

Mobile Application Security Assessment

Rayapati Tejasree, Runjhun Mathur, Trupti Sharma

B.E CSE IS

Chandigarh University, Punjab, India

Ms. Sheetal Laroiya Assistant Professor *Chandigarh University* Gharuan, Punjab, India
sheetal.e15433@cumail.in

Abstract: The growth of smart devices has led to a rise in security concerns, with flaws in established applications being exposed and the dissemination of mobile harmful software. Both the government and corporations offer a range of apps to the public. To ensure the dependability and security of these applications, security checks are necessary during the development process. This study focuses on identifying and analyzing vulnerabilities in the mobile device environment, specifically in the development of Android-based applications, in order to effectively address security risks.

This research study highlights the significance of mobile application security and offers guidance on identifying and mitigating security risks related to mobile applications. It specifically focuses on concerns such as insecure data storage, insecure communication. It is thought that this will enhance the safety of mobile applications by doing vulnerability analysis for Android application security.

Keywords: *OWASP, ZAP, PENTESTING, Privacy, MOBSF*

I. INTRODUCTION

Every software system is susceptible to having a significant portion of bugs when it is released. These faults are a result of various causes, such as the widespread and complex nature of the software, as well as the mounting market demands to quickly implement new services and features, which restricts the amount of time available for testing. In addition to the potential outcomes of these faults in terms of reliability and availability, which have been extensively studied in the literature, these defects might also result in software vulnerabilities.

A software vulnerability refers to any weaknesses or defects in software that can be exploited by an attacker who possesses the knowledge and intention to do so. Examples of frequent software vulnerabilities are buffer overflows, cross-site scripting (XSS) and SQL Injection. Developing vulnerability-free software could be impossible to achieve due to time and economical limits. However, there exist different approaches to disclose software vulnerabilities, such as data flow analysis, taint analysis or fuzzy testing, for just a few.

present an auditing technique for mobile applications covering the hazards pointed out by OWASP. We firstly evaluate, assess, and

The more vulnerability detection approaches are used, the higher the possibility that the software has fewer residual vulnerabilities. However, the application of these strategies is usually expensive and burdensome.

Numerous applications are available to carry out various everyday tasks, such as checking email, news, social media, or bank accounts, owing to the rapidly expanding mobile applications ecosystems.

Therefore, users rely on these applications sensitive data. Hence, any vulnerability in mobile applications can put in risk the user's privacy. Besides, mobile applications ecosystem promotes the first one-of-a-kind application released. So, mobile applications are usually released to the market as fast as possible, castigating essential aspects like security and reliability. In order to cope with vulnerabilities and security threats in software products, security auditing methodologies have been offered to certify certain security degree of confidence. To the best of our knowledge, there are neither recognized consensus nor guidelines about how to execute a security audit in mobile applications. However, there are common best practices like the IEEE standard for software audits. However, these basic best practices are not Considering the idiosyncrasy of mobile applications. The most closed approach to produce some best practices or recommendations to develop and maintain secure mobile applications is governed by Open Web Application Security Project (OWASP).

OWASP offers developers and security teams resources to build and maintain secure mobile applications, but no formal standard is provided. However, it is crucial to appreciate the effort conducted by OWASP in order to highlight the most representative risks during mobile application development. In this paper, we

classify the OWASP 2024 Top Ten Mobile Risks under different analysis blocks. Then, we offer a methodology that uses these analysis blocks to audit the security of mobile applications. The methodology is validated in practice by assessing the identical functional application produced for one different mobile platform: Google's Android. We focus on these platforms since they lead the worldwide smartphone market.

I. ANALYSIS BLOCKS TO IDENTIFY MOBILE RISKS

In this section, we describe the OWASP Top Ten Mobile Risks for 2024 to indicate where the common errors are located and how they might be surfaced during an application security auditing.

