

Ministry of Education and Science of Ukraine
Ternopil Ivan Puluj National Technical University

Faculty of Computer Information System and Software Engineering

(full name of faculty)

Department of Computer Science

(full name of department)

QUALIFYING PAPER

For the degree of

Bachelor

(degree name)

topic: **Information system for analyzing mobile application vulnerabilities**

Submitted by: student **IV** course, group **ICH-42**

specialty **122 Computer science**

(шифр і назва спеціальності)

Stanley Onyedikachi
Umesie

(signature)

(surname and initials)

Supervisor

Holotenko O.S.

(signature)

(surname and initials)

Standards verified by

(signature)

(surname and initials)

Head of Department

Bodnarchuk I.O.

(signature)

(surname and initials)

Reviewer

(signature)

(surname and initials)

Ternopil
2025

Ministry of Education and Science of Ukraine
Ternopil Ivan Puluj National Technical University

Faculty Faculty of Computer Information System and Software Engineering
(full name of faculty)

Department Department of Computer Science
(full name of department)

APPROVED BY

Head of Department

Bodnarchuk I.O.

(signature)

(surname and initials)

« »

2025

ASSIGNMENT
for QUALIFYING PAPER

for the degree of Bachelor
(degree name)

specialty 122 Computer science
(code and name of the specialty)

student Stanley Onyedikachi Umesie
(surname, name, patronymic)

1. Paper topic Information system for analyzing mobile application vulnerabilities

Paper supervisor Holotenko O.S., PhD
(surname, name, patronymic, scientific degree, academic rank)

Approved by university order as of « 07 » 05 2025 № 4/7-447

2. Student's paper submission deadline 07.07.2025

3. Initial data for the paper Literature sources about analyzing mobile application vulnerabilities

4. Paper contents (list of issues to be developed)

5. List of graphic material (with exact number of required drawings, slides)

6. Advisors of paper chapters

Chapter	Advisor's surname, initials and position	Signature, date	
		assignment was given by	assignment was received by
Life safety,			
basics of labor protection			

7. Date of receiving the assignment 07.07.2025

TIME SCHEDULE

[illegible]

Student

(signature)

Stanley Onyedikachi Umesie

(surname and initials)

Paper supervisor

(signature)

Holotebko O.S.

(surname and initials)

ANNOTATION

Information system for analyzing mobile application vulnerabilities // Diploma thesis Bachelor degree // Stanley Onyedikachi Umesie // Ternopil' Ivan Puluj National Technical University, Faculty of Computer Information System and Software Engineering, Department of Computer Science // Ternopil', 2025 // P. __, Fig. – __, Tables – __, Annexes – __, References – __.

Keywords: analyzing, mobile application, vulnerabilities, information system.

During the qualification work, the tasks were analyzed, the objectives were defined, and the requirements were set, following which the set goal can be achieved. The relevance of solving the problem of mobile application penetration testing in cybersecurity was analyzed.

An analysis of methods and existing tools for mobile application penetration testing was conducted. It was determined that the problem is quite relevant. A decision was made to improve the mobile application penetration testing process by developing information technology.

Methods and tools for testing mobile applications for penetration were analyzed. After analyzing and comparing methods and tools for testing mobile applications for penetration.

The developed information technology was tested.

This information technology can be used for penetration testing of mobile applications. Information technology consists of different stages, so it is easy to adapt it to the required tasks, expand or change the functionality.

CONTENT

LIST OF ABBREVIATIONS, SYMBOLS AND TERMS	5
INTRODUCTION.....	6
1. ANALYSIS OF THE SUBJECT AREA	7
1.1. Android operating system security analysis	7
1.2. Analysis of mobile application penetration testing methods and tools ...	23
1.3. Formalization of requirements and problem statement	31
2. DEVELOPMENT OF INFORMATION TECHNOLOGY.....	33
2.1. Information technology structure.....	33
2.2. Development of a mobile application penetration testing algorithm.....	40
2.2.1. Information collection stage.....	41
2.2.2. Mobile application testing stage	43
2.2.3. Results analysis stage and report creation process	45
3. 3. IMPLEMENTATION OF INFORMATION TECHNOLOGY MOBILE APPLICATION PENETRATION TESTING	50
3.1. Mobile application penetration testing.....	50
3.3. Creating a report.....	63
4 SAFETY OF LIFE, BASIC LABOR PROTECTION.....	68
4.1. Labor protection requirements when working with electrical equipment	68
4.2. Safety requirements during work.....	72
4.3. Safety requirements after completion of repair and maintenance of electrical equipment	74

CONCLUSIONS	76
REFERNCES	77

LIST OF ABBREVIATIONS, SYMBOLS AND TERMS

HEI - Higher education institution;

OS - Operating system;

ADT - Android Development Tools;

API - Application Programming Interfaces;

APK - Android Package;

ARM - Advanced RISC Machine;

SQL - Structured Query Language;

XML - Extensible Markup Language;

INTRODUCTION

The modern world of information technology is constantly evolving, offering a wide range of mobile applications that play an important role in people's daily lives and business areas. Mobile applications provide users with access to a variety of information, services and entertainment, and are an integral part of the digital environment. The growing popularity of mobile platforms has led to the emergence of new information security threats associated with the use of mobile applications.

One of the important tasks associated with the development and use of mobile applications is ensuring their security and reliability. Over time, there is an increasing need for information protection, as applications contain confidential information, personal data and access to network resources. It is important to remember that cybercriminals are constantly looking for new ways to penetrate mobile applications to steal information or cause other harmful effects.

To do this, you need to conduct penetration testing of mobile applications. Penetration testing is a necessary component of the information security process. The main goal is to identify weaknesses in systems and applications before attackers can exploit them. Mobile applications, like any other programs, may have potentially dangerous points in the security system that need to be identified and eliminated. Penetration testing allows you to assess the effectiveness of security measures and take steps to improve them.

1. ANALYSIS OF THE SUBJECT AREA

1.1. Android operating system security analysis

The Android operating system is an open source system. Android has the freedom to modify, invent, and implement its own device drivers and features. The Android operating system consists of six different layers (Figure 1.1).

1) System apps layer. Android comes with a set of core apps for email, SMS, calendars, internet, contacts, etc. Apps that are part of the platform have no special status among the apps that a user chooses to install. Thus, a third-party app can become the default web browser, SMS messenger, or even the default keyboard. There are some exceptions, such as the system Settings app. System apps function as both user apps and provide key capabilities that developers can access from their own apps. For example, if you want an app to send SMS messages, you don't need to write that functionality yourself. Instead, you can call any of the already installed SMS messaging apps to deliver the message to the specified recipient [2].

2) Java Application Framework Layer. The application framework provides several important classes that are used to build Android applications. It provides a common abstraction for accessing hardware and helps in managing the user interface with the application resources. In general, it provides services through which we can create a particular class and make it useful for building applications. It includes different types of services such as activity manager, notification manager, view system, package manager, etc., which are useful for developing our application according to the prerequisites [3].

3) The Application Framework layer provides many high-level services to applications in the form of Java classes. Application developers can use these services in their applications.

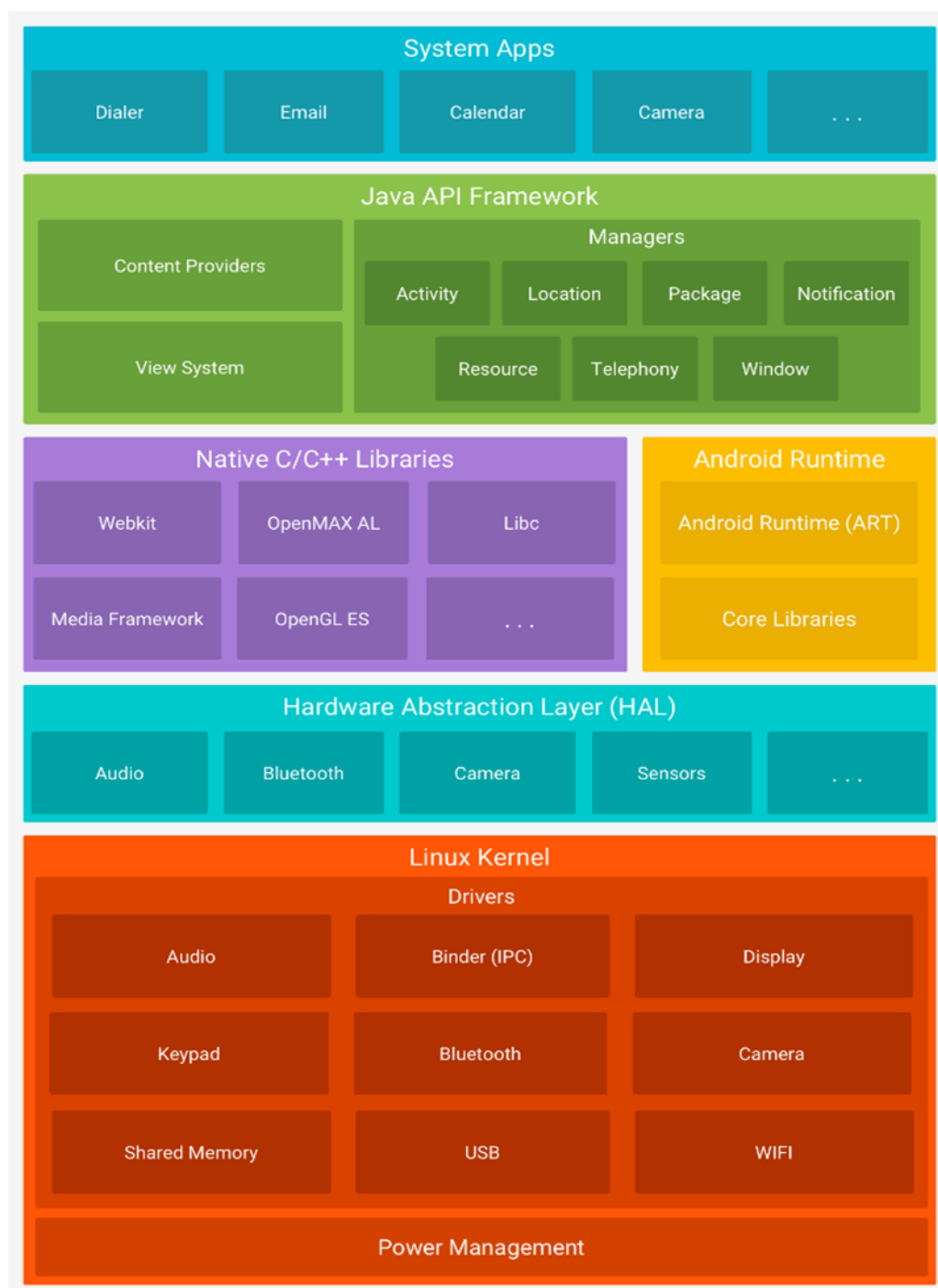


Figure 1.1 – Android platform architecture

The Android framework includes the following key services:

Activity Manager: Manages all aspects of the application lifecycle and activity stack.

Content Providers: Allows applications to publish and share data with other applications.

Resource Manager: Provides access to uncoded built-in resources such as strings, color settings, and UI layouts.

Notification Manager: Allows applications to show alerts and notifications to the user.

View System: An extensible set of views used to create application user interfaces [3].

4) Native C/C++ libraries layer. Many of the core Android system components and services, such as ART and HAL, are built on native code that requires native libraries written in C and C++. The Android platform provides APIs to the Java framework to expose the functionality of some of these native libraries to applications. Native libraries are required for Android development, and most of them are open source. This is a set of core C/C++ libraries, as well as Java-based libraries that support Android development, such as Graphics, Libc, SSL (Secure Socket Layer), SQLite, Media, Webkit, OpenGL (Open Graphic Library), Surface Manager, etc. Here is a detailed description of some of the key Android libraries that are available for Android development [3].

The Media library provides support for playing and recording audio and video formats;

The Surface Manager library provides display management functionality;

OpenGL (Open Graphic Library) and SGL (Scalable Graphics Library) libraries are used to create 2D and 3D graphics;

The SQLite library provides database support;

The FreeType library provides support for a variety of fonts;

The SSL (Secure Socket Layer) library provides security technologies for establishing an encrypted connection between a web server and a web browser;

WebKit provides a set of classes designed to embed web browser capabilities into applications.

4) **Android Runtime Layer.** The Android Runtime is one of the most important parts of Android. It contains components such as core libraries and the Dalvik Virtual Machine (DVM). It mainly provides the framework for the application and enables the application to run using the core libraries. Like the Java Virtual Machine (JVM), the Dalvik Virtual Machine (DVM) is a registry-based virtual machine specifically designed and optimized for Android to ensure that the device can run multiple instances efficiently. It relies on the Linux kernel layer for thread management and low-level memory management. The kernel libraries allow us to implement Android applications using standard programming languages such as JAVA or Kotlin [3].

5) **Hardware abstraction layer (HAL).** The hardware abstraction layer (HAL) provides standard interfaces that expose the hardware capabilities of a device to the high-level Java API framework. The HAL consists of several library modules, each of which implements an interface for a specific type of hardware component, such as a camera or a Bluetooth module. When the framework API makes a call to access the device hardware, the Android system loads the library module for that hardware component [3].

6) **Linux Kernel Layer.** The Linux kernel is the heart of the Android architecture. It manages all the available drivers such as display, camera, Bluetooth, audio, memory, etc. that are required during the execution of the application. The Linux kernel provides a layer of abstraction between the device hardware and other components of the Android architecture. It is responsible for managing memory, power, devices, etc. [3].

Features of the Linux kernel:

Security: The Linux kernel is responsible for application-to-application security and the system.

Memory Management: Manages memory efficiently, providing freedom in developing our applications.

Process Management: manages the process well, allocates resources between processes when they need them.

Network Stack: Efficiently manages network communication.

Driver Model: Ensures that the program works properly on devices and hardware from manufacturers responsible for including their drivers in the Linux build.

When creating Android applications, you need various components of the Android device, such as the camera, GPS, etc.

In order to use these features of an Android smartphone, you must first obtain permission from the user to use something on their phone [3]. In addition, permissions have different levels of protection (Figure 1.2), namely:

- 1) Normal Permissions - Default value. A low-risk permission that grants the requesting application access to isolated application-level functions with minimal risk to other applications, the system, or the user. The system automatically grants this type of permission to the requesting application during installation without asking for explicit user approval, although the user always has the opportunity to review these permissions before installation [4].
- 2) Signature Permissions - A permission that the system grants only if the request requesting the permission is signed with the same certificate as the request declaring the permission. If the certificates match, the system automatically grants the permission without notifying the user or asking for their explicit approval [4].
- 3) Dangerous Permissions - A high-risk permission that gives the requesting application access to the user's private data or control over the device, which could negatively affect the user. Because this type of permission is associated with potential risk, the system may not automatically grant it to the application requesting it.

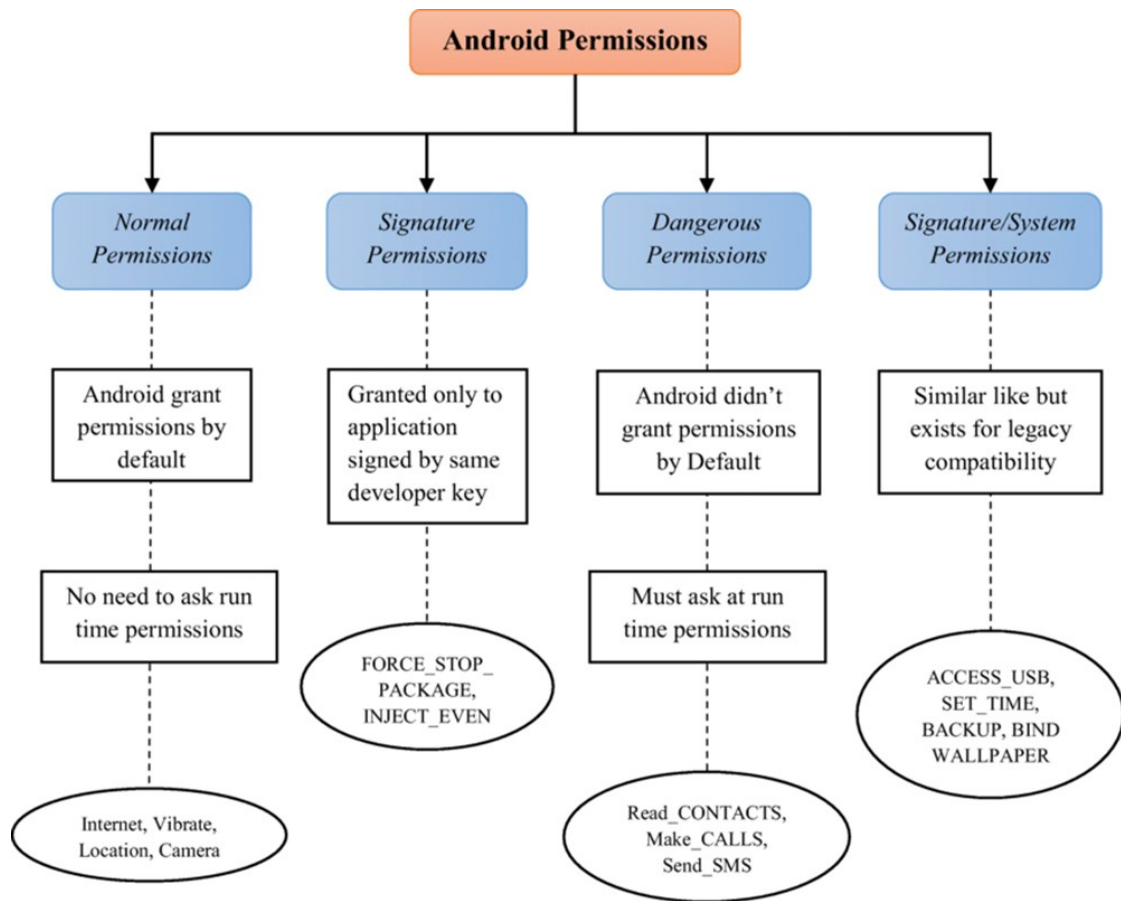


Figure 1.2 – Scheme of protection levels for permissions

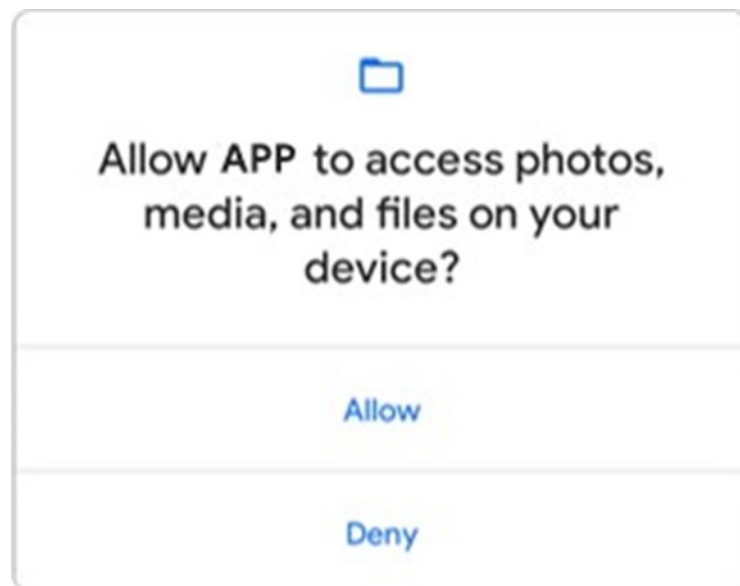


Figure 1.3 – An example of a system permission request that appears when a program requests execution permission.

