World Journal of Advanced Research and Reviews
eISSN: 2581-9615 CODEN (USA): WJARAI
Cross Ref DOI: 10.30574/wjarr
Journal homepage: https://wjarr.com/

(RESEARCH ARTICLE)

Developing a framework for enhancing security testing of android applications

Adedeji Olaniyi Lamina [1, *], Moshood Folawiyo Yussuf [2], Toyosi Oyinloye [3], Pelumi Oladokun [4] and Victor Kamalu Brown [5]

[1] *Department of Computer Science, School of Computer and Engineering Sciences, Faculty of Science, Business and Enterprise, University of Chester, Chester, Cheshire, UK*.*
[2] *Department of Decision Science, Faculty of Economics and Decision Sciences, Western Illinois University, Macomb, IL, USA.*
[3] *Department of Computer Science, School of Computer and Engineering Sciences, Faculty of Science, Business and Enterprise, University of Chester, Chester, Cheshire, UK.*
[4] *Department of Computer Science, College of Business and Computing, Southeast Missouri State University, MO, United States. Deep Learning/Artificial Intelligence Engineer.*
[5] *Department of Computer Science and Digital Technologies, School of Architecture, Computing and Engineering, University of East London, London, London, UK.*

World Journal of Advanced Research and Reviews, 2024, 23(02), 2585–2598

Publication history: Received on 17 July 2024; revised on 25 August 2024; accepted on 28 August 2024

Article DOI: https://doi.org/10.30574/wjarr.2024.23.2.2588

Abstract

Mobile applications have advanced a lot and now offer several features that help make our lives easier. Android is currently the most popular mobile operating system, and it is susceptible to exploitation attempts by malicious entities. This has led to an increased focus on the security of Android applications.

This dissertation proposed the development of a framework which provides a systematic approach to testing the security of Android applications. This framework was developed based on a comprehensive review of existing security testing methodologies and tools.

In achieving the study objectives, a test application was run on an emulator, Burp Suite was used as a proxy tool to capture HTTP and HTTPS traffic for analysis, reverse engineering was carried out, static and dynamic analysis were executed, network traffic was captured and analysed with tcpdump and Wireshark, intent sniffing was carried out, fuzz testing was discussed, and a proof-of-concept tool (automation script) was developed.

This work covers various aspects of Android applications' security testing, and the proposed framework provides developers with a practical and effective approach to testing the security of their Android applications, thereby improving the overall security of the Android application ecosystem.

Keywords: Penetration Testing; Cybersecurity; Framework; Security Testing; Android Application; Cyber-attack

1. Introduction

In recent years, the digital world has experienced a big change in focus towards mobile applications (Spencer, 2018). Mobile apps offer useful features and make our lives easier. They have advanced a lot and can now do several things like socialising, tracking fitness, providing entertainment, and even conducting business transactions (Spencer, 2018). Our phones have become our go-to devices for almost everything we need.

* Corresponding author: Adedeji Olaniyi Lamina

With the widespread use of mobile devices and applications, there has been an increased focus on the security of mobile applications. Android, being the most widely used mobile operating system as shown below in Figure 1, coupled with the knowledge of the simplified nature of its applications' approval process, has become more susceptible to exploitation attempts by hackers and cyber pirates (Spencer, 2018). Consequently, the need for effective security testing of Android applications has become increasingly critical.

Operating System	Quarterly Market Share (%)
Android	71.63
iOS	27.71
Samsung	0.35
KaiOS	0.12
Windows	0.02
Linux	0.01
Other	0.15

Figure 1 June 2023 Mobile Operating Systems Market Share (Turner, 2023)

A vulnerable application, if exploited, can expose users' private data such as account details, names, email addresses and so on, which can be used for malicious purposes. For example, MyFitnessPal, a popular mobile app for tracking diet and exercise owned by the Under Armour brand, experienced a data breach that exposed the usernames, email addresses, and hashed passwords of around 150 million users in 2018 (M. Baker, 2021). Attackers exploited a flaw in the data storage technology of the application. This opened Under Armour to an investigation for breaching the Data Protection Act and many users attempted to make data breach compensation claims (M. Baker, 2021).

Existing research shows that Android applications are vulnerable to various security threats, which include privacy-related vulnerabilities (Montealegre et al., 2018), much of which is due to hard-coded credentials and un-encrypted data transmission, flooding attacks (Naresh & Muhammad, 2011) which exploits the Open Systems Interconnection (OSI) layers, and the infiltration of malicious software. While there are a number of security testing tools and methodologies, they often lack the cross-compatibility and effectiveness required to address unique challenges posed by the Android platform. Moreover, according to Naresh and Muhammad (2011), numerous developers do not possess the expertise and necessary means to adequately assess the security of their applications, potentially leading to the deployment of a large number of potentially vulnerable apps into the ecosystem.

This study aims to address the following research questions: “What are the current security challenges Android mobile applications face?” and “What are the best practices in designing a security testing process for Android applications?”

The specific objectives of this research are as follows:

- Conduct a thorough review of existing security testing methodologies for Android applications. This will include analysing their strengths, weaknesses, and limitations.
- Develop a framework that encompasses various aspects of security testing, based on the comprehensive review of existing methodologies and tools, for efficient penetration testing of Android applications.
- Ensure the framework's effectiveness and adaptability to different application architectures.
- Validate the efficacy of the proposed framework through the development of a penetration testing tool. The tool, which will be a Python script, will be developed to assess the capabilities of the proposed framework.
- Automate aspects of Android application security testing with the developed tool, which might help save testing time and have increased compatibility with different Android architectures.

The proposed Framework aims to improve the overall security of the Android application ecosystem by providing developers with a practical and effective approach to testing the security of their applications. The proof-of-concept tool will also contribute to the existing body of knowledge on mobile application security testing, by automating aspects of the Android security testing process, which might help save security testing time.

2. Material and methods

The systematic approach employed to address the research objectives is described in this section. This section provides a comprehensive overview of the overall framework for conducting the study, data collection strategy, analysis tools, and research design.

2.1. Research Design

The research design employs a mixed-methods approach, integrating both qualitative and quantitative elements. The qualitative component is aimed at understanding current security testing methodologies and identifying associated challenges, while the quantitative component focuses on evaluating the effectiveness of the newly developed framework.

2.2. Framework Development

The security testing framework was developed using insights gathered from the qualitative phase of the study, as well as by addressing gaps identified in the existing literature. The framework follows a step-by-step approach to security testing and incorporates a comprehensive range of techniques, including dynamic analysis, static analysis, and fuzz testing. It is designed to be adaptable to various Android application architectures, ensuring broad applicability and effectiveness across different software environments.

2.3. Framework Validation Approach

The framework was validated through a step-by-step security testing process conducted on a deliberately vulnerable Android application. First, Android Studio was installed, and an Android emulator was run within the studio environment. The vulnerable application, InsecureShop, was then installed on the emulated device. Manual security tests were performed on InsecureShop using the developed framework. To enhance the testing process, a Python automation script was created to automate the security testing steps. The outcomes of these tests were thoroughly documented, demonstrating the effectiveness of the framework in identifying vulnerabilities within the application.