The OWASP Top Ten Mobile Risks identify the most common risks according to different aspects such as threatening agents, attack vectors, weaknesses, technical impact, and business effect. OWASP analyzes common mobile security vulnerabilities and provide assurance controls to reduce their impact or possibility of exploitation. The OWASP risks are briefly introduced:

M1) Improper Credential Usage

This is specific to the application. Threat agents may employ automated assaults with freely accessible or specially created tools to take advantage of hardcoded credentials and incorrect credential usage in mobile applications. These agents might be able to find and take advantage of hardcoded credentials or exploit flaws brought about by incorrect credential usage. They can also abuse credentials by, for example, obtaining access by using incorrectly stored or validated credentials, avoiding the requirement for authorized access.

M2) Inadequate Supply Chain Security

By taking advantage of weaknesses in the supply chain for mobile apps, an attacker can modify the functionality of the program. For instance, an attacker may change the code during the development process to add backdoors, spyware, or other malicious code, or they may inject harmful code directly into the mobile app's codebase. As a result, the attacker may be able to manage the mobile device, steal data, or spy on people. Additionally, an attacker can access the mobile application or the backend servers by taking advantage of flaws in third-party software libraries, SDKs, suppliers, or hardcoded credentials.

M3) Insecure Authentication/Authorization

Threat actors usually utilize automated attacks with readily accessible or specially designed tools to exploit the vulnerabilities in authorization and authentication.

M4) Insufficient Input/Output Validation

The lack of proper validation and sanitization of data obtained from external sources, such as user inputs or network data, in a mobile application can lead to significant security issues. Mobile applications that do not adequately validate and sanitize data are vulnerable to exploitation through attacks that are specifically targeted towards mobile environments. These attacks include SQL injection, Command Injection, and cross-site scripting (XSS) assaults.

Insufficient output validation can lead to data corruption or presentation flaws, enabling malicious individuals to insert malicious code or change critical information shown to users.

M5) Insecure Communication

Modern mobile applications typically communicate with one or many remote servers

exchange data. During data transfer, information usually passes through the carrier network of the mobile device and the internet. However, if the data is transmitted in plain text or using an outdated encryption standard, a malicious individual listening on the network can intercept and alter the data. Threat actors may possess several motivations, including the theft of sensitive information, engaging in espionage, and perpetrating identity theft, among others.

M6) Inadequate Privacy Controls

Protecting Personally Identifiable Information (PII), such as names, addresses, credit card numbers, email addresses, IP addresses, and details about one's health, sexual orientation, religion, and political beliefs, is the goal of privacy controls.

The aforementioned information holds significant value to potential attackers due to various factors. For instance, a malicious individual could assume the identity of the victim in order to engage in fraudulent activities, exploit the victim's financial information, extort the victim by leveraging sensitive data, or inflict harm upon the victim by tampering with or destroying their essential data.

M7) Insufficient Binary Protections

The binaries could contain important information, such as commercial API keys or hardcoded cryptographic secrets that an attacker could misuse. In addition, the code in the binary could be important on its own, for example, because it contains critical business logic or pre-trained AI models. Some attackers might even not target the app itself but use it to study potential flaws of the relevant backend to prepare for an attack. Technologies, and data emulsion ways employed to prognosticate parking vacuity directly, easing informed decision- making for motorists.

M8) Security Misconfiguration

Security misconfiguration in mobile apps refers to the poor configuration of security settings, permissions, and controls that can lead to vulnerabilities and illegal access. Threat agents who can exploit security misconfigurations are attackers aiming to acquire unauthorized access to sensitive data or execute damaging acts. Threat agents can be an attacker with physical access to the device, a malicious software on the device that leverages security misconfiguration to execute unauthorized operations on the target vulnerable application context.

M9) Insecure Data Storage

A mobile application's insecure data storage can attract in different threat actors that seek to take advantage of the weaknesses and obtain unauthorized access to private data. Competitors and industrial spies looking to gain a competitive advantage, activists or hacktivists with ideological motivations, state-sponsored actors conducting cyber espionage, cybercriminals seeking financial gain through ransomware or data theft, script kiddies using pre-built tools for simple attacks, data brokers looking to exploit insecure storage for the sale of personal information, and skilled adversaries targeting mobile apps to extract valuable data are some examples of these threat agents.

