



**Politecnico
di Torino**

Politecnico di Torino

Master's degree in Cybersecurity Engineering

A.a. 2025/2026

Graduation Session March 2026

**Overcoming the Limitations of
Automation in Spyware Detection:
Proactive Methodologies for Host
and Network Forensics**

Supervisors:

Andrea Atzeni
Paolo Dal Checco

Candidate:

Gabriele Gambaro

*«If you want to keep a secret,
you must also hide it from yourself.»*

— George Orwell, 1984

Acknowledgements

Sincere thanks to those who have supported me over the years and contributed to my personal and professional growth. Thanks to my family, friends, and Giorgia for all their support and motivation. Special thanks to my academic supervisors, Professor Andrea Atzeni and Professor Paolo Dal Checco, for allowing me to pursue such interesting research and for providing the necessary support, offering critical insights that allowed me to enhance the quality of my final work.

Table of Contents

List of Figures	VIII
1 Aim of the work	1
2 Background	5
2.1 Mobile Spyware: An Overview	5
2.1.1 Definition of spyware	5
2.1.2 Impact on user privacy and security	13
2.1.3 Legal background	13
2.1.4 Concern about social control	14
2.1.5 Stages of the attack	15
2.1.6 Infection vectors	17
2.1.7 Indicator of compromise (IOC)	23
2.2 Known Spyware Tools	25
2.2.1 Case Studies State-sponsored spyware	26
2.2.2 Commercial monitoring application	28
3 State-of-the-Art Spyware Detection	37
3.1 Spyware Detection Techniques	37
3.1.1 Dynamic Analysis	39
3.1.2 Tools for Network Dynamic Analysis	43
3.1.3 Static Analysis	48
3.1.4 Tools For Static Analysis	63
3.2 State-of-the-Art Spyware Detection Tools	65
3.2.1 SpyGuard	65
3.2.2 Interceptor	70
3.2.3 M2 Bridge New	71
3.2.4 Mobile Verification Toolkit (MVT)	72
3.2.5 IShutdown utility	74

4	Experimental Evaluation	76
4.1	Spyguard tests	76
4.2	Spyguard limitations and potential enhancements	85
4.2.1	Clustering technique	85
4.2.2	Solutions based on LLMs	96
4.2.3	VPNs identifier	97
4.2.4	Inducement operations	102
4.3	Test of MVT on Android	105
4.4	Test of MobSF	111
4.5	Spyware Discovery via ADB-Based Forensic Analysis	116
5	Conclusions and Future Perspectives	131
A	Command Line Reference	135
A.1	Network Log Acquisition and Processing	135
A.2	SSL Pinning Bypass (Frida and ADB)	136
A.3	ADB Telemetry	136
	Bibliography	138

List of Figures

2.1	A sponsored advertisement of a stalkerware application	7
2.2	Pegasus SMS command message. Source: [10]	8
2.3	Spyware capabilities	9
2.4	Attacker-controlled caavn[.]org domain imitating a news article from the South China Morning Post [41]	20
2.5	Obfuscation technique used by iKeyMonitor to hide its icon	31
2.6	iKeyMonitor dashboard accessed on the monitored device.	31
2.7	Two screenshots related to iKeyMonitor application	31
2.8	iKeyMonitor dashboard accessible via dedicated webpage	32
2.9	MTF installation succeeds	34
2.10	MTF downloads the demon after that Android Play Protect has been disabled	34
2.11	Two screenshots related to MTF installation	34
2.12	Obfuscation technique used by MTF to hide its icon	35
2.13	Installation precess: MTF request all the necessary permissions	35
2.14	MTF dashboard accessible via dedicated webpage	36
3.1	Classification system for mobile forensic acquisition	38
3.2	Random forest algorithm representation. Source [67].	47
3.3	Operating system and native app files. Source: [84].	60
3.4	Comparison of resource access levels with and without root privileges.	62
3.5	Spyguard backend page	66
3.6	Spyguard alert screen. Source: [89]	69
3.7	Spyguard pipeline	70
3.8	Representative photo of M2 Bridge New. Source: [95].	71
4.1	SSL Pinning. Source [105].	79
4.2	Elbow method plot for choosing the number of clusters	92
4.3	DBSCAN cluster plot (eps=2, min_samples=5)	94
4.4	DBSCAN results (eps=2, min_samples=5)	94
4.5	DBSCAN cluster plot (eps=2.5, min_samples=5)	95

4.6	DBSCAN results (eps=2.5, min_samples=5)	95
4.7	IKeyMonitor without VPN	100
4.8	IKeyMonitor with PlanetVPN	101
4.9	IKeyMonitor with TurboVPN	102
4.10	Screenshot of the MobSF home page.	111
4.11	MobSF app informations	126

Chapter 1

Aim of the work

The goal of this thesis is to analyse the state-of-the-art tools and techniques that digital forensic experts and law enforcement employ to identify mobile spyware and perform forensic analysis. Furthermore, after an in-depth analysis of existing technologies, the thesis aims to highlight potential problems in the tools that are adopted and propose possible solutions to improve them. The analysis considers the different options for mobile spyware analysis, focusing on both the major operating system vendors, Android and iOS.

The thesis aims to investigate the modern landscape of mobile spyware. However, due to the limited availability of resources related to state-sponsored spyware and the inaccessibility of their exploits, all tests and most considerations are based on commercially available spyware.

In a time when smartphones serve as the main source of financial and personal data, geolocation information, health measurements, and social contact, the existence of surveillance apps poses a serious threat to people's privacy and security.

According to recent statistics, in 2025 more than seven billion smartphones are in use globally, with the market split between Android (72,55%) and iOS (27,04%) devices [1]. The pervasiveness of these devices makes them an extremely attractive target for surveillance technologies, both commercial and state-sponsored.

Mobile spyware is a category of malicious software designed to covertly infect a device, collect sensitive information without the owner's consent, and transmit it to third parties. Such software can exfiltrate data including SMS messages, call logs, contacts, instant messaging content, files, photos, and GPS location, and in some cases enables real-time surveillance through access to the camera, microphone, or screen recording.

Accessibility is one of the most concerning aspects of commercial spyware, often referred to also as stalkerware. In fact, the inexpensive cost and easy installation, which takes only a few minutes, significantly reduce the technological barrier for

committing abuse. So, they make it possible a range of interpersonal abuse by enabling for malicious actors to covertly monitor victims' communications, movements, and private activities.

In addition to the evident privacy issue, commercial spyware can have broader security consequences. Many of these applications gather private information and send it via opaque or inadequately secured infrastructures, which could lead to data breaches or unauthorized third-party access. As a result, in addition to interpersonal harm, victims may be vulnerable to secondary threats such as identity theft or exploitation by third parties.

Furthermore, over the last decade, the cases of abuse involving state-sponsored spyware have increased significantly. Several documented cases report the use of spyware as a powerful weapon to conduct espionage, control and manipulate the news flow, and silence dissidents.

By leveraging zero-day vulnerabilities and unpatched software flaws, sophisticated spyware can be remotely distributed and controlled, reaching and compromising a significant number of devices.

In this particular scenario, mobile forensic analysis is essential for both individual security protection and legal investigations. Its role is to identify, collect, acquire, and evaluate the digital evidence while protecting their integrity and preserving a chain of custody that can be verified. When conducted correctly, forensic analysis can provide legally admissible evidence and support judicial proceedings, as well as assist victims and investigators in detecting and understanding covert surveillance activities.

This thesis begins by offering an in-depth analysis of contemporary surveillance technologies and the evolving landscape of mobile spyware. The research is structured to guide the reader through the technological, ethical, and forensic aspects of digital surveillance, starting from the classification of these threats to their practical detection.

In the first chapter is introduced the taxonomy of surveillance software, categorizing them based on their deployment and purpose: mercenary or state-sponsored spyware and commercial spyware. This part provides a technical basis for the analysis that follows by focusing on the underlying architecture of these technologies and analysing their extensive surveillance capabilities.

The chapter continues underlying the "gray area" of the surveillance market. It investigates the subtle distinctions between parental control applications and consumer spyware, commonly referred to also as stalkerware. Particular attention is given to the peculiarity of the iOS ecosystem. Furthermore, the research explores the legal ambiguities surrounding these applications, discussing the concerns for

social control, individual privacy, and security. The thesis then explains the stages of a spyware attack, dividing them into four crucial phases: reconnaissance, delivery and exploitation, agent installation and data exfiltration, in order to understand the operational logic of these software.

The text analyses different infection vectors, comparing physical access with remote techniques such as one-click and zero-click attacks. Real-world case studies from forensic reports of state-sponsored infections are used to demonstrate each attack infection method. The chapter concludes with the definition of Indicators of Compromise (IoCs), which are fundamental resources for modern detection tools, and an overview of well-known surveillance solutions. This presentation covers state-sponsored solutions, such as NSO Group's Pegasus and Cytrox's Predator, and widely accessible commercial applications like Mobile Tracker Free and iKeyMonitor.

Chapter 3 opens with the presentation of methodological framework for mobile forensics suggested by NIST SP 800-101 Rev. 1. Following this document, extraction techniques are divided into five hierarchical levels: manual extraction, logical extraction, HEX dumping and JTAG extractions, chip-off analysis, and micro-read. By defining these levels, the thesis demonstrate the trade-offs between technical complexity, invasiveness, and the depth of data recovery necessary to detect sophisticated spyware.

After that, the study moves to Dynamic Analysis solutions, which focus on real-time malware activity observation. This section presents host-based methods used for proactive threat hunting, with a focus on the function of live telemetry and the spyware's interface with the operating system's kernel and system calls. Furthermore, dynamic malware analysis within sandboxed environments is presented, which allows for the safe execution of suspicious payloads. In this section are covered live analysis methods for extrapolating and parsing system logs, artifacts that are often volatile but contain essential chronological evidence of an infection.

The chapter then assesses several network-based detection tools and methodologies. These include Intrusion Detection and Prevention Systems (IDS/IPS), Deep Packet Inspection (DPI), and traffic interception using Wireshark. The section concludes with an analysis of Machine Learning techniques and its use for spyware detection. Static analysis, which examines a device's data or an application binary without running the code, is the next topic covered in this chapter. This section highlights the differences between mobile operating systems, with a focus on the complexity and security restrictions of both Android and iOS. The analysis includes Android APK and iOS IPA inspection, permission mapping, which is particularly useful for consumer-grade spyware detection, and the examination of data artifacts acquired via logical and physical extractions.

Static analysis tools such as AndroidQF, MobSF, Autopsy, and commercial forensic tools are lastly introduced in this chapter.

In this chapter the State-of-the-Art in spyware detection technologies are also presented. SpyGuard is the first tool discussed, along with its operational pipeline, detection logic, and design.

Other network-based solutions that concentrate on traffic interception and analysis, such as Interceptor and M2 Bridge, are then covered. The chapter also introduces Mobile Verification Toolkit (MVT), a tool designed for extracting mobile data and comparing it to lists of known Indicators of Compromise (IoC). Lastly, Kaspersky Lab's iShutdown utility is introduced. It analyses the Shutdown.log file to find anomalous shutdown events that can be connected to spyware activities.

Chapter 4 regards experimental testing and evaluation of existing commercial-level spyware detection tools. Tests are conducted on consumer spyware applications such as MobileTracker Free, using the SpyGuard tool, including scenarios involving SSL pinning bypass. To address the inherent limitation of the IOC-based approach, the second section discusses potential enrichment to SpyGuard detection strategy. These include the integration of unsupervised machine learning techniques, using k-means and DBSCAN algorithms to isolate anomalous or beaconing connections, considerations for the use of Large Language Models (LLMs), VPN-based detection strategies, and inducement operations.

The focus then shifts to host-based forensics, evaluating the Mobile Verification Toolkit (MVT) for initial triage and the Mobile Security Framework (MobSF) for deep static inspection and IOCs extraction.

Finally, the chapter reports a proactive methodology for inspection via the Android Debug Bridge (ADB) that demonstrates the limitation of MVT and the detection of commercial spyware through permission mapping and Accessibility Services analysis. During this analysis also Android logs are logically extracted and used to find signs of the spyware infection by examining hidden foreground and background services and unconventional battery wake locks.

Lastly, this thesis demonstrates that while tools based on Indicators of Compromise (IOCs) are essential for rapid triage and initial evaluations, they suffer from limitations against novel, rapidly changing infrastructures and evasive threats. To overcome these limitations, this work proposes proactive behavioural-based solutions. The experimental results show that the paradigm shift from static to proactive methodologies is necessary to ensure robust and reliable mobile spyware detection.

Chapter 2

Background

To effectively analyse the process of mobile devices intrusions, it's necessary to identify the different typologies of surveillance software. The following section provides a detailed taxonomy of the spyware, highlighting the differences of mercenary spyware and commercial surveillance applications. After defining these software and detailing their capabilities, the context of privacy and security is analysed. Surveillance software infections occur with a predefined schema, made of sequential stages, designed to bypass security measures while maintaining persistence on the target device. Moreover, there are different infection vectors available to attackers. The second section of this chapter presents some of the well-known spyware applications, showing their infection vectors and presence on mobile devices.

2.1 Mobile Spyware: An Overview

2.1.1 Definition of spyware

Spyware is defined as a malicious software used to infect a device, collect information, and forward it to third parties without the owner's consent. According to NIST Computer Security Resource Center spyware is a "Software that is secretly or surreptitiously installed into an information system to gather information on individuals or organizations without their knowledge; a type of malicious code". [2] Mobile spyware are specifically designed to infect smartphones and tablets and can be divided in several categories: those used to make profit from stolen data, including keyloggers, that record keystrokes, information stealers, stalkerware, parental control applications, commercial spyware, and state-sponsored spyware.

Stalkerware refers to software designed to monitor, or surveil an individual without their informed consent. These applications are commonly deployed by

intimate partners in abusive relationships, on children by overly controlling family members, or within workplaces, where they may also be referred to as bossware. Stalkerware applications operate in a grey legislation zone. They are put on the market as parental control apps for the surveillance of minor children. Informed consent is necessary to remain within a legal operating zone, and these apps comply with notifications and screens which inform users that the device is being monitored. Usually, these types of applications offer subscription plans for a few euros per month, and the installation is affordable for everyone.

Their legitimacy on a piece of law that can also raise ambiguities: if the monitored person gives explicit consent to surveillance, it becomes legal.

Stalkerware apps, however, act silently and hide their presence through different obfuscation techniques.

It is important that legislation defines the line between legitimate monitoring and privacy violations. [3]

From the Coalition Against Stalkerware:

"In the coalition, we distinguish between stalkerware and parental control software, as the former remains hidden on the phone, while the latter does not. This does not mean that parental control software is not used in situations of digital violence, and we are well aware of this abuse." [3]

Some parental control apps available on the market, such as Qustodio, act similarly to stalkerware software, enabling the control over the device, and protecting their removal with a password. Even though these applications need to be paid attention, usually they are still visible by the surveilled user and periodically notify them about their presence.

Stalkerware is used by someone who wants to control or surveil another individual, including intimate partners, children or employees, in this case the monitoring app is also known as bossware.

Mobile spyware can be used to steal data such as short messages service (SMS), call logs and contact list, instant message applications data, the entire filesystem, photos, and position through GPS. Certain types can also perform real-time monitoring such as accessing the device's camera and microphone, or recording the screen. The attacker is also able to remotely control the compromised device, typically through a dashboard hosted on a command-and-control (C2) server, or by sending SMS containing specific commands. Those who remotely control the device can block applications and websites, as well as perform other intrusive actions. A spyware can be considered malicious if it is installed on a device without the consent of the owner.

Once the infiltration phase is ended, spyware agents are difficult to be detected and use techniques to hide themselves behind harmless system apps or processes, erasing

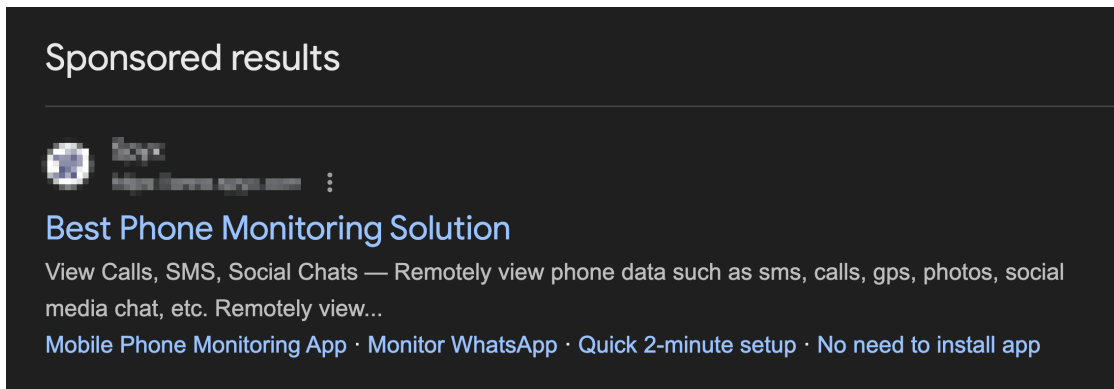


Figure 2.1: A sponsored advertisement of a stalkerware application

any trace of their presence. The increasing level of sophistication of these malware make them really difficult to be detected without a forensic investigation.

Understanding the architecture, infection vectors, and persistence mechanisms of mobile spyware is essential for developing effective detection and mitigation strategies. [4] [5]

From a forensic and technical perspective, mobile spyware must be addressed as sophisticated software, which acts at different layers of the mobile operating system, targeting the application sandboxing model.

In Android and iOS, sandboxing ensures that each application operates within restricted boundaries, with isolated memory and file system access. The purpose of spyware is to escape from sandboxes and escalate privileges to gain unauthorised access to other applications data or hardware sensors.

While the commercial spyware, such as stalkerware applications, are mainly based on system APIs abuse (for example, as will be further analysed, the abuse of Accessibilities Services [6]), mercenary or state-sponsored grade spyware are characterised by modular architectures which allow attackers to deploy specific payloads depending on the target’s environment. Usually spyware architecture includes the following components:

- **Core Orchestrator:** It is the unit that manages communication with the Command-and-Control (C2) server. It receives command via a web service (MITRE ATT&CK Mobile Matrix T1469 [7]) which commonly uses SSL/TLS encryption, giving adversaries an added level of protection. To enhance the protection layer adversaries may use SSL Pinning to protect the C2 traffic from being intercepted and analysed [8].
Spyware applications may also exploit the application layer (MITRE ATT&CK Mobile Matrix T1437 [9]) or communication to avoid detection or network

filtering. Many protocols can be used, including those used for web browsing, transferring files, electronic mail, or DNS.

For example many commercial spy applications use Firebase Cloud Messaging for receiving C2 instructions.

Another commonly used C2 channel is via SMS. To ensure obfuscation, usually command messages are deleted or hidden in communications that appear legitimate. An example is taken from the Lookout Pegasus Analysis of 2016, in which apparently legit Google password reset SMS were used to communicate with Pegasus.

```
Your Google verification code
is:5678429\nhttp://gmail.com/?z=FEcCAA==&i=MTphYWxhYW4udHY6NDQzLDE6bW
Fub3Jhb25saW51Lm5ldDo0NDM=&s=zpvzPSYS674=
```

Figure 2.2: Pegasus SMS command message. Source: [10]

- **Collection & Hooking Engine:** Different techniques are used to collect data. This module implements:
 - Abuse Accessibility Feature (MITRE ATT&CK Mobile Matrix T1453 [11]): accessibility services are abused to steal sensitive data.
 - Abuse of System APIs to capture audio, call logs, SMS messages, or to live access to location, screen and camera (MITRE ATT&CK Mobile Matrix TA0035 [12]).
 - Keylogging: used to intercept credential and sensitive information (MITRE ATT&CK Mobile Matrix T1417 [13]).
 - Access Notification: data is collected within notifications sent by the operating system or other applications (MITRE ATT&CK Mobile Matrix T1517 [14]).
 - Runtime Hooking: can be implemented via mechanisms known as Dynamic Linker Hijacking. This method forces the loading of a malicious library before or alongside lawful ones by subverting the OS dynamic linker (dyld on iOS or linker/linker64 on Android). While Amnesty International does not document the specific injection mechanism, their forensic analysis identified traces indicating that Pegasus infections result in malicious code executing within the context of legitimate system processes [15].
- **Data Exfiltration Module:** Responsible for the secure transmission of stolen data. It often employs Anti-Forensic techniques, such as encrypting

the exfiltrated data packets and mimicking legitimate network traffic (e.g., masquerading as standard HTTPS traffic to social media or cloud service domains) to bypass Network Intrusion Detection Systems (NIDS).

- **Persistence & Stealth Module:** This component ensures the agent remains stealth on the device and does not leave traces, and is also active after a device reboot. Technical methods include zero-day vulnerabilities exploitation, or by Subverting the Chain of Trust which consists of initialization scripts modification on rooted / jailbroken devices, to be automatically executed at boot or logon initialization (MITRE ATT&CK Mobile Matrix T1398 [16]). Another method that has been documented is the misuse of Mobile Device Management (MDM), a kind of security software that large companies use to regulate and enforce policies on the devices used by their employees. This technique prevents the application from being uninstalled and silently reinstalls components if the user deletes them. [17]

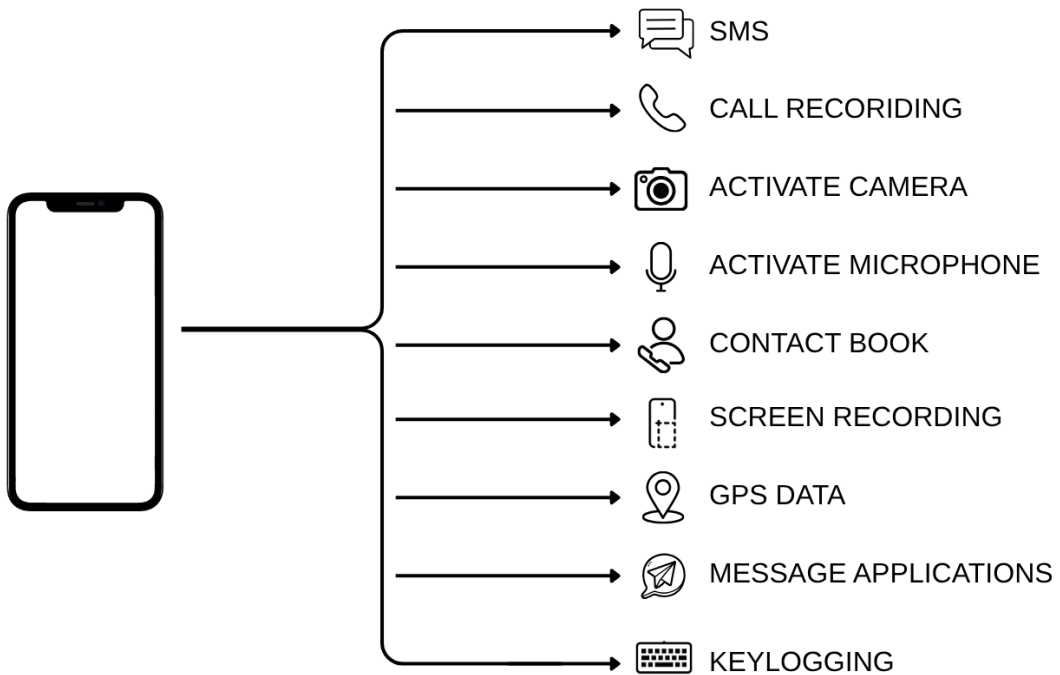


Figure 2.3: Spyware capabilities

The technical maturity of consumer-grade (commercial) spyware is best observed in its ability to abuse legitimate Android APIs to perform surreptitious data collection, because differently from Apple it supports side-loading off-store applications. The paper “No Privacy Among Spies: Assessing the Functionality and Insecurity

of Consumer Android Spyware Apps” [18] offers a valuable explanation on how commercial spyware achieve their functionalities:

- Camera: there are three documented techniques used to take a photo without showing a preview. One way is using the invisible preview: spyware apps use a transparent preview or reduce it to an invisible size (e.g. 1x1 pixel). A second way is to use Android features such as SurfaceView or TextureView, neither of which requires showing a preview. The acquired images are stored and sent or transmitted live to the spyware C2 infrastructure. The last technique, used by Spy24, is the utilization of a 1x1 pixel WebView to stream videos [19].
- Microphone and call recording: Spyware circumvents the restrictions on the *VOICE_CALL* audio source (introduced in Android 6) by using the Android NDK (Native Development Kit) [20]. By calling functions directly from *libmedia.so*, the spyware interacts with the lower-level media framework to record both uplink and downlink audio. For VoIP calls (WhatsApp, Signal), spyware abuses the AccessibilityService concurrent audio feature (Android 10+), which allows two apps to access the microphone simultaneously if one is registered as an accessibility service.
- Accessibility Services: Originally intended for users with disabilities, the Accessibility API is the most abused feature in the stalkerware ecosystem. By registering as an *AccessibilityService*, the spyware can use the *getText* method to scrape every UI element rendered on the screen. Spyware apps exploiting the *getText* method, part of Android’s accessibility framework, are also able to track all key presses.
- Notification Listening: By registering a *NotificationListenerService*, spyware can intercept data that never even reaches the foreground.
- Screenshots: They can be stealthy taken in two ways: by launching a transparent, 1x1 pixel *Activity* to satisfy the system’s requirement for a foreground task, initiates a *MediaProjection* session, takes the screenshot, and immediately kills the activity, or exploiting the Accessibility service API *AccessibilityService.takeScreenshot()*.