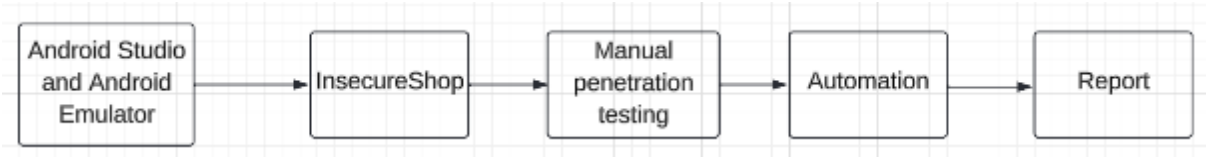

Figure 2 Framework Validation Approach

2.4. Ethical Considerations

The security testing was conducted using InsecureShop, an application intentionally built to be vulnerable for practicing security testing techniques and exploiting vulnerabilities, with its copyright license permitting unrestricted use and modification (Optiv, n.d.). The application does not contain any Personal Identifiable Information (PII), thereby eliminating any risk of breaching confidentiality and data protection during the research. The tests were carried out in a controlled environment, where Android Studio and an emulator were run on a Windows machine completely isolated from other systems. Additionally, when the testing was conducted on a Linux virtual machine, it was ensured that this machine was fully isolated to maintain a controlled testing environment.

Limitations of Study

The resources for this research were limited, as only a few related previous works were freely available. Additionally, attempts to run an Android Emulator on a Kali Linux machine resulted in CPU compatibility errors, restricting all tests to a Windows environment. Moreover, the analysis procedure was time-intensive, and the duration allocated for this work was insufficient to fully develop all the necessary skills.

2.5. Implementation of the Framework

The security testing framework for Android applications combines various tools and processes to ensure thorough assessment, benefiting both experienced and inexperienced developers. The testing environment involves using an emulator, Android Studio, and tools like Burp Suite for intercepting HTTP/HTTPS traffic. Reverse engineering is conducted using Jadx and APKTool to identify vulnerabilities in the source code. Static and dynamic analyses are performed using the Mobile-Security-Framework (MobSF), generating detailed security reports. Network traffic is

monitored with tcpdump and Wireshark, while Logcat aids in intent sniffing tests. Additional considerations include using tools like Qark for static analysis, Drozer for dynamic analysis, and bypassing SSL pinning to test vulnerabilities effectively. The framework also aligns with OWASP mobile top 10 vulnerabilities for comprehensive security evaluation.

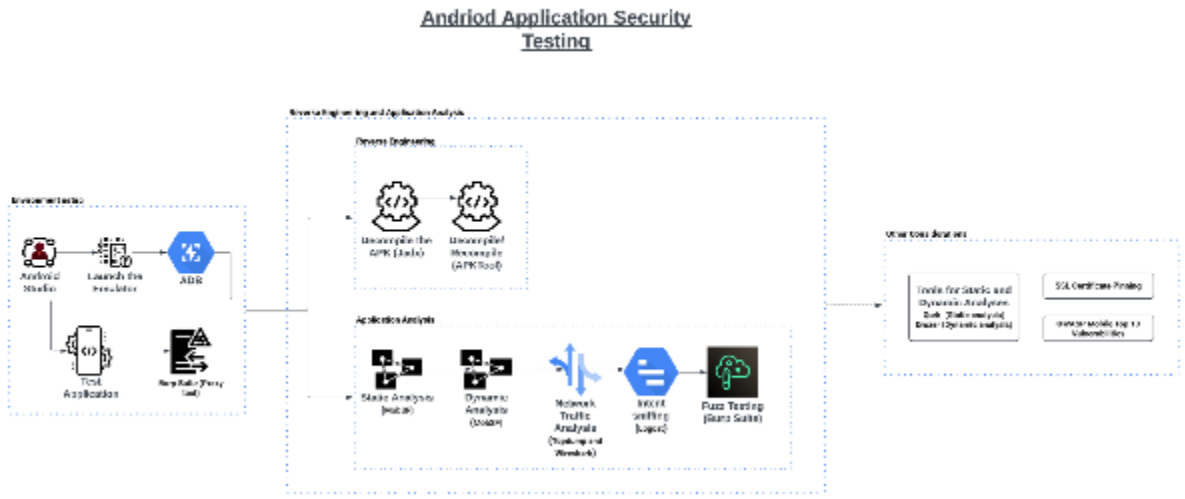

Figure 3 The Framework

2.6. Implementation of the Automation Tool

The proof-of-concept tool is an automation script written in Python to automate aspects of the security testing process for Android applications, thereby saving time. The script requires several dependencies, including Android Studio, an emulator, adb, Jadx, APKTool, Docker, tcpdump, Wireshark, and Burp Suite, all of which must be installed on the host machine before running the tool. The script does not perform security testing directly but automates the execution of these tools. It checks if necessary components are running, installs and decompiles APK files, sets up MobSF for analysis, captures network traffic, and opens it in Wireshark. The script also integrates Burp Suite for further testing. While not yet fully optimized, the tool demonstrates the potential for automation in security testing and suggests areas for future improvement.

```
def check_docker_installation():
    try:
        # Attempt to run the Docker version command
        subprocess.run("docker --version", stdout=subprocess.PIPE, stderr=subprocess.PIPE, shell=True, check=True)
        return True
    except subprocess.CalledProcessError:
        return False

def check_mobsf_docker_image():
    try:
        # Check if the MobSF Docker image is available locally
        subprocess.run("docker image inspect opensecurity/mobile-security-framework-mobsf:latest", shell=True, check=True)
        return True
    except subprocess.CalledProcessError:
        return False

def pull_mobsf_docker_image():
    if not check_mobsf_docker_image():
        try:
            # Pull the MobSF Docker image if it's not available locally
            subprocess.run("docker pull opensecurity/mobile-security-framework-mobsf", shell=True, check=True)
            print("MobSF Docker image pulled successfully.")
        except subprocess.CalledProcessError as e:
            print(f"Error: {e}")
            print("Failed to pull the MobSF Docker image.")
    else:
        print("MobSF Docker image is already available locally.")

def run_mobsf_docker_container():
    try:
        # Run MobSF in a Docker container
        subprocess.run("docker run -it --rm -p 8000:8000 opensecurity/mobile-security-framework-mobsf:latest", shell=True, check=True)
        print("MobSF Docker container started successfully.")
    except subprocess.CalledProcessError as e:
        print(f"Error: {e}")
        print("Failed to start MobSF Docker container.")

def open_mobsf_web_interface():
    # Open MobSF web interface in a web browser
    webbrowser.open('http://localhost:8000')
    print("MobSF web interface is running at http://localhost:8000.")
    print("Upload the APK for static analysis.")
    print("For dynamic analysis, refer to https://mobsf.github.io/docs/#/dynamic_analyzer?id=android-studio-emulator for instructions.")
```

Figure 4 Automation tool code snippet

3. Results and discussion

This chapter discusses the security tests carried out on the InsecureShop Android application, the results obtained from the various tests, and other security concepts.