These threat agents take advantage of flaws in data security, inadequate encryption, unsafe data storage practices, and inappropriate user credential management.

M10) Insufficient Cryptography

Threat actors have the ability to compromise the confidentiality, integrity, and authenticity of sensitive data by taking advantage of weak cryptography in mobile applications. These

threat actors include cybercriminals who use weak encryption to steal important data or commit financial fraud, attackers who target cryptographic algorithms or implementations to decrypt sensitive data, malicious insiders who alter cryptographic procedures or divulge encryption keys, and state-sponsored actors involved in cryptanalysis for intelligence purposes.

II. LITERATURE SURVEY

Existing related work

For the prevention of Mobile attacks, the various researchers have been implemented. Some major techniques are given studied below.

Penetration Testing: Concepts, Attack Methods (Mathew et al)

This article delves into various aspects of penetration testing tools, techniques, and methodologies. It discusses vulnerabilities exploitation and hacking of smart devices, including Bluetooth devices and Wi-Fi Protected Access (WPA). The paper explains different types of penetration tests, tools, and techniques. However, it lacks discussion on how effectively they can be used for the development of secure Android applications.

B. *Detecting Security Vulnerabilities in Web Applications Using Dynamic Analysis*

The paper proposes a framework employing dynamic analysis, black box testing, and static methods together to address weaknesses of individual methods. It claims to deliver better results by integrating the functions of the three methods. The framework effectively discovers vulnerabilities in applications and services. However, it does not cover how the proposed methods can be employed specifically in an Android environment.

C. Bacudio et al

This paper reviews penetration testing, its effectiveness, and methodology. It involves a thorough analysis of the testing methodology phases: test preparation, test, and test analysis. The application of the methodology is illustrated on two web applications. The author discusses penetration test methodologies and frameworks but does not delve into specific tools and techniques for pen-testing.

D. Sicari et al

This study focuses on the security of middleware, A. solutions for securing mobile devices, and how vulnerabilities can be handled. However, it lacks discussion on penetration test frameworks and developer- related challenges for Android app development.

E. Gerry et al

The paper discusses penetration testing and related factors. It elaborates on various techniques and tools for employing security. Additionally, it covers ISO 27000 security standards, guidelines, and ethics for developers for secure coding. However, it lacks discussion on penetration test frameworks and tools that can help in modeling security vulnerabilities.

F. Using a Protocol Analyzer to Introduce Communications Protocols (Mosenia et al)

This paper discusses how developers are made familiar with communication protocols such as Ethernet, ARP, IP, ICMP, UDP, and TCP. However, it lacks discussion on penetration test frameworks and does not cover how proposed methods can be employed specifically in an Android environment.

III. Methodology

This section outlines a thorough methodology that addresses the OWASP Top Ten Mobile Risks in order to perform a security assessment of mobile applications. Our approach is intended to facilitate the identification of security holes, and the evaluation of the functionality of an application. Pre-runtime, Runtime, and Post-runtime are its three primary phases. Each one consists of several tasks meant to evaluate various areas of mobile application security in-depth.

A. Pre-runtime

Preliminary Analysis, which incorporates information collection to obtain both passive data about the application, is the first step in the Pre-runtime phase. This contains information on the build, the Android version, and the description of the program. The next step is to reverse and decrypt the application in order to extract the binary code in a format that can be read by humans. This is necessary for additional analysis.

Static analysis is performed after the Pre- runtime stage to examine the application's code and structure in more detail. To find vulnerabilities and private data, this entails carefully examining strings extracted from source code, binary files, and configuration files.

B. Runtime

As we move to the Runtime stage, our approach expands on the static analysis to encompass important elements like intents, internal storage, and external storage, all of which are essential to comprehending the behavior of the program and its security threats.