Ways to spy on iOS devices

iOS devices are known for having stronger security policies compared to Android, which often makes them appear more secure. However, they are still vulnerable to spyware.

Differently from Android, common spy app are divided in four main categories:

- **Monitoring via iCloud credential:** some applications exploit access to iCloud to spy on a device by extracting information from the account or from its iCloud backup. Attackers utilize stolen or phished Apple ID credentials to authenticate via iCloud APIs. In this way they are allowed to perform a logical acquisition by downloading available iCloud backups or retrieve cloud-stored data exposed through iCloud services (e.g., Photos, iCloud Drive, Find My metadata).
One of the most important advantages of this method is that no installation or physical device access is required. However, the success of the attack relies on the frequency of the backups.
The advantage of this type of attack is that no device compromising is needed. However, the attacker must obtain valid Apple ID authentication, either by abusing trusted sessions or by bypassing two-factor authentication mechanisms.
- **Monitoring apps on jailbroken devices:** jailbreaking eliminates Apple's built-in security restrictions, allowing users to install applications from sources other than the App Store. On jailbroken devices applications can circumvent Apple security controls and access to different resources that are normally restricted, ensuring complete monitoring.
- **App Store monitoring applications:** on the marketplace there are apps that offer solutions like parental control and enable monitoring position and usage time. This type of application requires access to the device and requires the user to give the necessary permissions. However, these applications are limited and do not provide a complete surveillance of the device.
- **Zero-day and advanced spyware:** iOS systems are not immune to zero-day vulnerabilities. These are complex attacks and do not require physical access to the device nor the iCloud credential. By exploiting unknown security flaws, advanced spyware can escalate privileges and gain complete access to the device and its data.

Overcome app installation blocks

iOS systems block the direct installation of applications from external sources that are not part of the official App Store. However, some sophisticated spyware, such as Exodus, take advantage of the Apple Developer Enterprise Program, where firms can transfer their internal apps to their employees directly without passing through the App Store. This practice violates Apple's policies and can be exploited by attackers.

Another way to install apps coming from third party sources is by jailbreaking the device.

IOS jailbreaking refers to the use of privilege escalation in order to bypass the software limitations imposed by Apple on devices running iOS and iOS-based

operating systems. The procedure is typically carried out through a series of kernel-level patches. Once a device has been jailbroken, it grants root access to the operating system and allows for the installation of software that can't be installed via the official App Store.

Jailbreaking disables or bypasses the enforcement of iOS code-signing mechanisms, which normally prevent unsigned or improperly signed applications from executing. This gives the applications unrestricted access to the device, and also the ability to use undocumented APIs that Apple otherwise prohibits. [21]

Jailbroken devices create a major enterprise risk given their ability to run apps developed outside of Apple's review, which may be malicious or contain vulnerabilities.

Since iOS 17.4, Apple allows users in the European Union to install applications from alternative app marketplaces outside of the official App Store, in compliance with the Digital Markets Act (DMA) [22]. While this removes the need for jailbreaking to install third-party apps, such applications are still subject to Apple's notarization process, sandboxing, and code-signing requirements, and do not gain elevated privileges or access to undocumented APIs. As a result, this mechanism does not provide the same capabilities as jailbreaking and is not considered a viable vector for high-privilege spyware deployment.

The Lockdown Mode

To overcome the increasing number of spyware attacks, Apple in 2022 added a security feature on their devices, changing the paradigm to a proactive attack surface reduction.

Lockdown Mode is an advanced, optional security feature for iPhone, iPad and Mac that should only be used when a user believes they may be targeted by a highly sophisticated cyberattack, such as advanced spyware.

When activated, Lockdown Mode puts in place limitations on the normal functionality of the device to enhance security. Apps, websites and other features are restricted, and some user experiences are disabled. Lockdown Mode includes the following protections:

- *Messages*: Most message attachment types other than images are blocked. Some features, like link previews, are also disabled.
- *Web browsing*: Complex web technologies — including just-in-time (JIT) JavaScript compilation — are disabled, unless the user manually excludes a trusted website from Lockdown Mode.
- *Apple services*: Incoming invitations and service requests, including FaceTime calls, are blocked if the user has not previously sent the initiator a call or request.

- *Tethered connections*: When the device is locked, it blocks wired connections to computers and accessories.
- *Configuration profiles*: Configuration profiles can't be installed, and the device is unable to enrol into Mobile Device Management (MDM) while Lockdown Mode is turned on. However, any MDM profiles that were previously enabled prior to Lockdown Mode are still present on the device.

Lockdown Mode can be enabled by navigating to Settings > Privacy & Security and selecting Lockdown Mode.

Security researchers verified the effectiveness of this feature during the analysis of the BlastPass exploit chain (2023). It was confirmed that Lockdown Mode successfully prevented the infection by NSO Group's Pegasus, by blocking the execution of malicious code delivered via weaponized iMessage attachments.

2.1.2 Impact on user privacy and security

In the digital revolution, the right to privacy is a natural human right, vital to human freedom in democratic countries. The emphasis being placed on national security, however, creates tensions related to privacy since governments require access to information in order to counter threats.

In order to avoid penalizing one for the other, policymakers must balance between keeping privacy and guaranteeing security.

The abuse of surveillance software not only compromises individual privacy, but also undermines democratic processes and freedom, and therefore necessitates collective and regulatory efforts at the European level to address these matters. [23]

2.1.3 Legal background

The Italian legal framework concerning spyware includes legislation emanated by different regulatory bodies: on one side, the Italian state with its legal framework at the national level, and on the other, the European Union with regulation, such as GDPR [24], which are applied to all member states.

Italy provides guidelines on the application of surveillance and interception software with the Privacy Code (Legislative Decree 196/2003), updated and adapted by the Legislative Decree 101/2018, and other national legislation. At the same time the European Union implemented a regulation framework for the protection of personal data, through the GDPR, which includes consent and prohibits unlawful processing. [25]

Referring to the Privacy Code, article 122 [26] prohibits the access, the surveillance, or the interception not authorised of personal communication and data, except in cases expressly provided for by law. Wiretapping or electronic monitoring is

permitted only for specific purposes, typically for the security of the Republic, criminal investigations, or for reasons of public safety, and only under the methods and conditions defined by the law.

To summarize, the lawful use of the spyware is limited to:

- Activities authorized by juridical authorities or for national security.
- Monitoring of employees, provided that the data subjects are informed and have given their consent.
- Parental control on minors with adequate informed consent.

The illegal use is when:

- The spyware is installed without informed consent.
- Monitoring is used for unauthorized surveillance, for purposes different from the ones defined by the law, for example stalking, industrial espionage, and data stealing.
- Violation of the principles imposed by the GDPR.
- Wiretapping or data collecting for personal or criminal purposes.

2.1.4 Concern about social control

The Office of the United Nations High Commissioner for Human Rights highlights in a report [27] how spyware technologies like Pegasus can turn any smartphone into a continuous surveillance device. These software can access camera and microphone, as well as data, turning the phone into a means of spying on the user's daily life. Such abilities effectively blur the boundary between digital and physical privacy, and allow external entities to monitor intimate moments, personal conversations, and sensitive environments without the victim knowing. This kind of intrusion demonstrates how personal safety and psychological well-being can be compromised by modern spyware, in addition to information security.

Even though these instruments are, in general justified by governments as tools for fighting against crime and terrorism, surveillance software have consistently been used against dissidents, including journalists, opposition members of parliament, and human rights activists.

Several cases of power abuse are documented by Amnesty International and then analysed by their Security Lab. Technologies developed for national security can be redirected toward political control, suppressing investigative journalism, or limiting the capabilities of civil organizations to operate freely.

The report also expresses concern about the increasing surveillance of public spaces,

made possible by a large-scale data collection and biometric systems, which vastly expand the reach of monitoring activities.

Moreover, often governments do not provide exhaustive and transparent information about these practices and tools used, and even when they are initially used for legitimate goals, they can easily be re-proposed for secondary purposes. This lack of oversight creates an ecosystem in which surveillance infrastructures can grow incrementally and silently, adding new capabilities over time without public debate or democratic consent.

These practices pose significant threats to human rights and democratic freedoms. Without strong legal protections, independent monitoring, and strict rules about how data is used, advanced spyware and mass surveillance technologies could shift the power dynamic between governments and their citizens. This could have lasting implications for democratic governance.

2.1.5 Stages of the attack

1. **Reconnaissance:** in this phase the attacker tries to gain information about the target. Precious information that can be retrieved are: the phone number, that can be used to deliver the malware through messages and instant messages platforms, and the IMEI number, which is a unique identifier from which some useful information can be gained, for example the brand and the model. [28]

This stage of the attack is also useful to gain information about the target that can be used for example to perform social engineering during the delivery and infection.

When zero-click attacks are too expensive or not available, adversaries rely on social engineering, with techniques designed to trigger a specific action that may result in a one-click attack.

For commercial and advanced spyware getting information about the target device is necessary to map the vulnerabilities. Once the OS version and patch level are identified, the attacker selects a specific CVE (Common Vulnerabilities and Exposures) and crafts a compatible payload.

The information accessible to an attacker varies based on architecture:

- Android: Android is relatively permissive regarding hardware metadata. Applications can access the *android.os.Build* class. This class contains specific constants such as *Build.BOARD*, *Build.BOOTLOADER*, *Build.MANUFACTURER*, and *Build.FINGERPRINT* which provide valuable information about the hardware, the firmware build, and security patches [29].
- iOS: Apple's ecosystem is highly restrictive. Applications operate within

a strict sandbox where hardware-level identifiers (IMEI, Serial Number) are obfuscated. Legitimate third-party applications can typically only query the *UIDevice* class for basic info, such as model and systemVersion. While restricted, these two parameters are often sufficient for an attacker [30].

To bypass OS-level restrictions attackers employ remote fingerprinting via attacker-controlled websites. This technique applies to both Android and iOS devices. By analysing the User-Agent string attackers can retrieve useful information about the system. Another common technique to retrieve information is the Canvas Fingerprinting: forcing the device's GPU to render a specific hidden shape or text. Because every GPU and driver version has subtle differences in how it handles anti-aliasing and sub-pixel rendering, the resulting pixel data acts as a unique signature. This technique is less efficient on smartphones due to hardware homogeneity and modern browsers use data randomisation techniques to enhance privacy.

2. **Delivery and exploit:** delivery phase can take place in different ways, such as zero-click or one-click remote attack or via physical access.

Once delivered, the exploit must overcome the mobile operating system's security architecture, which is centred on sandboxing. Initial execution usually occurs within a restricted process with limited permissions. To gain full access, the spyware performs a sandbox escape by exploiting a secondary vulnerability in the Kernel or a high-privilege system service. After this step adversaries may exploit vulnerabilities to achieve higher privileges in order to execute their controlled code.

Bossware and stalkerware applications typically do not escalate privileges and escape from the sandbox, but abuse permissions, requesting excessive permissions to gain access to system APIs.

Sophisticated spyware usually has a checklist of behaviour that can be related to a forensic investigation such as having installed Cydra, Frida, proxies, tcpdump or developer mode. A module continuously checks for this behaviour and if one or more are detected, a notification is sent to the C2 server, and the exploitation process is terminated. As an example is reported the watcher module of Intellexa Predator [31].

This phase is very important from a forensic perspective because permission request and privilege escalation often leaves behind artifacts for example in the crash logs.

3. **Agent installation:** if the exploit succeeds, the spyware agents are installed with the root privileges. With the root privileges or kernel access spyware agents can access stored data and device resources, such as camera, microphone,

speakers, and gps bypassing standard OS permission prompts. In this phase the spyware put in place persistence techniques.

4. **Exfiltration:** in this last stage the spyware acts with all the privileges achieved to access the user data. A spyware can access location data, live recording microphone and camera, take photos, and record calls. In addition, it can access SMS, message applications, call logs, contacts and other data stored on the device.

Data is transmitted to the attacker command-and-control (C2) server, by encrypting it and using other techniques that aim to disguise their data exfiltration within legitimate network traffic. Some commercial tools use VPN to encrypt traffic and avoid detection via network analysis. During the exfiltration process some tools create temporary zip archives before sending them to the C2 infrastructure. Cisco Talos identified traces of data related to Intellexa Predator in temporary directories such as `/data/local/tmp/wd/` [32].

In addition, the agent remains in a low-power "listening" state, waiting for commands from the C2 server.

2.1.6 Infection vectors

To develop a spyware tool the vendors need a deep technical knowledge that can be used to find vulnerabilities, craft exploits, bypass security controls and deliver the agent.

The infection process involves multiple stages, starting from a reconnaissance phase to the exfiltration phase.

The infection method differs when talking about commercial spyware and state-sponsored spyware: in the first case the agent needs to be installed and try to obfuscate itself, in the second one, vulnerabilities are exploited to guarantee persistence and access to internal files and resources.

In 2022, a leaked document appeared to show a company that asked €8 million for services that include Android and iOS exploits.

Also, the cost for NSO Group's software was significant: in 2016, The New York Times reported that the firm charged an initial \$500,000 installation fee, plus an additional \$650,000 to compromise 10 devices. By 2019, the pricing model appears to have evolved, with India's Economic Times reporting that a single Pegasus license cost approximately \$7-8 million annually. [33]

There are many different ways in which a spyware can get installed on a device, such as phishing, malicious link, compromise password, direct physical access, and zero-day exploits. These techniques often use information about the target gained during the reconnaissance phase to deliver a personalised attack, enhancing the possibility of success.

Physical access attack

This type of infection can be considered both easy and challenging at the same time. This technique, in fact, is the one used in case of stalking violence or commercial spyware use, which often do not have sophisticated infection code that exploit zero-days vulnerabilities. Once the abuser has the physical control on the device infecting it with the spyware is relatively simple.

A prerequisite for most physical access infection is the ability to bypass the device's screen lock (PIN or password). In case of stalkerware or commercial spyware installation usually the abuser knows how to unlock the device, or use social engineering techniques to discover the secret. Modern mobile operating systems utilize File-Based Encryption (for example is mandatory from Android 10) and secure hardware modules, such as Trusted Execution Environment (TEE) or StrongBox on Android and Secure Enclave Processor (SEP) on iOS, to derive cryptographic keys from user credentials and hardware-bound secrets. Consequently, encryption keys and filesystem contents cannot be accessed in cleartext by the operating system. MITRE ATT&CK describes three different possible techniques to bypass the lock screen [34]:

- Biometric spoofing: attackers try to spoof a mobile device's biometric authentication mechanism. As additional protection layer Android and iOS require the device's passcode after the first boot and after a random amount of time.
- Brute-force: brute-force technique can be applied to discover the lock screen passcode. Mobile OS vendors implemented hardware-enforced security mechanisms. These components enforce minimum delays between attempts, exponential back-off timers after failures and maximum attempt counters.
- Vulnerability exploit: mobile devices' exploits are used to bypass the lock screen. The vulnerabilities are generally patched by the device or OS vendor once disclosed.

Advanced forensic tools like Cellebrite UFED and Magnet GrayKey are able to bypass passcodes through proprietary exploits, hardware interfaces, and brute-force methods.

These tools leverage hardware-level vulnerabilities to interact with the device. Additionally they exploit low-level chipset vulnerabilities to suppress the delay introduced by back-off timers and allow for continuous, high-speed attempts. Furthermore they can intercept communication preventing the maximum attempt counter to be incremented.

Not all devices are supported by these tools and ongoing updates are required to keep exploits valid after vendor patches.

This technique has been used also to install mercenary spyware. Amnesty International's document "A Digital Prison" [35] describes several cases in which physical tampering was carried out. For example, the document reports the installation of NoviSpy in October 2024 during an interview by BIA security service officials in Belgrade. The man was asked to unlock his Android phone in front of the officer, so the PIN code was captured by one of the many cameras in that room. After that the man left the phone out of the room and tampered. The phone was connected via USB cable and the Android developer mode enabled to control the phone over ADB. Between the USB cable connection and its disconnection fifteen minutes passed, and the spyware was installed.

Another example reported in the document is related to a Serbian journalist who after a traffic stop was taken to the police station under the pretext of taking drug and alcohol tests. During the test, the phone was tampered with and NoviSpy was installed.

This second example is also important because the forensic analysis uncovered evidence suggesting that Cellebrite's UFED [36] was used to unlock the device. Cellebrite's UFED (Universal Forensic Extraction Device) is a digital forensics tool that law enforcement and investigators use to extract, decode, and analyse data from mobile devices.

In this case was reported the exploit of a zero-day Android privilege escalation vulnerability CVE-2024-43047.

Once the device is unlocked the spyware can be installed the installation of the spyware agent follows distinct technical paths depending on the target operating system and the type of spyware itself. Commercial spyware, available for Android and jailbroken iOS, are sideloaded typically by the spyware's vendor webpage.

On android the primary vector for physical access installation is the Android Debug Bridge (ADB), a command line tool that communicates with the `adbd` demon, that runs on the device commands sent by a client [37]. For example the command `adb install -g <package_name>.apk` and the `"-g"` flag grants all runtime permissions defined in the app's manifest automatically.

On devices with an unlocked bootloader, physical access allows for the modification of the `/system` partition, and by injecting the spyware binary into `/system/priv-app/` grants it privileged system app status with elevated permissions [38] [39].

On iOS without jailbreak attackers can abuse the Enterprise provisioning, which involves misusing Apple's Developer Enterprise Program certificates to sideload unsigned or malicious apps outside the App Store, bypassing standard review processes. Another option is to exploit the USB mounting exploits.

Once installed the spyware puts in place different obfuscation techniques and the attacker can fully remotely control the device.

One-click attacks

To understand how these types of attacks succeed is important to understand what social engineering is. NIST defines social engineering as “the act of deceiving an individual into revealing sensitive information, obtaining unauthorized access, or committing fraud by associating with the individual to gain confidence and trust.” [40] By gaining trust from the victim, the abuser is able to retrieve personal information that are fundamental to deploy a spyware attack. This information is used both to know what types of vulnerabilities can be exploited in the delivery phase, and to create personalized phishing campaigns while deploying a one-click attack.

One-click attack takes this name because the victim is tricked to click a malicious link or download an infected attachment. These compromised elements are usually sent to the victim using direct messaging applications such as WhatsApp and Telegram using phishing techniques, and thanks to the information gained convincing messages are created.

One example of one-click attack is reported in the Amnesty International document "The Predator Files: Caught in the Net" [41], related to the investigation about Predator. During that surveillance campaign 50 social media accounts have been targeted with “1-click” spyware infection links connected to Intellexa’s Predator spyware attack infrastructure. As described in April 2023 a Twitter account shared a link of a spoofed news website containing Intellexa Predator infection link.

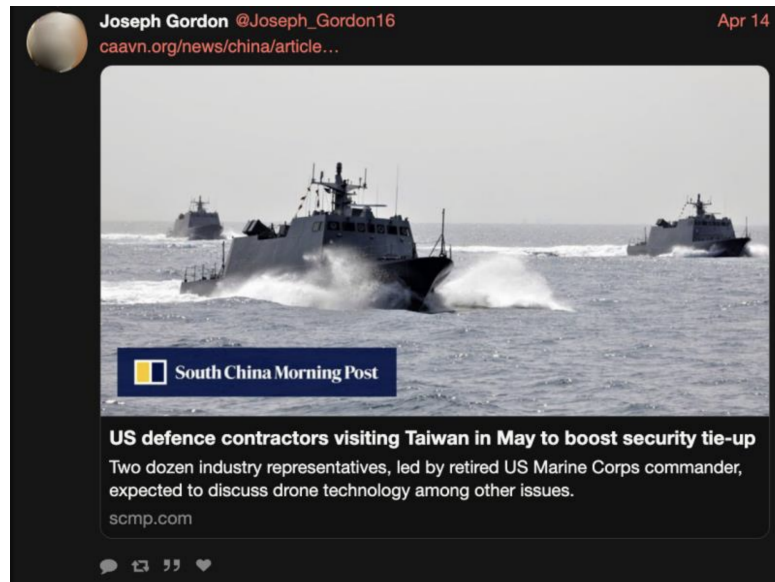


Figure 2.4: Attacker-controlled caavn[.]org domain imitating a news article from the South China Morning Post [41]

From a technical point of view, these types of attack start with a spear-phishing message containing a weaponised URL. Once the victim opens the malicious URL the goal of a one-click attack is to trigger Remote Code Execution (RCE) by exploiting vulnerabilities in the mobile device’s web browser engine. Infected URLs are often time-limited and geographically-restricted to make it more difficult for researchers to fully analyse the attack links or capture samples of the highly valuable zero-day exploit used to infect fully updated devices [41].

These types of attacks aim to exploit Just-In-Time (JIT) in WebKit/V8 to RCE without further interactions.

Examples of vulnerabilities exploited are identified by Google’s TAG. Discovered zero-day attacks targeted Chrome browser and proceeded with sandbox escape and privilege escalation. As an example CVE-2021-1048 used a Linux kernel bug in the `epoll()` system call and allows the attacker to escape the sandbox and compromise the system by injecting code into privileged processes [42].

Zero-click attacks

Zero-click attacks are the most sophisticated and dangerous infection vectors. Differently for one-click attacks which require social engineering techniques to trick the victim into clicking a malicious link, in this type of attack the victim doesn’t even know that the infection happened because his interaction isn’t required. A zero-click attack succeeds when the attacker exploits an unpatched vulnerability and gains access to the device.

Zero-click attacks could take advantage of data parser and media decoder, that are activated when the operating system tries to process an incoming message, for example to generate a preview.

An example is the zero-click exploit *FORCEDENTRY* (CVE-2021-30860), used by the NSO Group to deliver Pegasus. The attack was delivered via a hidden iMessage containing a malicious PDF disguised as a *.gif* file. When the Apple *CoreGraphics* library attempted to render the file, it triggered an integer overflow, resulting in a RCE [43].

