common vocabulary among the security practitioners. Knowing the CWEs that are related to the top security issues has some implications: (1) they allow comparisons regarding the effectiveness and efficiency of SATs in detecting relevant weakneesses; (2) they serve as anti-patterns for the developers of critical Android applications.

The results reported are preliminary. We plan to extend the analysis of more CWEs to create a set of top risk CWEs that can be utilized as guidelines for tool evaluation and

mobile apps development.

5. Acknowledgments

This research was carried out within the project "SoS-Agile: Science of Security in Agile Software Development", funded by the Research Council of Norway, under the grant 247678/O70. Marcos L. Chaim's was on a research stay in Norway and was funded by a personal guest researcher scholarship from the IKTPLUSS program and by the University of Sao Paulo, Sao Paulo, Brazil.

References

Chess, B. and McGraw, G. (2004). Static analysis for security. *IEEE Security & Privacy*, 2(6):76–79.

Chin, E., Felt, A. P., Greenwood, K., and Wagner, D. (2011). Analyzing inter-application communication in android. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, pages 239–252. ACM.

Corporation, I. D. (2017). Smartphone os market share, 2016 q3. http://www.idc.com/promo/smartphone-market-share/os. Visited on June, 13 2017.

Elenkov, N. (2014). *Android security internals: An in-depth guide to Android's security architecture*. No Starch Press.

Enck, W., Octeau, D., McDaniel, P. D., and Chaudhuri, S. (2011). A study of android application security. In *USENIX security symposium*, volume 2, page 2.

Jiang, Y. Z. X. and Xuxian, Z. (2013). Detecting passive content leaks and pollution in android applications. In *Proceedings of the 20th Network and Distributed System Security Symposium (NDSS)*.

Li, L., Bissyande, T. F. D. A., Papadakis, M., Rasthofer, S., Bartel, A., Octeau, D., Klein, J., and Le Traon, Y. (2016). Static analysis of android apps: A systematic literature review. Technical report, SnT.

Martin, T., Hsiao, M., Ha, D., and Krishnaswami, J. (2004). Denial-of-service attacks on battery-powered mobile computers. In *Pervasive Computing and Communications, 2004. PerCom 2004. Proceedings of the Second IEEE Annual Conference on*, pages 309–318. IEEE.

MITRE (2017). Common Weakness Enumeration (CWE)—a community-developed list of software weakness type. https://cwe.mitre.org/. Visited on June, 14 2017.

Mueller, B. (2017). OWASP Mobile Application Security Verification Standard v0.9.3: Foreword. Technical report, OWASP – Open Web Applications Security Project. Visited on June, 12 2017.

NIST (2017). Software Assurance Reference Dataset Project. https://www.owasp.org/index.php/Benchmark.

OWASP (2017a). Mobile Top 10 2016. https://www.owasp.org/index.php/Mobile_Top_10_2016-Top_10.

OWASP (2017b). OWASP benchmark Project. https://www.owasp.org/index.php/Benchmark.