For example, any dangerous permissions requested by the application may be displayed to the user and require confirmation before continuing, or another approach may be taken to avoid the user automatically granting permission to use such features (Figure 1.3) [4-5].

4) Permission level "Signature or System Permissions" (signatureOrSystem). Old name "signatureOrSystem", which is deprecated from Android

6.0 Marshmallow. A permission that the system grants only to apps that are in a special folder in the Android system image or are signed with the same certificate as the app that declared the permission. The "signatureOrSystem" permission is used in certain special situations where apps from different manufacturers are built into the system image and need to share certain features because they are built together [6].

APK is a file format used to distribute and install applications on the Android operating system. It is a compressed file that contains all the files needed to run an application on an Android device, including its code, resources, and the Android manifest file. APK files are similar to other package file formats, such as .exe files on Windows or .dmg files on macOS [7].

Users can install apps on their Android devices by downloading them from the Internet, transferring them via USB, or using an app store such as the Google Play Store. Once the app is installed, you can run and use like any other application. Developers can create APKs using the Android Studio development environment, which includes tools for building, testing, and packaging applications in APKs. Developers can also digitally sign their APKs to ensure the authenticity and integrity of the application. [7]

An APK consists of several components that are required for the application to run properly on an Android device. The details of these components are shown in Figure 1.4.

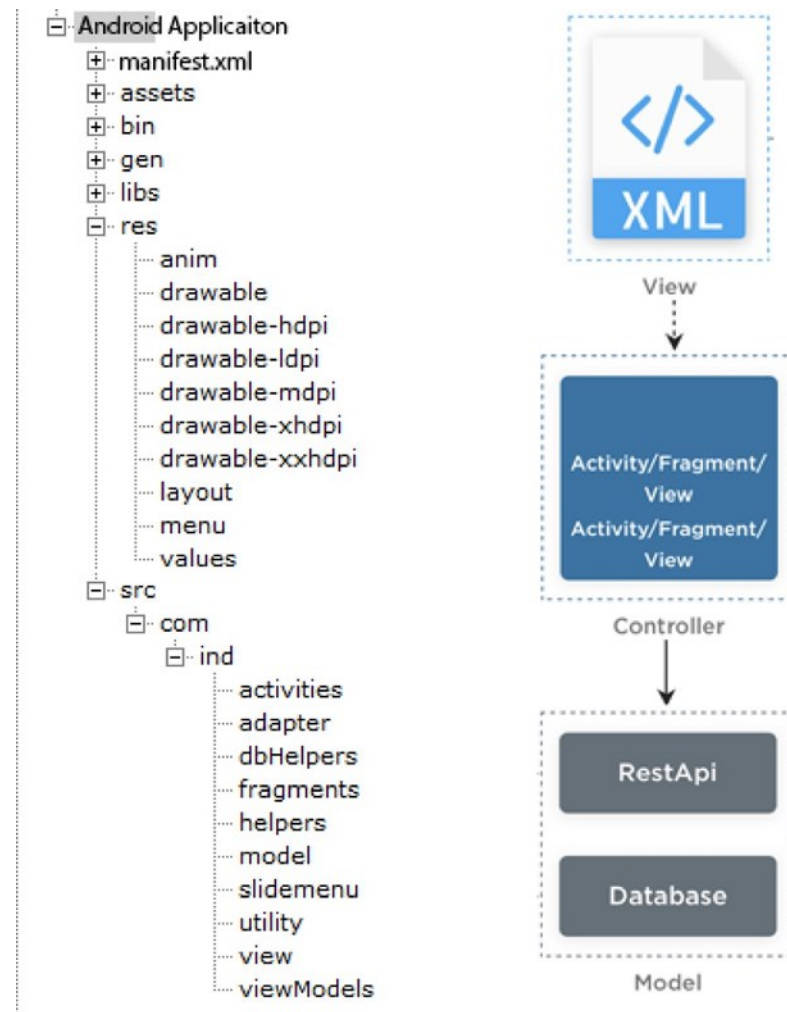


Figure 1.4 – Example of Android application package file structure – App code: This includes the Java or Kotlin source code that defines

Application functionality. This code controls the behavior of the application and handles user interaction.

- Resources: It files, which uses application, for example, images, layouts, and strings. These resources define the appearance of the application and contain text for different languages.
- Android manifest: This is a special XML file that contains important information about your app, such as its name, version, and the permissions it requires. The manifest also defines the app's components, such as actions, services, and broadcast receivers.

- External libraries: If the app uses external libraries, they are also packaged in the APK file. These libraries provide additional functionality and are usually open source projects integrated into the app.
- Assets: These are files that the app uses but does not compile, such as fonts, audio, and video files. They are stored in the APK in their native format and are read by the app at runtime.
- Android runtime (ART): This component runs an application on Android devices; it converts the application's bytecode into machine code so that the device's processor can execute it.

AAB is a new format introduced by Google for distributing Android applications. AAB is a new replacement for APK, introduced on August 6, 2018 with Android 9 [8], but it is designed to be more efficient and flexible. The main difference between AAB and APK is that AAB contains all the code and resources of an application, but does not contain compiled machine code for all devices. Instead, it contains the code and resources of an application in a form that can generate machine code for specific devices during installation. This makes the application smaller and more efficient, as it contains only the machine code needed for the specific device on which it is installed. In addition, AAB allows developers to create multiple APK files from a single Android project that can target different devices, screen configurations, and languages, reducing the size of the final APK file and improving the user experience [7].

According to the Common Vulnerabilities And Exposures database, many vulnerabilities have been discovered in the Android operating system since its inception. Since 2009, about 5,000 vulnerabilities have been discovered in the Android OS. Figure 1.5 shows the vulnerabilities discovered in recent years. It can be seen that the number of vulnerabilities discovered in the Android OS has been steadily increasing over the years [7].

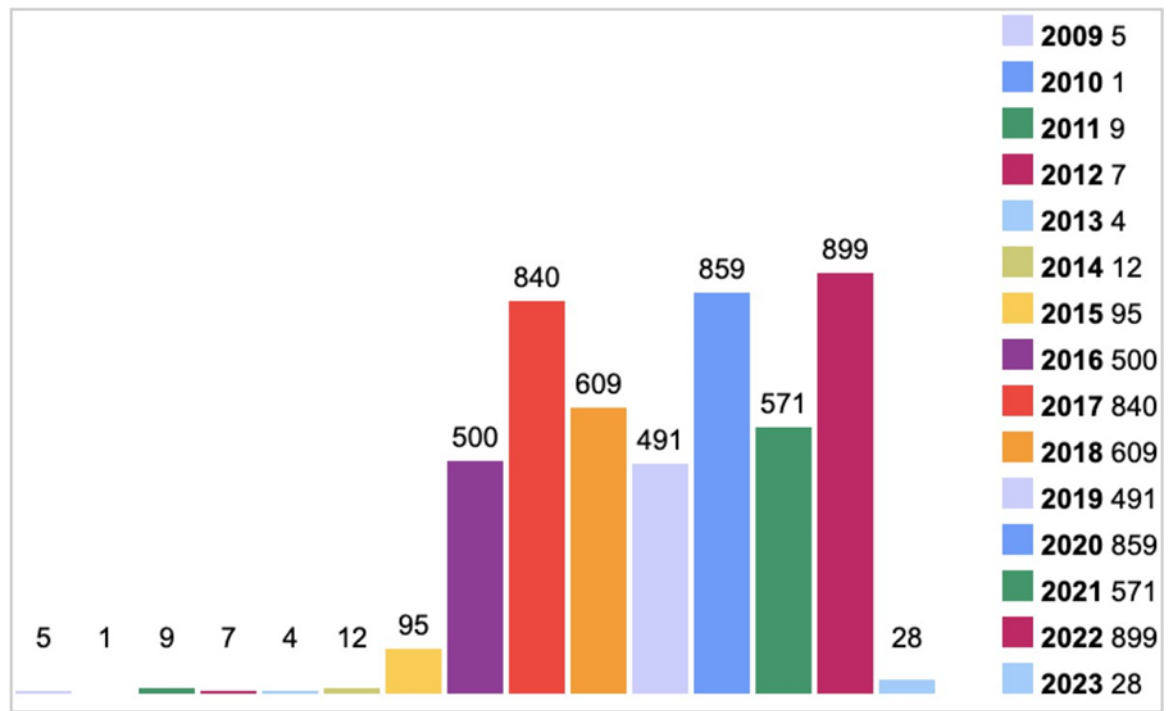


Figure 1.5 – Vulnerability statistics

The numbers demonstrate the ongoing security challenges faced by the Android OS. Despite the efforts of developers and researchers to improve the security of the Android operating system, the number of vulnerabilities discovered continues to increase. Furthermore, these discovered vulnerabilities can be divided into different types such as code execution, buffer overflow, and information retrieval. This categorization helps to identify which types of vulnerabilities are the most popular and common. Figure 1.6 provides details of the vulnerabilities by different types. The figure shows that the most common vulnerabilities that lead to exploitation are implementation code and overflow buffer. Details reported types vulnerabilities are listed below [7].

The most popular Android OS vulnerabilities:

1. Denial of service (DoS): This type of vulnerability occurs when an attacker floods a system with traffic or requests to overload its resources and make it inaccessible to legitimate users.

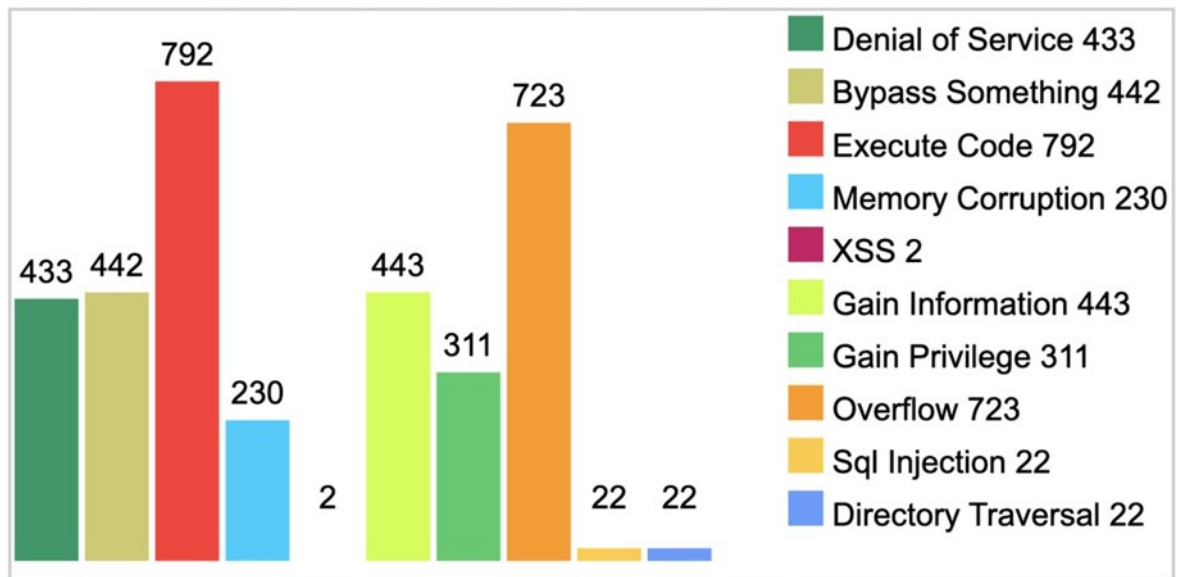


Figure 1.6 – Android vulnerability statistics

2. Bypass something: This vulnerability allows an attacker to bypass security controls, such as authentication or authorization, access protected resources, or perform unauthorized actions.

3. Execute code: This vulnerability could allow an attacker to execute arbitrary code on a targeted system or device, which could lead to data theft, unauthorized access, or other malicious actions.

4. Memory corruption: This vulnerability involves exploiting errors or flaws in an application's memory management system, such as buffer overflows or use-after-prohibition errors, to gain unauthorized access to data or execute malicious code.

5. Cross-site scripting (XSS): This vulnerability allows an attacker to inject malicious code, such as Java or Kotlin, into a web page or

a program that could potentially lead to data theft or other malicious actions.

6. Information disclosure: This vulnerability allows attackers to access sensitive information without authorization, such as passwords, personal data, or system configuration information.

7. Privilege escalation: This vulnerability allows an attacker to gain higher access or privileges than authorized, potentially allowing them to perform malicious actions or access sensitive data.

8. Buffer overflow: This vulnerability occurs when an attacker injects more data into an application's memory buffer than it can handle, potentially leading to arbitrary code execution or system failure.

9. SQL injection: This vulnerability allows an attacker to inject malicious SQL code into a web application or database, potentially leading to unauthorized access or modification of data.

10. Directory traversal: This vulnerability involves exploiting a web application's file path validation error to gain unauthorized access to files or directories outside of the intended application.

This information can be used to prioritize vulnerability remediation and identify patterns in the types of vulnerabilities that are most frequently targeted. It can also provide insight into how the industry is evolving and allow for preventive measures to be taken to address vulnerabilities that are likely to be targeted in the future [7].

The Google Play Store is the official app store for Android devices, where users can download and install apps, games, and other types of software. [9] Apps in the store are developed by third-party developers and are reviewed by Google before being made publicly available. Google has introduced Play Protect to scan apps for malware and other harmful content before publishing them to the store. This includes static and dynamic analysis of applications and checking for any known malicious behavior. However, despite these measures, attackers still infiltrate the store. There are several approaches through which attackers can download and distribute malware through the Google Play Store, the details of which are given below. [7].

1. Dynamic code loading: Attackers can use dynamic code loading techniques, such as Java mapping or Android DexClassLoader, to load and execute

code at runtime. This can be used to load and execute additional code or malicious payloads after an application is installed, which can be hidden within the legitimate application code or downloaded from a remote server. This can allow attackers to avoid detection by security scanners because the malicious code may not be present in the original version of the application.

2. Incremental malicious update attack (IMUTA): Attackers can use incremental updates to gradually add malicious code to an application over time. This can be done using a dropper program that initially appears legitimate but later downloads and installs additional components or payloads. Attackers can also use code obfuscation and dynamic code loading techniques to add malicious code in a way that is difficult to detect, such as through updates. This can allow attackers to evade detection by security scanners and extend the lifespan of the malware.

3. Code obfuscation: Obfuscation is the process of changing the source code of a program to make it more difficult to understand or analyze. Attackers can use obfuscation techniques, such as renaming variables and classes, adding unwanted code, or using encryption, to hide the functionality and purpose of the malicious code. This can make it more difficult for security researchers to detect and analyze malware.

4. Repackaging: Attackers can use repackaging techniques to take a legitimate app and add malicious code. This can be done by decompiling the app, adding malicious code, and then recompiling and re-registering the app. The repackaged app can then be uploaded to the Google Play Store, and users can be tricked into downloading it because it may appear legitimate and have good reviews.

5. Social engineering: Social engineering is the act of tricking people into taking certain actions or revealing sensitive information. Some cybercriminals use social engineering to trick users into downloading and installing programs that appear legitimate but are actually malicious software. These programs may look

like popular programs or games, but they contain malware that can steal personal information or perform other malicious actions.

The Google Play Store hosts numerous malware families that have been identified, reported, and published. Google has detected and blocked many Android malware families, but many recent incidents still indicate loopholes and vulnerabilities in the security mechanisms implemented by the Google Play Store [7,9].

Table 1.1 lists the 10 malware families that are distributed via the Google Play Store. These malware families are not limited to a single application or account, and their code can be associated with hundreds of applications that users may unwittingly download. Thus, it is of utmost importance that users and developers exercise caution and diligence when using and creating applications, respectively, and that the Google Play Store continues to implement robust security measures to detect and prevent the spread of malware [7,9].

Table 1.1 - List of the most common Android malware families detected in the Google Play Store.

Name	Number of applications	Total percentage (%)
Airpush	1574	23.8
Plankton	722	10.9
Adwo	583	9.0
Kuguo	492	7.4
Leadbolt	298	4.5
Dowgin	261	3.9
Fakeinst	214	3.2
Gingermaster	194	2.9
Wapsx	192	2.8
Youmi	110	1.7

These malware families can be found in various Google Play Store apps in several ways. Once the malware is downloaded and installed on a user's device, it can perform a variety of malicious actions. This can include stealing personal information, displaying unwanted ads, or even taking control of the device. These malware attacks can be harder to detect with Google Play Protect and other mobile antiviruses because they are embedded in the apps and may not have any visible signs of malicious behavior. The malware can also only launch after a certain amount of time, making it difficult for antiviruses to detect [7,9].