3.1. Setting up the Environment

3.2. Android Studio

To begin testing, an Android environment needed to be set up. Android Studio was installed and set-up, up to create the development environment needed to test an application on a virtual Android device.

3.2.1. Android Emulator

Primarily, two Android mobile devices were simulated using the emulator and the InsecureShop application was installed and tested on both devices. Other architectural dependencies for some tools required setting up specific emulators later in the process.

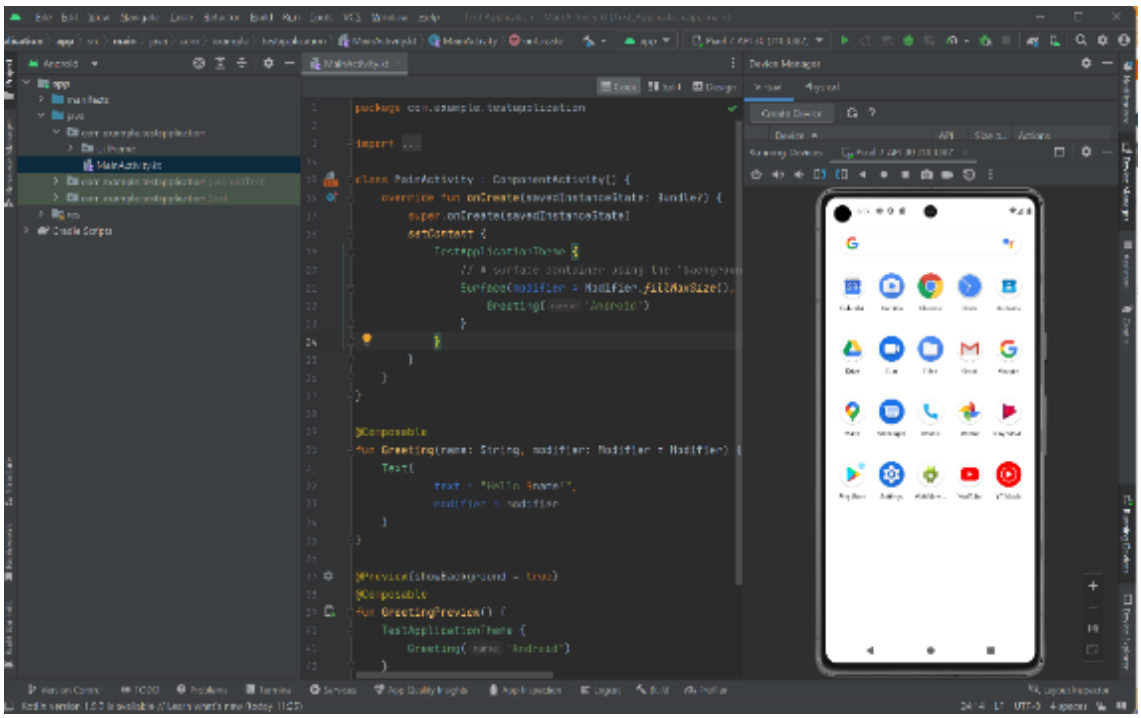

Figure 5 Android emulator - Google Pixel 7

Figure 5 shows an emulated Google Pixel 7 while Figure 6 shows an emulated Google Pixel 3a.

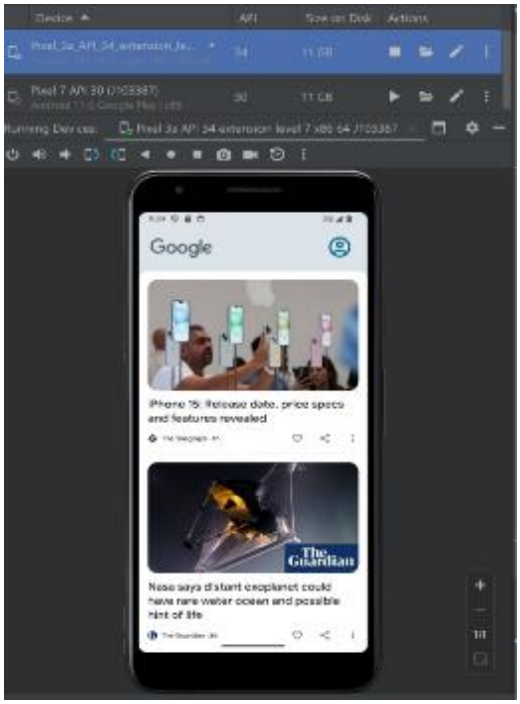

Figure 6 Android emulator - Google Pixel 3a

The emulated devices and executed tests were run only on a Windows machine, as opposed to an initial plan to run the same devices and tests on both a Windows machine and a Linux machine. This is because the Linux machine returned compatibility errors upon running an emulator.

3.2.2. Android Debug Bridge (adb)

According to Android Developers (n.d.), Android Debug Bridge (adb) is a flexible command-line utility that enables communication with a device. By default, adb is integrated with the Android studio as one of the software development kit (SDK) platform tools and can be activated by adding the SDK platform tools path to the system's environment variables.

3.2.3. InsecureShop

The InsecureShop application, a vulnerable Android app, was used for testing. The application was installed on an emulated device using adb, and the subsequent analysis reported was consistent across all tested devices.

3.2.4. Burp Suite

Burp Suite is a cybersecurity software tool used for conducting penetration tests on web applications. In penetration testing, traffic can be analysed to discover vulnerabilities. It was used here as a proxy to intercept and monitor emulator traffic. The setup involved configuring Burp Suite to listen on a proxy port, using the local machine's IP, and handling untrusted certificates by installing a Burp CA certificate on the emulated device. Post-installation, Burp Suite successfully intercepted traffic from the emulator's browser.