Zero-days infection can be performed through messages, calls or specific applications. In 2019, WhatsApp fixed CVE-2019-3568, a zero-click vulnerability in WhatsApp calling that NSO Group used against a large number of devices, and in 2020, NSO Group employed the *KISMET* zero-click iMessage exploit. Following the Amnesty International report “How To Catch NSO Group’s Pegasus” [15] in 2019 several vulnerabilities have been found in iOS iMessage and FaceTime. The forensic analysis revealed that Pegasus processes executed after suspicious iMessage account lookups [Table: 2.1].

DATE (UTC)	EVENT
2019-06-16 12:08:44	Lookup of <code>bergers.o79@gmail.com</code> by <code>com.apple.madrid</code> (iMessage)
2019-08-16 12:33:52	Lookup of <code>bergers.o79@gmail.com</code> by <code>com.apple.madrid</code> (iMessage)
2019-08-16 12:37:55	The file <code>Library/Preferences/com.apple.CrashReporter.plist</code> is created within RootDomain
2019-08-16 12:41:25	The file <code>Library/Preferences/roleaccountd.plist</code> is created within RootDomain
2019-08-16 12:41:36	Process: <code>roleaccountd</code>

Table 2.1: Pegasus event logs retrieved during the analysis. Source [15].

Another interesting example reported by Amnesty International is the use of Apple Music to compromise the device on 10th July 2020. In this case the investigation wasn't able to confirm if the first infection had been delivered by Apple Music, but confirmed that the HTTP request performed by Apple Music pointed to a Pegasus domain [Table: 2.2].

It is important to notice that this evidence of Apple Music abuse was retrieved by jailbreaking the device, otherwise the database file was not accessible.

DATE (UTC)	EVENT
2020-07-06	Lookup of f\x00\x00ip.bl82@gmail.com by iMessage (com.apple.madrid)
05:22:21	
2020-07-10	Pegasus request by Apple Music app:
14:12:09	https://x1znqjo0x8b8j.php78mp9v.opposedarrangement[.]net:37271/afAVt89Wq/stadium/pop2.html?key=501_4&n=7
2020-07-10	Process: roleaccountd
14:12:21	
2020-07-10	Process: stagingd
14:12:53	
2020-07-13	Pegasus request by Apple Music app:
05:05:17	https://4n3d9ca2st.php78mp9v.opposedarrangement[.]net:37891/w58Xp5Z/stadium/pop2.html?key=501_4&n=7

Table 2.2: HTTP request performed by Apple Music that pointed to a Pegasus domain. Source [15].

2.1.7 Indicator of compromise (IOC)

Indicators of compromise (IOC) [44] are data that suggest a system or a network may have been compromised by unauthorised activity.

IOCs act as flags that can be used to detect unusual activity that is evidence of an attack, cybersecurity professionals detect, investigate, and respond to potential threats.

IOC can be collected from a variety of sources including:

- Operating system - unusual system file changes, permissions or process activity.
- Memory - unusual use or behaviour of memory.
- Network traffic - unexpected outbound connections or suspicious data transfers.
- Log files - failed login attempts or changes in file access patterns.

The IOC can be of different types:

- File name: known to be associated with a malware
- IP address: associated to a malicious server or to a command-and-control (C2) server.
- Domain names or URLs: linked to malware distribution sites or to the spyware dashboard.
- Hash values: unique identifiers for malicious files, payloads or certification.
- Email addresses or suspicious accounts: used by a malware during the attack or to exfiltrate data.

IOCs are important because they can be useful in deception threats early in the attack lifecycle, reducing the time of the analysis. They are really useful to share information with other organizations to help in the attack detection and enhance collective security.

Example of IOC related to the mobile-tracker-free app:

```
1 - name: MobileTrackerFree
2   names:
3     - MobileTrackerFree
4     - MonitorLoverman
5     - MTrack
6     - CellTracker
7     - TrackMobil
8   packages:
9     - a.tck.lvmchi
10    - com.androidid.inteernet.aa21111227
11  [...]
12    - com.mtf.download
13    - com.protect.download
14  certificates:
15    - 021A3F097EDA780798DF5ECB16EF338C08236847
16    - 0568E0400308CBFC58E11A324EA233F5B2E923BF
17  [...]
18    - 09273A6004A46078991F3FBA2A4868DA26DBB508
19    - 2A84B79A7E17E1F49642E8D5EA9828CFEA763E8C
20  distribution:
21    - download.mobile-tracker-free.me
22  websites:
23    - br.mobile-tracker-free.com
24  [...]
25    - mobitrackapps.com
26  c2:
```

```
27 |     ips :  
28 |     - 51.15.183.209  
29 |     domains :  
30 |     - api1.easydoc.info  
31 |     - api3.easydoc.info  
32 |     [...]  
33 |     - mobile-tracker-free.com
```

Listing 2.1: Source: [45]

2.2 Known Spyware Tools

This section provides an overview of the most well-known mobile spyware solutions currently documented in the public domain.

As previously described, there are different types of spy software, depending on their purposes and architecture. The main distinction is based on diffusion models and architecture.

The most sophisticated category includes state-sponsored spyware, also referred as mercenary spyware. These tools have complex architecture combined with strategies to avoid the detection, maintain persistence, and exploit zero-day vulnerabilities. These software are strictly marketed and sold only to governmental agencies for national security purposes, and not legally available to private entities.

A second category is represented by commercial monitoring software. These tools are publicly available on the consumer market and usually require the access to the victim device for the installation, as they do not rely on sophisticated exploit chains or vulnerabilities for obfuscations.

As previously discussed, they act in a grey legislation area as vendors tend to shift the responsibility to the end user, who are required to notify the device's owner about the deployment of such monitoring tools. In some cases these applications are marketed as parental control tools, in others they are sold as tools for partner or employee monitoring.

Studying state-sponsored spyware is challenging due to the exceptional secrecy surrounding their development, deployment, and upkeep.

Unlike commercial monitoring applications, these tools are not publicly distributed, and lack accessible documentation. Most information available is sourced from isolated incidents of device compromise and forensic reconstructions.

2.2.1 Case Studies State-sponsored spyware

Pegasus by NSO Group

Pegasus, developed by the Israeli company NSO Group, is the best-documented example of mercenary spyware. It gained worldwide notoriety after a series of investigations revealed the abuse committed by government clients, who used the spyware against journalists, activists, diplomats, and political leaders. Public reports and forensic investigations show that Pegasus can compromise both Android and iOS devices using sophisticated exploit chains, including zero-click vulnerabilities delivered through messaging apps or system services. [46]

While the exact internal architecture or command-and-control structure remains unpublicized, forensic analyses have confirmed that Pegasus has access to device files, microphone, camera, messages-including end-to-end encrypted communication-and geolocation data. The tool is designed to minimize forensic traces and frequently exploits are updated to stay effective against new operating system countermeasures.

Studies from Amnesty International Security Lab show how Pegasus infrastructure mutated and changed over years.

Predator by Cytrox

Predator is a mobile spyware platform provided by the firm Cytrox. As reported by Citizen Lab, in 2021 Cytrox was part of an alliance known as Intellexa, which combined the Intellexa Group and Nexa, that worked on surveillance.

Its operations became more public with investigations by Citizen Lab and various news agencies reporting its use against journalists, opposition politicians, and civil society actors. In 2021, infact, Citizen Lab reported findings about Cytrox's Predator spyware being used to target two Egyptian dissidents.[47]

In May 2022, researchers at Google's Threat Analysis Group (TAG) reported that Predator bundled five zero-day exploits in one package and sold it to several government-backed. [48]

According to public sources, Predator is capable of infecting both Android and iOS devices. Once installed, it is reported that the spyware allows the extraction of files, contacts, messages, and browsing data, with the addition of activating microphones and cameras.

Evidence found suggests that Predator spyware has been sold to at least 25 countries.

Graphite by Paragon Solutions

Graphite is a surveillance tool developed by the Israeli company Paragon Solutions, one of the newer vendors in the mercenary spyware ecosystem. Due to its recent appearance and discovery, not much technical information has been documented. Reports indicate that Graphite can exfiltrate data from devices remotely, intercept communication, and even perform live monitoring. Similar to other tools within this class, it is supposed to utilize sophisticated mechanisms for the compromise of devices by exploits.

On January 31 2024 WhatsApp sent alerts of possible signs of compromise. According to WhatsApp and Citizen Lab ninety such notifications were sent simultaneously across several countries, seven of them in Italy. [49]

NoviSpy

NoviSpy is an Android spyware. Its traces are related to Serbia, country in which this spyware was used in combination with Cellbrite's UFED mobile forensic tools against Serbian environmental activists and protest leaders.

The public first became aware of NoviSpy following Amnesty International's report "A Digital Prison": Surveillance and the suppression of civil society in Serbia" [35] in December 2024, which revealed its use in the targeted state surveillance of Serbian journalists and activists. [50]

Despite Android's robust security architecture, attackers have leveraged physical access to compromise targeted devices.

NoviSpy seems to operate through two components: NoviSpyAdming, which relied on extensive system permission and Admin privilege, and NoviSpyAccess, which exploits Android's accessibility services.

Even though this spyware can be considered less technically sophisticated in respect to other solutions, it was widely used in Serbia.

Analysis of the IP infrastructure associated with NoviSpy revealed connections to state-operated Internet service providers, including Telekom Srbija, as well as to networks linked to governmental institutions such as the Security Information Agency (BIA).

This is an interesting example of spyware architecture because, despite its smaller infrastructure, it can create a serious concern for security and freedom.

Galileo by Hacking Team

Galileo, also known as Remote Control System (RCS), is a cross-platform tool available for Android, iOS, Windows and others, developed by the Italian company Hacking Team.

Once placed on a device, it enables operators to exfiltrate files, intercept communications, perform keylogging, and geolocation tracking, and even turn on microphones and cameras remotely.[51]

For several years, RCS was one of the most widely used spyware systems among law enforcement agencies and intelligence services worldwide.

Although this software was developed around 2010 and marketed until 2015, and is therefore less sophisticated than more recent spyware solutions, it remains important to discuss for at least three reasons. Galileo can be considered the precursor of modern spyware, introducing a new type of company that creates and sells surveillance products.

Secondly, the Hacking Team case started a broader debate on the legitimate purposes for which surveillance tools may be employed, as well as on the legal and technical limitations required to ensure adequate protection of privacy rights.

Finally, in 2015 Hacking Team was hacked. The leak revealed a huge amount of sensitive data published on the internet. The disclosed data included customers, such as government agencies all around the world, backups, private messages, sales contracts, and the spyware source code. This last episode marked the end of Galileo, and once again reopened the debate on the protection of individual privacy.[52]

2.2.2 Commercial monitoring application

Unlike state-sponsored spyware, which usually relies on zero-click exploits, sandbox escapes, and privilege escalation vulnerabilities, commercial stalkerware requires some amount of user interaction or physical access to install.

The security and design models of Android and iOS impose different constraints on commercial stalkerware developers. This results in an ecosystem of available applications and their technical capabilities that differ substantially between the two operating systems.

The Android ecosystem is much more permissive of third-party software, and consequently, Android is the primary target of most commercial stalkerware.

Most Android spyware applications do not require root permission to function, but can be sideloaded installing the APK. Once installed these applications require the user to disable security protections such as Play Protect, battery optimization, and background restrictions.

For most Android solutions, while rooting is not mandatory, having root access allows for more invasive features.

From data collected it is possible to assert that Android dominates the commercial spyware market due to its openness and permissive installation model.

Contrary to Android, iOS offers a significantly more hostile environment for third-party applications. For that reason, most iOS spyware applications need a jailbroken

device to be directly installed on the phone.

Several commercial stalkerware tools provide tutorials or links to external jailbreaking utilities. However, it has to be noted that: none of these methods work on recent iOS versions, making direct installation on modern devices unfeasible.

Some commercial tools claim to support iOS without needing jailbreak. In practice, such applications do not operate as traditional spyware, as they cannot run code on the device or extract real-time data locally. Instead, they rely on alternative data acquisition methods supported by Apple’s ecosystem. Most solutions utilize iCloud synchronization, which can be enabled without the direct access to the device and requires Apple ID credentials. For this solution the physical access is required only to pass the two factor authentication.

A small subset of legitimate parental-control apps is available on the App Store. However, they do not provide the covert surveillance capabilities associated with actual stalkerware.

The table 2.3 summarizes some of the well known monitoring solutions, showing their availability for both Android and iOS.

Product	Android	iOS (iCloud / no jailbreak)	iOS (jailbreak)	link
Mobile Tracker Free	✓	✗	✗	www.mobile-tracker-free.com/
iKeyMonitor	✓	✗	✓	www.ikeymonitor.com
SpyX	✓	✓	✗	www.spyx.com
FlexiSpy	✓	✓	✓	www.flexispy.com
mSpy	✓	✓	✗	www.mspy.mobi
XNSpy	✓	✓	✓	www.xnspey.com
ClevGuard	✓	✓	✗	www.clevguard.org
Hoverwatch	✓	✗	✗	www.hoverwatch.com
Eyezy	✓	✓	✗	www.eyezy.com
Cocospy	✓	✓	✗	www.cocospy.com
TiSpy	✓	✓	✗	www.tispy.net
Cerberus	✓	✗	✗	www.cerberusapp.com
Qustodio	✓	✓	✗	www.qustodio.com
FindMyKids	✓	✓	✗	www.findmykids.org

Table 2.3: Compatibility of the monitoring applications

iKeyMonitor

iKeyMonitor [53] works both on Android and iOS. As other solutions, this tool on Android supports more functionalities because on iOS it works through iCloud ID and iCloud backups. To unlock all the available features, such as live photos and screenshots, or gps position, on iOS it requires the jailbreak.

The app is downloaded from an external source from the original Apple and Google marketplaces. On Android it asks to disable Google Play Protect and allow background applications.

Once installed the app dashboard can be accessed by digitizing the pin #8888 in the phone app. As other solutions it has a complete dashboard that can be remotely accessed to view all the logs and settings related to all the connected devices. This tool also offers the possibility to access via SMTP to receive via e-mail periodic updates containing screenshots and logs.

The app uses obfuscation techniques hiding itself from notification bar, recent applications and home screen. It's still visible, under a spoofed name and icon, in the installed application list.

iKeyMonitor offers different plans starting from \$9.99 for three days to \$59.99 per month for the full plan, and the full version can be activated for free for three days.

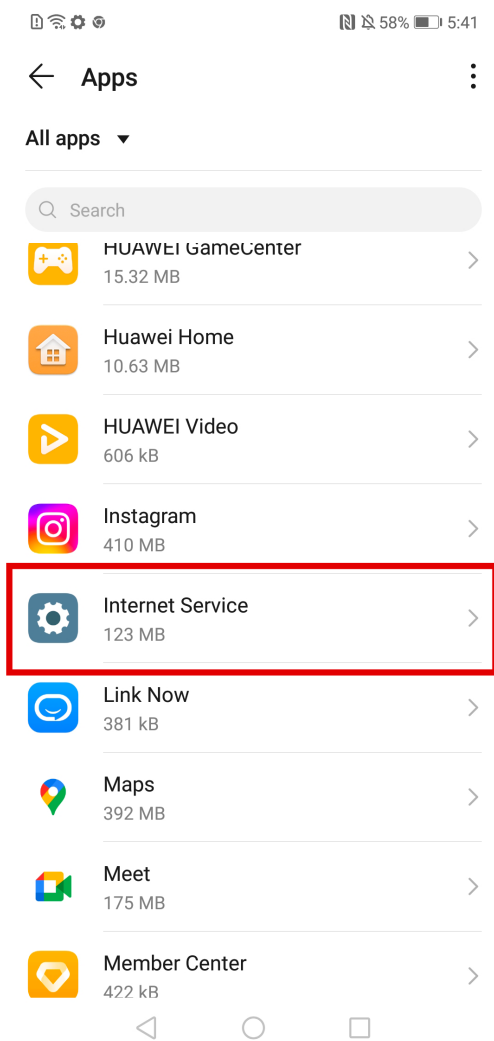


Figure 2.5: Obfuscation technique used by iKeyMonitor to hide its icon

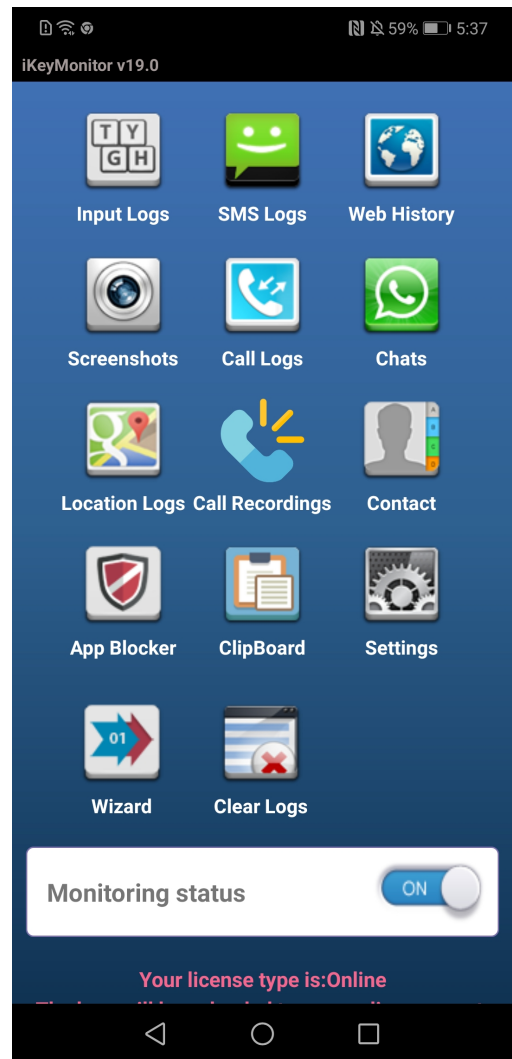


Figure 2.6: iKeyMonitor dashboard accessed on the monitored device.

Figure 2.7: Two screenshots related to iKeyMonitor application

Background

The screenshot displays the iKeyMonitor dashboard for a device named 'ANE-LX1's 9'. The interface includes a left-hand navigation menu with categories like Chats, SMS, and GPS. The main area is titled 'Menu - Dashboard' and features a grid of monitoring widgets. At the top, a status bar shows the device name, processed time (11:09 12-02), OS (Android 9), and license key (awpx9j5e6fascjd). Below this, a grid of 18 widgets shows counts for various communication methods: SMS LOGS (+0), CALL LOGS (+0), WEBHISTORY (+6), WHATSAPP (+0), FACEBOOK (+0), WECHAT (+0), VIBER (+0), LINE (+0), IMO (+0), HIKE (+0), PHOTOS (+0), and GPS (+0). There are also widgets for CLIPBOARD (+0), KEYSTROKES (+3), and SCREENSHOTS (+3). A 'TOP 10' section is currently empty. The 'LOCATIONS' section shows a satellite map. The 'RECENT ALERT WORDS' section displays two alerts for the word 'Mail' from 'Gabriele@gmail.com' on 09:26:12 2025-12-02. The 'RECENT PHOTOS' section shows three images of a laptop. The bottom of the dashboard has a navigation bar with options like Info, Search, Device, Settings, License, Download, and Help.

Figure 2.8: iKeyMonitor dashboard accessible via dedicated webpage

Mobile Tracker Free (MTF)

MTF is one of the most effective free monitoring solutions available for Android devices; however, it does not provide support for iOS platforms.

The application can be downloaded directly from the developer's website. Once installed, it requires to enable all the necessary permissions, to allow background execution, disabling battery and memory optimization, and to deactivate Google Play Protect.

Through the remote dashboard, the user can access all the logs related to installed applications and instant messaging services, block applications or websites, access live location data, and remotely view the device's camera, screen, and file explorer. Mobile Tracker Free also supports SMS commands, although these messages remain visible on the phone. Examples of available SMS commands include enabling mobile data, enabling GPS, or restarting the phone.

When the monitored device is offline or disconnected from the mobile network, collected data are stored locally and uploaded once connectivity is restored. The dashboard allows configuring periodic updates of the data collected and not yet transmitted.

In addition, one account can be used to monitor multiple devices.

The service is offered through several pricing tiers, ranging from a free plan to one-month and three-month packages. These are sold through one-time payments of €15, €45, or €150, accompanied by a recurring monthly fee.

Once installed on the phone the application is obfuscated under the name "Wi-Fi" with a spoofed icon, and it's only visible in the list of installed applications. The app settings are protected by a password, by default is *1234*.