Malware can be detected by analyzing the permissions it requests when installed on an Android device. Analyzing the permissions, a particular app requests can help determine whether it is potentially malicious. For example, if an app requests permissions that are not related to its

intended functions, such as the ability to access the user's contacts or call logs, this may indicate that the application is malicious. Additionally, if the application requests permissions that give it access to sensitive information, such as location data, this may be a sign of malware. Table 1.2 provides a resource for identifying malware by listing the permissions that malware most often requests after installation. This can be a useful resource for identifying malware because it lists the permissions that malware most often requests after installation [7,9].

By analyzing Tables 1.1 and 1.2, security experts and researchers can identify potentially malicious applications by analyzing the permissions they request. It is also important to note that not all applications that request these permissions are malicious, but it may be a red flag that requires further investigation.

Table 1.2 - List of the most common permissions requested by Android malware.

Permission name	Usage percentage	Permission group	Classification based on permissions
INTERNET	98.8	Chain	Dangerously
ACCESS_NETWORK_STATE	94.4	Chain	Normally
READ_PHONE_STATE	82.7	Telephone call	Dangerously
WRITE_EXTERNAL_STORAGE	70.4	Refuge	Dangerously
ACCESS_COARSE_LOCATION	56.2	Arrangement	Dangerously
ACCESS_WIFI_STATE	54.9	Chain	Normally
ACCESS_FINE_LOCATION	53.7	Arrangement	Dangerous
Permission name	Usage percentage	Permission group	Classification based on permissions
GET_ACCOUNTS	29.0	Accounts	Normally
C2D_MESSAGE	24.7	-	Signature
RECEIVE_BOOT_COMPLETED	24.7	Information about program	Normally
SYSTEM_ALERT_WINDOW	17.3	Display	Dangerously
CALL_PHONE	14.2	Calls to phone	Dangerously

Continuation of Table 1.2 - List of the most common permissions requested by Android malware.

Permission name	Usage percentage	Permission group	Classification based on permissions
CAMERA	13.0	Cell	Dangerously
RECORD_AUDIO	12.3	Microphone	Dangerously
EAD_HISTORY_BOOKMARKS	11.7	Bookmarks	Dangerously
SEND_SMS	11.7	Message	Dangerously
READ_EXTERNAL_STORAGE	14.8	Storage	Normally

1.2. Analysis of mobile application penetration testing methods and tools

Mobile application penetration testing encompasses a range of approaches, each tailored to specific testing requirements. These methodologies provide a clear picture of the application's security posture and vulnerabilities. There are three main types of mobile application penetration testing:

- Black box testing is conducted without the tester having any knowledge of the application under test. This process is sometimes referred to as "zero-knowledge testing." The main goal of this test is to allow the tester to act like a real attacker in the sense of exploring possible ways to exploit publicly available and open information.

- White box testing (sometimes called "full knowledge test") is the complete opposite of testing by "black box"

"box" in the sense that the tester has complete knowledge of the program. Knowledge can include source code, documentation, and diagrams. This approach allows testing to be done much faster than black box testing due to its transparency, and with the additional knowledge gained by the tester, much more complex and detailed test cases can be created.

- Gray box testing is all testing that falls between the two types of testing mentioned above: some information is provided to the tester (usually just credentials), and other information is intended for discovery. This type of testing is an interesting compromise between the number of tests, cost, speed, and scope of testing. Gray box testing is the most common type of testing in the security industry [10].

Mobile application penetration testing is the process of identifying and addressing potential security vulnerabilities in mobile applications. This process can be complex and involves multiple steps. The general steps of mobile application penetration testing are as follows:

- 1) preparation;
- 2) intelligence gathering;
- 3) drawing up an application map;
- 4) operation;
- 5) reporting.

During the preparation stage, the following tasks are solved:

- The scope of the security testing is determined, including identifying appropriate security controls, the organization's testing objectives, and sensitive data. In general, preparation includes all synchronization with the client, as well as legal protection for the tester (who is often a third party).

The next stage involves collecting intelligence information:

- an analysis of the environment and architectural context of the application is performed to obtain a general understanding of the context;

Application Mapping Phase – builds on information from previous phases; may be supplemented by automated scanning and manual application research. Mapping provides a deep understanding of the application, its entry points, the data it stores, and the main potential vulnerabilities. These vulnerabilities can then be ranked according to the damage that their exploitation could cause, so that the security tester can prioritize them. This phase involves creating test cases that can be used during test execution.

- In this phase, the security tester attempts to penetrate the application using the vulnerabilities identified in the previous phase. This phase is necessary to determine whether the vulnerabilities are real and truly positive.

The final reporting stage is important for the client, the security tester reports the vulnerabilities. This includes a detailed exploitation process, categorizes the type of vulnerability, documents the risk if an attacker is able to compromise the target, and outlines what data the tester had unauthorized access to [11].

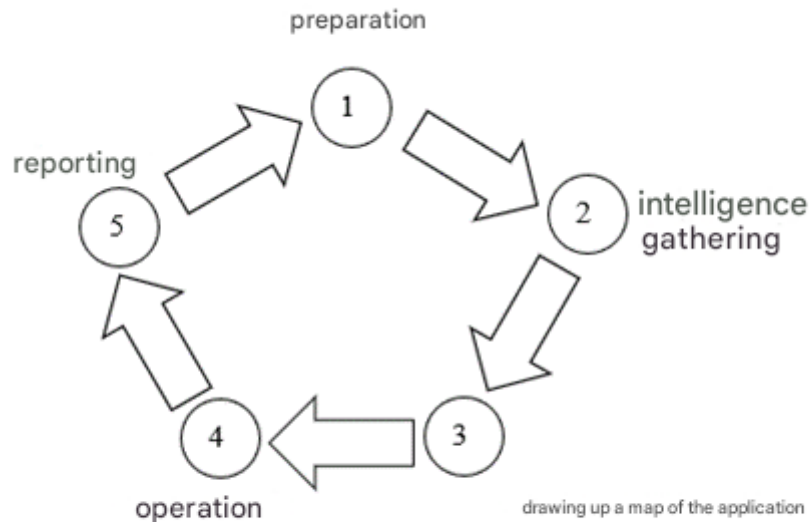


Figure 1.7 – Mobile application penetration testing process diagram After scanning the application map, the exploitation stage occurs:

Penetration testing of mobile applications requires compliance with various standards and guidelines. Some of the most important standards and documents that can be used when conducting penetration testing of mobile applications are:

Open Web Application Security Project (OWASP) - Mobile Security Testing Guide (MSTG): MSTG provides a detailed and practical guide on how to conduct security testing of mobile applications, covering the Android and iOS platforms. MSTG covers topics such as mobile application architecture, data storage, cryptography, authentication, networking, reverse engineering, code analysis, and penetration testing. MSTG also provides a checklist of security requirements and best practices that can be used to assess and improve the security of mobile applications [12].

NIST SP 800-163: Vetting the Security of Mobile Applications: The standard provides detailed guidance on assessing the security of mobile applications, including penetration testing [13].

Mobile Application Security Testing (MAST) Framework: This is a resource developed by an international banking industry organization that provides a common approach to mobile application security testing [14].

NIST SP 800-163 Revision 1, “Vetting the Security of Mobile Applications,” is a major update to NIST’s guidance on mobile application verification and security. The original document (January 2015) detailed the processes by which organizations assess mobile applications for cybersecurity vulnerabilities. Revision 1 expands on the original document by exploring resources that can be used to develop an organization’s mobile application security requirements. This includes reviews of relevant documentation from the National Information Protection Partnership (NIAP), the Open Web Application Security Project (OWASP), MITRE Corporation, and the National Institute of Standards and Technology (NIST) [15-16].

MITRE ATT&CK: This is a globally accessible knowledge base on tactics and adversary methods, based on real-world observations. The ATT&CK knowledge base is used as a basis for developing specific models and methodologies for countering threats in the private sector, in government, and in the community of developers of cybersecurity products and services.

Table 1.3 lists mobile application penetration testing methods, their clear advantages and disadvantages.

From the table above, it can be determined that the methods lack information on how to develop a testing report. To develop our own testing method, MSTG was taken as a basis. MSTG is the best and most understandable method for testing a mobile application for penetration due to the presence of clear documentation, the availability of penetration testing

tools, and the ease of understanding of the documentation.

Table 1.3 – Methods mobile application penetration testing.

Method name	OWASP - MSTG	NIST SP 800-163 Revision 1 "Vetting the Security of Mobile Applications»	Mobile Application Security Testing (MAST)	MITRE ATT&CK
Availability documentation	+	+	+	+
Availability of clear documentation	+	- (has links to OWASP and MITRE ATT&CK)	- (has links to OWASP and MITRE ATT&CK)	+
Method name	OWASP - MSTG	NIST SP 800-163 Revision 1 "Vetting the Security of Mobile Applications»	Mobile Application Security Testing (MAST)	MITRE ATT&CK
Ease of understanding of documentation	+(of course what is needed to do with information)	- (it is not clear what to do with the information)	-	- (excessive amount links to various resources)
Availability of funds penetration testing	+	-	-	+

The Mobile Application Security Verification Standard (MASVS) is a comprehensive security standard that developed by OWASP. The standard is divided into different groups that represent the most critical areas of the mobile attack surface. These control groups, designated MASVS-XXXXX, provide general recommendations and standards for the following areas:

1. MASVS-STORAGE: Secure storage of confidential data on the device (data-at-rest);

1.1 MASVS-STORAGE-1 The application securely stores confidential data; data.

1.2 MASVS-STORAGE-2 The application prevents the leakage of confidential.

2. MASVS-CRYPTO: Cryptographic functions used for protection of confidential data;

2.1 MASVS-CRYPTO-1 Application uses current reliable cryptography and uses it in accordance with industry best practices;

2.2 MASVS-CRYPTO-2 Application manages keys in accordance to industry best practices.

3. MASVS-AUTH: Mechanism's authentication and authorization, What used by a mobile application;

3.1 MASVS-AUTH-1 Application uses safe protocols authentication and authorization and follows appropriate best practices;

3.2 MASVS-AUTH-2 Application safely performs local authentication according to platform best practices;

3.3 MASVS-AUTH-3 The application protects sensitive transactions using additional authentication.

4. MASVS-NETWORK: Secure network communication between a mobile application and remote endpoints (data in transit);

4.1 MASVS-NETWORK-1 The application protects all network traffic in accordance with current best practices;

4.2 MASVS-NETWORK-2 The application performs identity anchoring for all remote endpoints under the developer's control.

5. MASVS-PLATFORM: Secure interaction with the underlying mobile platform and other installed applications;

5.1 MASVS-PLATFORM-1 The application securely uses mechanisms IPC;

5.2 MASVS-PLATFORM-2 The application uses Web Views securely.

5.3 MASVS-PLATFORM-3 The application securely uses the interface user.

6. MASVS-CODE: Best security practices for data processing and maintaining the program up-to-date;

6.1 MASVS-CODE-1 The application requires the latest platform version;

6.2 MASVS-CODE-2 The application has a mechanism for forced program updates;

6.3 MASVS-CODE-3 Application uses only software components without known vulnerabilities;

6.4 MASVS-CODE-4 The application checks and disinfects all unreliable data;

7. MASVS-RESILIENCE: Resistance to reverse engineering attempts and intervention;

7.1 MASVS-RESILIENCE-1 The application checks the integrity of the platform.

7.2 MASVS-RESILIENCE-2 The application implements mechanisms for protection against intrusion;

7.3 MASVS-RESILIENCE-3 Application implements mechanisms antistatic analysis;

7.4 MASVS-RESILIENCE-4 The application implements anti-dynamic analysis methods.

Mobile application penetration testing requires tools to accomplish the task at hand. The most popular mobile application testing tools include:

1) OWASP ZAP (Zed Attack Proxy) is a universal tool for testing the security of web and mobile applications [12].

- can perform automated scans to detect common vulnerabilities such as SQL injections, cross-site scripting (XSS), etc.;

- offers functions for intercepting and modifying requests and responses for application security testing;

- provides a user-friendly interface for both beginners and experienced testers.

2) MobSF (Mobile Security Framework) is a comprehensive tool for assessing the security of mobile applications [39].

- supports static analysis by decompiling and analyzing APK (Android) and IPA (iOS) files;
- performs dynamic analysis by interacting with the mobile application and tracking its behavior;
- detects vulnerabilities such as unsecured data storage, unsecured network connection, etc.

3) Drozer is specifically designed for Android security testing [28].

- can detect vulnerabilities such as incorrect permissions, exported components, and unsafe file storage;
- allows testers to interact with Android devices and applications through a command line interface;
- supports different modules for testing specific components of Android applications.

4) Frida is a dynamic toolkit for Android and iOS [29].

- allows analyze and manipulate behavior of applications in real-time, including function interception and scripting.
- useful for analyzing and modifying the behavior of running mobile applications at runtime.

5) AppUse is a virtual machine that comes with a set of pre-installed tools for mobile application security testing [40];

- provides a controlled environment for testing Android applications and their security features.

6) QARK (Quick Android Review Kit) - focuses on assessing the security of Android applications [41].

- can detect vulnerabilities such as insecure data storage, code execution, and improper permissions;
- creates detailed reports with information about found vulnerabilities.

- 7) AndroBugs Framework [42].
 - scans Android apps for security issues;
 - detects vulnerabilities such as code execution, unprotected components, and data leakage.
 - offers a user-friendly interface for scanning and generating reports.
- 8) Xposed Framework - allows testers to create and install modules that can intercept and modify the behavior of Android applications. [43].
 - useful for configuring and testing application behavior and security features.
- 9) Burp Suite Mobile Assistant is an extension of Burp Suite used for testing the security of mobile application APIs. [44].
 - intercepts and analyzes requests and responses to detect security vulnerabilities in API calls made by mobile applications.

1.3. Formalization of requirements and problem statement

One of the mechanisms for protecting mobile applications is penetration testing. Methods and tools for penetration testing of mobile applications were analyzed. Creating your own information technology for penetration testing allows you to save time searching for testing methods and testing tools and optimize this process, because these standards are usually quite voluminous. Depending on the type of complexity and set of application functions, testing methods and tools differ.

During the analysis, it was found that OWASP is the best collection of methodologies for testing mobile applications for penetration, but the disadvantage of each of the methods is that they do not provide information as

create a mobile application testing report and what method and tool to use for different types of applications.

In this paper, the object is the process of improving penetration testing methods for mobile applications during penetration testing.

The main goal is to improve the penetration testing process at the data collection and application operation stages using testing tools.

After analysis means For MASVS testing, a set of requirements for mobile application penetration testing technology was formed:

- testing the configuration of cryptographic standard algorithms;
- checking local storage for confidential data;
- verification of confidential data;
- testing backups for confidential data;
- testing data encryption on the network;
- testing program permissions;
- testing of local storage to verify entered data. Based on the analyzed

data, the following tasks were identified:

- Security Analysis: Conduct a detailed security analysis of the mobile application to identify potential vulnerabilities.
- Penetration Testing: Develop and implement penetration testing using a variety of attack methods to verify the effectiveness of the security system.
- Vulnerability identification and documentation: Identify and document vulnerabilities, including a description of possible attack scenarios and recommendations for their remediation.
- Development of recommendations: Develop recommendations for improving the security system of the mobile application, taking into account the testing results.

2. DEVELOPMENT OF INFORMATION TECHNOLOGY

2.1. Information technology structure

Information technology is a set of methods and tools through which a certain task is realized. Information technology consists of processes that are independent parts. This approach allows you to identify problems at the early stages of development and quickly solve them.

The information technology of mobile application penetration testing consists of several processes, each of which has its own functional purpose. The information technology process diagram is shown in Fig. 2.1.

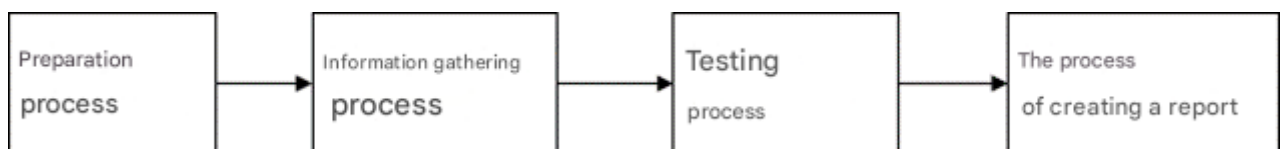


Figure 2.1 – Process flow diagram of penetration testing technology

1) Preparation process. At this stage, you need to assess the complexity of the task and estimate the necessary resources.

The process includes the following stages (Fig. 2.2).

- search for the number of hours spent analyzing the source code;
- analysis of the number of specialists who need to be involved in the preparatory work;
- analysis of the cost of preparatory work.

The specified process is mandatory when testing a mobile application for penetration. The process model can be described as:

$$I = \langle T_{\text{prep}}, N_{\text{specialists}}, C_{\text{prep}} \rangle$$

where,

T_{prep} is the number of hours for source code analysis,

$N_{\text{specialists}}$ is the number of specialists,

C_{prep} is the cost of preparatory work.

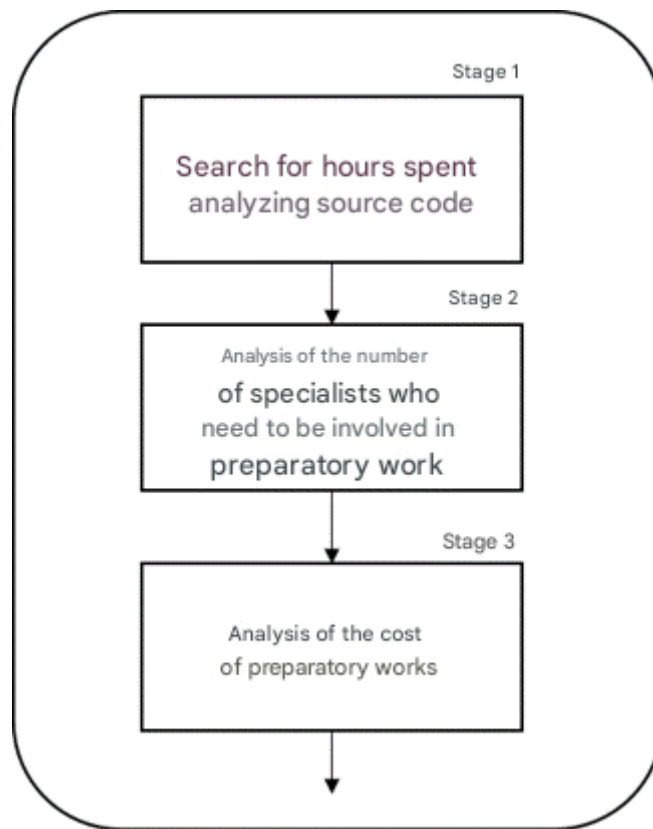


Figure 2.2 – Schematic of the preparation process

2) Information collection process. Scanning the application for vulnerabilities, analyzing the scan results, collecting information.