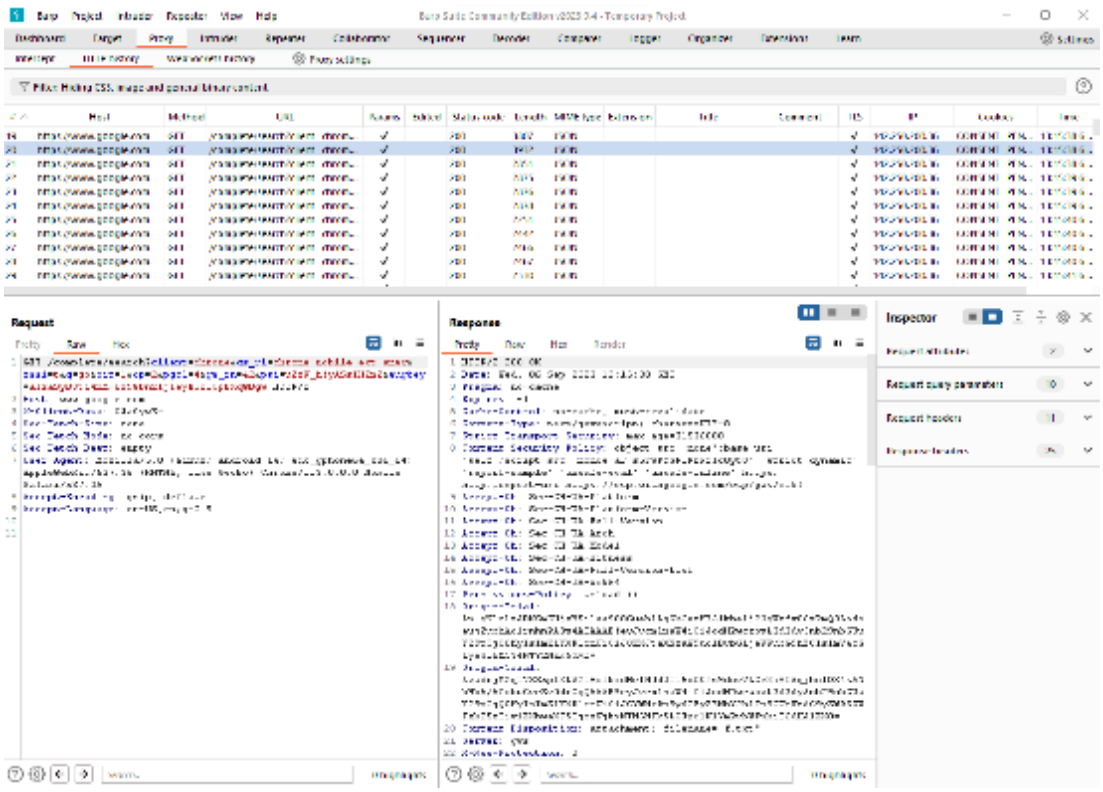

Figure 7 Burp Suite - packet capture

3.3. Reverse Engineering and Application Analysis

The essence of reverse engineering involves extracting information from the existing software elements and transforming it into abstract forms that are more comprehensible to humans (Canfora & Di Penta, 2007). Application analysis involves examining software applications to gain insights into their attributes such as behaviour, security, and structure.

3.3.1. Decompiling with Jadx

Jadx contains command line and graphical user interface (GUI) tools that help produce the Java source code from Android APK and Dex files (Skylot, n.d.). Here, it was used to view the source code of the InsecureShop application and a manual source code analysis was carried out. The analysis revealed vulnerabilities such as hardcoded credentials and inadequate URL and host validation, which could be exploited by malicious entities.

3.3.2. Decompiling and Re-compiling Using APKTool

APKTool is a reengineering tool that can disassemble resources into an almost original state and reconstruct them after making some adjustments (Apktool | Kali Linux Tools, n.d.). These adjustments could be to change the behaviour of the application in any way desired. It does not access the Java files. Instead, it accesses the smali files which are assembly language files. It was used to modify the InsecureShop application. After making changes to the app's behavior, it was recompiled and signed, allowing it to be run on an Android device.

3.3.3. Static and Dynamic Analysis with Mobile-Security-Framework (MobSF)

Mobile Security Framework (MobSF) is a comprehensive, automated framework designed for conducting security assessments, malware analysis, and penetration testing of mobile applications on Android, iOS, and Windows platforms (MobSF, n.d.). It can be used to perform both static and dynamic analysis. MobSF was used for both static and dynamic analysis of the InsecureShop application. The static analysis involved examining the app's APK without running it, while dynamic analysis tested the app in real-time on an emulator.

3.3.4. Dynamic Analysis Using Drozer

One of the advantages of using an emulator environment such as Android Studio is the flexibility to emulate different devices for specific purposes. This can be seen in the more detailed report for this work showing the dynamic analysis process using MobSF. However, if such flexibility is not available during the testing process, e.g. testing with a physical device, a tool such as Drozer is very sufficient for dynamic analysis.

Drozer enables the discovery of security vulnerabilities in applications and devices by acting as an application and communicating with the Dalvik VM, IPC endpoints of other apps, and the core operating system (WithSecure Labs, n.d.).

In a more robust report for this work, Drozer was introduced as an alternative for dynamic analysis. While it wasn't demonstrated, its setup and basic requirements were outlined, highlighting its use in environments where an emulator like Android Studio isn't available.

3.3.5. Tcpdump and Wireshark for Intercepting Traffic

Tcpdump is a command line tool with which network traffic going through a system can be captured and analysed (Gerardi, 2020). According to Elahmad (2023), the major difference between Wireshark and Burp Suite is that Wireshark is intended for network protocol analysis while Burp Suite is built specifically for web application security testing. Wireshark enables users to collect and analyse network traffic from all protocols and applications while Burp Suite focuses on the examination of HTTP/S traffic, intercepting and modifying requests and responses to detect vulnerabilities (Elahmad, 2023).

Tcpdump was used on an ARM-emulated device to record traffic. The recorded traffic was saved as a ‘capture.pcap’ file which was later analyzed using Wireshark. This analysis focused on identifying potential data exposure in the captured network traffic.

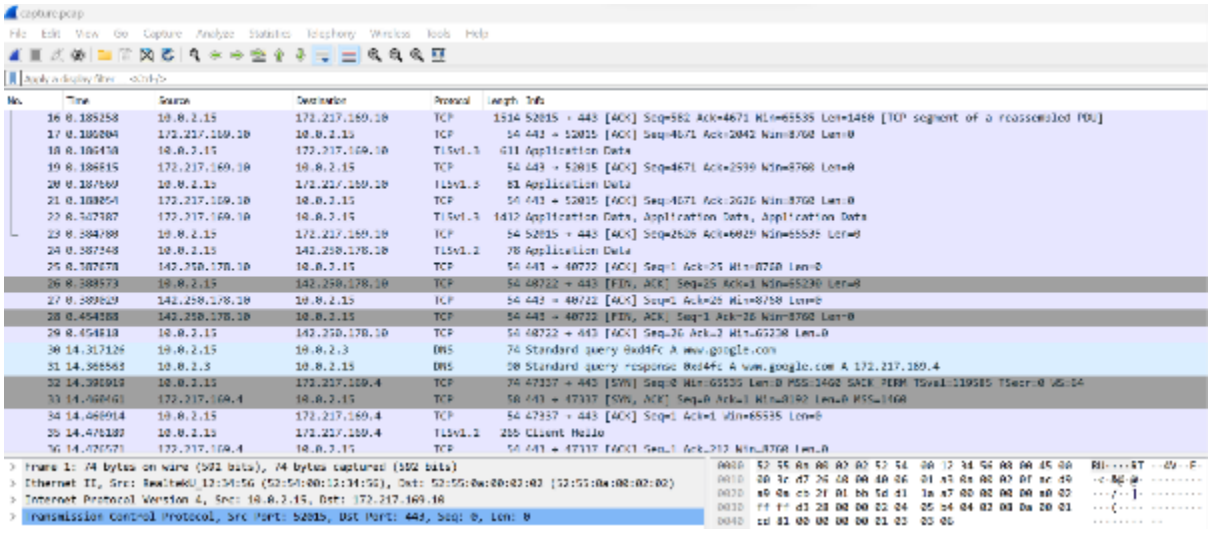

Figure 8 Capture.pcap file on Wireshark

When analysing with Wireshark, specific strings of packet bytes can be searched for, to narrow down suspected or possible exposed data. For example, while analysing, a search for keywords such as "user", "password", "username", "secret" can be executed.