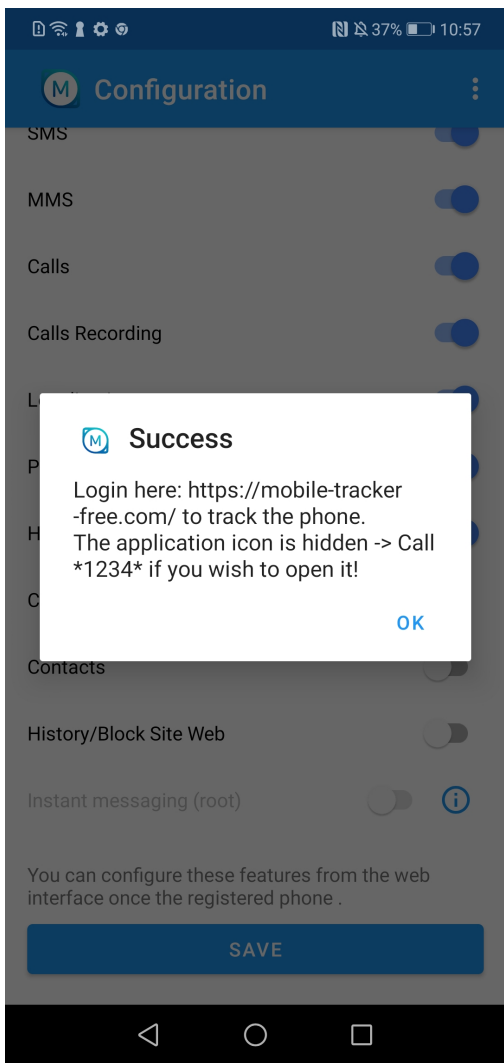


Figure 2.9: MTF installation succeeds

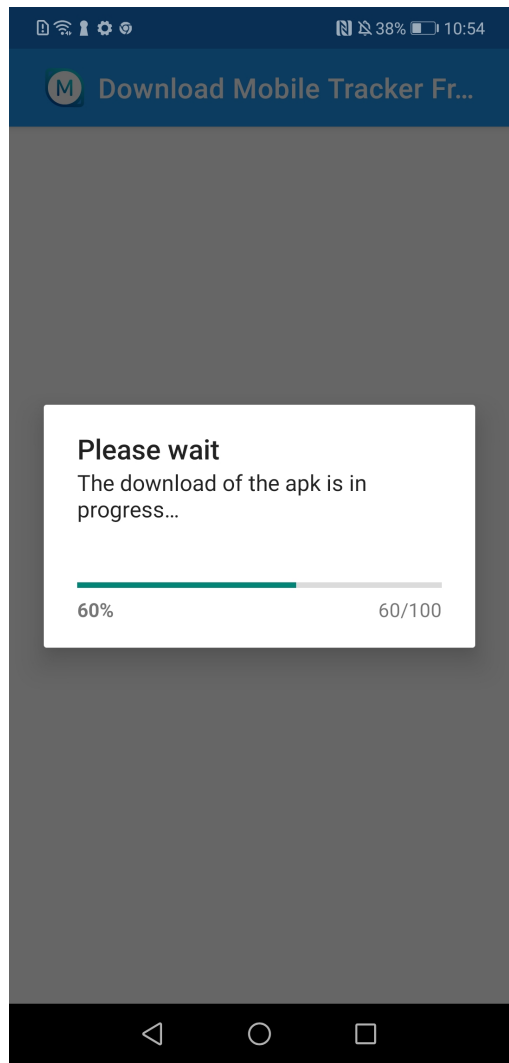


Figure 2.10: MTF downloads the demon after that Android Play Protect has been disabled

Figure 2.11: Two screenshots related to MTF installation

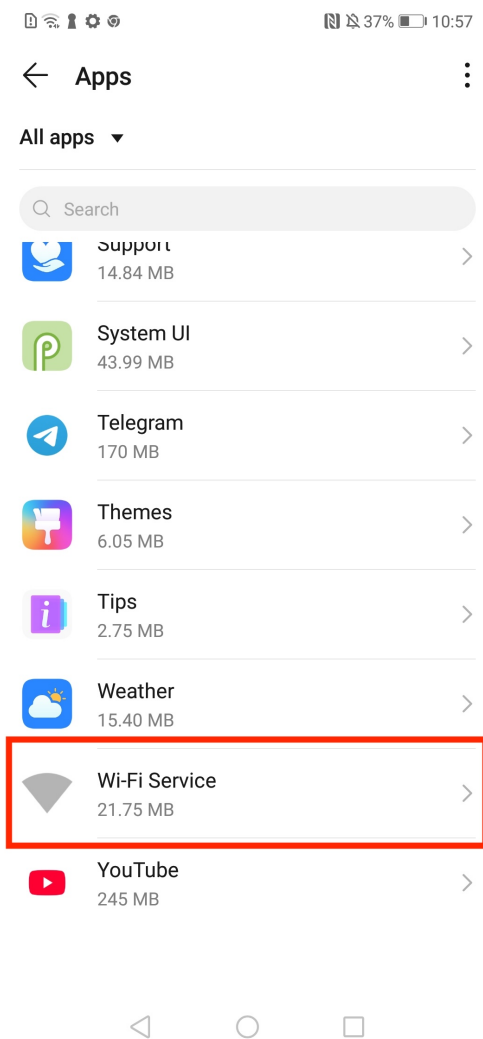


Figure 2.12: Obfuscation technique used by MTF to hide its icon

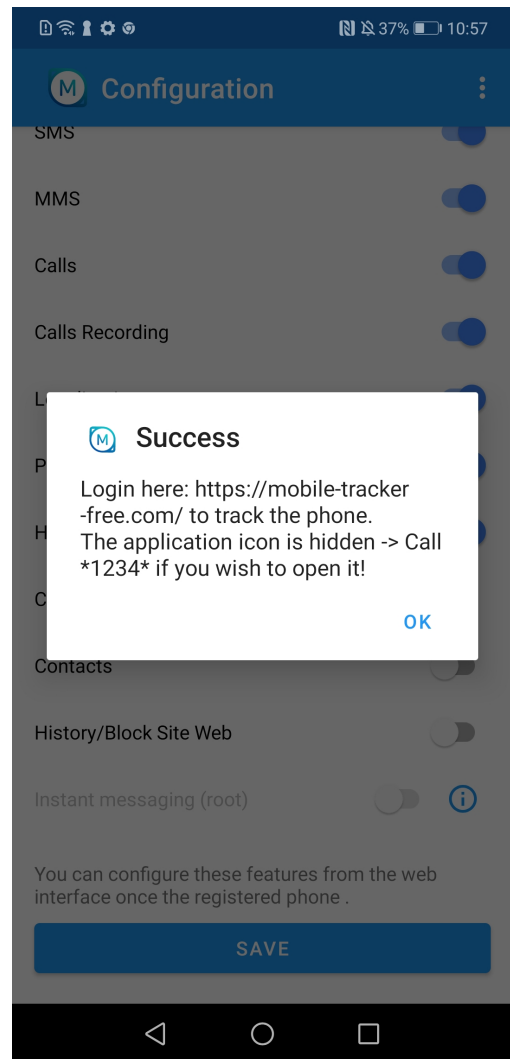


Figure 2.13: Installation process: MTF request all the necessary permissions

Background

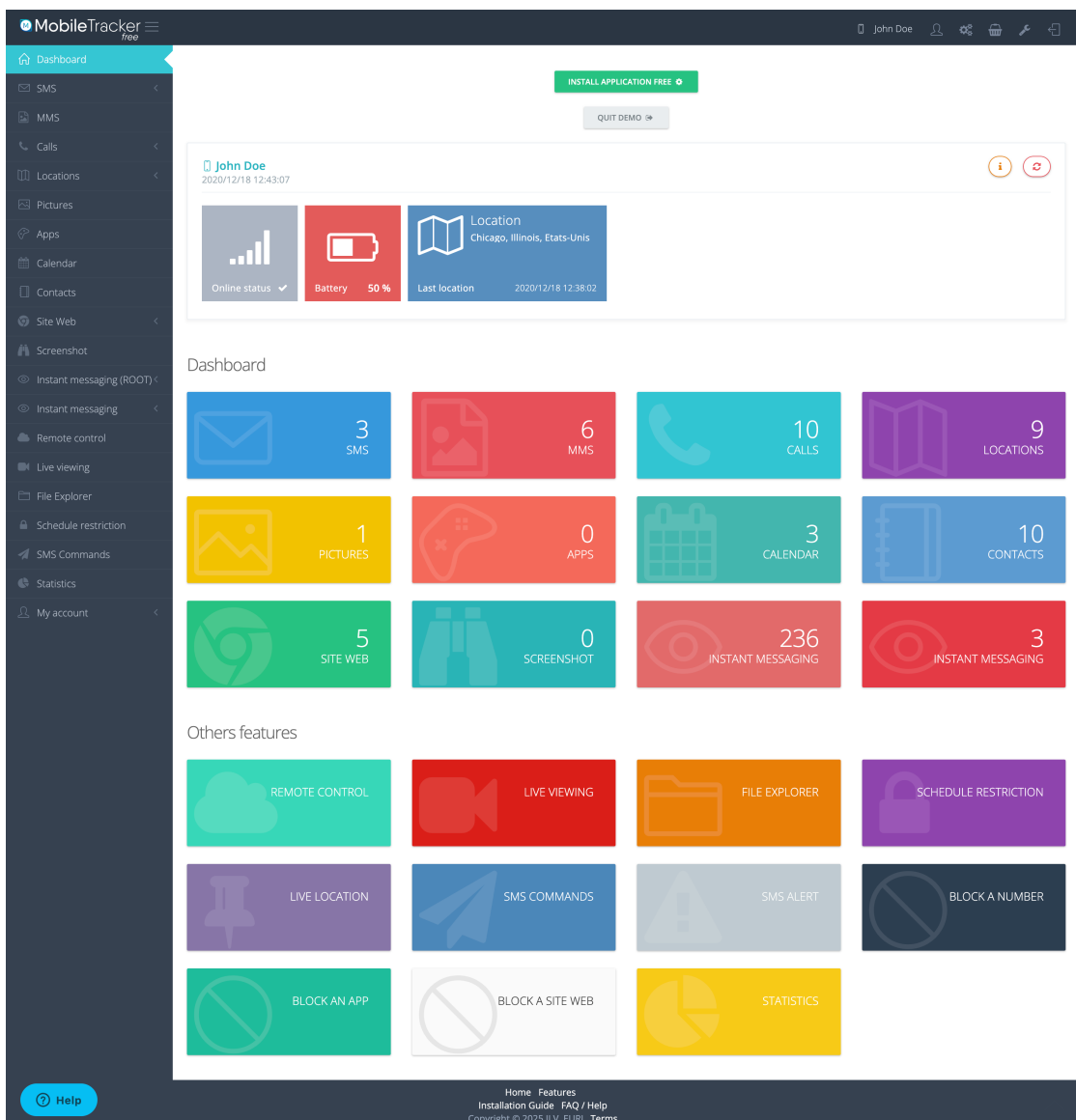


Figure 2.14: MTF dashboard accessible via dedicated webpage

Chapter 3

State-of-the-Art Spyware Detection

Mobile forensics for spyware detection represents a challenge due to the advanced obfuscation techniques put in place by these applications, designed to work without user consent. There are many approaches for spyware identification, and they operate at different levels with different techniques.

This section aims to illustrate the methodologies and tools adopted in both the scenarios of dynamic and static analysis.

The presentation starts with the NIST SP 800-101 approach to standardise mobile evidence recovery. The following paragraphs offer an overview on the dynamic analysis tools and techniques, followed by the static analysis presentation.

The last part of this section is about the ready to use tools and utilities that are considered the state of the art technologies for spyware detection on mobile devices.

3.1 Spyware Detection Techniques

Spyware apps often leave signs of compromise on the infected device. Common indicators include decreased mobile performance, rapid battery drain, weird or unexpected pop-ups, and inexplicable increases in data or bandwidth usage. [54]

The analysis on spyware typically involves different stages. When a mobile device is suspected of being infected by a spyware, there are two primary strategies to detect and analyse the spyware: Dynamic Analysis and Static Analysis.

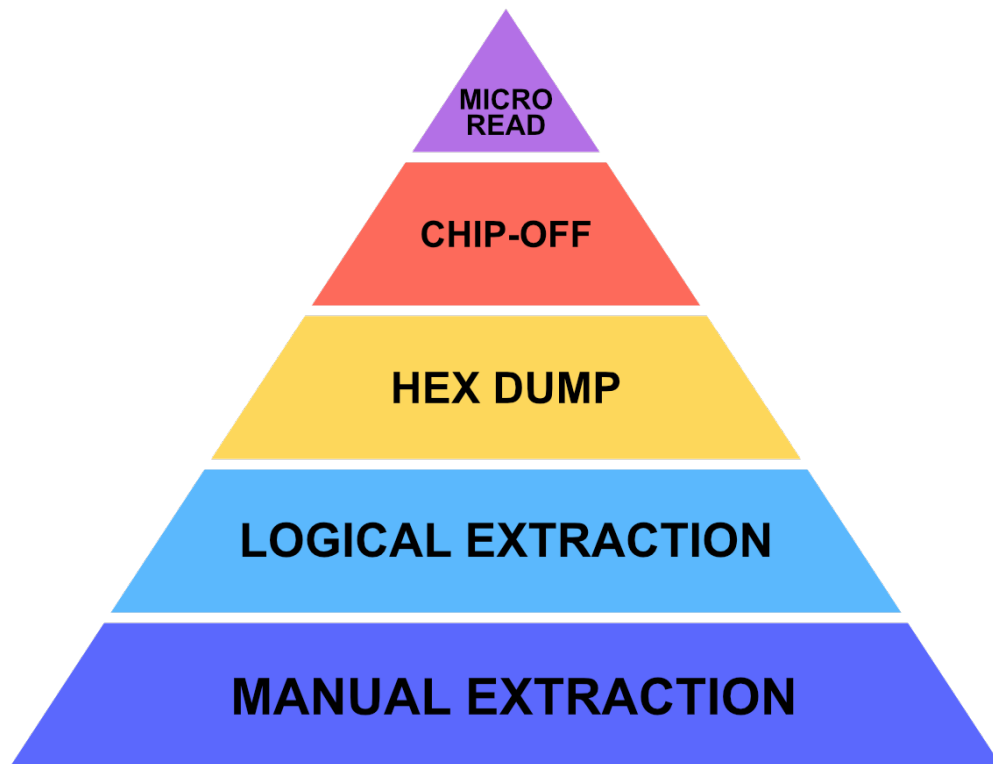


Figure 3.1: Classification system for mobile forensic acquisition

To standardize the approach to mobile evidence recovery, the NIST SP 800-101 [55] guidelines established a classification system for forensic acquisition 3.1. This hierarchy, often visualized as a pyramid, categorizes extraction methods into five levels based on their complexity, invasiveness, and the depth of data recovered. The levels are:

- **Level 1 - Manual Extraction:** the first level involves viewing the data content stored on a mobile device. This involves the direct interaction with the device's user interface by manipulating buttons, keyboard or touchscreen to view the contents of the device screen.
- **Level 2 - Logical Extraction:** this is one of the most common forensic methods, and involves the connection of the device to a workstation via USB

or wireless protocols. The tool sends API commands to the operating system to request specific data (e.g. call logs, messages, application installed). This method captures logical storage objects (files and directories) but is limited to what the OS restrictions enable the tool to see, often missing unallocated space or system-protected components. **Level 3 - Hex Dumping and JTAG Extraction:** from this stage acquisition moves to physical level. Two techniques can be used at this level:

- **Hex Dumping:** this technique is the more commonly used method by tools at this level, and involves uploading a modified bootloader into a protected area of memory (e.g., RAM). This process is performed by linking the mobile device's data interface to a flasher box, which is then connected to the forensic workstation. The flasher box issues a sequence of commands that place the mobile device into a diagnostic mode. Once in this state, the flasher box performs a bit-by-bit acquisition of the device's flash memory, capturing not only active data but also slack space that may contain remnants of deleted files. The acquired data is then transmitted to the forensic workstation using the same communication channel.
- **JTAG (Joint Test Action Group):** Many manufacturers support the JTAG standard, which defines a common test interface for processor, memory, and other semiconductor chips. This utilizes physical test points on the motherboard to communicate directly with the processor, requiring invasive access and wired connections. These methods allow for the recovery of memory including deleted data and unallocated fragments.
- **Level 4 - Chip-off Analysis:** This highly invasive method requires the physical removal (desoldering) of the memory chip from the device's circuit board. The chip is then read using specialized hardware. Chip-Off provides examiners with the ability to create a binary image of the removed chip. This is used when the device is physically damaged or when software-based physical extractions are blocked by security hardware.
- **Level 5 - Micro Read:** involving the use of an electron microscope to analyze the physical state of the logic gates on the silicon. It's extremely technically complex, time and resource consuming.

While logical acquisition is applicable in most of the cases, rather than physical acquisition, it provides limited scopes and limited access to memory, for example excluding slack space.

3.1.1 Dynamic Analysis

Dynamic or live analysis can be divided into two macro detection categories: host-based and network based.

Host-based analysis concerns examining the internal live state of the device to identify anomalies in the execution environment. Monitoring processes and system calls could reveal the interaction between the malware and the OS kernel. By tracing these calls it is possible to identify which APIs and system libraries are invoked at runtime. By monitoring low-level system calls, it's possible to detect processes that request unsafe operations, such as a sudden request for access to the device's microphone or camera buffers.

A more advanced pillar of host-based detection is the acquisition and analysis is the acquisition and analysis of live volatile memory (RAM) [56]. Since malicious code, decrypted payloads, cryptographic material, and runtime data structures must often reside in memory during execution, volatile memory analysis can reveal artifacts that are not observable through static analysis, including plaintext configuration data, command-and-control parameters, injected code, and in-memory-only malware. However, volatile memory acquisition is a technically complex technique that requires elevated privileges and is therefore generally limited to rooted devices. Furthermore, modern malware may employ memory obfuscation, hardware-backed secure execution environments, or just-in-time decryption to reduce the effectiveness of memory analysis.

Eventually, a strategy for host-based analysis is the monitoring of system telemetry and hardware resource consumption. Spyware, despite its stealth-oriented design, must consume CPU cycles, memory, and bandwidth to perform its surveillance tasks. This results in measurable physical and digital anomalies.

Significant CPU spikes, atypical network throughput, or unexplained battery drain often serve as the first indicators of a concealed infection.

Network-based detection intuition is to treat the device as a black box and monitor only its external communications.

The core of this type of detection resides in the implementation of a Man-in-the-Middle (MITM) environment. The analysed device is connected to an ad-hoc wireless network where all incoming and outgoing packets are routed through a forensic workstation acting as a transparent proxy. To make it possible it is required to install a custom CA certificate on the device to trust the MITM framework. This setup allows the investigator to intercept network packets and perform Deep Packet Inspection (DPI).

While modern network communications typically rely on Transport Layer Security (TLS) to encrypt data, MITM interception techniques can still provide valuable insight on the communication. Information such as the Server Name Indication (SNI) are visible in clear text during TLS handshake, as well as Domain Name Server (DNS) queries (in the absence of DNS-over-HTTPS or DNS-over-TLS) used to resolve spyware infrastructure domains. Furthermore, modern applications and advanced spyware frequently employ TLS certificate pinning mechanisms to bind

network connections to specific certificates or public keys, thereby preventing man-in-the-middle interception even when a trusted root certificate is installed on the device. As a result, effective traffic interception generally requires runtime hooking, dynamic instrumentation, or binary modification, which is typically feasible only on rooted or jailbroken systems [57].

Dynamic malware analysis involves executing the malware’s code within a strictly controlled environment named sandbox [58]. This approach allows researchers to observe how it interacts with the system in real time and which evasion techniques are put in place by the malicious application.

By analysing the application while it is active, investigators can profile its runtime behaviour. This includes monitoring the process lifecycle, observing calls to sensitive system libraries, and recording the exfiltration of data.

The execution environment is typically connected to an ad-hoc created network to capture exfiltration attempts without allowing the spyware to communicate with its real Command-and-Control (C2) server.

Sandboxes can mimic various user activities and system configurations to analyse how malware responds, providing insights into its behaviour. Sandbox addresses the gap of behaviour-based or signature-based detection systems, enabling the discovery of zero-day threats.

There are two main limitations about executing the malware in a sandbox environment:

- Evasion technique: sophisticated malware and, as well as sophisticated spyware, are designed to detect in which environments are executed, stopping or altering their execution to avoid detection. Malicious applications often include code to check if they are executing in a virtual environment.
- Environment differences: malware can behave differently depending on which system configuration, version and setting are executed. For example if the environment isn’t correctly replicated, some exploits might not work as designed by the attacker.

Sandboxing techniques can additionally provide information about runtime telemetry, including CPU utilization, memory consumption, and network usage, which can be leveraged for behavioural analysis.

System logs can be considered dynamic/live analysis, as they reflect runtime events produced during execution and are inherently volatile, being subject to deletion by the operating system over time. Logs are a crucial part of the analysis because they provide a chronological record of system events.

iOS utilizes the unified logging system [59] and generates extensive logs, which

store historical records of process execution and network transitions.

The forensic value of iOS logs depends strongly on how long they are retained. Empirical comparison between logs collected directly via the Unified Logging System and those included in a sysdiagnose archive shows that events can disappear from sysdiagnose archives within 24 hours, while the same events remain accessible in directly collected Unified Logs for at least a week. This limited retention emphasizes the importance of timely log acquisition when investigating iOS devices, as critical evidence can be lost if collection is delayed [60] [61].

A sysdiagnose is a comprehensive diagnostic archive that can be generated manually by the user through a button combination (pressing both volume buttons and the side/top button for 1–1.5 seconds) or via the Analytics & Improvements menu in the iOS settings. After the snapshot is triggered the generating process starts and can last up to thirty minutes.

For a deeper reconstruction, the *log collect* command-line interface (CLI) can be utilized. Unlike the manual snapshot, this method allows for a more extensive extraction of the logs stored on the filesystem. To perform a log collection via CLI, the iPhone must be connected via USB, unlocked at least once, and the computer must be trusted by the device.

In Android forensics, unlike iOS’s centralized Unified Logging, the data is fragmented across different system services. The main tool for logs acquisition is Android Debug Bridge (ADB) which query live system services and volatile buffers:

- *logcat*: *adb logcat* command is the primary method for extracting the OS’s circular buffers. This command dumps the current state of logs storing buffers. Logcat is useful for capturing real-time indicators of compromise (IOCs), such as unauthorized permission requests or background service executions occurring during the forensic examination.
- *dumpsys*: provides a detailed snapshot of system services and states. Specifying the target is possible to get valuable snapshots of the system. Some of them are: *dumpsys package*, to get installed apps; *dumpsys netstats*, to show network usage statistics; *dumpsys location*, to display GPS and location service activity.
- *getprop*: queries the Android property service, which acts as a centralized repository for system-wide configuration settings and build information.

In case of spyware one of the principal techniques adopted is the network traffic analysis. It’s used to firstly detect whether the device is effectively infected, before proceeding with an in depth analysis of the device.

During network traffic analysis, the device is connected to an ad-hoc created network in which another device acts as Man-in-the-Middle (MITM), intercepting and analysing all the incoming and outgoing traffic from the analysed device. This type of approach allows experts to study the behaviour of the spyware, identify network calls performed and determine the destination servers to which the traffic is directed.

Advanced spyware often utilise obfuscation techniques to exfiltrate data, by hiding themselves behind legitimate application. In these cases while the specific data content may remain hidden, abnormal traffic patterns, such as unusually large volumes of transmitted data, can still reveal suspicious activities.

In addition to network traffic analysis, system log analysis plays a fundamental role in spyware investigations, as it can reveal execution traces, system interactions, and anomalous behaviors that may not be directly observable through network data alone.

3.1.2 Tools for Network Dynamic Analysis

Intrusion Detection and Intrusion Prevention Systems (IDS and IPS)

An intrusion detection system [62] is an application that monitors network traffic and searches for known threats or attack patterns to detect malicious activities. When a malicious activity is detected, the IDS produces an alert.

Most IDS solutions are listen-only devices that simply monitor and report suspicious traffic, while some advanced can take action blocking the suspicious traffic.

An intrusion prevention system (IPS) [63] goes beyond IDS solutions that are limited to the monitoring and detection by blocking or preventing security risks. An IPS can both monitor for malicious events and take action to prevent an attack from taking place.

So, IPS solutions offer a proactive approach for mitigating threats as soon as possible.

There are different types of IDS, the most common are network-based (NIDS) and host-based (HIDS) intrusion detection systems. There are also other categories of IDS like: protocol-based (PIDS), application-protocol-based (APISD), virtual machine-based (VMISD) and stack-based (SBISD).

Intrusion detection system can also be divided in two variants based on the detection method:

- Signature-bases (SIDS): monitors all packets on a network and looks for specific patterns and sequences that match known attack signatures. While it's efficient in detection of known pattern attacks, it can't recognise attacks

with no known patterns.

- Anomaly-based (AIDS): monitors traffic on a network and compares it with a predefined baseline that is considered "normal." It detects anomalous activity and behaviour across the network, including bandwidth, devices, ports, and protocols. This type of detection uses machine learning to analyse large amounts of network data and traffic.

For spyware detection the focus goes on the network-based detection system. The NIDS monitors all traffic flowing to and from devices on the network, making determinations based on packet contents and metadata. These solutions can report false alarms, giving false positives, and false negatives, mistaking an actual security threat for legitimate traffic.

Furthermore, there are numerous techniques that can be used to avoid detection by IDS [64]. Some of them are:

- Fragmentation: it divides packets into smaller fragmented packets that will be later reconstructed at the IP layer by the recipient node. This allows the attacker to hide the attack signature.
- Flooding: by spoofing User Datagram Protocol (UDP) and Internet Control Message Protocol (ICMP) is possible to camouflage the anomalous activities of the perpetrator, creating an overwhelming volume of traffic.
- Obfuscation: it can be used to avoid being detected by making a message difficult to understand, thereby hiding an attack.
- Encryption: it can be used to evade detection by encrypting protocols that cannot be read by the IDS.

Deep Packet Inspection (DPI) techniques

Deep Packet Inspection (DPI) is an advanced network security technique that goes beyond normal packet filtering by examining the header and the data payload of data packets travelling over a network [65]. By predefined rules configured by administrators, service providers, or security systems, DPI examines metadata as well as content to enable more accurate threat and anomaly detection.

Unlike classical stateful packet inspection that bases its examination on source and destination ports, IP addresses, and headers, DPI provides insight into the actual data being sent. It is therefore able to detect malicious payloads, attempts at data exfiltration, policy breaches, malware, spam, or other hidden threats.