The process includes the following stages (Fig. 2.3).

- searching for potential vulnerabilities;
- analysis of found vulnerabilities.

The following steps occur during the search for potential vulnerabilities: – Gathering information about the application architecture: this includes code analysis, research into the libraries and frameworks used.

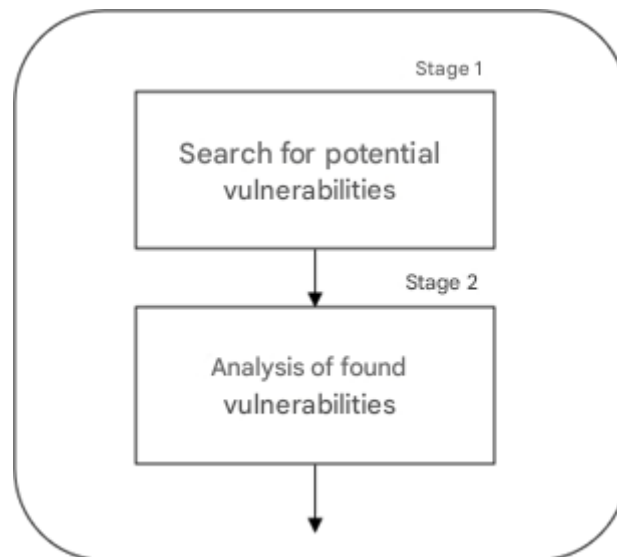


Figure 2.3 – Diagram of the information collection process

- Identifying vulnerabilities in security implementation: this may include checking for insufficient authentication, the possibility of entering incorrect data, the use of outdated encryption methods, and more.

- Analysis of existing weaknesses in the application: research into possible problems related to memory usage, data processing, network protocols, and other aspects of the software.

At the stage of analyzing the found vulnerabilities, the following information is collected:

- Description of discovered vulnerabilities: a detailed description of each vulnerability found, including the type, location, possible impact on the system, and ways to fix it.

- Risk assessment: determining the degree of threat it poses to the security of the system and assessing the potential consequences if these vulnerabilities are exploited by attackers.

- Remediation recommendations: developing specific recommendations and strategies to fix the vulnerabilities found, including practical steps and techniques that can be used to ensure greater security application.

This process is mandatory when testing a mobile application for penetration.

The process model can be described as:

$Rdata = \langle Tset, Iset, Tapp \rangle$

where,

$Rdata$ is a set of detected vulnerabilities,

$Tset$ is a set of technologies with which the mobile application will be tested, for example, such as: OWASP, NIST SP 800-163, MAST, MITRE ATT&CK, etc.,

$Iset$ is a set of tools that will be used to test the mobile application, for example, such as: OWASP ZAP, MobSF, Drozer, Frida, etc.

$Tapp$ is the type of application, the type of application can include technologies used by the application, for example, such as: Rest API, local Database, gRPC, etc.

3) Testing process. Performing attacks on the application, checking for vulnerabilities, analyzing test results. The process includes the following stages (Fig. 2.4).

- selection of attack vectors and necessary tools;
- application vulnerability testing;
- analysis of the results of attacks and identified vulnerabilities.

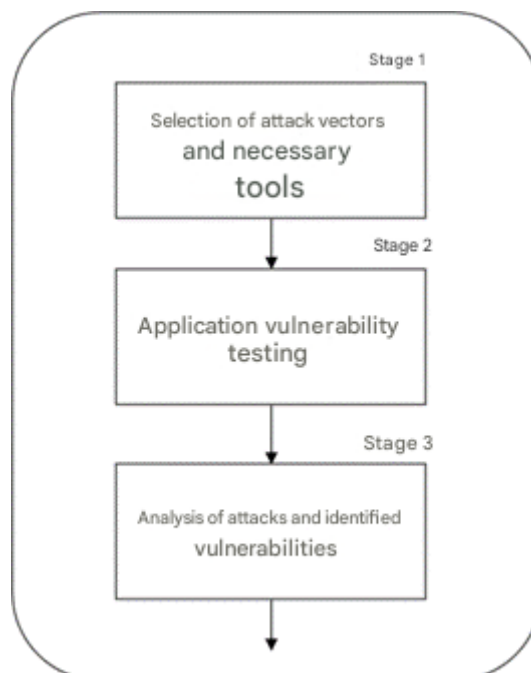


Figure 2.4 – Testing process diagram

At the stage of selecting attack vectors and necessary tools, the following actions take place:

- Based on the information collected and potential vulnerabilities identified in the previous step, we select the necessary set of attack vectors and tools. For example, for an application that does not work with a local database, it is not necessary to attack the database for testing. For different types of applications, different types of attacks and different tools need to be used. Next, the process of testing the application for vulnerabilities occurs.

During the testing phase, penetration testing of the mobile application takes place.

- After collecting information about the application, we will perform the following tests: testing data encryption on the network, checking the configuration of cryptographic standard algorithms, checking local storage for sensitive data, checking sensitive data, testing backups for sensitive data, testing application permissions, testing local storage to verify entered data.

- Details of each stage: description of the process of performing each testing stage, performing vulnerability testing, and evaluating the results.

- Tools and methods used: a list of tools and methods used to perform attacks and assess vulnerabilities.

The stage of analyzing attacks and identified vulnerabilities describes:

- Details detected vulnerabilities: description detected vulnerabilities, including type, location, and possible impact on the system.

- Details of detected attacks: assessment of the results of each attack, their success, and possible impact on application functionality.

This process is mandatory when testing a mobile application for penetration.

The process model can be described as:

$$At = \{ Rdata, Va \}$$

where

At is the test result,

Rdata is the set of detected vulnerabilities,

Va is the attack vectors.

Attack vectors can be described as:

$V_t = \{T_{1,2...N}, I_{1,2...N}\}$,

where

$T_{1,2...N}$ is a set of tests

$I_{1,2...N}$ is a set of tools.

4) Report creation process. Compiling a report on identified vulnerabilities and recommendations for their neutralization. The process diagram is shown in Figure 2.5.

- analysis of identified problems;
- risk assessment and prioritization;
- report formatting.

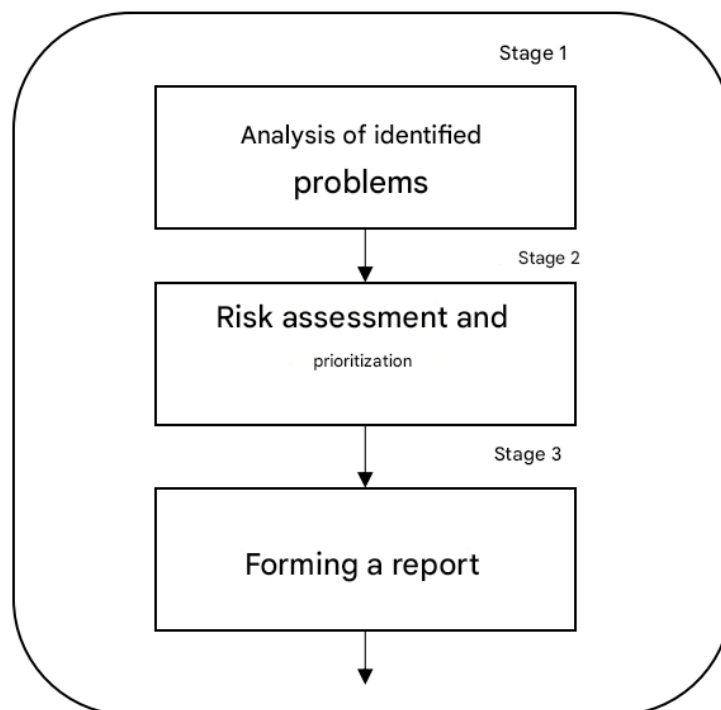


Figure 2.5 – Report creation process diagram

The first stage involves:

- Description of found vulnerabilities: a detailed description of each found vulnerability, including their type, possible impact on the system, and exploitation methods.

- Remediation recommendations: Specific suggestions and recommendations for correcting each identified issue, including practical steps and methods that can be used to ensure greater security.

In the second stage, the following occurs:

- Risk assessment: an assessment of the degree of threat that the identified issues pose to the security of the system and an assessment of the possible consequences if these vulnerabilities are exploited.

- Prioritization: Determine the order of priority for fixing identified issues based on their importance and potential impact on application operations.

In the third stage, the following occurs:

- Creating a report structure: describing the structure of the report, including headings, sections, and subsections that help in logical organization of information.

- Report formatting: formatting details, such as the use of tables, graphs, lists, and other elements that make the information easier to understand.

The process model can be described as:

$$T_{result} = \{ I_{vulnerabilities}, R_{assessment}, R_{neutralizing} \}$$

where,

T_{result} - the result of creating a report,

$I_{vulnerabilities}$ - identified vulnerabilities,

$R_{assessment}$ - criticality assessment,

$R_{neutralizing}$ - a set of recommendations for neutralizing vulnerabilities.

As a result, models and diagrams of information technology processes for mobile application penetration testing have been created. The next step is to develop a mobile application penetration testing algorithm for information technology implementation.

2.2. Development of a mobile application penetration testing algorithm

The algorithm for testing a mobile application for penetration consists of the following stages (Fig. 2.6):

preparation stage;

information collection stage;

mobile application penetration testing stage;

the results analysis stage and the report creation process.

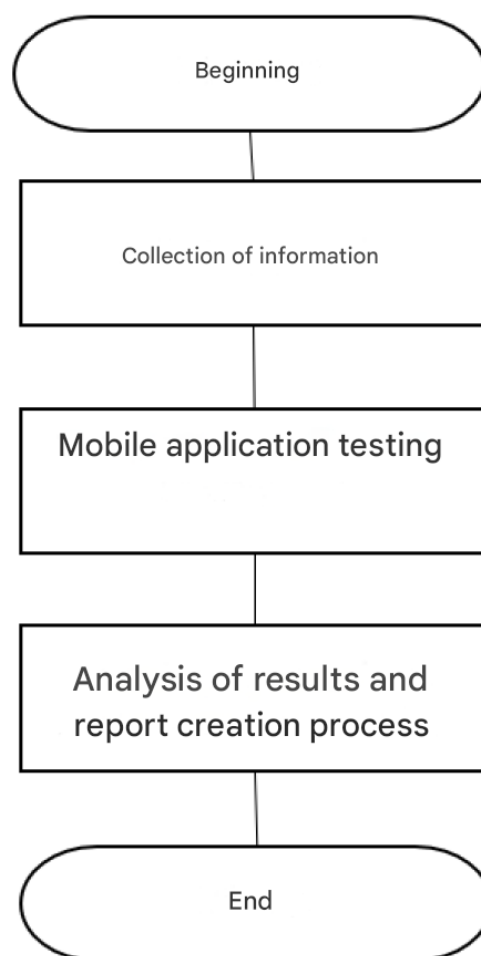


Figure 2.6 – General diagram of the mobile application penetration testing process

The next stage after creating a mobile application penetration testing process diagram is the information collection stage.

2.2.1. Information collection stage

The main goal of this stage is to collect all the necessary information regarding the mobile application, its architecture, design, and security (Figure 2.7).

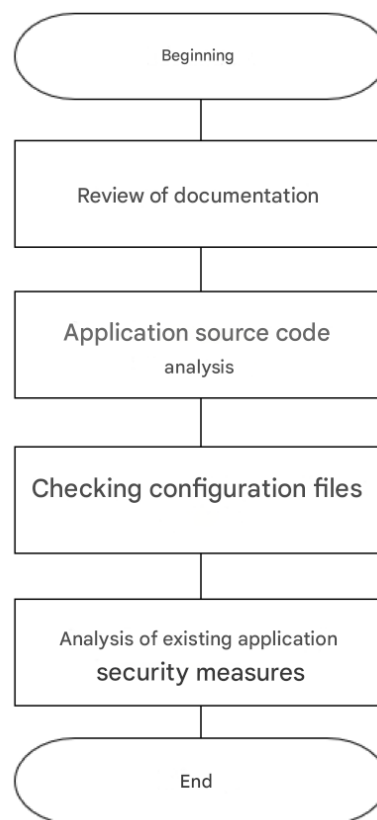


Figure 2.7 – Diagram of the information collection algorithm

The first step is to start by reviewing all documentation related to the mobile application. This includes technical specifications, functionality descriptions, security requirements, system architecture, etc.

But since testing is done on the principle of "black and white box", we have the source code of the application. It is necessary to analyze the source code, conduct an analysis of it to identify possible weaknesses and vulnerabilities.

To ensure that all security requirements have been properly implemented in the source code. Source code can be analyzed using static analysis tools such as SonarQube[18], Checkmarx [19], Fortify [20], as well as integrated development environments (IDEs) such as Android Studio [23], IntelliJ IDEA [24]. For code analysis verification, I prefer Android Studio [23] and IntelliJ IDEA [24] because of their ease of use and free access.

The next step is to check all configuration files to ensure that security settings and other parameters are configured properly. To check configuration files, you can use tools such as ConfigCheck [21] or built-in tools in development environments. The main advantage of ConfigChecker is its ability to retrieve configuration parameters from various sources. For example, it can read or check the values of:

- ☐ plain text files;
- ☐ xml files;
- ☐ java properties files;
- ☐ manifest files;
- ☐ java windows ini files;
- ☐ Windows registry;
- ☐ apache configuration files;
- ☐ ldif files.

After collecting information, you need to analyze the existing security measures already implemented in the mobile application and evaluate their effectiveness.

The analysis should identify potential security threats that may arise when operating a mobile application. This includes threats related to misuse, data leaks, network attacks, unauthorized access, etc.

2.2.2. Mobile application testing stage

Conducting penetration testing of a mobile application involves a number of stages and methods that help identify potential vulnerabilities and ensure a high level of security. The stages of testing a mobile application include the following steps (Fig. 2.8):

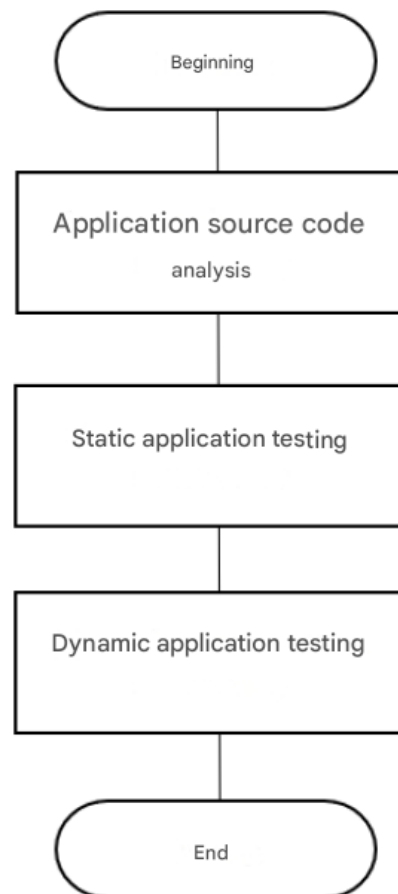


Figure 2.8 – Mobile application testing algorithm diagram

1) Analyzing the application source code: You need to perform the following steps: Checking local storage for sensitive data; Checking for sensitive data exposure through the user interface; Searching for sensitive information in automatically generated screenshots; Testing local storage to verify input data; Testing application permissions; Testing backups for sensitive data. All of this can

be done using Android Studio [23] and IntelliJ IDEA [24] due to their ease of use and free availability.

2) Static Application Testing: You need to examine the structure and functionality of a mobile application to identify potential vulnerabilities. To analyze a mobile application, you need to use analysis tools such as MobSF (Mobile Security Framework) and AndroBugs. When compared to these tools, MobSF: has advanced functionality for analyzing mobile application security; includes features such as vulnerability scanning, API testing, permission analysis, and many others; supports both Android and iOS, making it a universal tool for mobile application analysis; there is an active user community and constant updates as AndroBugs has not received as many updates compared to MobSF. The best tool is Mobile Security Framework (MobSF).

3) Dynamic testing:

- For dynamic application analysis, you should use Frida[27] and Drozer [28]. If you need flexibility and the ability to inject your own code: You should use Frida. It allows you to intercept and modify the dynamic behavior of applications. If you need a tool for auditing and vulnerability detection: Use Drozer. It provides vulnerability scanning and penetration testing, but does not have the flexibility of Frida. Sometimes it is best to use both tools. For example, scan for vulnerabilities using Drozer and then use Frida to further analyze and test the application while it is running.

- Testing for vulnerabilities in inter-component interactions. For this testing, you need to use traffic interception tools such as Burp Suite[25] and Wireshark[26]. The advantages of Burp Suite include: intercepting and modifying HTTP requests and responses, as well as detecting vulnerabilities; ideal for testing the security of web applications, but not so powerful for general analysis of all network traffic. In turn, Wireshark can: intercept and analyze any network traffic, including HTTP, TCP, UDP, etc.; allows you to analyze all aspects of network interactions in detail, which makes it useful for various tasks, including network

protocol analysis and troubleshooting communication problems. We need to analyze general network traffic and our task is not related to web applications, Wireshark may be a more suitable tool.

2.2.3. Results analysis stage and report creation process

This stage is crucial in the mobile application penetration testing process as it helps to understand the severity of the identified vulnerabilities and determine steps to fix them.

Analyzing the results of mobile application penetration testing can include steps such as:

- Analysis of identified vulnerabilities. Identified vulnerabilities should be analyzed in detail in terms of their potential impact on the security of the application and user data. Understanding the causes of vulnerabilities is key to subsequent remediation.
- Vulnerability Severity Assessment: Each vulnerability should be assessed in terms of its potential impact on system security. Vulnerability severity helps determine the priority order for remediation.

Risk Factors: Each outcome is assigned two factors to measure its risk. The factors are measured on a scale of 1 (low), 2 (medium) to 3 (high).