3.3.6. Intent Sniffing

According to Salva and Zafimiharisoa (2013), Intents represent a form of messages that is composed of actions and data dispatched from one component to another with the purpose of executing several operations, such as initiating a user interface. By analysing intents sent and received by applications, issues such as data leakage and improper handling of sensitive information can be uncovered.

Figure 9 below shows a part of the codes contained in “AboutUsActivity” within “Com.InsecureShop”, as viewed with jadx.

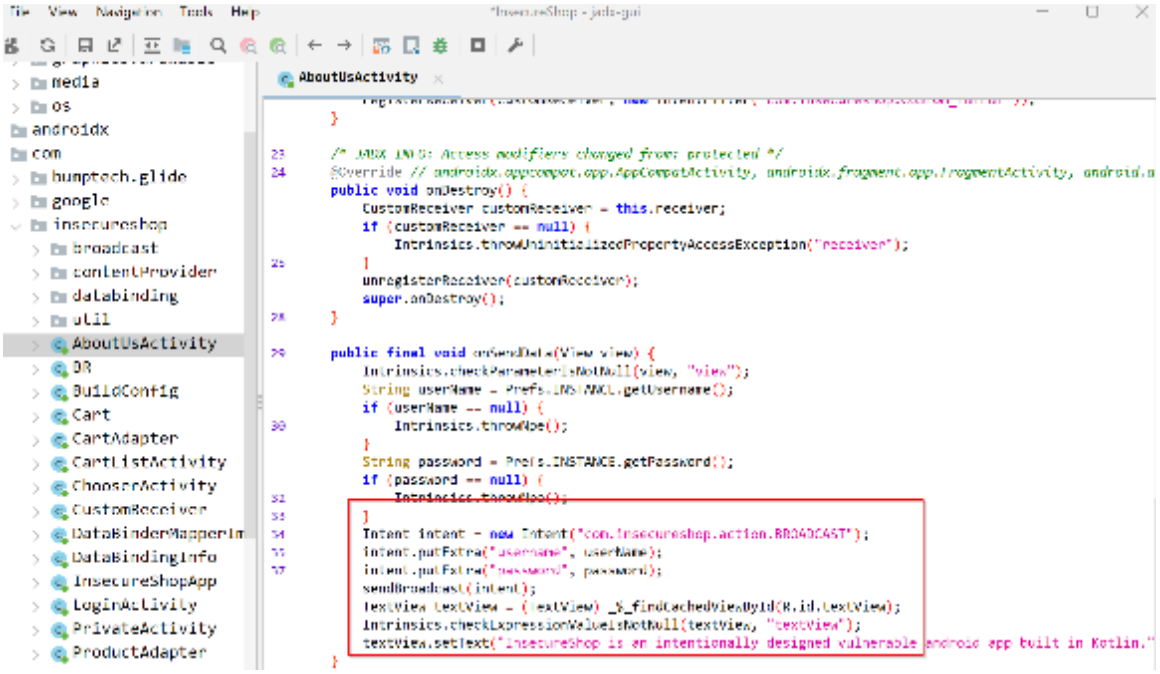

Figure 9 Jadx - broadcast action

The highlighted set of codes is seen to define an action "com.insecureshop.action.BROADCAST", and it sends a broadcast that contains the username and password of the logged-in user, using implicit intent.

To listen for and analyse these intents, the InsecureShop application is run on the emulator, the About Us page is viewed, and the log is monitored real-time using logcat on the Android Studio.

Figure 10 shows the username of the logged in user captured on logcat while accessing the "About InsecureShop" page of the application on the emulator, while Figure 11 shows the password of the logged in user captured under same circumstances.