DPI can employ many inspection methods, such as protocol anomaly detection (blocking content that fails to meet pre-specified protocol standards), pattern

or signature matching (matching packet payloads against a database of known attacks), and integration with IPS to prevent attacks in real-time.

DPI is also utilized beyond enterprise security: ISPs and governments can employ it for censorship, monitoring, or traffic management, through intercepting communications, filtering out unauthorized sources, or extracting internet data.

One of the main limitations of DPI is its inability to accurately inspect packets when communications are encrypted. With the widespread deployment of protocols such as TLS, HTTPS and VPN, increasingly network communications are protected by end-to-end encryption systems. In that scenario, DPI can observe metadata of the flow, such as IP addresses, destination ports, packet size, and frequency, but cannot observe the content of the application layer.

Wireshark tool

Wireshark is a cross-platform, free, and open-source network packet analyser.

It uses PCAP (Packet Capture), an application programming interface, to capture the network traffic. With Wireshark, it is possible to intercept packets in real-time by selecting the network interface at the start of the application and setting it in promiscuous mode.

Additionally, Wireshark allows users to import and analyse .pcap files produced by other applications.

Wireshark key feature are:

- Packet Capture: Wireshark enables listening to a network connection in real-time and captures entire streams of traffic.
- Filtering: Wireshark provides two levels of filters:
 - Capture filters, which are applied at the initial data collection stage to filter only the relevant types of network traffic.
 - Display filters: which help isolate data of interest after the traffic has been captured.
- Visualization: It's possible to isolate and visualize entire conversation and network streams for detailed analysis.

Machine Learning Techniques

As demonstrated by numerous researches, machine learning models can be used for malware detection, without doing pattern-matching or basing the detection on IOCs.

Machine learning techniques can be mainly divided in two categories: supervised machine learning and unsupervised machine learning [66]. Supervised machine

learning is a type of machine learning where the model is trained on a label dataset. Several types of algorithms are used in this context. Regression algorithms predict output values by identifying linear relationships between real or continuous values. Classification algorithms aim to predict categorical output variables and include methods such as logistic regression, k-nearest neighbors and support vector machines. The random forest algorithms predict a value or category by combining the results from a number of decision trees. Finally, neural networks, which simulate the functioning of the human brain, and Naïve Bayes classifiers enable classification tasks on large datasets.

Unsupervised machine learning employs algorithms to analyze and draw inferences from unlabelled data. The model is not given specific instructions on what to look for, but rather is tasked with recognising patterns, structures and anomalies. These strategies are intended to facilitate data processing and enable pattern recognition. Unsupervised machine learning has the advantage of acting directly on data without the needing of a dataset for training.

Cluster analysis is the most common unsupervised learning approach, which uses clustering algorithms to group data points based on similarity in their values, as seen in applications such as anomaly detection. Furthermore, association algorithms enable data scientists to discover relationships between data objects inside large datasets, enabling tasks such as data visualisation and dimensionality reduction. There are different clustering techniques. Centroid-based clustering uses a pre-defined number of centres, or centroids, to group data points according to their distance from each center, with K-means being a common example. The goal of K-means is to minimize the sum of distances between data points and their cluster centroids, although it is not robust to outliers. Hierarchical clustering connects data points one by one by determining their nearest neighbors until a single large cluster is created, making it particularly useful for hierarchical data such as taxonomies. Distribution-based clustering assumes that data is generated from probabilistic distributions and creates "bands" representing confidence intervals, but it is often difficult to implement.

In the current landscape of network threat hunting supervised machine learning represents the mainly adopted solution for identifying patterns, detecting anomalies, and predicting potential threats [67]. Supervised ML models are integrated into modern Intrusion Detection Systems (IDS) to move beyond static signature-matching. Instead of looking for a specific byte sequence, these tools look for the behavioural signature of the malware.

In particular, random forest is one of the most widely deployed algorithms in network security due to its robustness against overfitting. It works by constructing multiple decision trees during training and assigning classes by the majority vote of the trees during scanning [Figure: 3.2].

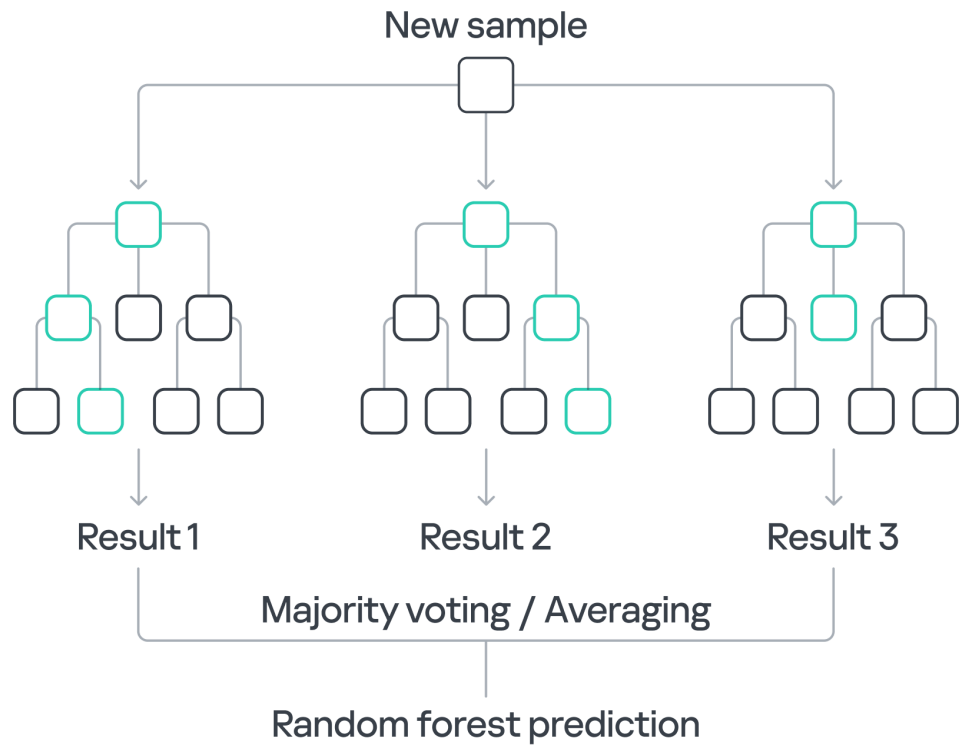


Figure 3.2: Random forest algorithm representation. Source [67].

A machine learning dataset is a collection of data used to train, validate, and test ML models.

$$\text{TPR} = \frac{TP}{TP + FN} \quad \text{Precision} = \frac{TP}{TP + FP} \quad (3.1)$$

TP: true positive
 TN: true negative
 FP: false positive
 FN: false negative

The training data must be carefully labelled in order to achieve an adequate Recall, which represents the capacity to identify all malicious instances, and Precision, which represents the capacity to reduce false positives. Overfitting is a risk for a supervised model that was trained on an extremely rigid or particular dataset. Instead of discovering the underlying patterns, the algorithm "memorizes" the training data in this condition.

The paper "Android Spyware Detection Using Machine Learning: A Novel Dataset" [68] proposes the use of Random Forest algorithm to identify commercial Android spyware through network traffic analysis. The authors collected network traffic from five spyware applications (UMobix, TheWiSPY, MobileSPY, FlexiSPY, and mSPY) ensuring identical experimental conditions for each tool. The data is categorized into three classes:

- Normal smartphone traffic
- Traffic during the spyware installation phase (pre-infection)
- Traffic during spyware operation, such as data exfiltration (post-infection)

The research highlights that while the overall average accuracy was 79% for binary classification and 77% for multi-class classification, some systems were much harder to detect than others. Tools like FlexiSPY and MobileSPY showed lower recall and precision because they possess a high ability to conceal their digital footprints, hiding a significant portion of their network traffic activity.

This paper emphasizes that developing effective detection models for commercial Android spyware involves several significant challenges regarding dataset creation, feature selection, and parameter tuning. Also it is not possible to rely on existing mobile malware dataset, because their technical operations differ significantly, and there is no dataset for spyware.

3.1.3 Static Analysis

Static analysis involves the examination of a device's data or an application's binary without executing the code. This methodology focuses on the structural and declarative properties of the software and the operating system's filesystem. By analysing the malware code, logs, backups or a forensic image of the system, it is possible to extract features useful for the creation of indicators of compromise (IOCs) databases. Furthermore, this approach allows reconstructing the spyware's configuration without the risks associated with live execution.

Differently from network analysis, static analysis varies depending on the operating system, IOS or Android, and its version. Each system, in fact, has differences

in architecture, produces different types of log, and applications are written with different code languages.

Static analysis is more complex, because a lot of elements can be considered and many logs need to be evaluated. However, thanks to the large amount of data that can be extrapolated from a device, this type of analysis is more complete and exhaustive.

Most of the static analysis tools work with a signature-based approach. This approach bases the malware detection on matching an app's byte-level signature against known malware databases. It is commonly used in mobile antivirus tools for real-time scanning.

Although their simple and resource-efficient implementation, they are ineffective against spyware and advanced malware which introduce metamorphic engines, code obfuscation, or dynamically code loading, where key features are hidden or appear only at runtime.

As a result, behaviour-based and automated dynamic analysis techniques are required to detect novel and zero-day malware at scale, replacing traditional manual analysis [69].

As noted, static analysis is highly dependent on the underlying architecture of the operating system (iOS vs. Android) and the specific format of its application packages:

- **Android APK Analysis:** Android applications are distributed as APK (Android Package Kit) files, which are essentially ZIP containers. Static analysis involves decompressing the APK to examine the `AndroidManifest.xml`, which contains application's requested permissions, and the source code. Android application's code can be disassembled into Smali or decompiled into Java/Kotlin, allowing for a deep inspection of the application's logic.
- **iOS IPA Analysis:** iOS IPA files, which are zipped files that include the application executable and other application's resources, utilize the Mach-O binary format. Apple encrypts the main binary when the package is uploaded to the App Store, and signs the app with an AppStore certificate, so if the application is retrieved from the AppStore need to be decrypted before being analysed.

An important artifact in iOS are property list (`.plist`) files, which define the application's capabilities and environment.

Permission mapping

A first step into static analysis could be the permission mapping. On Android, this is performed by auditing the `AndroidManifest.xml`. Unlike Android, iOS splits

permission-related information across two layers: *Info.plist*, which contains user-facing usage disclosures and configuration metadata, and application entitlements, which are cryptographically bound to the app's code signature and grant access to privileged system capabilities.

These files include all the permissions asked by an application. Identifying apps that request permissions beyond their expected scope can be a useful trigger for deeper analysis. Spyware, in particular, is often identified by over-privileged permission requesting access to sensitive APIs (e.g. camera, microphone) that are not consistent with the app's declared functionality.

Considering Android applications, one of the most important files is the `AndroidManifest [70].xml`. Each application project has one, and it describes the essential information of the app. These information include:

- The components of the app, including all activities, services, broadcast receivers, and content providers.
- The permissions that the app needs to access sensitive user data. Here are also declared the permissions that the other app needs to access content from this app.
- Software and Hardware requirements, in order to check the compatibility with a device.

There are three types of permission on Android [71]:

- Install-time permissions: allow the app to access restricted data or perform restricted operations that minimally affect the system or other app. These are automatically guaranteed by the system when the app is installed.
- Runtime permissions: are the dangerous ones that give the application additional access to restricted data or allow it to perform restricted operations. These permissions are requested each time before each access.
- Special permissions: can be defined only by the platform and OEMs to protect particularly dangerous actions.

The information contained in the manifest file is a powerful source of information in spyware and monitoring application detecting, because starting from this file it's possible to understand the behaviour of the APK.

Example of Mobile-Tracker-Free application manifest file:

```
1 SHA2-256(app.apk)=
2 a9be474958857c0698b4526089412e34173fc1406b00b4b3327b7a83db0b4fbb
```

```
1 <activity-alias
2     Android:enabled="@bool/app_icon_value"
3     Android:exported="true"
4     Android:name="com.app.activity.FirstActivityAlias"
5     Android:targetActivity="com.app.activity.FirstActivity">
6     <intent-filter>
7         <action Android:name="Android.intent.action.MAIN"/>
8         <category Android:name="Android.intent.category.LAUNCHER"/
9     >
10    </intent-filter>
11 </activity-alias>
```

Listing 3.1: Mechanisms used by the application to enable or disable the icon

This is one of the common mechanisms used by the application to enable or disable the icon in the launcher. This alias checks the value stored in `res/values/booleans.xml` `<bool name="app_icon_value">true</bool>` and when the alias is disabled the application icon becomes invisible for the user.

```
1 activity Android:Android:excludeFromRecents="true"
```

Listing 3.2: Implementation of the `excludeFromRecents` attribute

Most of the activities of this application are configured to not appear in the recent list, this technique is used to enhance the obfuscation.

```
1 <receiver Android:exported="false" Android:name="com.app.received
2     .BootReceived">
3     <intent-filter Android:priority="999">
4         <action Android:name="Android.intent.action.
5     BOOT_COMPLETED"/>
6     </intent-filter>
7 </receiver>
```

Listing 3.3: Start application after the boot

This setting is used to automatically start the application after a boot.

```
1 <receiver Android:exported="false" Android:name="com.app.received.  
  PolicyActivate$PolicyAdmin" Android:permission="Android.  
  permission.BIND_DEVICE_ADMIN">  
2   <meta-data Android:name="Android.app.device_admin"  
  Android:resource="@xml/device_admin"/>  
3   <intent-filter>  
4     <action Android:name="Android.app.action.  
  DEVICE_ADMIN_ENABLED"/>  
5     <action Android:name="Android.app.action.  
  ACTION_PASSWORD_FAILED"/>  
6   </intent-filter>  
7 </receiver>
```

Listing 3.4: Device Admin Receiver definition

This block of the manifest defines a Device Admin Receiver, a component that can receive events and have special privileges on the system.

This component can activate the root functionalities and can be used to prevent the app uninstallation.

```
1 <service Android:enabled="true" Android:exported="true"  
  Android:label="@string/app_name_accessibility" Android:name="  
  com.app.service.AccessService" Android:permission="Android.  
  permission.BIND_ACCESSIBILITY_SERVICE">  
2   <intent-filter>  
3     <action Android:name="Android.accessibilityservice  
  .AccessibilityService"/>  
4   </intent-filter>  
5   <meta-data Android:name="Android.accessibilityservice"  
  Android:resource="@xml/accessibility_service"/>  
6 </service>
```

Listing 3.5: accessibility service access request

In this section of manifest the application requires the access to the accessibility service.

Many malicious applications abuse the accessibility service [72]. This service can see everything that appears on the screen, perform inputs, trigger action at specific moments and steal sensitive information.

```
1 service Android:enabled="true" Android:exported="false"  
    Android:name="com.app.service.NotificationListener"  
    Android:permission="Android.permission.  
    BIND_NOTIFICATION_LISTENER_SERVICE">  
2         <intent-filter>  
3             <action Android:name="Android.service.notification  
    .NotificationListenerService"/>  
4         </intent-filter>  
5 </service>
```

Listing 3.6: Access to notifications

Here are reported services that enable the ability to read all the notifications.

```
1 <application Android:allowBackup="false"  
    Android:appComponentFactory="Androidx.core.app.  
    CoreComponentFactory" Android:extractNativeLibs="false"  
    Android:icon="@mipmap/ic_app" Android:label="@string/  
    name_hide_app" Android:largeHeap="true"  
    Android:manageSpaceActivity="com.app.activity.FirstActivity"  
    Android:name="com.app.activity.MyApp"  
    Android:requestLegacyExternalStorage="true" Android:theme="  
    @style/AppTheme">
```

Listing 3.7: Disable of automatic backups

Another layer of obfuscation is achieved in these lines. Here, in fact, the application excludes its private data (as settings, files and databases) from the automatic backups on Google Drive that are used for the Android backup and restore service.

```
1 <uses-permission Android:name="Android.permission.  
    FOREGROUND_SERVICE"/>  
2 <uses-permission Android:name="Android.permission.  
    FOREGROUND_SERVICE_MEDIA_PROJECTION"/>  
3 <uses-permission Android:name="Android.permission.  
    FOREGROUND_SERVICE_DATA_SYNC"/>  
4 <uses-permission Android:name="Android.permission.  
    FOREGROUND_SERVICE_LOCATION"/>
```

```
5 <uses-permission Android:name="Android.permission.  
    FOREGROUND_SERVICE_CAMERA"/>  
6 <uses-permission Android:name="Android.permission.  
    FOREGROUND_SERVICE_MICROPHONE"/>  
7 <uses-permission Android:name="Android.permission.READ_CALENDAR"/>  
8 <uses-permission Android:name="Android.permission.  
    REQUEST_DELETE_PACKAGES"/>  
9 <uses-permission Android:name="Android.permission.  
    USE_FULL_SCREEN_INTENT"/>  
10 <uses-permission Android:name="Android.permission.CAMERA"/>  
11 <uses-permission Android:name="Android.permission.FLASHLIGHT"/>  
12 <uses-permission Android:name="Android.permission.READ_CONTACTS"/>  
13 <uses-permission Android:name="Android.permission.GET_ACCOUNTS"/>  
14 <uses-permission Android:name="Android.permission.  
    ACCESS_COARSE_LOCATION"/>  
15 <uses-permission Android:name="Android.permission.  
    ACCESS_FINE_LOCATION"/>  
16 <uses-permission Android:name="Android.permission.  
    ACCESS_BACKGROUND_LOCATION"/>  
17 <uses-permission Android:name="Android.permission.RECORD_AUDIO"/>  
18 <uses-permission Android:name="Android.permission.  
    MODIFY_AUDIO_SETTINGS"/>  
19 <uses-permission Android:name="Android.permission.READ_PHONE_STATE  
    "/>  
20 <uses-permission Android:name="Android.permission.READ_CALL_LOG"/>  
21 <uses-permission Android:name="Android.permission.  
    PROCESS_OUTGOING_CALLS"/>  
22 <uses-permission Android:name="Android.permission.CALL_PHONE"/>  
23 <uses-permission Android:name="Android.permission.  
    READ_PHONE_NUMBERS"/>  
24 <uses-permission Android:name="Android.permission.READ_SMS"/>  
25 <uses-permission Android:name="Android.permission.RECEIVE_SMS"/>  
26 <uses-permission Android:name="Android.permission.RECEIVE_MMS"/>  
27 <uses-permission Android:name="Android.permission.SEND_SMS"/>  
28 <uses-permission Android:name="Android.permission.  
    WRITE_EXTERNAL_STORAGE"/>  
29 <uses-permission Android:name="Android.permission.  
    READ_EXTERNAL_STORAGE"/>  
30 <uses-permission Android:name="Android.permission.  
    MANAGE_EXTERNAL_STORAGE"/>  
31 <uses-permission Android:name="Android.permission.  
    READ_MEDIA_IMAGES"/>  
32 <uses-permission Android:name="Android.permission.READ_MEDIA_VIDEO  
    "/>  
33 <uses-permission Android:name="Android.permission.  
    QUERY_ALL_PACKAGES"/>  
34 <uses-permission Android:name="Android.permission.INTERNET"/>  
35 <uses-permission Android:name="Android.permission.  
    ACCESS_NETWORK_STATE"/>
```

```
36 <uses-permission Android:name="Android.permission.  
    ACCESS_WIFI_STATE"/>  
37 <uses-permission Android:name="Android.permission.  
    CHANGE_WIFI_STATE"/>  
38 <uses-permission Android:name="Android.permission.  
    CHANGE_NETWORK_STATE"/>  
39 <uses-permission Android:name="Android.permission.VIBRATE"/>  
40 <uses-permission Android:name="Android.permission.DISABLE_KEYGUARD  
    "/>  
41 <uses-permission Android:name="Android.permission.WAKE_LOCK"/>  
42 <uses-permission Android:name="Android.permission.  
    RECEIVE_BOOT_COMPLETED"/>  
43 <uses-permission Android:name="Android.permission.  
    REQUEST_IGNORE_BATTERY_OPTIMIZATIONS"/>  
44 <uses-permission Android:name="Android.permission.ACCESS_SUPERUSER  
    "/>  
45 <uses-permission Android:name="Android.permission.REBOOT"/>  
46 <uses-permission Android:name="Android.permission.  
    PACKAGE_USAGE_STATS"/>  
47 <uses-permission Android:name="Android.permission.  
    SYSTEM_ALERT_WINDOW"/>  
48 <uses-permission Android:name="Android.permission.  
    SCHEDULE_EXACT_ALARM"/>  
49 <uses-permission Android:name="Android.permission.  
    POST_NOTIFICATIONS"/>  
50 <uses-permission Android:name="com.google.Android.c2dm.permission.  
    RECEIVE"/>  
51 <uses-permission Android:name="com.google.Android.finsky.  
    permission.BIND_GET_INSTALL_REFERRER_SERVICE"/>  
52 <uses-permission Android:name="com.google.Android.gms.permission.  
    AD_ID"/>  
53 <uses-permission Android:name="Android.permission.  
    ACCESS_AD SERVICES_ATTRIBUTION"/>  
54 <uses-permission Android:name="Android.permission.  
    ACCESS_AD SERVICES_AD_ID"/>
```

Listing 3.8: Full list of permissions

Eventually, it is reported the full list of permissions that the app requires from the system to perform a complete monitoring over all the device functionalities. The list includes services such as receiving, reading and sending SMS, access to both the cameras, access to the location service, and others.

Regarding iOS Info.plist file the following information are included: Core application information: the Info.plist defines fundamental metadata such as the Bundle Identifier, which uniquely identifies the application within the iOS ecosystem.

App's version: (CFBundleShortVersionString) and build number (CFBundleVersion), used by the operating system and the App Store for update management and app identification. Sensitive resources access declaration: access to sensitive system resources (e.g. camera, microphone, location) is declared through NS*UsageDescription keys, which provide user-facing justification and are mandatory to avoid runtime crashes when invoking the corresponding APIs.