Impact: Indicates the impact of the identified factor on technical and business operations. It covers aspects such as confidentiality, integrity and availability of data or systems, as well as financial or reputational losses.

Likelihood: Indicates the potential for exploiting the research findings. It takes into account aspects such as the attacker's skill level and relative ease of use.

Low: Vulnerabilities that have either a very low business impact or a very high probability, or a very low probability but a high business impact are considered low severity vulnerabilities. Additionally, vulnerabilities that have both

a low impact and a low probability are considered low severity. The risk score ranges from 0 to 3.

Medium: Vulnerabilities with moderate business impact and probability are considered "Medium". This also includes vulnerabilities that have either a very high business impact combined with a low probability, or a low business impact combined with a very high probability. Risk score 3 to 6.

High: Vulnerabilities with a high or greater business impact and a high or greater probability of occurrence are considered high severity vulnerabilities. Risk score of at least 6.

Table 2.1 provides an example of a "Risk Score" calculation. A risk score can be defined as the product of impact and probability. For example, a risk score is calculated using the formula:

$$\text{Risk assessment} = \text{Impact} * \text{Probability}.$$

Table 2.1 – Definition of risk factors

Risk factor	Influence (Impact)	Probability (Likelihood)	Risk assessment
Low code quality	2 (medium)	3 (high)	6 (medium)
Insufficient data confidentiality	3 (high)	2 (medium)	6 (medium)
Using outdated libraries	2 (low)	2 (medium)	4 (low)
Insufficient authentication users	3 (high)	3 (high)	9 (high)

The report creation process consists of the following steps:

- Creating a report structure. The report should have a clear structure, including a description of the vulnerabilities found, their severity, possible consequences, and recommendations for remediation.
- Detailed description of vulnerabilities. Each vulnerability should be described in detail, including methods of exploitation, possible consequences, and

recommendations for remediation. Examples of attacks and appropriate security measures should be included.

- Recommendations for remediation. You should provide specific recommendations for remediating each vulnerability you find. Include detailed steps or recommendations for improving your code, application architecture, or security processes.

- Conclusions and recommendations. Finally, you need to draw conclusions from the analysis and provide general recommendations for further steps to improve the security of the application.



Figure 2.9 – Diagram of the report creation algorithm

Table 2.1 shows a matrix of the correspondence of testing tools to testing stages.

Table 2.1 – Matrix of correspondence of testing tools to testing stages

Processes Tools	Preparation	Information collection	Testing	Creating a report
Android Studio	+	+	–	+
Frida	–	+	+	+
Xposed	–	–	+	+
Drozer	+	+	+	+
MobXSS	–		+	+
MobSF	+	+	+	+
OWASP ZAP	–		+	+
Burp Suite Mobile Assistant	–	–	+	+
QARK	–	–	+	+
Andro-Bugs Framework	–	–	+	+

Table 2.2 – Matrix of compliance of testing tools with the OWASP MASVS security standard

MASVS Instruments	STORAGE		CRYPTO		AUTH			NETWORK		PLATFORM			CODE				RESILIENCE			
	1	2	1	2	1	2	3	1	2	1	2	3	1	2	3	4	1	2	3	4
Android Studio	+	+	+	+	+	+	+	–	–	–	+	+	+	+	+	+	+	–	–	+
Frida	+	–	–	–	–	–	–	–	–	–	+	–	–	–	–	+	–	+	+	+
Xposed	+	–	–	–	–	–	–	–	–	–	+	–	–	–	–	+	–	–	–	+
Drozer	+	+	–	–	–	–	–	–	–	–	+	+	–	–	–	–	+	–	+	+
MobXSS	+		–	–	–	–	–	–	–	–	+	–	–	–	–	–	–	–	–	–
MobSF	+	+	–	–	–	–	–	+	+	–	+	–	–	–	–	+	+	+	+	+
OWASP ZAP	–	–	–	–	–	–	–	+	+	+	+	–	–	–	–	–	–	–	–	–
Burp Suite Mobile Assistant	–	–	–	–	–	–	–	–	+	+	–	–	–	–	–	–	–	–	–	–
QARK	+	+	–	–	–	–	–	–	+	+	–	–	–	–	–	–	–	–	–	–
Andro-Bugs Framework	+	–	–	–	–	–	–	–	+	–	+	–	–	–	–		+	–	–	–

These MASVS testing tools demonstrate that they can test different aspects of mobile application security testing. Depending on your specific needs and the

platforms you are testing, you can choose the most suitable tools for mobile application penetration testing.

At the preparation and information collection stages, it is necessary to determine what technologies the application works with, such as Rest Api, gRPC, Local Database, etc. to determine test sets.

This section analyzed the methods and tools for testing a mobile application for penetration, and provided recommendations in the form of step-by-step stages of testing a mobile application for penetration.

3.3. IMPLEMENTATION OF INFORMATION TECHNOLOGY MOBILE APPLICATION PENETRATION TESTING

3.1. Mobile application penetration testing

For black box and white box penetration testing, Kali Linux [22] was chosen as the test environment and Android Studio [23] was chosen for analyzing the source code of the application. Testing will be conducted on a commercial application that is hosted on Play Market and is regularly updated.

At the information gathering stage, it was discovered using Android Studio that the following actions would need to be performed on the application (Table 3.1):

- verification of the configuration of cryptographic standard algorithms;
- checking local storage for confidential data;
- testing backups for confidential data;
- testing program permissions;
- testing local storage to verify entered data;
- testing data encryption in the REST API network.

Using Android Studio, you need to analyze the source code of a commercial application and find potential vulnerabilities.

Verify the configuration of cryptographic standard algorithms. This requires using Android Studio.

At analysis not was revealed cryptographic copies. ToExamples include:

- classes Cipher, Mac, MessageDigest, Signature;
- Key, PrivateKey, PublicKey, SecretKey interfaces;
- getInstance or generateKey functions;
- exclusion KeyStoreException, CertificateException,

NoSuchAlgorithmException;

- classes, What use packages java.security.*, javax.crypto.*, android.security.* and android.security.keystore.*.

Table 3.1 – Matrix compliance tools testing to application vulnerabilities found

Tests	Configuration check and cryptographic standard algorithms	Checking local about storage on confidential data	Checking backups on confidential data	Testing program permissions	Testing local storage to verify input data	Encryption testing data transmission in the network "REST API".
MASVS Instruments	CRYPTO-1	STORAGE-1	STORAGE-2	PLATFORM-1	STORAGE-1	NETWORK-1
Android Studio	+	+	+	+	+	+
MobXSS	+	+	–	–	–	–
Frida	+	–	+	+	–	–
Xposed	–	–	+	–	–	–
Drozer	+	+	+	+	+	–
MobSF	–	+	+	–	–	+
OWASP ZAP	–	–	–	–	–	+
Burp Suite Mobile Assistant	–	–	–	–	–	+
QARK	+	–	+	+	–	–
Andro-Bugs Framework	+	–	+	–	–	–

You need to use Android Studio to check local storage for sensitive data and local storage for input validation. Analysis revealed that the data is stored in SharedPreferences.

- Data storage is done using SharedPreferences (Figure 3.1). The SharedPreferences API is typically used to persistently store small collections of key-value pairs. The data stored in a SharedPreferences object is written to a plain XML text file (Figure 3.2). A SharedPreferences object can be declared public

(accessible to all applications) or private, in our case SharedPreferences is declared private.

```
single { provideSharedPref(androidApplication()) }  
  
fun provideSharedPref(app: Application): SharedPreferences {  
    return app.applicationContext.getSharedPreferences(  
        SHARED_PREFERENCE_NAME,  
        Context.MODE_PRIVATE  
    )  
}
```

Figure 3.1 – Shared Preferences implementation code in private mode

Data stored in SharedPreferences is not encrypted by default. This means that if someone gains access to the device files, they can easily view and modify the stored data. Trying to store user passwords using SharedPreferences is not secure. Passwords should be protected, and it is better to use more secure mechanisms such as KeyStore or authentication management libraries for this. SharedPreferences can be easily accessed by other applications or even the user if they have rooted devices.

SharedPreferences does not provide a mechanism to control access to data if it has access type `MODE_WORLD_READABLE` "allows other applications to read settings" or `MODE_WORLD_WRITEABLE` "allows other applications to write new settings". Other applications may have access to your SharedPreferences, which may lead to unwanted disclosure of sensitive information.

You can see that the data is stored in plain text.

- When opening the Logcat menu item in Android Studio, sensitive data logging was found, which displays a log of system messages, including messages that work with the REST API (Fig. 3.3).

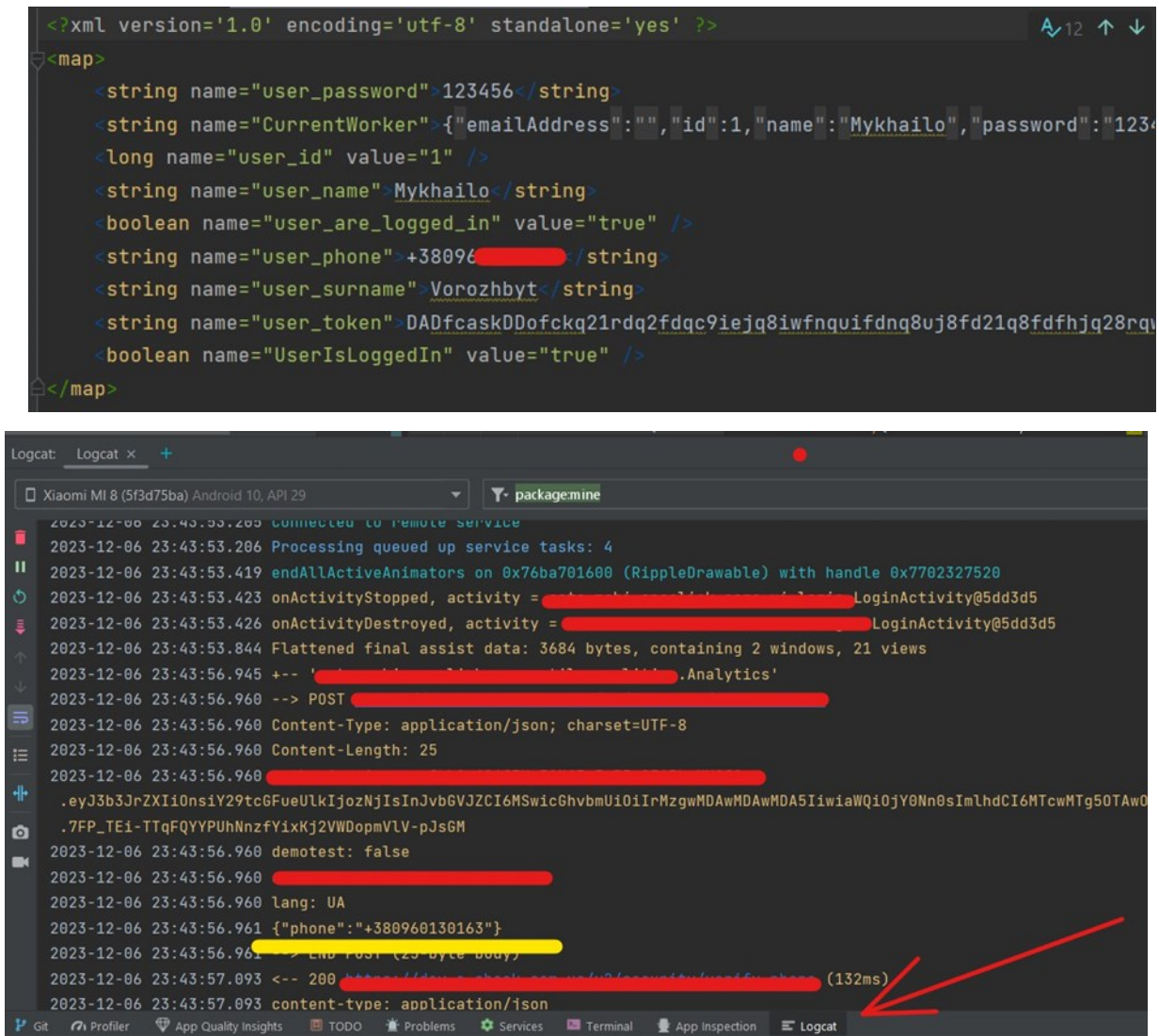


Figure 3.3 – Confidential user data

- Cached images were detected in the application file system at the path `data/app_name/cache/images` (Fig. 3.4).

Checking backups for sensitive data. To do this, you need use Android Studio.

- In the `AndroidManifest.xml` file, an `allowBackup` setting was found that was set to `allowed` mode (Figure 3.5). This setting allows anyone to backup the application data using `adb`. This will allow users who have enabled USB debugging to copy the application data from the device.

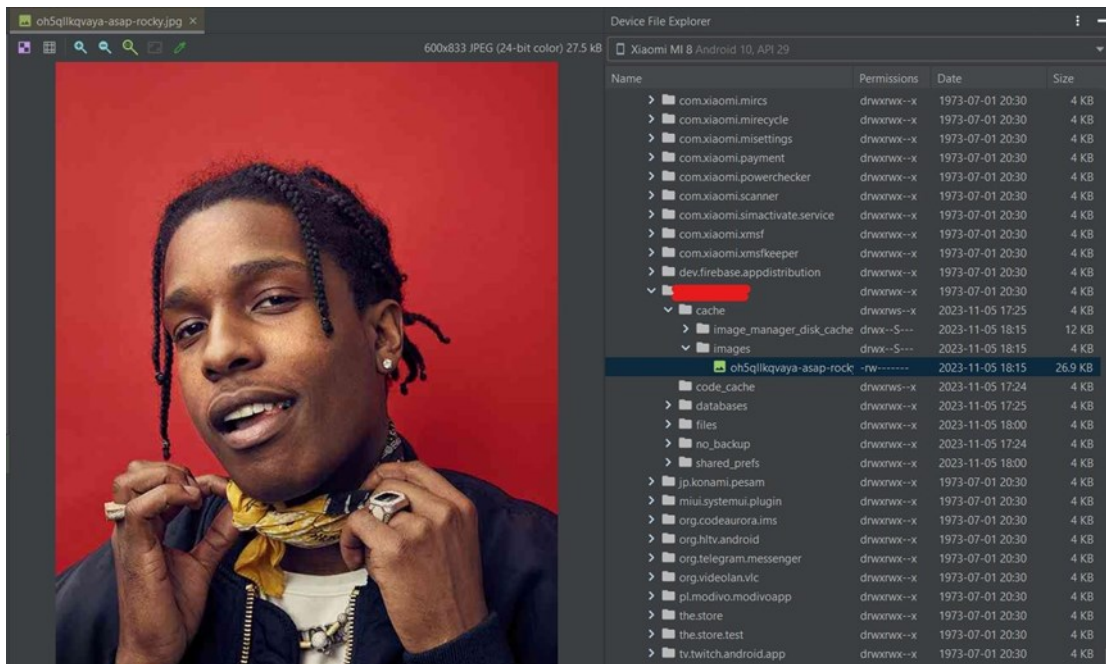


Figure 3.4 – View of cached image in application files

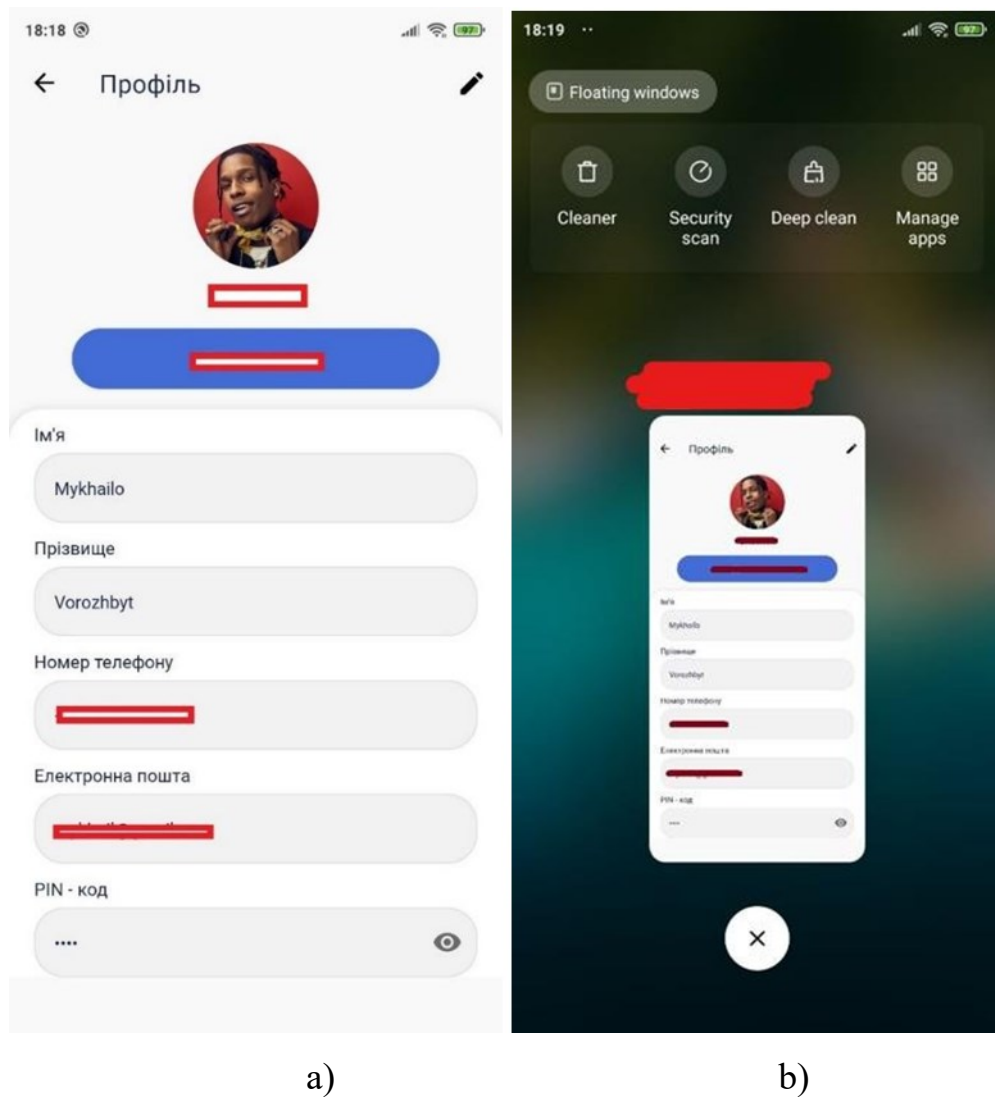
```

<manifest ... >
    ...
    <application android:allowBackup="true" ... >
        ...
    </application>
</manifest>

```

Figure 3.5 – Full backup capability in Android

– A screenshot of the current screen when an Android application goes into the background and is displayed in the list of open applications may lead to the leakage of confidential user information. (Fig. 3.6 (a), 3.6 (b)).