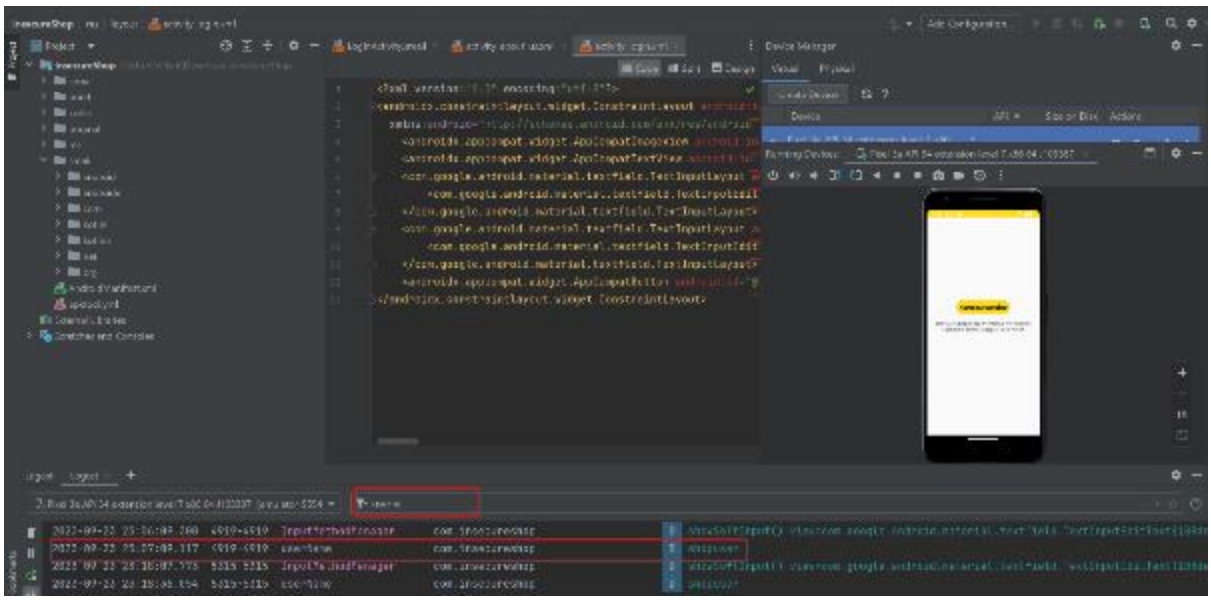

Figure 10 Intent sniffing – username

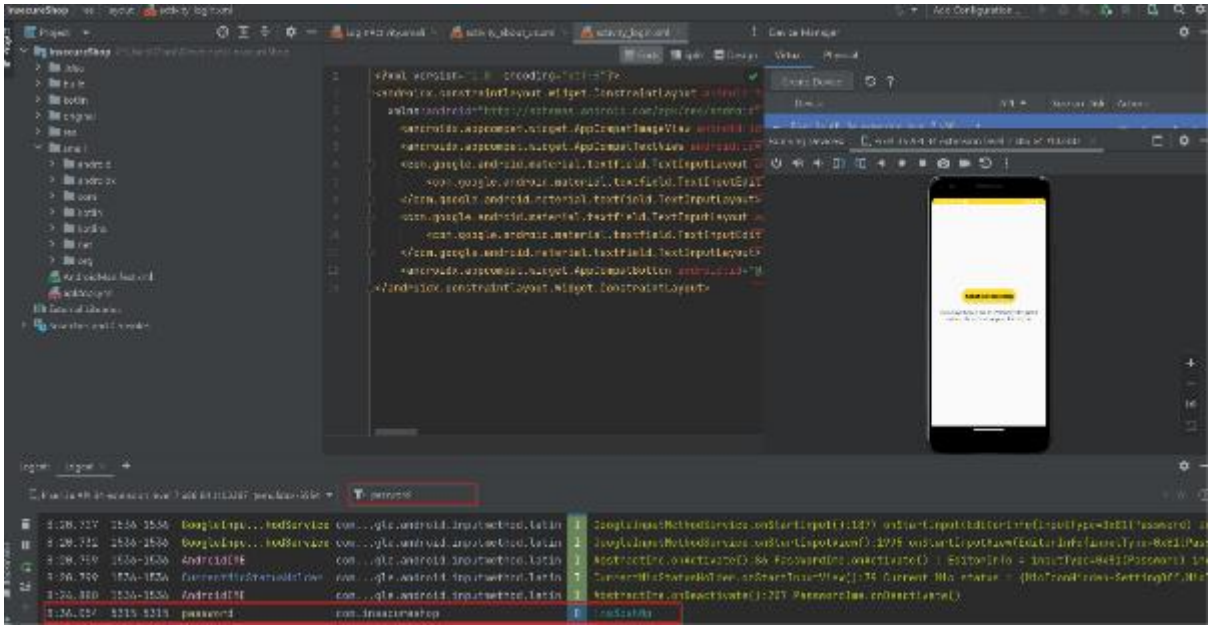

Figure 11 Intent sniffing - password

As seen in both Figure 10 and Figure 11, keyword searches can be made on logcat to view specific logs that might contain key information based on such key words.

Another way to capture and analyse intents sent and received by specific applications is to install an application built to listen for intents on the Android device or emulator. With such an application installed, as soon as the application being analysed is running and intents are exchanged, the intent sniffing application listens and stores or displays such intents.

3.3.7. Fuzz Testing

Fuzzing is a software testing approach that subjects a target system to a barrage of generated test cases (Zhao & Pang, 2018). This is done with the aim to reveal software vulnerabilities and defects. The test cases are usually invalid or unexpected inputs.

Burp Suite can be used for fuzz testing, especially in terms of username enumeration and brute-forcing passwords. This concept is not demonstrated in this report. However, the use of burp suite for this concept is summarised below.

Firstly, ensure burp suite is properly configured as a proxy between the application to be tested and the server.

After achieving this configured environment, HTTP/S requests from the application being tested can be intercepted by burp suite and this forms the basis for fuzz testing. For example, without knowing the right username and password, a log in attempt can be made to see the response returned by the application. Burp Suite has an inbuilt tool called "intruder" which can be used to run a list of generated usernames against a password and display the application's response to each of these attempts. This is then monitored for responses that can indicate that a username is valid, such as "wrong password to user 'shopuser'". If such a response is gotten and a valid username is recognised, the same tool can then be used to brute force a list of passwords against such a valid username. Lists of usernames and passwords can be downloaded off the internet for such tests.

Using Burp Suite, the intruder tool can also be used to fuzz test for various other vulnerabilities e.g. SQL injection. The basic idea is inputting various data into different fields within the application, to check for responses that can show vulnerabilities.

3.4. Other Security Concepts

While testing Android applications for security vulnerabilities, there are a few other concepts to note. These concepts are described here.

3.4.1. SSL (Secure Socket Layer) Certificate Validation and Pinning

SSL Certificate Pinning is a process that associates a host with its public key or certificate (Rai, 2022). It associates a specific SSL certificate with a particular server or website in a client's trusted certificate store. This ensures that the client only communicates with the server if it presents the expected, pinned certificate. It does this by checking if the hash of the server certificate is the same as the one hardcoded in between the application (Rai, 2022). This helps prevent man-in-the-middle attacks by making it difficult for attackers to intercept and impersonate the server with a different certificate.

If an application utilises SSL certificate pinning, it is difficult to discover a number of potential vulnerabilities when performing security tests, as decrypting and analysing the traffic generated by the application becomes difficult. For example, intercepting the HTTP/HTTPS traffic and fuzz testing earlier described using Burp Suite would have been difficult if the application implemented certificate pinning, as the burp certificate used would have been flagged as invalid.

To resolve this difficulty in a scenario where pinning is implemented, a certificate pinning bypass is carried out. Some tools such as MobSF automatically bypass certificate pinning when used for application analysis. However, there are cases where more effort is required to bypass certificate pinning, and this involves reverse engineering the application, editing the codes to override certificate pinning, and then recompiling the edited application for analysis.

Figure 12 below shows a piece of code within the "CustomWebViewClient" of the Webview class source code of the InsecureShop application. The code ignores SSLError and continues with the certificate provided by the attacker, exposing the app to potential Man-in-the-Middle (MITM) attacks.