The Info.plist file is structured as an XML document composed of key-value pairs. Below is a simple example [73]:

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
3   "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
4 <plist version="1.0">
5   <dict>
6     <key>CFBundleName</key>
7     <string>MyApp</string>
8
9     <key>CFBundleIdentifier</key>
10    <string>com.company.appname</string>
11
12    <key>CFBundleVersion</key>
13    <string>1.0.0</string>
14
15    <key>NSCameraUsageDescription</key>
16    <string>This app requires camera access to take photos.</
17    string>
18  </dict>
19 </plist>

```

The analysis of data at rest allows for the discovery of a malware's persistent traces and artifacts left within the device's non-volatile memory. In mobile forensics, this typically involves the systematic inspection of Forensic Images or Offline Backups [74].

On iOS, backups (created via Finder, iTunes, or iCloud) are a primary source of evidence. iCloud and standard (non cyphered) backups are both copies of the data on the device. Both contain the same amount of data, but differ how they are stored: the first is divided in cyphered blocks on cloud, while the second one is stored locally.

Encrypted backup is a variant of the standard backup which is encrypted with a password chosen by the user. Differently from the other two, encrypted backup contains more data, including sensitive data such as the complete keychain, Safari browsing history and passwords.

Four standard files are typically found within an iOS backup directory:

- **Info.plist**: provides backed-up device-specific information, including the device name, iOS version, phone number, ICCID, IMEI, and backup timestamps.

- ***Manifest.plist***: describes the overall content and properties of the backup, including metadata such as the encryption status, the backup version, and information about installed applications.
- ***Manifest.mdbd* / *Manifest.db***: is the main index of the backup, storing metadata for each backed-up file, including domain, relative path, file size, permissions, and MAC (Modified, Accessed, Created) timestamps.
- ***Status.plist***: Contains information about the backup process itself, indicating whether the backup was completed successfully and providing status-related metadata.

Android Root vs non Root Forensic acquisition

In modern Android ecosystems, static analysis increasingly overlaps with live acquisition, as data accessibility is tightly bound to the device's unlocked state. On Android devices static forensic analysis enables the extrapolation of many useful data. Furthermore the amount and the type of the data is conditioned by the privileges available on the device. [75]

A non-rooted device, in fact, enables to perform only logical acquisition, that means that can be acquired whatever stored and available logical information. This operation can be performed through the ADB interface, by extracting some piece of information, or by performing a backup, either directly on the device or via ADB interface.

Additionally, Android forensic tools such as Oxygen and Belkasoft X use agent-based acquisition, where an agent is installed on the device after USB or Wi-Fi connection. This method typically allows acquisition of data like calls, messages, contacts, calendars, device profile, Wi-Fi and Bluetooth data, file structure, and app data from external storage. However, as presented, accessing application files and app data from internal memory requires a full file system acquisition.

Rooted devices can be acquired either logically and physically, capturing logical partition, system partition, and volatile memory (RAM). With root privileges mounted partitions can be checked with the command “cat partitions”.

It is important to distinguish between permanent rooting techniques and temporary or exploit-based privilege escalation. The latter, often employed by forensic tools (such as Cellebrite UFED, Oxygen Forensic Detective and Magnet AXIOM), aims to obtain transient root access in memory without permanently modifying the system or boot partitions, thereby reducing the forensic analysis footprint on the device. This is achieved by leveraging specific vulnerabilities to inject an agent, a binary payload, in RAM that can be executed with root privileges and execute a full acquisition bypassing system APIs restrictions. However, success depends heavily on device model, Android version, and security patch level.

Partition on a rooted device can be acquired manually with the command-line tool `dd` [76], a Linux powerful utility for low-level data copying and conversion, used for disk cloning, creating disk images, and partition backups.

When performing a low level disk copy, for example with `dd` command, is important to put in place a write blocker [77]. A write blocker is a fundamental tool that can be either hardware or software that ensures the data integrity during the acquisition of the memory, blocking any write attempt on the analysed device. Its utilization is crucial to avoid the accidental or involuntary tampering of the digital evidence. There are two types of write blocker:

- Physical write blocker: physically or electronically disable all the write attempts, enabling only reading operations.
- Virtual write blocker: are software that intercepts and blocks all the write commands directed to the analysed device. It's less reliable than a physical one, and must be correctly configured. Write blocker software are usually included in live forensic distros, such as CAINE [78], Tsurugi [79], and Paladin [80].

Rooting is technically "invasive" because it requires modifying the system partition, and sometimes the recovery partition, to gain administrative privileges. This raises concerns about violating the forensic principle of not altering the evidence.

The paper "Impact of Android Phone Rooting on User Data Integrity in Mobile Forensics" [81] investigates the impact of rooting Android phones on the integrity of user data and the search for any damage resulting from the rooting of the device and concludes that any effect is noticed on the user data during the process of rooting.

Furthermore, the paper "Digital Forensic Investigation of WhatsApp on Android" [82] shows that in a forensic analysis of WhatsApp, open-source tools like Autopsy and WinHex failed to retrieve critical evidence (such as account information, contact lists, and message logs) from a non-rooted device. However, once the device was rooted, the same tools successfully extracted accounts, messages, and call logs.

Another relevant aspect that needs to be considered to perform a physical acquisition on devices is the encryption mechanism, such as Full Disk Encryption (FDE) or File Based Encryption (FBE).

Full Disk Encryption (FDE) is the standard used up to Android 7 and uses a single encryption key to protect the entire data partition. If the device is powered on and unlocked, or if the encryption key can be retrieved from volatile memory, it is technically possible to perform a physical acquisition and obtain decrypted data for forensic analysis. In such scenarios, a low-level bit-by-bit copy of the storage may still retain significant evidentiary value.

Newer Android versions, starting from Android 7, introduce the File Based Encryption (FBE) standard, which became mandatory from Android 10 onward. Under this new standard each file is encrypted with its own key, which is further protected by hardware-backed security mechanisms, such as Trusted Execution Environment (TEE) and StrongBox. On an FBE-enabled device, each user of the device has two storage locations: Device Encrypted (DE), accessible as soon as the device powers on (before user unlock); Credential Encrypted (CE) storage available both during Direct Boot mode and after the user authentication. [83]

Additionally, metadata encryption protects filesystem metadata, ensuring that filenames and directory structures are secured along with the file contents.

To define the device states, the two terms AFU (After First Unlock) and BFU (Before First Unlock) are critical: BFU: the state in which Device Encrypted files are accessible and Credential Encrypted files remain locked.

AFU: the state in which both Device Encrypted and Credential Encrypted files can be accessed, assuming the device is unlocked or the unlock key is known.

When the device is in the BFU state, and the unlock code is unknown, tools capable of extracting Device Encrypted files, probably attempting to crack the unlock code, are needed. Without appropriate tools, no data can typically be extracted.

A different scenario emerges when the state is AFU and the unlock code is unknown. This case requires tools capable of AFU acquisition, accessing Credential Encrypted files without cracking the unlock code.

A final scenario is reached when the state can be AFU or BFU, and the unlock code is known.

With tools capable of bypassing Android restrictions, often temporarily achieving root access, data accessibility is possible.

Otherwise it is possible to perform a logical acquisition through the native protocol.

In accordance with the principle of order of volatility, the sequence of data acquisition should be guided by the volatility of the data. A notable example is provided by the study *Android Dumpsys Analysis to Indicate Driver Distraction* [84], which demonstrates that artifacts obtained through the `dumpsys wifi` and `dumpsys telecom` commands are volatile in nature and do not persist across device reboots.

In the table 3.3 are summarized which operating system and native app data is available for the different acquisition types.

Category	Description	Source	BFU	Logical	FFS
Device Information	Build Information	build.prop		ADB	
Device Information	Manufacturer	Various		ADB	
Device Information	Model Number	Various		ADB	
Device Information	Chipset	Various		ADB	
Device Information	Android ID	settings_secure.xml		ADB	
Device Information	Bluetooth Mac Address	settings_secure.xml		ADB	
Device Information	Bluetooth Device Name	settings_secure.xml		ADB	
Device Information	Carrier	Various		ADB	
Device Information	Serial Number	Various		ADB	
Device Information	SIM Card Information (IMSI, ICCID, Carrier)	telephony.db			
Device Information	IMEI(s)	Various		ADB	
Device Information	Wi-Fi Mac Address	Various		ADB	
OS Information	OS Version	Various		ADB	
OS Information	Security Patch	Various		ADB	
Device Settings	Time Zone	Various		ADB	
Device Settings	Reset Date/Time	bootstat folder		ADB	
Device Settings	Global/Secure/System Settings	settings_secure.xml, settings_global.xml, settings_system.xml		ADB	
Installed Applications	Installed Packages List	packages.list			
Installed Applications	Installed Packages Details	packages.xml			
Installed Applications	Application Runtime Permissions	runtime-permissions.xml			
Installed Applications	Application Roles	roles.xml			
Installed Applications	Google Play Installed Applications	localappstate.db			
Installed Applications	Google Play Searches	suggestions.db			
Installed Applications	Installed apps (historical)	gass.db			
Installed Applications	Applications updates	frosting.db			
User Accounts	Accounts (Device Encrypted)	accounts_de.db			
User Accounts	Accounts (Credential Encrypted)	accounts_ce.db			
User Accounts	Google Play User Library	library.db			
Application Usage	Usagestats	usagestats folder		Dumpsys	
Application Usage	Recent Tasks	recent_tasks			
Application Usage	Device Health Services (Application Usage)	app_usage_stats.xml			
Application Usage	Digital Wellbeing	Depends on manufacturer			
Application Usage	Network Stats	netstats folder			
Application Usage	Firebase Cloud Messaging	fcm_queued_messages.ldb			
Application Usage	Privacy Dashboard	appops folder			
Application Usage	Samsung Package Predictions	PkgPredictions.db			
Device Usage	Power Events	bootstat folder		Dumpsys	
Device Usage	Device Health Services (Battery Usage)	turbo.db			
Device Usage	App Battery Usage Events	battery-usage-db-vx			
Device Usage	SSRM Heating Logs	com.sec.android.sdhms package			
Emulated SD Card	Emulated Storage (SD Card)	media folder		MTP/ADB	
Emulated SD Card	Emulated Storage (Metadata)	external.db			
Connected Devices	ADB Hosts	adb_keys			
Connected Devices	Connected Bluetooth Devices	bt_config.conf		Dumpsys	
Connected Devices	Wi-Fi Connections/Profiles and PSK	WifiConfigStore.xml		Dumpsys	
Connected Devices	Wi-Fi Hotspot	WifiConfigStoreSoftAp.xml			
Connected Devices	Airtag Alerts/Scan	personalsafety_db			
Device Locations	Google Timeline Events	app_semanticlocation_rawsignal_db			
Device Locations	Samsung Weather Clock	WeatherClock			
Device Locations	Samsung CMH	cmh.db			
Native Applications	Android Calendar	calendar.db		Agent	
Native Applications	Android Call Log	calllog.db		Agent	
Native Applications	Android Contacts	contacts2.db		Agent	
Native Applications	Android Messages	bugle_db/mmsms.db		Agent	
Native Applications	Android Downloads	downloads.db			

Figure 3.3: Operating system and native app files. Source: [84].

As a consequence, even with root privileges, a traditional physical acquisition may result in large portions of encrypted and inaccessible data if the device is locked. In modern Android devices, forensic acquisition therefore shifts from classical physical extraction toward advanced logical or live acquisition techniques performed while the device is unlocked.

As a result, for modern Android devices, distinguishing between rooted and non-rooted acquisition is no longer sufficient to accurately assess the quantity and

quality of extractable data. As previously discussed, factors such as the File-Based Encryption (FBE) architecture and device state, specifically AFU (After First Unlock) or BFU (Before First Unlock), play a significant role in determining data accessibility, often outweighing the impact of root privileges themselves.

This evolution demonstrates how rooting alone is no longer sufficient to provide full data access, and how encryption architecture and device state play a crucial role in determining the feasibility and evidentiary value of a forensic acquisition.

In the context of static anti-spyware analysis, it is typically assumed that the device is acquired in a powered-on and unlocked state. Under these conditions, the forensic relevance of the AFU (After First Unlock) state becomes central, as it enables access to Credential Encrypted (CE) data that would otherwise remain inaccessible. Consequently, the amount of data available for analysis is not determined solely by the presence or absence of root privileges, but rather by the combined effect of File-Based Encryption (FBE), device state, and the timing of acquisition. In such scenarios, root access primarily enhances the depth of analysis rather than defining its feasibility.

Even with root privileges, a physical acquisition performed via `dd` on an FBE-enabled Android device yields predominantly encrypted data. Root access allows the inspection and export of data while the system is up and unlocked, in a context where the decryption mechanisms are already running in memory. Therefore, the benefit of root access is accessing areas of the file system not visible with standard user privileges, and doing so before the runtime state changes or is lost. The table 3.4 shows a comparison of resource access levels with and without root privileges. Green indicates full access, yellow indicates partial access, and red indicates no access.

Category	Description	Non-Root (ADB / Logical)	Root
/sdcard/ External storage	Medias, downloads, documents		
Media	photos / videos / audios		
Backup ADB	* depending on allowBackup	Limited *	
App databases			
App caches			
App configuration files			
/data/system/	user and security data		
/data/misc/	wifi, bluetooth and config data		
/system	system app and libraries	Limited	
/vendor	OEM		
/cache			
Logcat		Limited	
Dumpsys	runtime states	Limited	
Active processes / services			
Access to /data/data/			

Figure 3.4: Comparison of resource access levels with and without root privileges.

Lastly, it is important to notice that some spyware, especially advanced ones, may implement anti-root techniques that detect elevated privileges on the device. Upon detecting root access, these malware may disable their functionality, hide themselves, or delete sensitive data. Such techniques are noticed in the *LANDFALL*'s analysis performed by Unit 42 researchers [85] and includes the detection of root management applications such as Magisk or SuperSU, checking the presence or the alteration of directories typically modified during root process such as */system/bin/su* binary, or via API call to detect the system status.

Similar checks are noticed in commercial spyware applications such as mSpy and FlexiSPY to modify their behaviours based on root privileges presence [86].

Databases analysis

Finally, an important source of evidence are the databases. Parsing records from numerous system and apps databases can reveal traces of compromise or attempt of the spyware to hide its traces. Both Android and iOS rely on SQLite databases. For example, iOS maintains a detailed record of processes execution and processes network utilization in two SQLite files named *DataUsage.sqlite* and *netusage.sqlite*. The analysis of these two databases represented a key point in the individuation of NSO's Pegasus artifacts during Amnesty International forensic analysis. Their analysis also confirmed that Pegasus manipulates system databases to hide its tracks. However, analysts were able to identify the infection precisely because of

the inconsistencies between different database tables left by the spyware.

3.1.4 Tools For Static Analysis

AndroidQF

One essential tool for Android device investigation is ADB. Activating the USB debugging on the mobile, is possible to perform a variety of actions on the device such as retrieving information about the system, the application installed and performing backups.

AndroidQF (Android Quick Forensics) is a tool written in Go and maintained by Amnesty International's Security Lab. [87]

This tool exploits the adb interface and it's designed to simplify the acquisition of key forensic information from Android devices. The tool also provides information about the acquisition that can be used to guarantee the chain of custody of forensic evidence.

AndroidQF, differently from MVT, does not perform the analysis of extracted files, but is a tool that aims to simplify the acquisition through the ADB. As reported on the GitHub page of the project, the tool can extract the following data:

- A full backup or backup of SMS and MMS messages.
- The output of “adb shell getprop”, providing build information and configuration parameters.
- All system settings
- The output of the ps shell command, providing a list of all running processes.
- The list of system's services.
- A copy of all the logs from the system.
- The output of the dumphsys shell command, providing diagnostic information about the device.
- A list of all packages installed and related distribution files.
- Copy of all installed APKs or of only those not marked as system apps.
- A list of files on the system.
- A copy of the files available in temp folders.
- A bug report containing system and app-specific logs, with no private data included.

Mobile Security Framework (MobSF)

A further step in the static analysis can be reached with the Mobile Security Framework (MobSF) tool [88]. MobSF is an open-source security research platform for static and dynamic analysis of mobile applications used for penetration testing, privacy analysis and malware analysis. This tool automatically performs a static analysis on mobile applications, examining the source code. The analysis starts by unpacking the APK (Android) or IPA (iOS) and checking the manifest of the application. Then MobSF checks the code for vulnerabilities, detects harmful code and obfuscation techniques, performs behaviour analysis, and extracts emails and URLs from source code.

In the case of Android applications the tool decompiles the APK to Java with JADX. MobSF also can also perform the dynamic analysis by executing the application in a sandbox environment to track the behaviour of the application in runtime. Among the principal features there are: the monitoring of the network traffic, track the app interactions, and monitoring of system calls.

Furthermore, the tool offers APIs to automate scans and can be integrated in a script or a pipeline of other applications.

Autopsy

Autopsy is an end-to-end open source digital forensics platform created by Sleuth Kit Labs.

The tool supports both logical and physical acquisition. After the creation of a new case, a new data source is added. Once the file is selected, we are asked what type of analysis to perform. Autopsy performs some automated scans to detect possible interesting data.

Once the image is opened, it is possible to view information about the partitions, documents found within the image and explore the filesystem.

Commercial Forensic Tools

Commercial forensic platforms such as Cellebrite UFED, Oxygen Forensic Detective, and Magnet AXIOM are industry leaders and offer advanced capabilities for data extraction to support in-depth investigations. These tools are capable to perform multi-level acquisition either logical and physical on supported devices. They parse and reconstruct volatile and persistent system data, including app data, system logs, configuration files, messaging histories, and network metadata. Some of them offer timeline reconstruction feature, which could be a support in reconstructing spyware infection. Additionally, these platforms correlate extracted artifacts from

both local and cloud sources, allowing for the detection of anomalous account activities, suspicious sessions, unauthorized privilege escalation, and persistence mechanisms that indicates sophisticated spyware infections.

3.2 State-of-the-Art Spyware Detection Tools

3.2.1 SpyGuard

SpyGuard [89] is an improved version of TinyCheck created by the same developer. This tool is designed to easily capture network communications from any device that can be associated with a custom-created Wi-Fi access point, like smartphones, laptops, IoT devices, and workstations. It can be installed on a separate device, also a simple single-board computer like the Raspberry Pi. Like TinyCheck, SpyGuard is not restricted to any device and works on a wide range of operating systems, such as Android, IOS and Windows.

SpyGuard scans devices connected to the generated Wi-Fi network to detect network traffic for Indicators of Compromise (IOCs).

This tool is based on its own detection engine with specific detection rules. The detection rules include the use of HTTP/HTTPS protocols over non-standard ports, communication with TOR nodes, connections to domains registered within the past year, and network traffic directed toward IP ranges associated with malicious activity.

The advantage of this approach, when used with a well-formed set of rules and a complete set of IOC, is the easy detection of stalkerware applications or known spyware without the need to install technical apps or perform forensic analysis on the victim's device. In addition, scanning a device with these tools does not alert the abuser of that scan.

SpyGuard's analysis engine leverages several layers of detection techniques, including:

- Heuristic detection to identify suspicious behaviour patterns.
- Indicators of Compromise (IOCs) to match known threats.
- Whitelist to prevent false positives.
- Active analysis such as DNS lookups (dig), domain registration checks (whois), and SSL certificate inspection (openssl).

SpyGuard network analysis core is powered by Suricata [90], an advanced open-source Network IDS, IPS and Network Security Monitoring engine. Suricata is used for real-time deep packet inspection (DPI) and signature-based detection applying

a rule set.

To start using SpyGuard is very simple: is sufficient to clone the repository and install the script with the following commands: `git clone https://github.com/SpyGuard /SpyGuard && cd SpyGuard && sudo bash install.sh` After the installation two pages are accessible:

- <https://localhost:8443/> for the backend and
- <http://localhost:8080/> for the frontend.

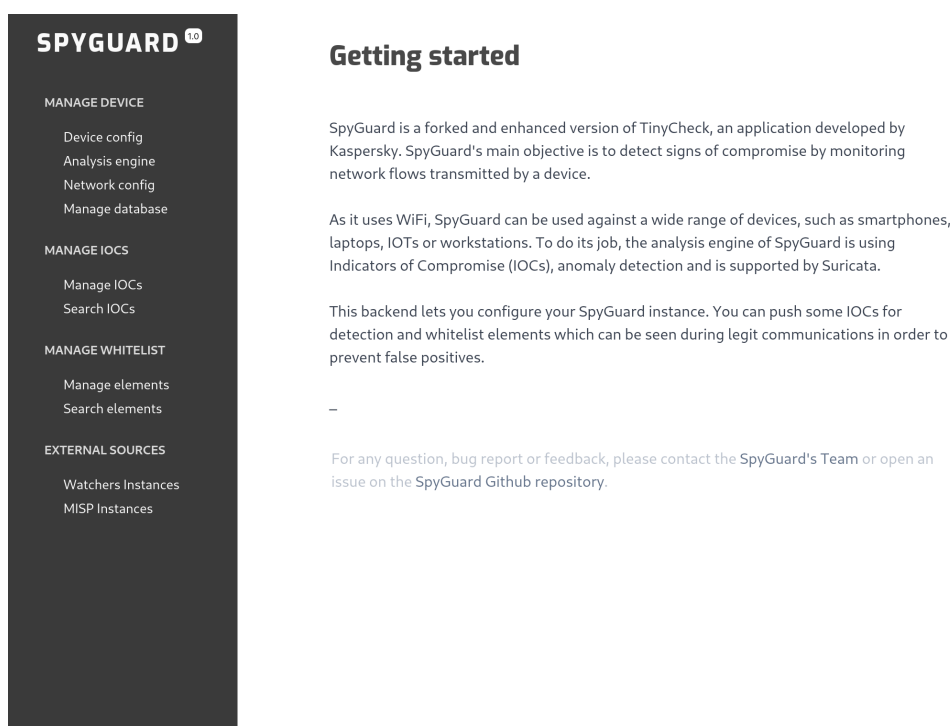


Figure 3.5: Spyguard backend page

On the backend page, two different network interfaces need to be specified in the “Network Config” section. The first one, the Wireless AP interface, is used by the system to create the ephemeral Wi-Fi access point. The second one, Internet link interface, is the interface connected to the Internet. The ephemeral access point is created with the code located at “/usr/share/spyguard/server/frontend/app/classes/network.py”.

On the backend is also possible to edit different settings to personalise the tool, such as enabling and disabling detection methods (heuristic, IOCs, whitelist and active analyst).

Furthermore, it is possible to personalise IOCs, importing new databases, manage

whitelist and external sources, adding watchers and MISP¹ instances.

After configuration, the analysis can be started from the frontend page by creating the ephemeral access point.

To obtain a valuable report, SpyGuard needs to be kept running for about 15 minutes in order to collect sufficient traffic.

By default, SpyGuard has three entries in the `watchers.yaml` file: “SpyGuard IOCs repository”; “ECHAP stalkerware repository”, maintained by ECHAP non-profit organization against surveillance; “SpyGuard whitelist”, a default whitelist that includes the common known services, such as `google.com`, `iCloud.com`, `apple.com`. Many other known and legitimate network infrastructures are represented in the whitelist using CIDR notation, allowing SpyGuard to efficiently identify entire IP ranges that are considered safe and avoid false positives. In the whitelist, resources are represented with four different notations: `ip4addr`, `ip6addr`, `cidr` and `domain`. As an example here are reported two entries of the whitelist file:

```
"type": "domain", "element": "google.com"
```

```
"type": "cidr", "element": "239.0.0.0/8"
```

This json is an example of IOC:

```
1      {
2          "type": "ns",
3          "tag": "suspicious",
4          "tlp": "white",
5          "value": "topdns.com"
6      }
```

The IOCs entries are composed by four values:

- “type”: describes the type of the system. According to `definition.py`, this value can assume ten different values: `ip4addr`, `ip6addr`, `cidr`, `domain`, `shalcert`, `jarm`, `snort`, `ns`, `freedns` and `tld`.
- “tag” is a semantic label that describes the information known about the target. In the IOC database eight tags are defined: `apt`, `stalkerware`, `suspicious`, `malicious`, `spyware`, `cybercrime`, `doh` (DNS over HTTPs) and `dual`.
- “tlp” represents the Traffic Light Protocol² and can assume four values: `red`,

¹The MISP is an open-source software solution for collecting, storing, distributing and sharing cyber security indicators and threats about cyber security incident analysis and malware analysis [91].

²The Traffic Light Protocol (TLP) was created to facilitate greater sharing of potentially sensitive information and more effective collaboration [92].

amber, green, clear (or white).

- “value”: the domain or the IP address of the resource. In many cases, multiple IPs are represented with the CIDR notation.

After the ephemeral network is generated, SpyGuard starts to capture packets. The class Capture, at “server/frontend/app/classes/capture.py”, is responsible for packet capture flow. It integrates the Wireshark utility (dumpcap) and Suricata.