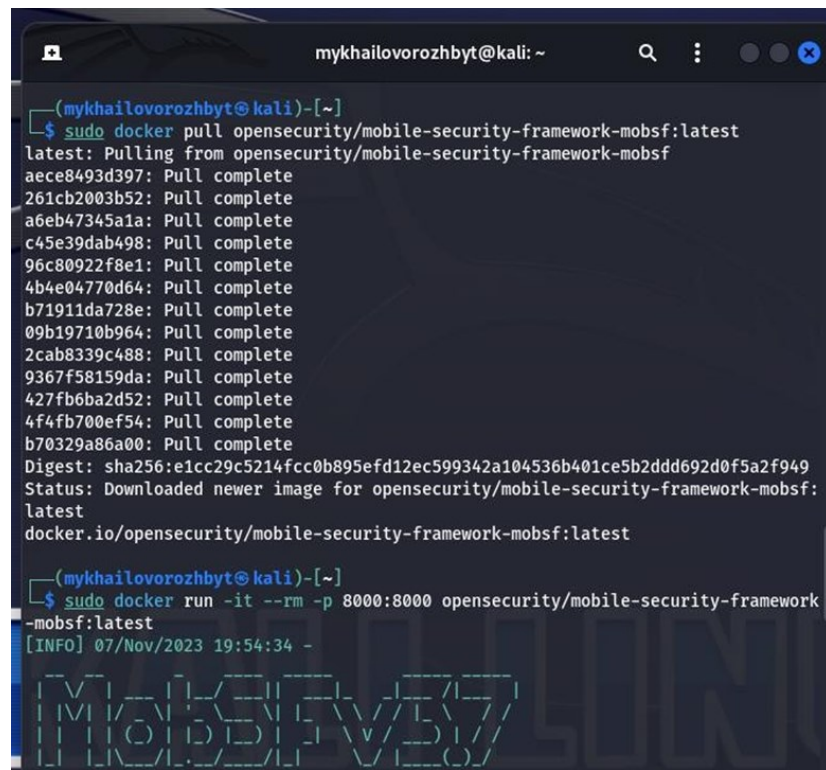
Figures 3.6 – Screen with user conference data: a – in the application, b – in minimized mode

Static analysis is based primarily on obtaining the source code, binary code, studying it to identify vulnerabilities, and analyzing the functioning of the program.

Using the MobSF (Mobile Security Framework) utility, we perform a static analysis of our application (Fig. 3.7).

After downloading the application APK, MobSF provides the following information:

Information about security rating, file information, and application information (Fig. 3.8).

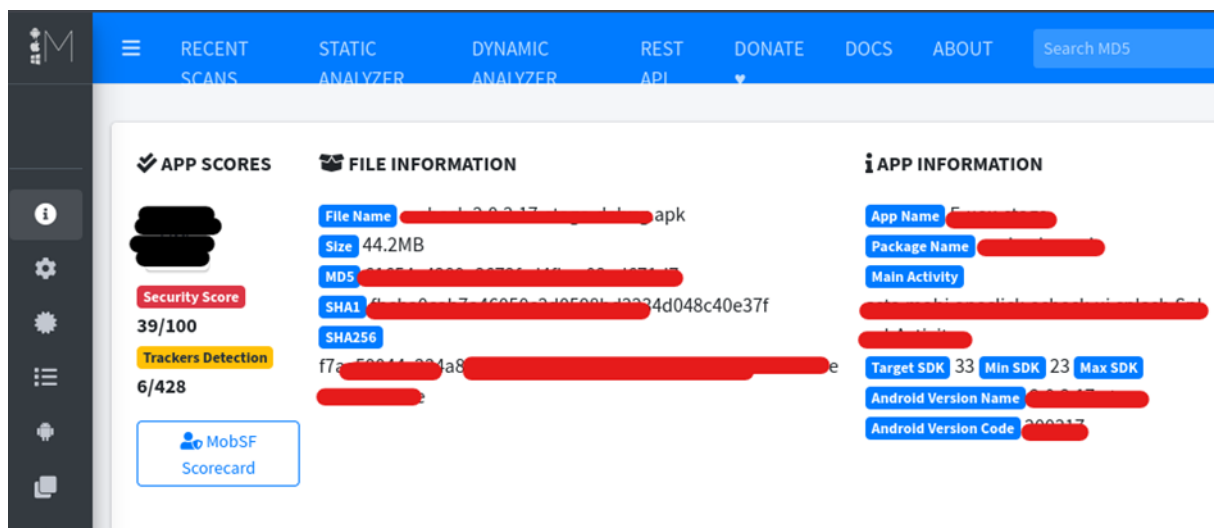


```

mykhailovorozhbyt@kali: ~
(mykhailovorozhbyt@kali)-[~]
$ sudo docker pull opensecurity/mobile-security-framework-mobsf:latest
latest: Pulling from opensecurity/mobile-security-framework-mobsf
aece8493d397: Pull complete
261cb2003b52: Pull complete
a6eb47345a1a: Pull complete
c45e39dab498: Pull complete
96c80922f8e1: Pull complete
4b4e04770d64: Pull complete
b71911da728e: Pull complete
09b19710b964: Pull complete
2cab8339c488: Pull complete
9367f58159da: Pull complete
427fb6ba2d52: Pull complete
4f4fb700ef54: Pull complete
b70329a86a00: Pull complete
Digest: sha256:e1cc29c5214fcc0b895efd12ec599342a104536b401ce5b2ddd692d0f5a2f949
Status: Downloaded newer image for opensecurity/mobile-security-framework-mobsf:latest
docker.io/opensecurity/mobile-security-framework-mobsf:latest

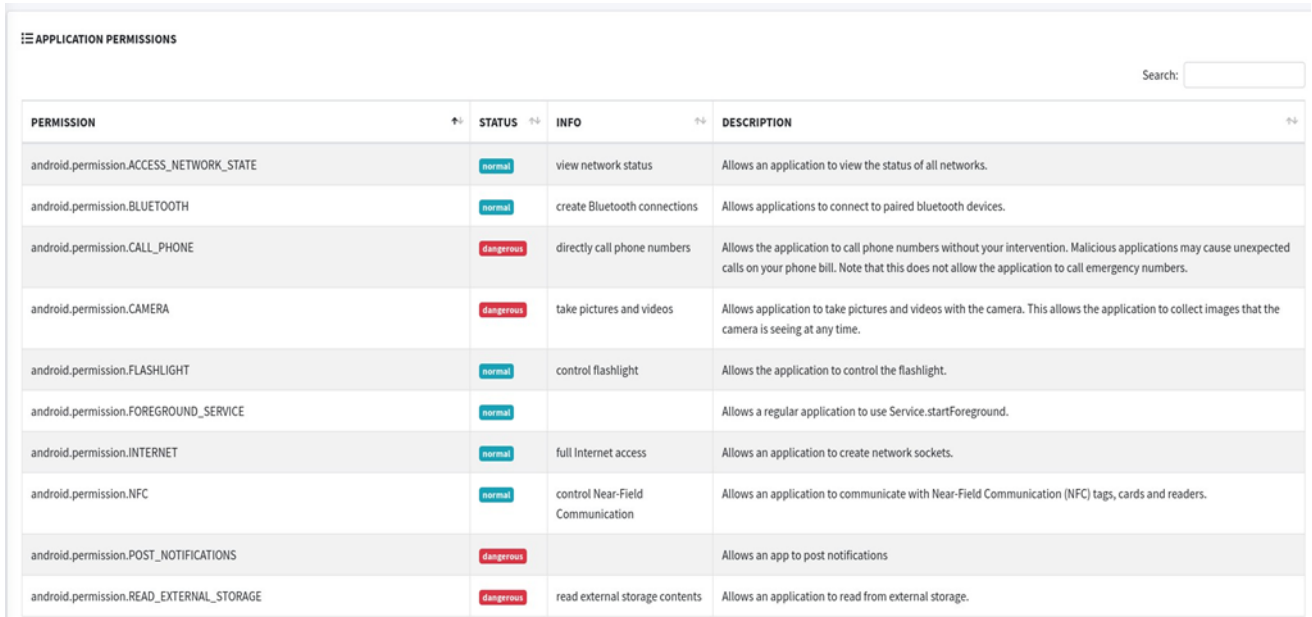
(mykhailovorozhbyt@kali)-[~]
$ sudo docker run -it --rm -p 8000:8000 opensecurity/mobile-security-framework-mobsf:latest
[INFO] 07/Nov/2023 19:54:34 -
  
```

Figures 3.7 – MobSF utility settings



Figures 3.8 – MobSF utility settings

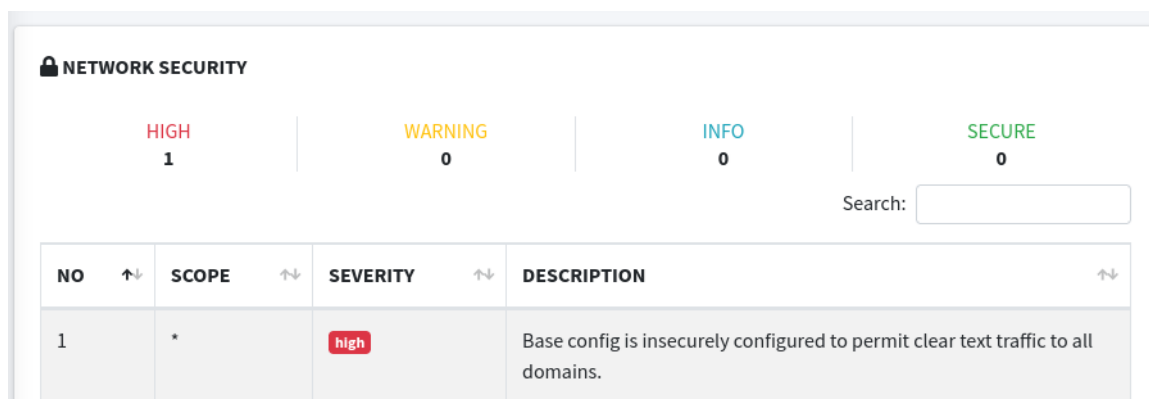
To check and test the permissions of an application, you need to use the MobSF utilities. Different types of permissions that the application will use are identified (Fig. 3.9).



PERMISSION	STATUS	INFO	DESCRIPTION
android.permission.ACCESS_NETWORK_STATE	normal	view network status	Allows an application to view the status of all networks.
android.permission.BLUETOOTH	normal	create Bluetooth connections	Allows applications to connect to paired bluetooth devices.
android.permission.CALL_PHONE	dangerous	directly call phone numbers	Allows the application to call phone numbers without your intervention. Malicious applications may cause unexpected calls on your phone bill. Note that this does not allow the application to call emergency numbers.
android.permission.CAMERA	dangerous	take pictures and videos	Allows application to take pictures and videos with the camera. This allows the application to collect images that the camera is seeing at any time.
android.permission.FLASHLIGHT	normal	control flashlight	Allows the application to control the flashlight.
android.permission.FOREGROUND_SERVICE	normal		Allows a regular application to use Service.startForeground.
android.permission.INTERNET	normal	full Internet access	Allows an application to create network sockets.
android.permission.NFC	normal	control Near-Field Communication	Allows an application to communicate with Near-Field Communication (NFC) tags, cards and readers.
android.permission.POST_NOTIFICATIONS	dangerous		Allows an app to post notifications
android.permission.READ_EXTERNAL_STORAGE	dangerous	read external storage contents	Allows an application to read from external storage.

Figures 3.9 – Application Permissions

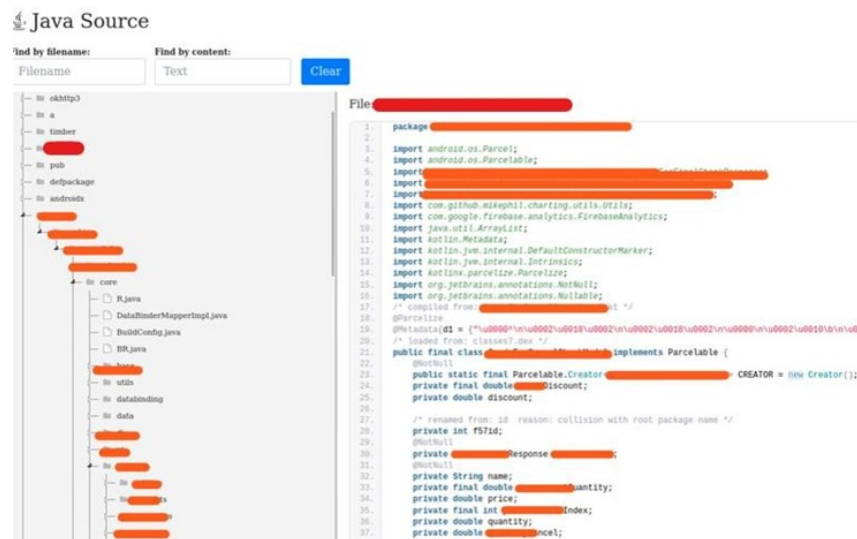
To test data encryption in the REST API network, you need to use the MobSF utilities. A configuration issue has been identified with network traffic encryption when using the REST API (Figure 3.10). Error description: The base configuration is not securely configured to allow clear text traffic to all domains.



NO	SCOPE	SEVERITY	DESCRIPTION
1	*	high	Base config is insecurely configured to permit clear text traffic to all domains.

Figures 3.10 – Data encryption on the network

The Application was found to be without proper code obfuscation. The decompiled Application shows all class names and parameters (Fig. 3.11).

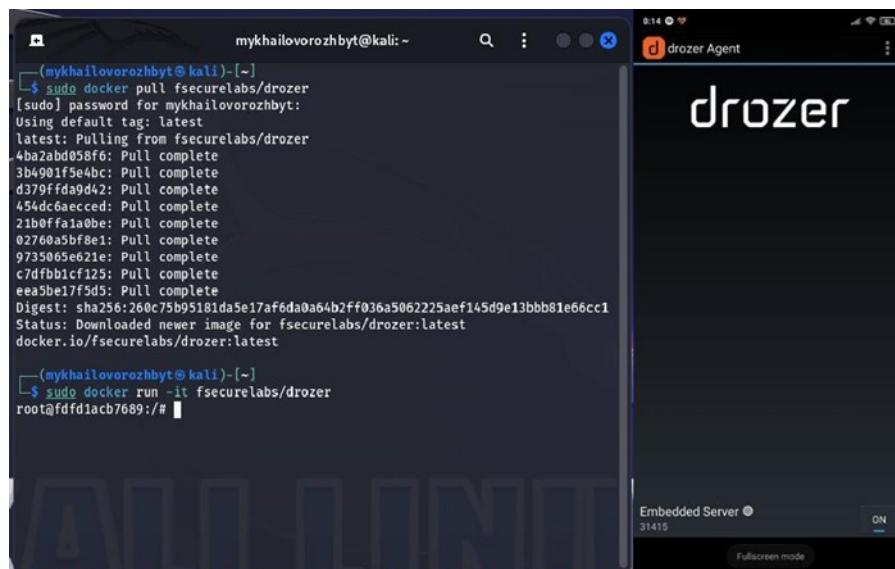


Figures 3.11 – Class without proper obfuscation

Using Kali Linux, using the drozer (Fig. 3.11 (a)) and drozer agent (Fig. 3.11 (b)) utilities, we will test the application for penetration:

First you need to go to the drozer console and configure the drozer agent with the following commands:

```
sudo docker pull fsecurelabs/drozer sudo docker run -it fsecurelabs/drozer
sudo adb forward tcp:31415 tcp:31415.
```



a)

b)

Figures 3.12 – Result of drozer agent configuration: a - in kali linux, b - in smartphone

We connected the drozer agent to our smartphone (Fig. 3.13) using the command: `drozer console connect --server 192.168.0.173`.



```
root@83b4d6d00f84:/# drozer console connect --server 192.168.0.173
Selecting ea7f94c675b439fc (Xiaomi MI 8 10)

..                               ...
..o..                           .r..
..a.. . . . . . . . . . . . . .nd
    ro..idsnemesisisand..pr
    .otectorandroidsneme.
    .,sisandprotectorandroids+.
    ..nemesisisandprotectorandroidsn:.
    .emesisisandprotectorandroidsnemes..
    ..isandp,..,rotectorandro,..,idsnem.
    .isisandp..rotectorandroid..snemesis.
    ,andprotectorandroidsnemesisandprotec.
    .torandroidsnemesisandprotectorandroid.
    .snemesisandprotectorandroidsnemesisan:
    .dprotectorandroidsnemesisandprotector.

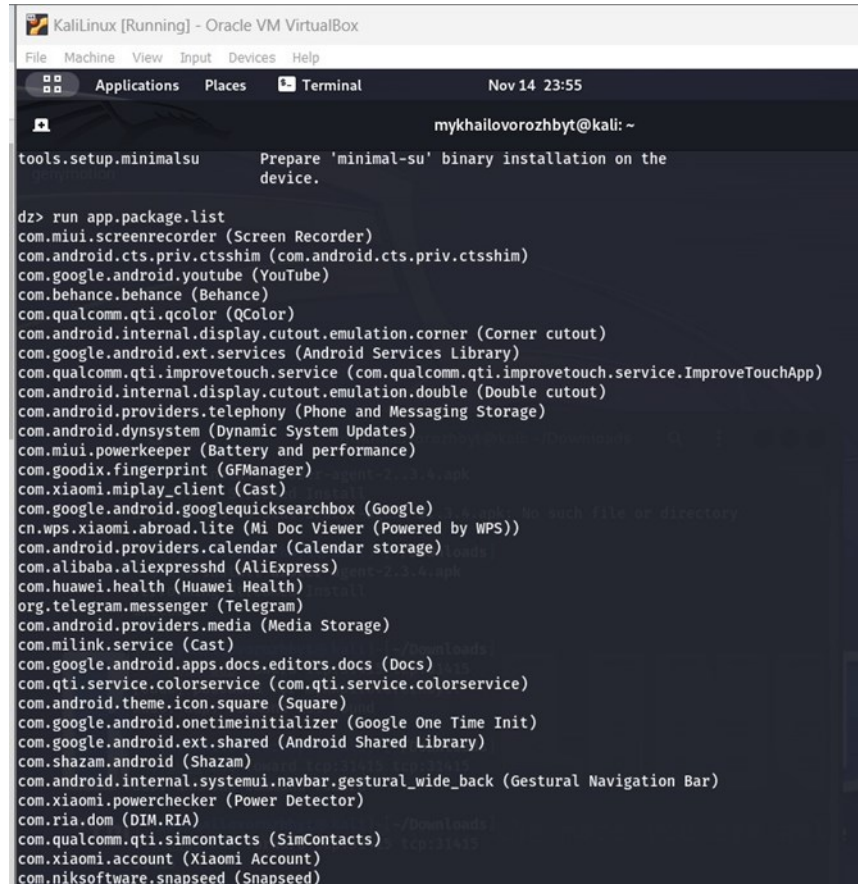
drozer Console (v2.4.4)
```