In runtime static analysis, intents—a crucial component of inter-component communication in Android applications—are examined in detail. Data flow across various services, activities, and broadcast receivers is made easier by intentions. We seek to identify intent misuse issues, such as faulty intent data validation, unsafe implicit intent processing, and unintentional component access, by closely examining how intentions are declared, managed, and used inside the application.

Moreover, we also analyze file actions performed within the application's internal storage. Private data that is only accessible by the application itself is kept in internal storage. We can spot any security problems including inappropriate permissions on sensitive files, careless file handling, and the possibility of data leaking from careless storage procedures by keeping an eye on file read and write operations in this storage environment. In a similar vein, we examine how the program interacts with external storage, or shared storage that is available to users and other apps.

C. Post-runtime

We hope to identify vulnerabilities such as insecure file permissions, the possibility of other applications gaining unauthorized access to sensitive data, and possible privacy breaches brought on by insecure data storage practices by closely examining how the application interacts with external storage, including reading and writing data to shared directories. The results of the Pre-runtime and Runtime analyses are used to make conclusions in the post-runtime phase. The analysis results are


summarized in a comprehensive report. We hope to offer a thorough analysis of the application's behavior during runtime by incorporating these elements into our methodology. This will make it possible to spot security flaws and possible areas for improvement, strengthening the application's security posture and safeguarding user data from abuse and unauthorized access.

SYSTEM ARCHITECTURE AND RESULTS

The Android application package (APK) scanning and analysis process is made easier by the APK Scanner system's architecture, which consists of multiple interrelated components. The solution is designed to provide thorough vulnerability scanning for the threats listed in the OWASP Top 10 Mobile Risks.

Command-Line Interface (CLI):

The APK Scanner system offers a command-line interface (CLI) as its user interface, permitting efficient interaction with the instrument through text-based commands. Through the CLI, users may upload APK files for scanning and examine the results smoothly.



```
APKSCANNER

usage: APKScanner.py [-h] -apk APK [-source_code_path APK] [-report {json,pdf,html}]

options:
  -h, --help            show this help message and exit
  -apk APK              Path to the APK file to be analyzed.
  -source_code_path APK Enter a valid path of extracted source for apk.
  -report {json,pdf,html} Format of the report to be generated. Default is JSON.
  (venv) PS C:\Users\tej\Desktop\APKScanner>
```

Figure 1 Interface

The CLI commands are designed to be basic, allowing users to launch scans by entering the file path of the APK they wish to investigate. Once the scanning procedure is complete, the CLI delivers the results right within the interface, displaying them in an organized way for easy understanding. Users may move through the scanning process intuitively, with clear instructions and prompts directing them through each step. Overall, the CLI offers as an easy and accessible mechanism for users to analyze the security of Android applications, providing a streamlined experience for conducting scans and reviewing results within the command-line environment.

Static analysis:

Static analysis offers detailed information that provides insights into the security posture of Android applications. This result is delivered in a structured way, allowing users to quickly analyze the findings and take relevant actions to remedy any discovered vulnerabilities.

The static analysis output often includes a summary of the scan results, highlighting major findings and security concerns found inside the APK file. This summary provides as an overview of the application's security state, giving users with an immediate awareness of any potential threats.

Report Generation:

The system creates extensive reports summarizing the outcomes of both static analysis. These reports contain specific information about permissions, intents, and storage consumption within the application. The reports are designed to be useful and actionable, giving customers with the insights needed to improve the security of their Android applications.