```
1 sp.Popen(["dumpcap", "-n", "-i", self.iface, "-w", self.pcap])
2 sp.Popen(["suricata", "-c", "/etc/suricata/suricata.yaml", "-i",
           self.iface, "-l", self.assets_dir, "-S", self.rules_file])
```

- dumpcap: a Wireshark utility to capture raw packets and save them in a .pcap file.
- suricata: launched in parallel on the same interface. It is run in real time with the original configuration of Suricata and applies custom rules derived from IOCs.

In this class, some IOCs are translated into Snort rules that Suricata can use as IDS rules. When the Capture phase ends, the analysis phase begins. At the end of the capture phase, analysis.py is launched with the capture token as a parameter. This block of code collects and unifies all the logs produced in a readable report.

The internal engine of SpyGuard is defined in the file “spyguard/analysis/classes/engine.py”. This code contains methods for enrichment and heuristic analysis that support IOCs analysis and cover the cases in which the domain is unknown. After the Suricata analysis the output file eve.json is parsed and various checks are performed.

The enrichment and heuristics phase follows these steps:

1. WHITELIST: `check_whitelist()` method verifies even if the IP or the associated domain is in the whitelist. If so, the "whitelisted" key of the record is set to True.
2. DNS: a query is executed to obtain the nameserver of the domain. It checks if the domain is in the blacklist and if it matches the domain is reported as suspicious.
3. WHOIS: the whois library is used to verify if the domain's registration date is less than one year. If true, then the domain is reported as suspicious.

```
1   whois_record = whois.whois(domain)
2   creation_date = whois_record.creation_date if type(
whois_record.creation_date) is not list else whois_record.
creation_date[0]
3   creation_days = abs((datetime.now() - creation_date).days)
4   if creation_days < 365:
5       suspicious = True
6
7
```

4. FLOW: checks a network flow against a set of IOCs and heuristics. It ensures that UDP or ICMP protocol isn't going outside of the local network, that HTTP protocol uses the default port and that the flow is not using a port greater than 1024. It also checks if the communication goes to a Tor node included in the list retrieved from "https://www.dan.me.uk/torlist/".
5. TLS certificate: `active_check_ssl()` verifies that TLS uses the default port, checks for the use of free or self-signed certificates and ensures that if the certificate has an SNI, the domain is validated.

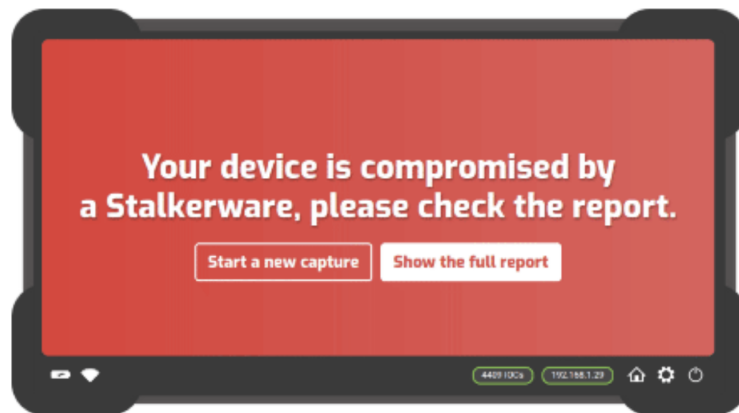


Figure 3.6: Spyguard alert screen. Source: [89]

The last step of the SpyGuard script is the generation of various reports in JSON format. A report is produced on the frontend when capture and analysis phases are completed. The different JSON files are saved along with the raw .pcap file and Suricata reports. The *Alarm.json* file includes all the alarms classified by score.

SPYGUARD PIPELINE

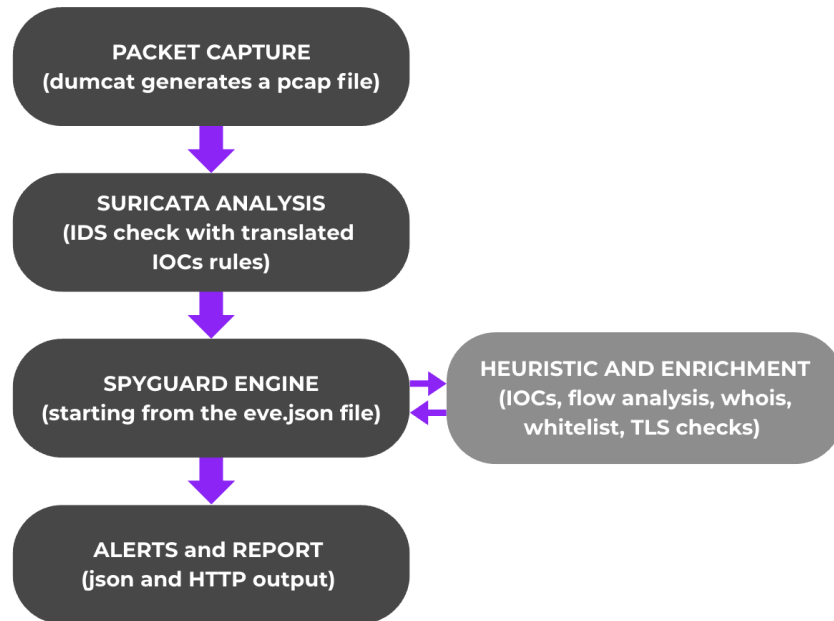


Figure 3.7: Spyguard pipeline

3.2.2 Interceptor

Interceptor [93] is a remote traffic flow analysis service. The system allows users to determine whether a device has been intercepted by spyware without physical access. This eliminates the need for users to hand over devices to third parties, optimizing privacy and control.

The service is provided on a pay-per-use basis and employs the WireGuard application to establish a secure connection through a VPN tunnel. The platform will guide the user during all the steps needed for the connection. Once connected, Interceptor performs a scan of the outgoing data traffic from the device. The scan typically takes ten minutes, and the user should use their device normally during this time.

The remote analysis system tracks outgoing connections and communications, examining their destination and source. The platform verifies the protocol, domain, destination IP address, and port in use. The software also scans for indications of recognized spyware activity (IOCs). Along with traffic analysis through Zeek [94] and Suricata monitoring, Interceptor involves a proprietary database developed from the in-house experience of the company.

So, SpyGuard, which is the primary analysis engine of the platform, is paired with

the proprietary database and Zeek. Unlike Suricata, which is an active defense system, Zeek operates passively on a sensor, monitoring network traffic in real time, providing a deep insight into network behaviour.

3.2.3 M2 Bridge New



Figure 3.8: Representative photo of M2 Bridge New. Source: [95].

The M2 Bridge New [96], developed by MgExtreme and made in Italy, operates through passive sniffing and man-in-the-middle techniques. Although there are no official online materials explaining its workings, available information indicates that it uses an analytical engine similar to Sprayguard. The application is capable of producing a detailed forensic report and a .pcap capture file, probably using dumpcap Wireshark utility.

One of the greatest advantages of the M2 Bridge New is its compact and portable design: the system contains preinstalled network cards, a mini-computer and a

monitor. With this device investigators are able to have a complete analysis within minutes in any environment or situation. The system can be run on the standard SIM card without requiring a connected Wi-Fi network, or it can be connected directly via an RJ45 cable.

The forensic integrity of the generated PDF report is ensured through a BLAKE2s hash string. This ensures that the documents are probative, and they are made admissible and valid in legal proceedings.

The manufacturer [97] recommends that during the analysis phase the operator must make the mobile phone user perform the following steps, each for approximately five minutes, without launching any browser applications:

- Place a conventional voice call.
- Use messaging apps (e.g., WhatsApp, Messenger, Telegram) by sending a message, photo, and placing a voice call.
- Send an SMS.
- Take a photo with the phone.
- Power off and then reboot the phone.

These activities are meant to simulate real-world usage scenarios so that the system can capture and analyze communication information in a controlled manner.

In the final report are also included links to these resources: Whois Domain Tools, Domain link, IP Tracker Online and SECURI Security Analysis.

3.2.4 Mobile Verification Toolkit (MVT)

One useful tool for static analysis is Mobile Verification Toolkit (MVT) [98] that intends to facilitate forensic analysis by searching traces of compromise. This tool has been created and it's maintained by the Amnesty International Security Lab in July 2021 during the forensic investigation for Pegasus.

It has a set of utilities, different for IOS and Android, and performs a scan over the files to find correspondence with a set of IOCs. Among the utilities that MVT offers are the description of IOS backups, the parsing of IOS system and app databases and logs, the extraction of installed applications and information from Android devices, the check over ADB or over an Android backup, the identification of traces of compromise, and generation of reports and a timeline of all malicious activities found.

The IOCs include indicators for the principal known mercenary spyware: KingsPawn, Novispy, Pegasus, Wyrmspy, DragonEgg, Wintego's Helios and

Predator. They also include IOCs related rcs lab, Android hacking campaign linked to mercenary spyware, operation triangulation against IOS, and a list of indicators for stalkerware application.

```

1  "ioc_files": [
2  "/home/any/.local/share/mvt/indicators/raw.
githubusercontent.com_mvt-project_mvt-indicators_main_2023
-04-11_quadream_kingspawn.stix2",
3  "/home/any/.local/share/mvt/indicators/raw.
githubusercontent.com_AmnestyTech_investigations_master_2024
-12-16_serbia_novispy_novispy.stix2",
4  "/home/any/.local/share/mvt/indicators/raw.
githubusercontent.com_AmnestyTech_investigations_master_2021
-07-18_nso_pegasus.stix2",
5  "/home/any/.local/share/mvt/indicators/raw.githu
busercontent.com_mvt-project_mvt-indicators_main_2023-07-25
_wyrmspy_dragonegg_wyrmspy_dragonegg.stix2",
6  "/home/any/.local/share/mvt/indicators/raw.
githubusercontent.com_AmnestyTech_investigations_master_2024
-05-02_wintego_helios_wintego_helios.stix2",
7  "/home/any/.local/share/mvt/indicators/raw.
githubusercontent.com_AmnestyTech_investigations_master_2023
-03-29_Android_campaign_malware.stix2",
8  "/home/any/.local/share/mvt/indicators/raw.
githubusercontent.com_mvt-project_mvt-indicators_main_2022
-06-23_rcs_lab_rcs.stix2",
9  "/home/any/.local/share/mvt/indicators/raw.
githubusercontent.com_mvt-project_mvt-indicators_main_2023-06
_01_operation_triangulation_operation_triangulation.stix2",
10 "/home/any/.local/share/mvt/indicators/raw.
githubusercontent.com_mvt-project_mvt-
indicators_main_intellexa_predator_predator.stix2",
11 "/home/any/.local/share/mvt/indicators/raw.
githubusercontent.com_AssoEchap_stalkerware-
indicators_master_generated_stalkerware.stix2"
12 ],
13

```

Among with the report MVT produces different files:

- *dumpsys.txt*: which provides information about system services and then is structured in single json files divided in battery, packages, receivers, and activities.[99]
- *logcat.txt*: logcat is a tool that dumps a log of system messages. [100]

- *command.log*: includes the sequence of all the commands executed by MTV during the analysis
- *file.json*: includes the list of all the file retrieved on the device
- *getprop.json*: is the result of getprop command and reports all the properties of the system
- *packages.json*: is the list of all the APK packages found on the device
- *setting.jons*: the list of all the setting configured on the device
- *timeline.csv*: reports the timeline of all the system events. Each event is described by four columns: Device Local Timestamp, Plugin, Event and Description.

Another operation offered by MVT is to download all available installed packages from the device with the command *mvt-Android download-apks*. This function allows the further analysis and reverse engineering on the APK. The code that retrieves all the packages is in the file `mvt/src/mvt/Android/modules/adb/packages.py`, and use the `adb pm list packages packages = self._adb_command("pm list packages u i f")` after that in the `src/mvt/Android/cmd_download_apks.py` `adb pull` is used to copy the selected packages. `adb pm list packages` [101] can be used with different flags, among them:

- f: report the associated APK file
- i: report the installer for the packages
- U: show the package UID
- u: include uninstalled packages

3.2.5 IShutdown utility

KasperskyLab cybersecurity researchers developed a script called iShutdown [102]. This method extracts, analyses, and parses Shutdown.log forensic artifacts from iOS Sysdiagnose archives. It can be used to perform an initial recognition for reliably identifying signs of spyware on Apple iOS devices.

The Shutdown.log log file is stored within the sysdiagnose archive of iOS devices. It records every reboot event with its environmental characteristics and is a generally overlooked forensic artefact.

When a user initiates a reboot, the operating system terminates running processes, flushing memory buffers, and waiting for a normal reboot. The script analyses anomalies during reboot procedures to find potential infections with high accuracy.

With the iOS 26 update, Apple changed the way with which the system handles the shutdown.log file [103]. This update poses a new challenge for the forensic investigation, reducing the effectiveness of analysing this file. With the iOS 26 update, the system overwrites the shutdown.log file at startup.

However, with the introduction of iOS 26.2, Apple has made yet another modification in the management of this artefact [104]. With the introduction of iOS 26.2, rather than overwriting the already existing file, the operating system now generates a new log file for each reboot.

Although this change restores the ability to track consecutive reboots moving forward, it does not recover past data; the transition to the iOS 26 architecture effectively wiped all pre-existing forensic traces stored in the log format. Consequently, while the iShutdown methodology remains viable for new events, there is a permanent "blind spot" in the device's history corresponding to the period immediately following the first iOS 26 installation.

Chapter 4

Experimental Evaluation

In the previous chapters the basis for detecting mobile spyware and the theoretical framework are discussed.

In this chapter an empirical research is conducted to evaluate and discuss the efficiency of spyware identification approaches. The next sections cover different tools such as Spyguard, MVT (Mobile verification Toolkit), MobFS, and ADB for log analysis.

The main goal of these tests is to evaluate these tools and their effectiveness against contemporary obfuscation methods. Moreover, attention is put on the correctness and capability of spyware identification of tools that rely on IOCs.

Each section details experimental setup and the results obtained.

4.1 Spyguard tests

This first set of tests are performed on Spyguard application, and aim to evaluate its functionality and its capability to identify the presence of surveillance software on the device through the network traffic analysis.

The tests are performed both on Android and iOS to verify that network analysis does not have differences and can provide the same results on both the system.

The tests are conducted on two different parental control applications: Mobile Tracker Free for Android, and Qustodio for iOS.

On Android another test is performed by using Frida to bypass the SSL pinning and read the encrypted traffic sent to the C2 of the application.

Test n.1 - Mobile Tracker Free on Android

The analyses have been conducted on the following hardware:

- A Kali Linux emulated machine on which has been installed SpyGuard
- An Android device, running Android 12, emulated with Android Studio.
- An Alfa Network adapter (Model: AWUS036NH).

Intercepting the network traffic of a physical device using the Spyguard script. Spyguard successfully identified the installed spyware application and generated a comprehensive report. The output from Spyguard includes various files, in particular a PCAP file containing all intercepted network traffic.

The network traffic clearly reveals the DNS requests to "mobile-tracker-data.com".

Table 4.1: DNS requests to "mobile-tracker-data.com"

No.	Time	Source	Destination	Protocol	Len
11072	2025-03-28 11:55:00.641179303	10.42.0.227	10.42.0.1	DNS	83
Info: Standard query 0xb819 A mobile-tracker-data.com					
11073	2025-03-28 11:55:00.646006557	10.42.0.1	10.42.0.227	DNS	99
Info: Standard query response 0xb819 A mobile-tracker-data.com A 51.158.154.183					

The PCAP file also contains a request to a secondary Mobile Tracker Free server dedicated to the file explorer

Table 4.2: DNS requests to "fileexplorer.mobile-tracker-free.com"

No.	Time	Source	Destination	Protocol	Len
10926	2025-03-28 11:54:48.026054915	10.42.0.227	10.42.0.1	DNS	96
Info: Standard query 0xa733 A fileexplorer.mobile-tracker-free.com					
10927	2025-03-28 11:54:48.064118036	10.42.0.1	10.42.0.227	DNS	99
Info: Standard query response 0xa733 A fileexplorer.mobile-tracker-free.com A 104.22.47.254 A 172.67.7.104 A 104.22.46.254					

The PCAP file also shows the TLS handshake for establishing the encrypted channel. The Server Name Indication (SNI) is visible in clear text, revealing `SNI=fileexplorer.mobile-tracker-free.com` and `SNI=mobile-tracker-free.com`. Once the TLS handshake is completed, all traffic to and from the server is fully encrypted and, therefore, unreadable.

It is possible to confirm that the traffic is directed to and originates from the Mobile Tracker Free server by examining the IP addresses.

During my tests, I identified four distinct IP addresses: 51.158.154.183 for the base application, and 104.22.47.254, 172.67.7.104, and 104.22.46.254 for the file explorer.

Test n.2 - Mobile Tracker Free on Android with SSL pinning bypass

The analyses have been conducted on the following hardware:

- MacOS system on which has been installed Burp Suite, ADB, frida and Android Studio
- A Kali Linux emulated machine on which has been installed SpyGuard
- An Android device, running Android 12, emulated with Android Studio.
- An Alfa Network adapter (Model: AWUS036NH).

Due to the TLS encryption it is not possible to read the payload of the packets. Despite the security provided by SSL/TLS, is still possible to attacks encrypted traffic, particularly through man-in-the-middle (MITM) attacks. In a MITM attack, an attacker intercepts communication between a web client and a web server, allowing them to eavesdrop on the conversation or even modify the data being exchanged.

SSL pinning is a technique that helps to prevent MITM attacks by hardcoding the SSL/TLS certificate's public key into the app or device. This means that when the app or device communicates with the server, it will compare the server's SSL/TLS certificate's public key with the one that is hardcoded into the app or device. If once a host is identified or is in view with a certificate or public key, then the certificate or public key gets associated or 'pinned' to the host. This will make the application trust only the valid or pre-defined certificates or public Keys. If SSL Pinning is not used, application trusts custom certificate and proxy tools are able to intercept the traffic. This is achieved in 3 ways: Certificate Pinning, Public Key Pinning and Hash Pinning [105].

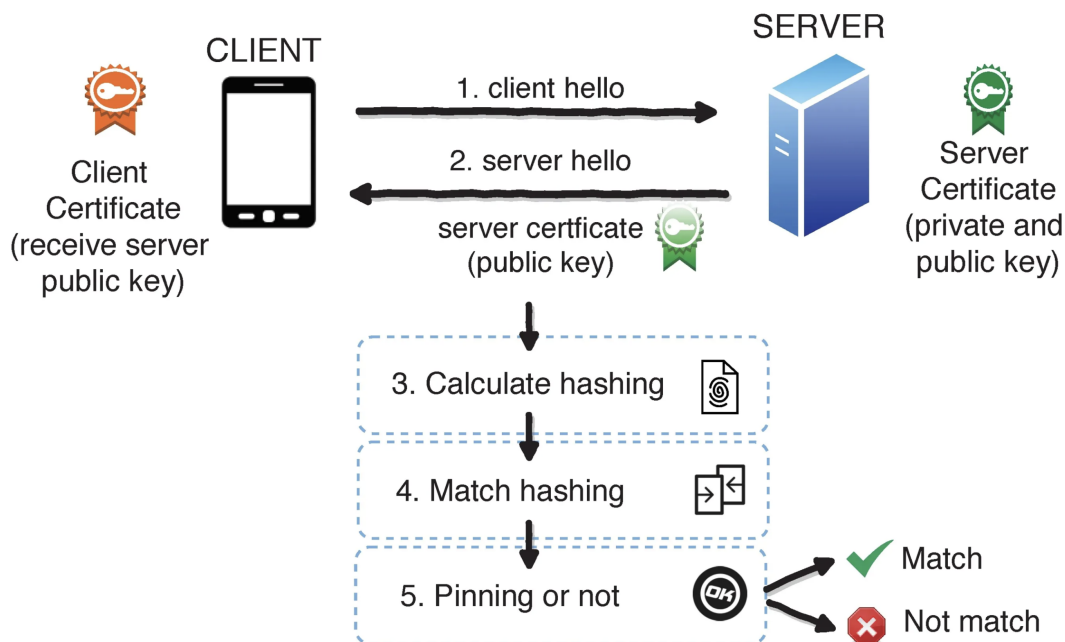


Figure 4.1: SSL Pinning. Source [105].

On Windows, it is possible to use the `SSLKEYLOGFILE` environment variable to log SSL session keys.

In other operating systems, such as Android, logging the SSL keys used is not feasible.

Additionally, the application likely employs SSL pinning to prevent traffic decryption, so a simple app as PCAPdroid with the packet “PCAPdroid MITM” is insufficient to read the traffic in cleartext.

Deciphering TLS traffic between the application and the server with high probability requires a rooted device.

For the subsequent tests, I used an emulated device running Android 12 through Android Studio.

To obtain root access on the device, I utilized a script [106] that enables rooting Android Studio Virtual Device (AVD) with Magisk.

The next steps required Frida, a toolkit for developers, reverse-engineers, and security researchers.

From the GitHub repository [107] it is possible to download the appropriate Frida Server, for this test `frida-server-16.7.3-android-arm64.xz`. To communicate with the device ADB is needed. Android Debug Bridge (ADB) [37] is a versatile command-line tool that lets you communicate with a device. The `adb` command facilitates a variety of device actions, such as installing and debugging apps. `adb` provides access to a Unix shell that you can use to run a variety of commands on a

device.

To get Frida working, it is also necessary to install three python modules on the machine.

This can be done with the command `pip3 install frida frida-tools objection`.

First, the previously downloaded Frida Server must be pushed to the device using the adb command: `adb push [frida_server_file_name] /data/local/tmp`.

Next, navigate to the `/data/local/tmp` directory, give to frida the permission for execution with `chmod +x [frida_server_name]` and then execute the server. In this step it's probably required to run the server in root mode to ensure the proper functionality.

The next step is to configure the proxy on android. In the Settings menu, set the computer IP as proxy server and then add the listeners in the proxy setting in Burp Suite. If everything is configured correctly, navigating to `http://burp` on the mobile device is possible to download and install the CA certificate.

At this point, the interceptor is configured, but it's still necessary to bypass the SSL pinning of the application.

Frida requires the unpinning script provided online [108]. Copy it and save it in a `script.js` file.

Now, the process of the app needs to be identified to perform the ssl pinning bypass. Initially, the app may not appear in a standard scan due to process-hiding techniques. A more in-depth scan can be performed using the command `adb -s emulator-5554 shell ps -A | grep -i protect`.

This command shows all the processes active on the mobile.

In the case of mobile spyware, a thorough analysis must be performed to find every potential process that is not Android native and should not be running on the device.

In the case of Mobile Tracker Free the hidden processes are:

u0_a149	7557	21723	15032416	67376	0	0	S	com.protect2025:fileexplorer
u0_a149	9459	21723	15381844	84496	0	0	S	com.protect2025

So now is possible to run the js script on these two processes with the command `frida -U -p 9459 -l fridascript.js` and `frida -U -p 7557 -l fridascript.js`.

At this point, it becomes possible to observe the traffic in clear text.

When a user logs into the dashboard, certain information is transmitted from the mobile device via a POST request:

```
1 tokenSecurity=zQNrPpyZ6GN15k95CUmtcyVDv0TRrjDaS -
  DLesMyNQzINvxD12WCy731N9Ck5XSBhJmkTPVViKKo0Roe1WYFe2NH2e -5
  CHyewhMC&data=-1&idPhone=7674028&type=playProtect
```

Listing 4.1: Connection to the dashboard

Battery request: in this case data=100.0 indicates that the battery is at 100

```
1 tokenSecurity=zQNrPpyZ6GN15k95CUmtcyVDv0TRrjDaS -
  DLesMyNQzINvxD12WCy731N9Ck5XSBhJmkTPVViKKo0Roe1WYFe2NH2e -5
  CHyewhMC&data=100.0&dateConnexion=2025%2F03%2F29%2021%3A05%3A16
  &idPhone=7674028&type=batterie
```

Listing 4.2: Request for battery status

The response that MTF gives when the filesystem is loaded in the dashboard:

```
1 [{"new response", "MixX8wHSBnESFE8wa3Lw", "en", [{"id": "409229424063
  a30b8b6d19d7bc092db4", "parentFile": "\/storage\/emulated\/0", "
  name": "Alarms", "type": "FOLDER"}, {"id": "95286
  c6edb7d393c59082a507b7ca78b", "parentFile": "\/storage\/emulated
  \/0", "name": "Android", "type": "FOLDER"}, {"id": "828142
  e090f45f324e391b3c4ca7c50d", "parentFile": "\/storage\/emulated
  \/0", "name": "Audiobooks", "type": "FOLDER"}, {"id": "0
  ee32c4cc70264cb3fb9e64eee4bbf54", "parentFile": "\/storage\/
  emulated\/0", "name": "DCIM", "type": "FOLDER"}, {"id": "5
  a1eff6a3560e75b807fd48626363c5d", "parentFile": "\/storage\/
  emulated\/0", "name": "Documents", "type": "FOLDER"}, {"id": "
  884778479c9a0a6715034ca628ee5b2d", "parentFile": "\/storage\/
  emulated\/0", "name": "Download", "type": "FOLDER"}, {"id": "2
  b94444bda238a7027a260fc36c54134", "parentFile": "\/storage\/
  emulated\/0", "name": "Movies", "type": "FOLDER"}, {"id": "
  b7559c498bfca2d0391b3ce1f380c881", "parentFile": "\/storage\/
  emulated\/0", "name": "Music", "type": "FOLDER"}, {"id": "7
  bb1c3f6d9bff131c4acddd93f2a13ab", "parentFile": "\/storage\/
  emulated\/0", "name": "Notifications", "type": "FOLDER"}, {"id": "5
  f4e0af4a4bb88c451f7d33136b967b9", "parentFile": "\/storage\/
  emulated\/0", "name": "Pictures", "type": "FOLDER"}, {"id": "16
  f7639cfef326abcd5be885468b1e98", "parentFile": "\/storage\/
  emulated\/0", "name": "Podcasts", "type": "FOLDER"}, {"id": "0123
  a4e8a211bb67704595db21ef2fb4", "parentFile": "\/storage\/emulated
  \/0", "name": "Recordings", "type": "FOLDER"}],
```

```
2 [{"id": "7abbec16eb8212cf55bbe89bd5165b", "parentFile": "\/storage
  \/emulated\/0", "name": "Ringtones", "type": "FOLDER"}]]
```

Listing 4.3: Request for battery status

When a live photo is requested from the dashboard, the file is returned in JPEG format (among others).

JPEG file format can be identified by their magic number, which is FF D8 FF at offset 0. When this data is encoded in Base64, you'll always get /9j/.

The data should be URL-decoded to reconstruct the original file (only first lines are reported).