Figure 3.13 – Result of connecting drozer agent to smartphone via IP

Command `list` - shows a list of all drozer modules that can be running in the current session. Hides modules that you do not have the appropriate permissions to run (Fig. 3.13).

apktool	
	
mykhailovorozhbyt@kali: ~/Downloads	mykhailovorozhbyt@kali: ~/Downloads
dzx list	
app.activity.forintent	Find activities that can handle the given intent
app.activity.info	Gets information about exported activities.
app.activity.start	Start an Activity
app.broadcast.info	Get information about broadcast receivers
app.broadcast.send	Send broadcast using an intent
app.broadcast.sniff	Register a broadcast receiver that can sniff particular intents
app.package.attacksurface	Get attack surface of package
app.package.backup	Lists packages that use the backup API (returns true on FLAG_ALLOW_BACKUP)
app.package.debuggable	Find debuggable packages
app.package.info	Get information about installed packages
app.package.launchintent	Get launch intent of package
app.package.list	List Packages
app.package.manifest	Get AndroidManifest.xml of package
app.package.native	Find Native libraries embedded in the application.
app.package.shareduid	Look for packages with shared UIDs
app.providers.columns	List columns in content provider
app.provider.delete	Delete from a content provider
app.provider.download	Download a file from a content provider that supports files
app.provider.finduri	Find referenced content URIs in a package
app.provider.info	Get information about exported content providers
app.provider.insert	Insert into a Content Provider
app.provider.query	Query a Content provider
app.provider.read	Read from a content provider that supports files
app.provider.update	Update a record in a content provider
app.service.info	Get information about exported services
app.service.send	Send a Message to a service, and display the reply
app.service.start	Start Service
app.service.stop	Stop Service
auxiliary.webcontentresolver	Start a web service interface to content providers.
exploit.jdwp.connect	Open @jdwp-control and see which apps connect
exploit.pilfer.general.appprovider	Reads APW content provider
exploit.pilfer.general.settingsprovider	Reads Settings content provider
information.datetime	Print Date/Time
information.deviceinfo	Get verbose device information
information.permissions	Get a list of all permissions used by packages on the device
scanner.activity.browsable	Get all BROWSABLE activities that can be invoked from the web browser
scanner.misc.native	Find native components included in packages

Using the `run app.package.list` command, in the drozer console we find all the packages of various applications on the smartphone (Fig. 3.14).



```

KaliLinux [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Applications Places Terminal Nov 14 23:55
mykhailovorozhbyt@kali: ~
tools.setup.minimalsu Prepare 'minimal-su' binary installation on the
device.

dz> run app.package.list
com.miui.screenrecorder (Screen Recorder)
com.android.cts.priv.ctsshim (com.android.cts.priv.ctsshim)
com.google.android.youtube (YouTube)
com.behance.behance (Behance)
com.qualcomm.qti.qcolor (QColor)
com.android.internal.display.cutout.emulation.corner (Corner cutout)
com.google.android.ext.services (Android Services Library)
com.qualcomm.qti.improvetouch.service (com.qualcomm.qti.improvetouch.service.ImproveTouchApp)
com.android.internal.display.cutout.emulation.double (Double cutout)
com.android.providers.telephony (Phone and Messaging Storage)
com.android.dynsystem (Dynamic System Updates)
com.miui.powerkeeper (Battery and performance)
com.goodix.fingerprint (GFManager)
com.xiaomi.mipay_client (Cast)
com.google.android.googlequicksearchbox (Google)
cn.wps.xiaomi.abroad.lite (Mi Doc Viewer (Powered by WPS))
com.android.providers.calendar (Calendar storage)
com.alibaba.aliexpresshd (AliExpress)
com.huawei.health (Huawei Health)
org.telegram.messenger (Telegram)
com.android.providers.media (Media Storage)
com.milink.service (Cast)
com.google.android.apps.docs.editors.docs (Docs)
com.qti.service.colorsense (com.qti.service.colorsense)
com.android.theme.icon.square (Square)
com.google.android.onetimeinitializer (Google One Time Init)
com.google.android.ext.shared (Android Shared Library)
com.shazam.android (Shazam)
com.android.internal.systemui.navbar.gestural_wide_back (Gestural Navigation Bar)
com.xiaomi.powerchecker (Power Detector)
com.ria.dom (DIM.RIA)
com.qualcomm.qti.simcontacts (SimContacts)
com.xiaomi.account (Xiaomi Account)
com.niksoftware.snapseed (Snapseed)

```

Figure 3.14 – List of all packages on the smartphone

Using the `run app.service.info` command in the drozer console, we obtain information about the exported services and using the `run app.activity.start` – component `SplashActivity` command, we launch the `SplashActivity` activity (Fig. 3.15).

Using the `run app.package.attacksurface` command in the drozer console, we get the package attack surface (Figure 3.16).

Using the `run app.package.manifest app.prod` command in the drozer console, we obtain the contents of the `AndroidManifest.xml` file, Figure 3.17 (a) shows which permissions the application uses, and Figure 3.17 (b) shows the main activity of the application.

```

dz> run app.service.info -a [REDACTED]
Package: [REDACTED]
[REDACTED] LogoutJobIntentService
Permission: android.permission.BIND_JOB_SERVICE
[REDACTED] LogoutJobIntentService
Permission: android.permission.BIND_JOB_SERVICE

dz> run app.activity.info -a [REDACTED]
Package: e_check.prod
[REDACTED] SplashActivity
Permission: null
com.facebook.CustomTabActivity
Permission: null
[REDACTED] FederatedSignInActivity
Permission: [REDACTED] LAUNCH_FEDERATED_SIGN_IN

dz> run app.activity.start --component [REDACTED] SplashActivity
dz>

```

Figures 3.15 - The result of an attack on SplashActivity

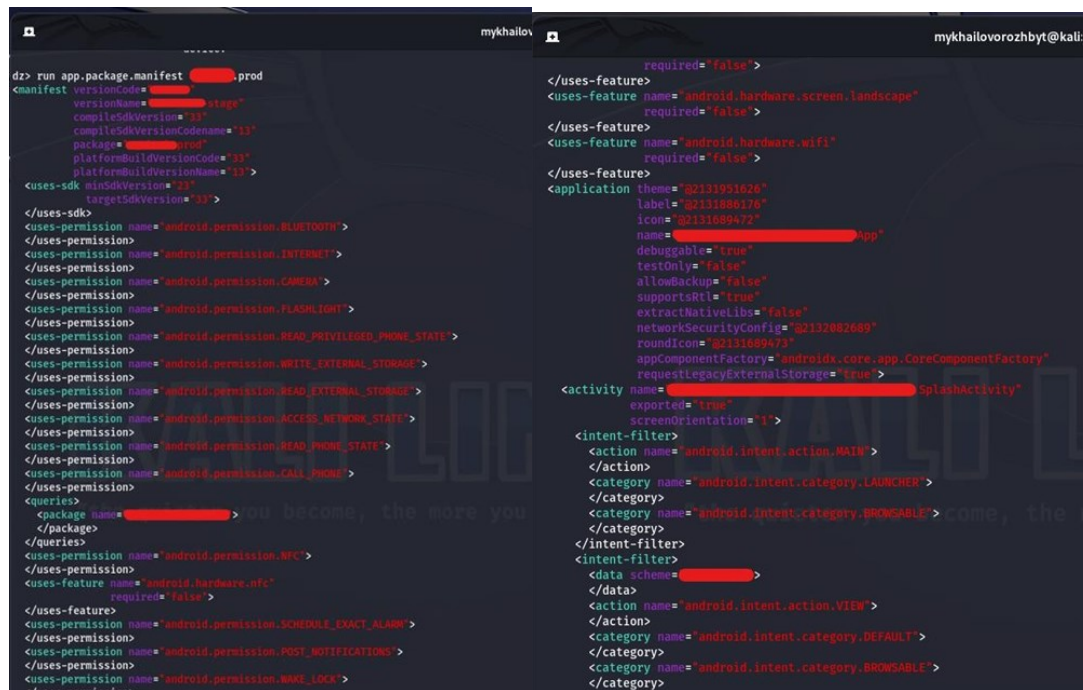
```

drozer Console (v2.4.4)

dz>
dz> run app.package.attacksurface [REDACTED]
Attack Surface:
  3 activities exported
  2 broadcast receivers exported
  0 content providers exported
  2 services exported
  is debuggable
dz>

```

Figures 3.16 – Application Packages



Figures 3.17 – Contents of the AndroidManifest.xml file: a – 1st part of the file, b – 2nd part of the file

The run scanner.provider.injection command in the drozer console is used to check for content provider injection vulnerabilities in an Android application.

```
dz> run scanner.provider.injection -a [redacted].prod
Scanning [redacted].prod...
Not Vulnerable:
content://com.facebook.orca.provider.MessengerPlatformProvider/versions
content:// Uri/
content:// [redacted] imagepicker.provider/
content:// [redacted] FacebookInitProvider/
content:// or file:// uri/
content:// or file:// uri/
content://com.facebook.app.FacebookContentProvider/
content:// [redacted] lifecycle-process/
content:// Uri
content:// [redacted] imagepicker.provider
content:// [redacted] AttributionIdProvider/
content:// [redacted] lifecycle-process
content://com.facebook.app.FacebookContentProvider
content://com.facebook.wakizashi.provider.AttributionIdProvider/
content://sms/inbox/
content:// [redacted] firebaseinitprovider/
content://com.facebook.katana.provider.AttributionIdProvider
content:// [redacted] firebaseinitprovider
content:// [redacted] AttributionIdProvider
content:// [redacted] provider.MessengerPlatformProvider/versions/
content://sms/inbox

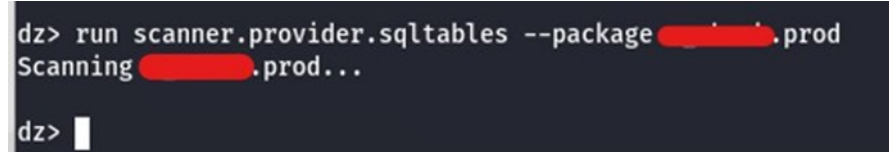
Injection in Projection:
No vulnerabilities found.

Injection in Selection:
No vulnerabilities found.
dz>
```

Figures 3.18 – Content Provider Vulnerability Injection

Result It can be seen that no vulnerabilities were detected using this attack injection of content providers into an Android application.

The `run scanner.provider.sqltables --package app.prod` command in the drozer console is used to extract information about SQLite database tables opened by content providers in an Android application.



```
dz> run scanner.provider.sqltables --package [redacted].prod
Scanning [redacted].prod...
dz>
```

Figures 3.19 – Result of SQLite table attack

It is clear that the attack was not successful, no SQLite tables were detected.

After testing the application, you need to analyze the application for penetration testing and provide recommendations for their neutralization.

3.3. Creating a report

The report should have a clear structure, including a description of the vulnerabilities found, their severity, possible consequences, and recommendations for remediation. Each vulnerability should be described, including methods of exploitation, possible consequences, and recommendations for remediation. Each vulnerability should be rated in terms of its potential impact on the security of the system. The severity of the vulnerabilities helps determine the order of priority for remediation. Specific recommendations for remediation for each vulnerability found should be provided. Add detailed steps or recommendations for improving the code, application architecture, or security processes.

The content of the mobile application penetration testing report includes the following subsections:

- ☐ 1.0 Risk Calculation Methodology
- ☐ 2.0 Summary

- ☐ 2.1 Sensitive data in logs
- ☐ 2.2 Lack of proper obfuscation of application code
- ☐ 2.3 Automatically generated screenshots of conference data actions
- ☐ 2.4 Lack of fault tolerance configuration
- ☐ 3.0 Conclusions

Section 1.0 describes the methodology for calculating risks.

Risk Factors: Each outcome is assigned two factors to measure its risk. The factors are measured on a scale of 1 (low), 2 (medium) to 3 (high).

Impact: Indicates the impact of the identified factor on technical and business operations. It covers aspects such as confidentiality, integrity and availability of data or systems, as well as financial or reputational losses.

Likelihood: Indicates the potential use of the research results. It takes into account aspects such as the level of attacker's skills the attacker's skill level and relative ease of use.

High: Vulnerabilities with a high or greater business impact and a high or greater probability of occurrence are considered high severity vulnerabilities. Risk score of at least 6.

Medium: Vulnerabilities with a moderate business impact and probability are considered Medium. This also includes vulnerabilities that have either a very high business impact combined with a low probability, or a low business impact combined with a very high probability. Risk score from 3 to 6.

Low: Vulnerabilities that have either a very low business impact or a very high probability, or a very low probability but a high business impact are considered low severity vulnerabilities. Additionally, vulnerabilities that have both a low impact and a low probability are considered low severity. The risk score ranges from 0 to 3.

Section 2.0 describes the Executive Summary.

A penetration test was conducted on the Android application PROD_APP.
INFORMATION ABOUT THE APPLICATION

Application name: PROD_APP Package name: prod

Main Activity: SplashActivity Target SDK: 33

Minimum SDK: 23

Android version name: 1.0.0.0.0-prod Android version code: 100000

APK signed v1 signature: True v2 signature: True v3 signature: False

This white box and black box assessment was conducted to identify loopholes in the application from a security perspective. This assessment aimed to identify vulnerabilities present in the application that could lead to injection, information leakage, and other risks that could cause potential business losses.

This report presents the results of the assessment.

Overall, several findings were identified during the evaluation, which will be described in detail in the "Conclusions" section.

Sections 2.1 – 2.4 describe all vulnerabilities found in the mobile application.

Section 2.1 describes confidential data in logs. Risk factors: Medium

Probability: Medium Impact: Medium

Attack Vector: Individual User

Description: The application uses a record to register sensitive information, such as the user's JWT token, user data, and user name and surname (Figure 3.3).

Consequences: If an attacker gains access to the phone (physical or remote), he can set up log collection.

Recommendation:

- Disable collection of sensitive data.

Reference: Sensitive Data - OWASP Mobile Application Security [32].

Section 2.1 describes the lack of proper obfuscation of the application code.

Risk Severity: Low Probability: Medium Impact: Low

Attack Vector: Individual User

Description: The application was compiled without proper code obfuscation. The decompiled program shows all class names and parameters. Code obfuscation

can significantly increase the complexity of understanding the logic of the program (Fig. 3.11).

Impact: An attacker can easily decompile the code, understand the program logic, and use this knowledge for further attacks.

Recommendation:

- Code obfuscation should be done before compilation. Reference: Sensitive Data - OWASP Mobile Application Security [33].

Section 2.3 describes automatically generated screenshots of conference data actions.

Risk Severity: Low Probability: Low Impact: Low

Attack Vector: Individual User

Description: A screenshot of the current activity is taken when an Android app goes to the background and is displayed for aesthetic purposes when the app returns to the foreground. However, this may lead to the leakage of sensitive information.

Impact: An attacker may be able to access a folder containing automatically generated screenshots, which may lead to information disclosure.

Recommendation:

- Installflag "SAFELY" for actions with confidential information

Reference: Finding Sensitive Information in Auto-Generated Screenshots - OWASP Mobile Application Security [34].

Section 2.4 describes the lack of fault tolerance configuration. Risk severity: Low

Probability: Low Impact: Low

Attack Vector: Individual User

Description: The program does not detect rooted, emulated, or unencrypted devices.

Impact: An attacker could exploit this flaw to test an application to find vulnerabilities or manipulate it.

Recommendation:

- Configure detection of rooted, emulated, unencrypted devices

References: Testing Root Detection - OWASP Mobile Application Security [35], Testing Emulator Detection - OWASP Mobile Application Security [35].

In section 3.0, you need to write a conclusion on the penetration testing of a mobile application.

During the penetration test of the Android application, several critical vulnerabilities were discovered that could pose serious risks to the security of the application and the user's confidential information.

4 SAFETY OF LIFE, BASIC LABOR PROTECTION

4.1. Labor protection requirements when working with electrical equipment

General provisions

The labor protection instructions for an electrician when performing repair and maintenance work on electrical equipment were developed in accordance with the Law of Ukraine “On Labor Protection” (Resolution of the Verkhovna Rada of Ukraine dated 10/14/1992 No. 2694-XII) as amended on 01/20/2018, based on the “Regulations on the Development of Labor Protection Instructions”, approved by the Order of the Labor Protection Supervision Committee of the Ministry of Labor and Social Policy of Ukraine dated January 29, 1998 No. 9 as amended on September 1, 2017, taking into account the “Rules for the Technical Operation of Consumer Electrical Installations”, approved by the Order of the Ministry of Fuel and Energy dated July 25, 2006. No. 258 (as amended by the order of the Ministry of Energy and Coal Industry of Ukraine dated 13.02.2012 No. 91, “Rules for the safe operation of electrical installations of consumers”, approved by the order of the State Supervision Service of Ukraine dated 09.01.1998 No. 4.