Basic Information

Report Date: 2024-04-04
Package: com.android.insecurebankv2
Build: 22
Android Version: 1

Permissions

- android.permission.INTERNET
- android.permission.WRITE_EXTERNAL_STORAGE
- android.permission.READ_PHONE_STATE
- android.permission.USE_CREDENTIALS
- android.permission.GET_ACCOUNTS
- android.permission.READ_SMS
- android.permission.READ_CONTACTS
- android.permission.READ_PHONE_STATE
- android.permission.READ_EXTERNAL_STORAGE
- android.permission.READ_CALL_LOG
- android.permission.ACCESS_NETWORK_STATE
- android.permission.ACCESS_COARSE_LOCATION

Figure 2 Report Sample (Taken from ApkDeeplens)

V. CONCLUSION

In conclusion, the APK Scanner system provides a great tool for checking the security posture of Android applications. By static and dynamic analysis approaches, the system effectively identifies vulnerabilities, rights misuse, and other security problems within apps.

The static analysis component of the system analyzes the application's code and structure to discover typical security vulnerabilities such as inappropriate credential usage, unsecured storage methods, and inadequate privacy measures. Additionally, dynamic analysis assesses the application's behavior during runtime to uncover potential dangers such as network vulnerabilities and data leakage.

Through the compilation of thorough reports, the APK Scanner system helps developers and security professionals to understand the security implications of their applications and take suitable measures to prevent risks.

Overall, the APK Scanner system plays a critical part in the ongoing endeavor to improve the security of mobile applications and preserve the privacy and integrity of user data in the ever-evolving world of mobile technology.

VII. REFERENCES

- [1] Botas, Álvaro & García, Juan F. & Alonso Lopez, Javier & Balsa, Jesús & Lera, Francisco & García, Christian & Matellán, Vicente & Riesco, Raúl. (2016). Security Assessment Methodology for Mobile Applications. K. Ewusi-Mensah, Software Development Failures. MIT Press, 2003.
- [2] Charette, "Why software fails [software failure]," IEEE Spectrum, vol. 42, no. 9, pp. 42–49, September 2005.
- [3] O. H. Alhazmi, S. Woo, and Y. K. Malaiya, "Security vulnerability categories in major software systems," in Procs. IASTED 2006, 2006, pp. 138–143. [Online]. Available: <http://www.cs.colostate.edu/~malaiya/pub/CNIS-547-097.pdf>
- [4] M. Dowd, J. McDonald, and J. Schuh, The Art of Software Security Assessment: Identifying and Preventing Software Vulnerabilities. Addison-Wesley, 2006.
- [5] W. Jimenez, A. Mammar, and A. Cavalli, "Software Vulnerabilities Prevention and Detection Methods: A Review," in Procs. SEC-MDA 2009. CTIT Workshop Procs. Series, Jun 2009, pp. 6–13.
- [6] [Online] Available: <http://www.utwente.nl/ewi/ecmda2009/workshops/ECMDA2009-SEC-MDA.pdf>
- [7] IEEE STD 1028-2008, "IEEE Standard for Software Reviews and Audits," IEEE STD 1028-2008, pp. 1–52, 2008.
- [8] OWASP, "OWASP 2014 Top Ten Mobile Risks," [Online; accessed at January 28, 2014], January 2014, https://www.owasp.org/index.php/OWASP_Mobile_Security_Project.
- [9] International Data Corporation, "Smartphone OS Market Share, Q32014," [Online; accessed at November 28, 2014], 2014, <http://www.idc.com/prodserv/smartphone-os-market-share.jsp>.
- [10] M. G. Cimino and F. Marcelloni, "An efficient model-based methodology for developing device-independent mobile applications," J. Syst. Architect., vol. 58, no. 8
- [11] Y.-J. Jeong, J.-H. Lee, and G.-S. Shin, "Development Process of Mobile Application SW Based on Agile Methodology," in Procs. of ICACT
- [12] M. Lindorfer, M. Neugschwandtner, L. Weichselbaum, Y. Fratantonio, V. van der Veen, and C. Platzer, "ANDRUBIS - 1,000,000 Apps Later: A View on Current Android Malware Behaviors," in Procs. BADGERS.
- [13] S. Fahl, M. Harbach, T. Muders, L. Baumgartner, B. Freisleben, and M. Smith, "Why Eve and Mallory Love Android: An Analysis of Android SSL (in)Security," in Procs