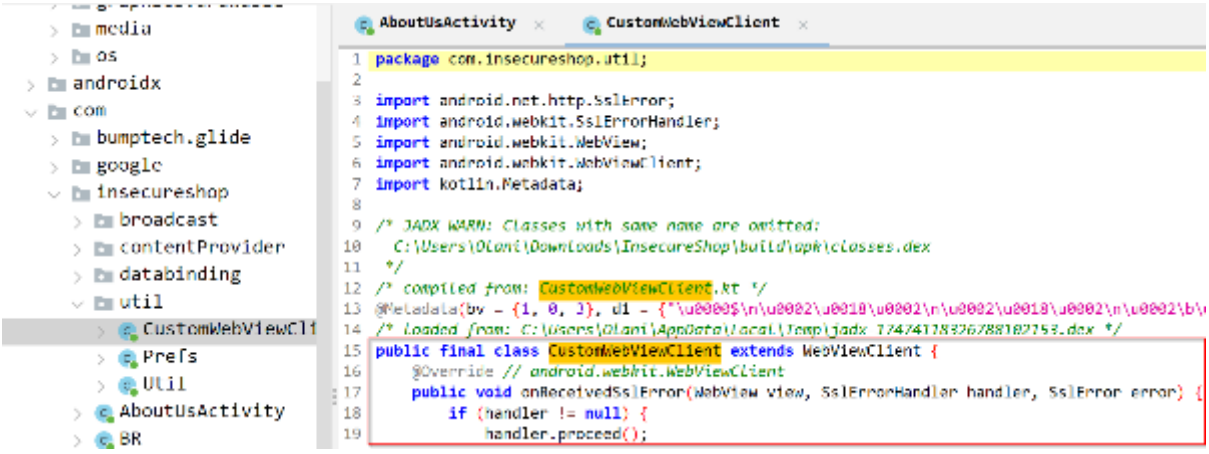

Figure 12 InsecureShop - improper handling of SSL error

3.4.2. OWASP Mobile Top 10 Vulnerabilities

"The Open Worldwide Application Security Project (OWASP) is a nonprofit foundation that works to improve the security of software." (OWASP, n.d.). The OWASP Mobile Top 10 is a list of the ten most critical security risks for mobile applications. It highlights common vulnerabilities and threats that developers and organisations should be aware of when designing, developing, and testing mobile applications. To generate this list, OWASP gathers reliable and consistent data from diverse sources such as databases of vulnerabilities, incident reports, and security assessments, and the vulnerabilities are then ranked based on their impact and likelihood of occurrence (OWASP, n.d.).

When testing and analysing mobile applications for security vulnerabilities, the OWASP mobile top 10 vulnerabilities should be considered. The OWASP mobile top 10 vulnerabilities can be found on the OWASP official website.

3.5. Summary

This chapter elaborates the security testing process and tools based on the earlier defined framework. This process: uses Burp Suite to intercept HTTP and HTTPS traffic from the emulator and to fuzz-test an application; reverse engineers an application to get the source codes and assembly language files; carries out robust security analysis (static and dynamic); runs network traffic analysis with tcpdump and Wireshark; and tests an application for data leakage through intents. The architectural cross-compatibility of the framework is limited, as some tools require specific

architectures to function properly. Overall, this process achieves detailed security tests, utilising publicly available tools for various aspects of the security testing process.

4. Conclusion

4.1. Study Summary

In a time where mobile applications have become an important part of our daily lives, ensuring their security is of utmost importance. Considering the popularity of the Android mobile operating system, this dissertation focuses on the security testing of Android applications, with the aim to develop a well-structured step-by-step Android penetration testing process that improves the way the security of Android applications are tested.

A comprehensive literature review was conducted in order to gain an understanding of the current security testing methodologies, tools, and best practices for Android applications.

In carrying out this study, specific objectives were stated and followed. However, one of the objectives which was to ensure the framework's adaptability to different application architectures was not achieved. As stated in the tests and results chapter of this dissertation, MobSF requires a specific range of Android versions and API structure to run dynamic analysis on an application and Tcpdump requires an ARM Android architecture for it to run and capture network traffic.

Overall, through a combination of tools, from APKTool for decompiling, to MobSF for static and dynamic analysis, and the integration of Wireshark for network traffic inspection, the framework proved its versatility in covering various aspects of Android applications' security testing and provided a holistic view of an application's security posture.

4.2. Future Work

While this dissertation has described and demonstrated a process for testing the overall security of Android applications, it does have some limitations, which provide opportunities for future research.

In executing the tests, this dissertation primarily utilised a Windows based host machine, limiting the tests to a Windows host environment. Though the tools utilised should remain effective regardless of the host operating system, the commands run to utilise these tools might slightly differ and dependencies might change. Future work should consider other operating systems (OS) such as the Mac OS and the Linux OS as test environments, alongside the Windows OS.

Some tools incorporated in this dissertation's framework are limited in terms of the Android architectures on which they can run. Future research can be done to improve this by implementing a process with little or no architectural dependencies, thereby eliminating environment-based restrictions.

Also, the automation tool (python script) developed in this work shows that automating various aspects of the penetration testing process is feasible. However, the tool has a lot of dependencies, and it is not optimal. Future work can be done to optimise the tool and reduce its dependencies.

Compliance with ethical standards

Disclosure of conflict of interest

The authors declare no conflicts of Interest.

References

[1] Aditya, C., Kurniawan, Y., & Rhee, K. (2016). Security analysis testing for secure instant messaging in android with study case: Telegram. *IEEE International Conference on System Engineering and Technology (ICSET)*. https://doi.org/10.1109/icsengt.2016.7849630

[2] Amin, A., Eldessouki, A., Magdy, M. T., Abdeen, N., Hindy, H., & Hegazy, I. (2019). AndroShield: Automated Android Applications vulnerability detection, a hybrid static and dynamic analysis approach. *Information*, *10*(10), 326. https://doi.org/10.3390/info10100326

[3] *Android architecture - eLinux.org*. (n.d.). https://elinux.org/Android_Architecture

[4] Android Developers. (n.d.). *Android Debug Bridge (adb)*. https://developer.android.com/tools/adb

[5] *apktool | Kali Linux Tools*. (n.d.). Kali Linux. https://www.kali.org/tools/apktool/#:~:text=A%20tool%20for%20reverse%20engineering,smali%20code%20step%20by%20step.

[6] Avancini, A., & Ceccato, M. (2013). Security testing of the communication among Android applications. *IEEE 8th International Workshop on Automation of Software Test (AST)*. https://doi.org/10.1109/iwast.2013.6595792

[7] Baker, J. D. (2016). The Purpose, Process, and Methods of Writing a Literature Review. *AORN Journal*, *103*(3), 265–269. https://doi.org/10.1016/j.aorn.2016.01.016

[8] Baker, M. (2021, October 1). *Everything you need to know about the MyFitnessPal data breach - UK Tech News*. UK Tech News. https://uktechnews.co.uk/2021/10/01/everything-you-need-to-know-about-the-myfitnesspal-data-breach/

[9] Bojjagani, S., & Sastry, V. N. (2017). VAPTAi: A Threat Model for Vulnerability Assessment and Penetration Testing of Android and iOS Mobile Banking Apps. *IEEE 3rd International Conference on Collaboration and Internet Computing*. https://doi.org/10.1109/cic.2017.00022

[10] Borja, T., Benalcazar, M. E., Caraguay, Á. L. V., & López, L. I. B. (2021). Risk Analysis and Android Application Penetration Testing Based on OWASP 2016. In *Springer eBooks* (pp. 461–478). https://doi.org/10.1007/978-3-030-68285-9_44

[11] Canfora, G., & Di Penta, M. (2007). New Frontiers of Reverse Engineering. *IEEE*. https://doi.org/10.1109/fose.2007.15

[12] *"Docker overview."* (2023, September 8). Docker Documentation. https://docs.docker.com/get-started/overview/