All provisions of this labor protection instruction apply to electricians of an educational institution who perform repair and maintenance work on electrical equipment.

Persons not younger than 18 years old who have undergone training in the specialty and who are also allowed to perform repair and maintenance work on electrical equipment independently are:

- a medical examination and do not have contraindications due to health to perform this work;

- introductory and primary workplace briefings on labor protection;

- training in safe methods and techniques of work;

testing of knowledge of the rules for installing electrical installations, safety rules for operating electrical installations, labor protection requirements;

when repairing and maintaining electrical equipment voltage up to 1000V have an electrical safety group not lower than III, and over 1000V - not lower than IV.

Electricians must know and comply with the requirements of the labor protection instructions when performing work on the repair and maintenance of electrical equipment, instructions for working with hand tools, power tools and ladders.

Electricians when performing work on the repair and maintenance of electrical equipment must comply with the requirements of the Rules for the safe operation of electrical installations of consumers and the Rules for the technical operation of electrical installations of consumers, and have an appropriate electrical safety group in accordance with the requirements of these Rules.

When performing work on the repair and maintenance of electrical equipment, the impact of the following harmful and dangerous production factors may be observed:

- fall from a height;
- electric shock;
- increased electric field strength;
- increased dustiness of the air in the work area;
- increased vibration level;
- insufficient illumination of the work area;
- physical overload;
- neuropsychic overload.

Electricians when performing repairs and maintenance of electrical equipment must use the following PPE:

- cotton overalls - for 12 months;
- gloves for - 3 months;

leather boots for - 24 months;

dielectric galoshes - on duty;

dielectric gloves - on duty;

dielectric mats - on duty.

An electrician when repairing and maintaining electrical equipment is obliged to:

keep his workplace clean and tidy;

comply with the Rules of Internal Labor Regulations;

be able to use personal and collective protective equipment, fire extinguishing equipment;

be able to provide first aid to accident victims;

know and comply with all requirements of regulatory acts on labor protection, fire protection rules and industrial sanitation.

immediately inform your immediate supervisor about any accident that occurred at work, about signs of an occupational disease, as well as about a situation that poses a threat to the life and health of people;

know the testing dates of protective equipment and devices, the rules for their operation, care and use. It is not allowed to use protective equipment and devices with an expired inspection period;

perform only the assigned work;

comply with the requirements of the equipment operating instructions;

know where the first aid facilities, primary fire extinguishing equipment, main and emergency exits, evacuation routes in the event of an accident or fire are located;

know the telephone numbers of a medical institution (103) and fire department (101).

An electrician may refuse to perform the work assigned to him if a production situation arises that poses a threat to his life and health of others, or to the environment, and report this to his immediate supervisor.

Smoking, drinking alcoholic beverages and other substances that have a narcotic effect on the human body are prohibited in the workplace.

In order to prevent injuries and the occurrence of dangerous situations, the following requirements must be observed: it is impossible to involve third parties in the work;

- do not start work if there are no conditions for its safe performance;

- perform work only on serviceable equipment, with serviceable devices and tools;

- if a malfunction is detected, immediately report it directly to the manager or eliminate them on their own, if this applies to their job duties;

- not to touch uninsulated or damaged wires;

- not to perform work that is not part of their professional duties.

Be able to provide first aid for bleeding, fractures, burns, electric shock, sudden illness or poisoning.

Follow the rules of personal hygiene:

- outerwear, hats and other personal belongings should be left in the wardrobe;

- work in clean overalls;

- eat in the designated place.

Be able to correctly use PPE and collective protection equipment, primary fire extinguishing equipment, fire-fighting equipment, know where they are.

Persons who violate this labor protection instruction for an electrician when performing repair and maintenance work on electrical equipment shall bear disciplinary, administrative, material and criminal liability in accordance with the current legislation of Ukraine.

Safety requirements before starting work

- Wear overalls, inspect and prepare the workplace, remove unnecessary objects.

Remove unauthorized persons from the work area and clear the workplace of foreign materials and other objects, fence off the work area and install safety signs.

Make sure that the workplace is sufficiently illuminated, that there is no electrical voltage on the repaired equipment.

Inspect the serviceability of switches, electrical outlets, power cords, electrical wires, connecting cables, make sure that PPE (personal protective equipment) and warning devices (dielectric gloves, safety glasses, galoshes, mats, etc.) are available and in good condition.

When working with a tool, it is necessary to make sure that it is in good condition, that there is no mechanical damage to the insulating coating and that the tool has been tested in a timely manner.

Inspect the workplace for compliance with fire safety requirements and for adequate workplace lighting.

If you find any deficiencies or violations in electrical and fire safety, immediately report them to your immediate supervisor.

4.2. Safety requirements during work

When performing your duties, an electrician must have a certificate of knowledge testing on labor protection. In the absence of a certificate or a certificate with an expiration date, the employee is not allowed to work.

Work in electrical installations is divided into 3 categories in terms of safety measures:

- with voltage relief;
- without voltage relief on or near live parts;
- without voltage relief away from live parts.

Employees performing special types of work that require additional safety requirements must be trained in the safe conduct of such work and have a corresponding entry in the knowledge test certificate.

An employee who serves electrical installations assigned to him with a voltage of up to 1000 V alone must have a III group on electrical safety.

When performing work in electrical installations, it is necessary to carry out organizational measures that ensure the safety of work:

- draw up work orders-permits, orders in accordance with the list of works performed in the order of current operation;

- prepare workplaces;

- admittance to work;

- exercise control over the performance of work;

- transfer to another workplace;

- establish breaks in work and its completion.

To prepare the workplace for work that requires voltage relief, it is necessary to apply, in a certain order, the following technical measures:

- perform the necessary shutdowns and take all measures that exclude erroneous or unauthorized switching on of switching equipment;

- hang prohibition posters on the drives of manual and remote control keys of switching equipment;

- check for the absence of voltage on conductive parts that must be grounded to protect people from electric shock;

- install grounding (turn on grounding knives, use portable grounding);

- install fences, if necessary, near workplaces or live parts that remain under voltage, and also hang safety posters on these fences.

- depending on local conditions, fence live parts before or after their grounding.

At least two workers should work without removing voltage on or near live parts, one of whom, the work supervisor, must have group IV; the others must have group III with mandatory registration of the work with a work permit or order.

When removing and installing fuses under voltage in electrical installations with voltage up to 1000 V, all loads connected to the specified fuses should be disconnected in advance; use insulating pliers or dielectric gloves, and if there are open fuse inserts, then safety glasses.

Work using ladders must be carried out by two people, one of the workers must be at the bottom. Standing on boxes or other objects is prohibited. P

When installing extension ladders on beams, elements of metal structures, etc., the upper and lower parts of the ladder should be securely fixed to the structures.

During maintenance and repair of electrical installations, it is prohibited to use metal ladders.

4.3. Safety requirements after completion of repair and maintenance of electrical equipment

Disconnect (disconnect) the necessary electrical equipment, power tools from the network.

Clean up the workplace, remove parts, material, garbage and waste to special places.

Remove all tools and devices to the designated place.

Remove and remove overalls, PPE, wash hands thoroughly.

Inspect the workplace for compliance with all fire protection requirements.

Notify your immediate supervisor of any deficiencies and malfunctions that occurred during the work. Record this in the operational log.

Safety requirements in emergency situations

In case of fire:

turn off electrical equipment, supply and exhaust ventilation, if any;

notify the fire department by calling 101 and report this to your supervisor, and in his absence, to another official;

proceed to eliminate the source of the fire, using the fire extinguishing agents provided for this purpose. Extinguish electrical equipment that is under voltage can only be extinguished with carbon dioxide fire extinguishers of the OU type or sand. It is prohibited to extinguish them with water or foam fire extinguishers.

The electrician must remember that in the event of a sudden power outage, it can be supplied again without warning.

Mechanisms and devices should be quickly turned off:

in the event of a sudden power outage;

if their further operation threatens the safety of employees;

in the event of a feeling of electric current when touching metal parts of the starting equipment;

in case of sparking;

at the slightest sign of ignition, smoke, or a burning smell;

if an unfamiliar noise appears.

In the event of a short circuit in the power supply network, it is necessary to de-energize the equipment and notify your immediate supervisor.

If an electric shock occurs, the victim should be released from the action of the electric current, for which purpose the electrical network should be turned off or the victim should be disconnected from the conductive parts using dielectric protective equipment and other insulating items and objects (dry clothing, dry pole, rubberized material, etc.), or the wire should be cut (chopped) with any tool with an insulating handle, carefully, without causing additional injuries to the victim. Before the arrival of a medical worker, it is necessary to provide the victim with first aid.

In the event of accidents (injury to a person), immediately notify the immediate supervisor.

CONCLUSIONS

During the qualification work, the tasks were analyzed, the objectives were defined, and the requirements were set, following which the set goal can be achieved. The relevance of solving the problem of mobile application penetration testing in cybersecurity was analyzed.

An analysis of methods and existing tools for mobile application penetration testing was conducted. It was determined that the problem is quite relevant. A decision was made to improve the mobile application penetration testing process by developing information technology.

Methods and tools for testing mobile applications for penetration were analyzed. After analyzing and comparing methods and tools for testing mobile applications for penetration, it was determined that OWASP is the best method for testing mobile applications for penetration, but the disadvantage of each method is that they do not provide information on how to create a test report and which method and tool should be used for testing different types of applications.

The developed information technology was tested. After testing, a number of errors were identified, which included a description of the severity of the threat risk, the probability of the threat, the impact on the application, and recommendations were provided on what to do to avoid this or that threat to the security of the application and user data.

This information technology can be used for penetration testing of mobile applications. Information technology consists of different stages, so it is easy to adapt it to the required tasks, expand or change the functionality.

REFERENCES

1. Mobile App Security: A Comprehensive Guide to Penetration Testing: Website. URL: <https://qualysec.com/mobile-application-penetration-testing-a-guide/> (date of application 12.2024).
2. Android Platform architecture: website. URL: <https://developer.android.com/guide/platform> (date of application 2024).
3. Android OperatingSystem: website. URL: <https://www.javatpoint.com/android-operating-system> (date of application 12.2024).
4. Android <permission>: website. URL: <https://developer.android.com/guide/topics/manifest/permission-element> (date of application 12.2024).
5. What are the Different Protection Levels in Android Permission? >: website. URL: <https://www.geeksforgeeks.org/what-are-the-different-protection-levels-in-android-permission/> (access date 02.2025).
6. Permissions on Android: website. URL: <https://developer.android.com/guide/topics/permissions/overview> (date of application 03.2025).
7. Smartphone Security and Privacy: A Survey on APTs, Sensor-Based Attacks, Side-Channel Attacks, Google Play Attacks, and Defenses: Website. URL: <https://www.mdpi.com/2227-7080/11/3/76> (date of application 03.2025).
8. About Android App Bundles: website. URL: <https://developer.android.com/guide/app-bundle> (date of application 04.2025).
9. How Google Play works:website. URL: <https://play.google/howplayworks/> (date of application 04.2025).
10. Mobile App Security: A Comprehensive Guide to Penetration Testing: Website. URL: <https://qualysec.com/mobile-application-penetration-testing-a-guide/> (date of application 04.2025).

11. OWASP Mobile Application Security Mobile Application Security Testing - Penetration Testing (aka Pentesting): Website. URL: <https://mas.owasp.org/MASTG/General/0x04b-Mobile-App-Security-Testing/#penetration-testing-aka-pentesting> (date of application 04.2025).
12. OWASP Mobile Application Security: website. URL: <https://mas.owasp.org/> (date of application 04.2025).
13. NIST SP 800-163 Rev. 1 - Vetting the Security of Mobile Applications: website. URL: <https://csrc.nist.gov/pubs/sp/800/163/r1/final> (date of application 05.2025).
14. Mobile Application Security Testing (MAST) - Challenges & Tools: Website. URL: <https://snyk.io/learn/application-security/mobile-application-security/mast-mobile-app-sec-testing/> (date of application 05.2025).
15. Vetting the Security of Mobile Applications: NIST Publishes SP 800-163 Revision 1: website. URL: <https://csrc.nist.gov/news/2019/nist-publishes-sp-800-163-rev-1> (accessed 05.2025).
16. NIST SP 800-163 Rev. 1 (Initial Public Draft): website. URL: <https://www.iso.org/standard/75652.html> (date of application 05.2025).
17. PCI DSS (Payment Card Industry Data Security Standard): website. URL: <https://csrc.nist.gov/pubs/sp/800/163/r1/ipd> (accessed 05.2025).
18. Sonarhttps: website. URL: www.sonarsource.com/ (date of application 05.2025).
19. Checkmarx: Website. URL: <https://checkmarx.com/> (date of application 05.2025).
20. Fortify: website. URL: <https://www.microfocus.com/en-us/cyberres/application-security> (date of application 05.2025).
21. config-check: website. URL: <https://www.npmjs.com/package/config-check> (date of application 05.2025).
22. Kali - The most advanced Penetration Testing Distribution: website. URL: <https://www.kali.org/> (date of application 06.2025).

23. Android Studio: Website. URL: <https://developer.android.com/studio>(date of application 06.2025).
24. IntelliJ IDEA – the Leading Java and Kotlin IDE: Website. URL:<https://www.jetbrains.com/idea> (date of application 06.2025).
25. Burp Suite - Application Security Testing Software: Website. URL:<https://portswigger.net/burp> (date of application 06.2025).
26. Wireshark: Website. URL:<https://www.wireshark.org/> (date of application 06.2025).
27. Frida. Dynamic instrumentation toolkit for developers, reverse-engineers, and security researchers.: website. URL: <https://frida.re/>(date of application 06.2025).
28. Drozer. Comprehensive security and attack framework for Android.: website. URL: <https://labs.withsecure.com/tools/drozer>(date of application 06.2025).
29. EncryptedSharedPreferences.: website. URL: <https://developer.android.com/reference/androidx/security/crypto/EncryptedSharedPreferences> (accessed 06.2025).
30. Android Keystore system.: website. URL: <https://developer.android.com/privacy-and-security/keystore> (accessed 06.2025).
31. Glide: website. URL: <https://github.com/bumptech/glide> (date(application 06.2025).
32. Sensitive Data - OWASP Mobile Application Security.: website. URL:<https://mas.owasp.org/MASTG/tests/android/MASVS-STORAGE/MASTG-TEST-0003/>(date of application 06.2025).
33. Testing Obfuscation - OWASP Mobile Application Security.: Website. URL: <https://mas.owasp.org/MASTG/tests/android/MASVS-RESILIENCE/MASTG-TEST-0051/>(date of application 06.2025).
34. Finding Sensitive Information in Auto-Generated Screenshots - OWASP Mobile Application Security: Website. URL:

<https://mas.owasp.org/MASTG/tests/android/MASVS-PLATFORM/MASTG-TEST-0010/#static-analysis>.

35. Testing Root Detection - OWASP Mobile Application Security: Website. URL: <https://mas.owasp.org/MASTG/tests/android/MASVS-RESILIENCE/MASTGTEST-0045/> (date of application 06.2025).

36. Testing Emulator Detection - OWASP Mobile Application Security: website. URL: <https://mas.owasp.org/MASTG/tests/android/MASVS-RESILIENCE/MASTG-TEST-0049/> (accessed 06.2025).

37. Mobile Security Framework (MobSF): website. URL: <https://github.com/MobSF/Mobile-Security-Framework-MobSF> (accessed 06.2025).

38. AppUse PRO - Let AppUse Pro do the work for you! : website. URL: <https://appsec-labs.com/appuse/> (access date 06.2025).

39. Quick Android Review Kit: website. URL: github.com/linkedin/qark (accessed 06.2025).

40. AndroBugs Framework: Website. URL: https://github.com/AndroBugs/AndroBugs_Framework (access date 06.2025).

41. Xposed Framework: What It Is and How to Install It: website. URL: lifewire.com/xposed-framework-4148451 (accessed 06.2025)

42. Burp Suite Mobile Assistant: website. URL: yw9381.github.io/Burp_Suite_Doc_en_us/burp/documentation/desktop/tools/mobile-assistant/index.html (accessed 06.2025).

43. Y. Leshchyshyn, L. Scherbak, O. Nazarevych, V. Gotovych, P. Tymkiv and G. Shymchuk, «Multicomponent Model of the Heart Rate Variability Change-point,» 2019 IEEE XVth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH), Polyana, Ukraine, 2019, pp. 110-113, doi: 10.1109/MEMSTECH.2019.8817379

44. Lytvynenko, S. Lupenko, O. Nazarevych, G. Shymchuk and V. Hotovych, «Mathematical model of gas consumption process in the form of cyclic

random process,» 2021 IEEE 16th International Conference on Computer Sciences and Information Technologies (CSIT), LVIV, Ukraine, 2021, pp. 232-235, doi: 10.1109/CSIT52700.2021.9648621

45. Bodnarchuk, I., Kunanets, N., Martsenko, S., Matsiuk, O., Matsiuk, A., Tkachuk, R., Shymchuk, H.: Information system for visual analyzer disease diagnostics. CEUR Workshop Proceedings 2488, pp. 43-56 (2019).

46. Шимчук Г. В. Дослідження методів захисту відомих хмарних платформ : кваліфікаційна робота освітнього рівня „Магістр“ „125 – Кібербезпека“ / Г. В. Шимчук. – Тернопіль : ТНТУ, 2022. – 74 с.

47. Методичні вказівки розроблені у відповідності з навчальним планом для студентів освітнього рівня бакалавр спеціальності 126 «Інформаційні системи та технології» / Уклад.: О. Б. Назаревич, Г. В. Шимчук, Н. М. Шведа. – Тернопіль : ТНТУ 2020. – 22 с.

48. V. Kozlovskyi, Y. Balanyuk, H. Martyniuk, O. Nazarevych, L. Scherbak and G. Shymchuk, «Information Technology for Estimating City Gas Consumption During the Year,» 2022 International Conference on Smart Information Systems and Technologies (SIST), Nur-Sultan, Kazakhstan, 2022, pp. 1-4, doi: 10.1109/SIST54437.2022.9945786.