[13] Elahmad, J. (2023, June 8). Wireshark vs. Burp Suite: Unveiling Network Insights and Web Application Security. *Medium*. https://jadelahmad.medium.com/wireshark-vs-burp-suite-unveiling-network-insights-and-web-application-security-82b8fd044b10

[14] Faruki, P., Bharmal, A., Laxmi, V., Ganmoor, V., Gaur, M. S., Conti, M., & Rajarajan, M. (2015). Android Security: A survey of issues, malware penetration, and defenses. *IEEE Communications Surveys and Tutorials*, *17*(2), 998–1022. https://doi.org/10.1109/comst.2014.2386139

[15] *Fuzzing for vulnerabilities*. (n.d.). PortSwigger. https://portswigger.net/burp/documentation/desktop/tools/intruder/uses/fuzzing

[16] GeeksforGeeks. (2021). Android Architecture. *GeeksforGeeks*. https://www.geeksforgeeks.org/android-architecture/

[17] Gerardi, R. (2020, September 1). *An introduction to using tcpdump at the Linux command line*. Opensource.com. https://opensource.com/article/18/10/introduction-tcpdump

[18] Hubbard, J. P., Weimer, K., & Chen, Y. (2014). A study of SSL Proxy attacks on Android and iOS mobile applications. *IEEE 11th Consumer Communications and Networking Conference (CCNC)*. https://doi.org/10.1109/ccnc.2014.6866553

[19] Hunt, R. (2013). Security testing in Android networks - A practical case study. *IEEE*. https://doi.org/10.1109/icon.2013.6781950

[20] Jing, G., Wei, H., Zi-Ying, W., Chao, Z., & Haitao, J. (2019). Research on Android Component Security Testing Method for Mobile Application in Power System. *IEEE Conference on Industrial Electronics and Applications (ICIEA)*. https://doi.org/10.1109/iciea.2019.8834355

[21] Katoch, S., & Garg, V. (2023). Security Analysis on Android Application Through Penetration Testing using Reverse Engineering. *International Conference on Smart Data Intelligence (ISCSDI) | IEEE*. https://doi.org/10.1109/icsmdi57622.2023.00048

[22] *Kotlin and Android*. (n.d.). Android Developers. https://developer.android.com/kotlin#:~:text=Develop%20Android%20apps%20with%20Kotlin&text=Kotlin%20is%20a%20modern%20statically,developer%20satisfaction%2C%20and%20code%20safety.

[23] Mahmood, R., Esfahani, N., Kacem, T., Mirzaei, N., Malek, S., & Stavrou, A. (2012). A whitebox approach for automated security testing of Android applications on the cloud. In *Automation of Software Test* (pp. 22–28). https://doi.org/10.5555/2663608.2663613

[24] Malek, S., Esfahani, N., Kacem, T., Mahmood, R., Mirzaei, N., & Stavrou, A. (2012). A Framework for Automated Security Testing of Android Applications on the Cloud. *IEEE International Conference on Software Security and Reliability Companion*. https://doi.org/10.1109/sere-c.2012.39

[25] Microsoft. (2022, July 14). *Manually signing the APK - Xamarin*. Microsoft Learn. https://learn.microsoft.com/en-us/xamarin/android/deploy-test/signing/manually-signing-the-apk

[26] *Mobile Security Framework - MOBSF Documentation*. (n.d.). Mobile Security Framework - MobSF Documentation. https://mobsf.github.io/docs/#/dynamic_analyzer?id=android-studio-emulator

[27] MobSF. (n.d.). *GitHub - MobSF/Mobile-Security-Framework-MobSF: Mobile Security Framework (MobSF) is an automated, all-in-one mobile application (Android/iOS/Windows) pen-testing, malware analysis and security assessment framework capable of performing static and dynamic analysis.* GitHub. https://github.com/MobSF/Mobile-Security-Framework-MobSF

[28] Montealegre, C., Njuguna, C., Malik, M. I., Hannay, P., & McAteer, I. N. (2018). Security vulnerabilities in android applications. *In Proceedings of the 16th Australian Information Security Management Conference*, 14–28. https://doi.org/10.25958/5c5274d466691

[29] Naresh, K., & Muhammad, E. U. H. (2011). *Penetration Testing of Android-based Smartphones*. http://publications.lib.chalmers.se/records/fulltext/146812.pdf

[30] OWASP. (n.d.). *About the OWASP Foundation | OWASP Foundation*. https://owasp.org/about/

[31] Palma, F., Realista, N., Serrão, C., Nunes, L., Oliveira, J., & Almeida, A. (2020). Automated security testing of Android applications for secure mobile development. *IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. https://doi.org/10.1109/icstw50294.2020.00046

[32] *psutil*. (2023, April 17). PyPI. https://pypi.org/project/psutil/

[33] Rai, A. (2022, January 6). Everything you need to know about SSL Pinning - Anuj Rai - Medium. *Medium*. https://medium.com/@anuj.rai2489/ssl-pinning-254fa8ca2109#:~:text=SSL%20Certificate%20Pinning%2C%20or%20pinning,pin%20it%20to%20that%20host.

[34] RedHat. (2022, July 21). *What is an ARM processor?* https://www.redhat.com/en/topics/linux/what-is-arm-processor#:~:text=ARM%20processors%20are%20a%20family,stands%20for%20Advanced%20RISC%20Machine.

[35] Salva, S., & Zafimiharisoa, S. R. (2013). Data vulnerability detection by security testing for Android applications. *IEEE*. https://doi.org/10.1109/issa.2013.6641043

[36] Skylot. (n.d.). *GitHub - skylot/jadx: Dex to Java decompiler*. GitHub. https://github.com/skylot/jadx

[37] Spencer, W. (2018, April 18). *HokuApps Helped Roofing Southwest to Digitize Their Operations* [Video]. HokuApps. https://www.hokuapps.com/blogs/android-app-development-security-risks-what-you-should-know/

[38] Turner, A. (2023, June 5). Android vs. Apple Market Share: Leading Mobile OS (2023). *BankMyCell*. https://www.bankmycell.com/blog/android-vs-apple-market-share/

[39] WithSecureLabs. (n.d.). *GitHub - WithSecureLabs/drozer: The Leading Security Assessment Framework for Android.* GitHub. https://github.com/WithSecureLabs/drozer#windows

[40] Yang, Y., Cai, L., & Zhang, Y. (2016). Research on non-authorized privilege escalation detection of android applications. *IEEE*. https://doi.org/10.1109/snpd.2016.7515959

[41] Zafimiharisoa, S. R., Salva, S., & Laurençot, P. (2012). An Automatic Security Testing approach of Android Applications. *International Conference on Software Engineering Advances.*

[42] Zhao, J., & Pang, L. (2018). Automated Fuzz Generators for High-Coverage Tests Based on Program Branch Predications. *IEEE*. https://doi.org/10.1109/dsc.2018.00082