```
1 /9j/4AAQSkZJRgABAQAAQABAAD/4
  gIoSUNDX1BSTOZJTEUAAQEAAAIIYAAAAAIIQAABtbnRyUkdCIFhZWiAAAAAAAAAA
2 AAAAAABhY3NwAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAQAA9tYAAQAAADTLQ
3 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
4 AA1kZXNjAAAA8AAAAHRyWFlaAAABZAAAABRnWFlaAAABeAAAAABRiWFlaAAABjAAAAB
5 RyVFJDAAABoAAAAChnVFJDAAABoAAAAChiVFJDAAABoAAAACh3dHB0AAAAByAAAABRj
6 cHJOAAAB3AAAADxtbHVjAAAAAAAAAAEAAAAMZW5VUwAAAFgAAAAcAHMAUgBHAElIAAA
7 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
8 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAFhZWiAA
```

Listing 4.4: Live photo request

Test n.3 - Qustodio on iOS

The analysis has been conducted on the following hardware:

- A Kali Linux emulated machine on which has been installed SpyGuard
- iPhone 12 running IOS 26
- An Alfa Network adapter (Model: AWUS036NH).

Qustodio is a monitoring solution sold as a parental control app to control children. [109]

As other parental control apps, it can be used as a stalkerware. Differently from other Android solutions, this app does not hide itself and does not obfuscate its presence, in fact the icon is visible and when a site is blocked or usage time is up the name of the app is visible. To protect itself from uninstallation, the application locks it with a password.

The app enables the controller to access the real-time position of the mobile, and through a VPN it logs browser history and app usage. Furthermore, web filters and routines can be configured, and applications can be blocked or uninstalled from the dashboard.

This app is not detected by Spyguard because it is not present in the IOCs lists, also Virus Total considers the domains of the app as legit.

Apparently Qustodio is considered a harmless application for parental control, but as seen with other parental control applications, it performs a continuous monitoring on the intercepted device and can be abused in cases of digital violence. This test highlights the IOC-based nature of Spyguard, which is severely limited in identifying known surveillance software.

Proof of the communications found in the capture.pcap file:

```
1      454 10576.059179528 10.42.0.75 10.42.0.1 DNS 84 Standard
      query 0x9367 AAAA mdm-service.qustodio.com
2 455 10576.062739150 10.42.0.1 10.42.0.75 DNS 132 Standard
      query response 0x567b A mdm-service.qustodio.com A 3.93.105.88
      A 44.207.150.238 A 34.195.231.27
3 464 10576.231316565 10.42.0.75 3.93.105.88 TLSv1.3 5 571 Client
      Hello (SNI=mdm-service.qustodio.com)
4 1383 10726.774782097 10.42.0.75 18.208.104.148 TLSv1.3 14 119
      Client Hello (SNI=api.qustodio.com)
```

Connections performed during test:

ID	src_ip	dst_ip	server_name	conn_count
33	10.42.0.75	44.207.150.238	mdm-service.qustodio.com	6
30	10.42.0.75	34.195.231.27	mdm-service.qustodio.com	6
28	10.42.0.75	3.93.105.88	mdm-service.qustodio.com	6
22	10.42.0.75	18.208.104.148	api.qustodio.com	5
23	10.42.0.75	18.66.195.85	static.qustodio.com	3

Table 4.3: Connection performed to Qustodio APIs

4.2 Spyguard limitations and potential enhancements

Like other existing tools, Spyguard's detection capability mostly depends on the presence of the spyware in the list of Indicators of Compromise (IOCs). This signature-based approach creates a reactive defence posture: the system relies on databases of known spyware, limiting the detection of new, unknown or not yet categorised threats.

In the landscape in which polymorphic code is used and command-and-control servers are frequently changed to avoid detection, a IOC-based approach can represent a limitation in the detection of spyware. The lists of IOCs also require continuous updates to made these tools effective.

4.2.1 Clustering technique

The main constraint that prevents the effective application of advanced machine learning techniques for network-based threat hunting lies in the short duration of the data collection process. Since the traffic capturing window is limited there is no long-term baseline of normal behaviour for comparison and training. As a result, algorithms that classify or predict anomalies cannot achieve high statistical confidence.

If network traffic is captured over an extended period, features such as the number of connections per minute, average session duration, protocol distribution, frequency of HTTP POST requests, packet size variations, and the presence of periodic beaconing activity can be analysed to detect anomalies indicative of malicious activity.

As discussed in the paragraph section 3.1.2 supervised learning techniques struggle in spyware detection during network analysis. For the following test I used clustering techniques, rather than supervised learning techniques. Spyware tend to deliberately hide their digital prints, modifying behaviour, signatures, and network patterns to evade detection.

Moreover, the behaviour varies significantly from application to application, making it difficult to define stable and generalizable classes.

There is currently no reliable and comprehensive database of spyware network behaviour, both commercial and mercenary, necessary for training supervised models.

Added to this is the complexity of feature engineering, also reported in the mentioned study [68].

In this context, clustering allows to identify anomalies in data without requiring predefined labels, making it more suitable for exploring and identifying suspicious

behaviors.

In the next lines, I will propose an approach based on a machine learning clustering solution. After the analysis SpyGuard produces two useful files for further analysis: capture.pcap and the Suricata output eve.json.

The first tool used is Zeek [94], which is not an active defence mechanism, but acts as a sensor and captures high-fidelity transaction logs, analysing traffic in real-time. I used Zeek utility to generate log files starting from the pcap file generated by SpyGuard. Zeek produces different logs, including those related to dhcp, dns, http, quic, ssl and x509. For the purpose of this test are considered conn.log, which is one of the most important logs created and includes both TCP and UDP connection, and ssl.log which parse TLS traffic.

To translate the logs produced by Zeek in dataframe I used Zeek Analysis Tool (ZAT) [110], a python package which supports the processing and analysis of Zeek data with Pandas, utility “LogToDataFrame”. After the translation it is possible to manage the dataframe with Pandas, so I performed an inner join of conn.log and ssl.log based on the UID, the unique identifier created by Zeek for each connection. I grouped all the connections with the same source IP, destination IP and server name, and I created a new column that represents the amount of time passed from the previous connection to the same server.

Then, I created a new dataframe that group all the connection for source IP, destination IP, and server name, computing the following features:

- conn_count: the total number of connections.
- mean_interval: the mean time in seconds between consecutive connections.
- std_interval: standard deviation of time between connections. Higher values mean irregular (or in bursts) connection, which underlines that the time intervals between connections are very different from each other. Lower values, instead, represent regular connections typical of the beaconing behaviour.

I loaded the Spyguard whitelist to filter the known secure traffic, checking the presence of domains both in whitelist and dataframe.

At this point, the dataframe contains all the filtered connections, and if there are enough elements clustering techniques can be used to filter them.

“Clustering is an unsupervised machine learning technique designed to group unlabeled examples based on their similarity to each other.” [111]

This machine learning technique is very powerful to perform anomaly detection. Particularly, k-means has been used to group similar connection isolating spyware, and the right number of clusters is computed on each dataset using the “elbow

method”, a visual approach to determine the ideal “K”.

With this method, during my tests, I was able to isolate the spyware connection in a single cluster, separated from “normal” connection.

This approach helped in cases like the Qustodio one in which SpyGuard had not reported the stalkerware presence.

The script correctly isolated the spyware connections in different dataset tested. Test has been performed on Mobile-Tracker-Free, Qustodio, SpyOne, FlexSPY, mSpy, TheWiSpy, and uMobix (some of these pcap file has been retrieved AL-Qabalin, Eng Majdi (2022), “Android Spyware Detection Using Machine Learning: A Novel Dataset”, Mendeley Data, V1, doi: 10.17632/mhvgtywrx1.1 [112]). While reporting suspicious clusters, the script also reports legitimate domains that can be considered false positives. Sometimes harmless domains are inserted in the same cluster of the suspicious ones.

Another similar solution can be implemented using DBSCAN instead of k-means. DBSCAN handles the outliers better, isolating the anomalies. This solution overcomes the limitation imposed by the k-means algorithm, which tends to move the centroids of the cluster to include the anomalies. DBSCAN finds core samples of high density and expands clusters from them, does not require the number of clusters in advance and can identify outliers as noise points, categorising them as noise with the label “-1”. DBSCAN results depend on two parameters:

- `eps`: is the maximum distance between two samples for one to be considered as in the neighbourhood of the other.
- `min_sample`: is the number of samples in a neighbourhood for a point to be considered as a core point.

In the context of this analysis, the density-based approach offered by DBSCAN could be particularly effective. Dense, legitimate user traffic (e.g. browsing) is grouped in clusters, while the rapid, periodic or beaconing connections generated by the spyware are flagged as noise (-1), making the identification more accurate. While this algorithm is more accurate and less prone to false positives compared to k-means, results can include legitimate traffic, and spyware traffic can act as "standard" traffic, thus evading detection. The last consideration on this algorithm is that the result of the successful isolation of spyware connection heavily depends on the `eps` parameter: by increasing or decreasing this parameter the result may include other domains in the outliers, so the correct parameter should be found and varies for each dataset analysed.

Taking these considerations the two proposed scripts cannot be considered as an autonomous detection system, but should be considered as an enrichment tool

that helps in analysing the hundreds of connections present in the pcap file that SpyGuard produces, isolating the suspicious ones for further analysis.

Eventually, suspicious results are scanned with Virus Total [113] to detect the link with known malwares. Virus Total can be considered an enrichment on the result because not always spyware or stalkerware domains are reported as suspicious or malicious.

src_ip	dst_ip	server_name	conn	mean_interval	std_interval	clust
10.42.0.75	18.208.104.148	api.qustodio.com	5	48.41361928	36.33671783212	1
10.42.0.75	18.66.195.85	static.qustodio.com	3	7.27047205	1.311760979640	2
10.42.0.75	3.93.105.88	mdm-service.qustodio.com	6	68.9392545702	105.2410910364	3
10.42.0.75	34.195.231.27	mdm-service.qustodio.com	6	48.019367218	64.95333648654	1
10.42.0.75	44.207.150.238	mdm-service.qustodio.com	6	38.7900285722	56.20015341160	1

Table 4.4: Cluster division of Qustodio connections using k-means

src_ip	dst_ip	server_name	conn	mean_interval	std_interval	clust
10.42.0.75	18.208.104.148	api.qustodio.com	5	48.41361928	36.33671783212	-1
10.42.0.75	18.66.195.85	static.qustodio.com	3	7.27047205	1.311760979640	0
10.42.0.75	3.93.105.88	mdm-service.qustodio.com	6	68.9392545702	105.2410910364	-1
10.42.0.75	34.195.231.27	mdm-service.qustodio.com	6	48.019367218	64.95333648654	-1
10.42.0.75	44.207.150.238	mdm-service.qustodio.com	6	38.7900285722	56.20015341160	-1

Table 4.5: Cluster division of Qustodio connections using DBSCAN

```
1 --- Virus total scanning anomalous domains ---
2
3 === Domain: mdm-service.qustodio.com ===
4 Reputation: 0
5 Last analysis stats: {'malicious': 0, 'suspicious': 0, 'undetected
6   ': 33, 'harmless': 62, 'timeout': 0}
7 Categories: {}
8
9 === Domain: api.qustodio.com ===
10 Reputation: 0
11 Last analysis stats: {'malicious': 0, 'suspicious': 0, 'undetected
12   ': 33, 'harmless': 62, 'timeout': 0}
13 Categories: {}
14
15 === Domain: static.qustodio.com ===
16 Reputation: 0
17 Last analysis stats: {'malicious': 0, 'suspicious': 0, 'undetected
18   ': 34, 'harmless': 61, 'timeout': 0}
19 Categories: {}
```

Mobile Tracker Free has been detected as malicious

```
1 === Domain: mobile-tracker-data.com ===
2 Reputation: 0
3 Last analysis stats: {'malicious': 2, 'suspicious': 1, 'undetected
4   ': 31, 'harmless': 61, 'timeout': 0}
5 Categories: {'alphaMountain.ai': 'Information Technology,
6   Suspicious (alphaMountain.ai)'}
```

Virus Total detection			
Domain	Detected	Vendor	Is in SpyGuard / Echap IOCs
mdm-service.qustodio.com	False		false
api.qustodio.com	False		false
static.qustodio.com	False		false
mobile-tracker-data.com	True	CRDF, Seclookup	True
ue.spyone.pl	False		False
push.mobilefonex.com	True	CRDF, Seclookup	True
a.thd.cc	True	Dr.Web, Seclookup, Forcepoint ThreatSeeker	True
cp.thewispy.com	True	CRDF, Webroot	True
us.umobix.com	True	CRDF	True
app4.space	True	alphaMountain.ai, Webroot	False
family.mmguardian.com	False		False

Table 4.6: Virus Total detection results for the analyzed domains.

The following section presents the output generated by the developed analysis script. The script processes as input the packet capture file FlexSPY.pcap, which was obtained from the public dataset available at Mendeley Data [112].

The FlexSPY_Traffic.pcap file contains network traffic samples associated with the FlexiSPY spyware, a commercial surveillance application designed to monitor user activity on compromised devices.

```
1 Whitelist loaded: 1129 domains
2 Rows hit with whitelist: 184
3 Rows remaining for analysis: 27
4
5 TOP 10 TLS:
6      src_ip          dst_ip  \
7 171 10.215.173.1     52.205.117.34
8 139 10.215.173.1     23.79.129.52
9 26 10.215.173.1     119.8.47.97
10 154 10.215.173.1     35.201.71.192
11 103 10.215.173.1     176.29.114.155
12 182 10.215.173.1     54.220.62.216
13 77 10.215.173.1     172.217.171.194
14 83 10.215.173.1     172.217.171.227
15 120 10.215.173.1     199.127.193.108
16 156 10.215.173.1     35.210.53.219
17
18      server_name      conn_count
19 171      c.deployads.com      10
20 139      www.aljazeera.net    10
21 26      push.mobilefonex.com 7
22 154      c.pub.network        6
23 103      live-hls-web-aja.getaj.net 6
24 182      c.deployads.com      4
25 77      adservice.google.jo  4
26 83      www.google.jo        4
27 120      62ea30a39dede4f6d44ac289591051076c294526.cws.c... 4
28 156      pool.admedo.com      3
```

Starting clustering analysis...

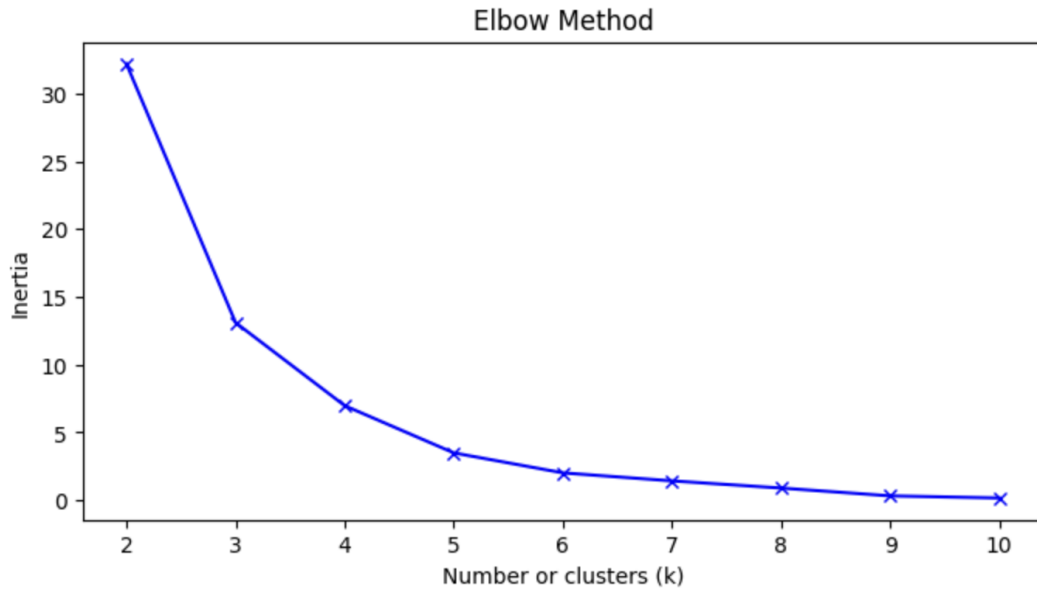


Figure 4.2: Elbow method plot for choosing the number of clusters

```

1 KMeans with 3 clusters...
2
3 --- Cluster dimension ---
4 cluster
5 0    19
6 1     6
7 2     2
8
9
10 --- Cluster centroids ---
11          conn_count  mean_interval  std_interval
12 cluster
13 0          1.947368         0.329094         0.461893
14 1          6.666667         16.381480         23.146175
15 2          5.500000         66.328583         75.949371
16
17 --- Suspicious Connections from Anomalous Clusters (Clusters: [2])
18 ---
19          src_ip          dst_ip          server_name
20          conn_count \
21 26  10.215.173.1          119.8.47.97  push.mobilefonex.com
22 7
23 83  10.215.173.1  172.217.171.227          www.google.jo
24 4

```

```
22 |     mean_interval  std_interval  cluster
23 | 26      88.455629      75.596556      2
24 | 83      44.201538      76.302187      2
25 | --- Virus total scanning anomalous domains ---
26 |
27 | === Domain: push.mobilefonex.com ===
28 | Reputation: -1
29 | Last analysis stats: {'malicious': 2, 'suspicious': 0, 'undetected
30 |   ': 31, 'harmless': 62, 'timeout': 0}
31 | Categories: {'alphaMountain.ai': 'Business/Economy (alphaMountain.
32 |   ai)', 'Sophos': 'general business', 'Forcepoint ThreatSeeker':
33 |   'information technology'}
34 |
35 | === Domain: www.google.jo ===
36 | Reputation: 0
37 | Last analysis stats: {'malicious': 0, 'suspicious': 0, 'undetected
38 |   ': 32, 'harmless': 63, 'timeout': 0}
39 | Categories: {}
```

As is possible to see with the k-means solution also `www[.]google[.]jo` has been included in the cluster of anomalous connections. This result is obtained by dividing the connections into three clusters; by decreasing the clusters number no anomalous connection is detected, while increasing it includes too many false positives. The detection result can be improved by using the DBSCAN techniques.

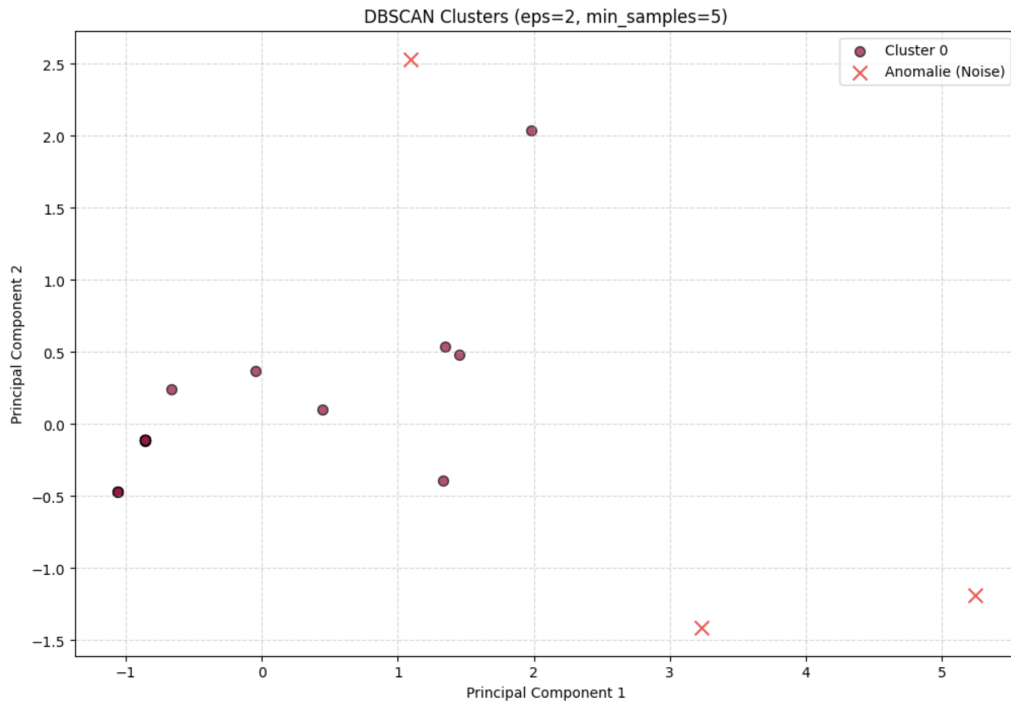


Figure 4.3: DBSCAN cluster plot (eps=2, min_samples=5)

Total point: 27

Anomalies detected (Noise): 3

	src_ip	dst_ip	server_name	conn_count
26	10.215.173.1	119.8.47.97	push.mobilefonex.com	7
83	10.215.173.1	172.217.171.227	www.google.jo	4
171	10.215.173.1	52.205.117.34	c.deployads.com	10

Figure 4.4: DBSCAN results (eps=2, min_samples=5)

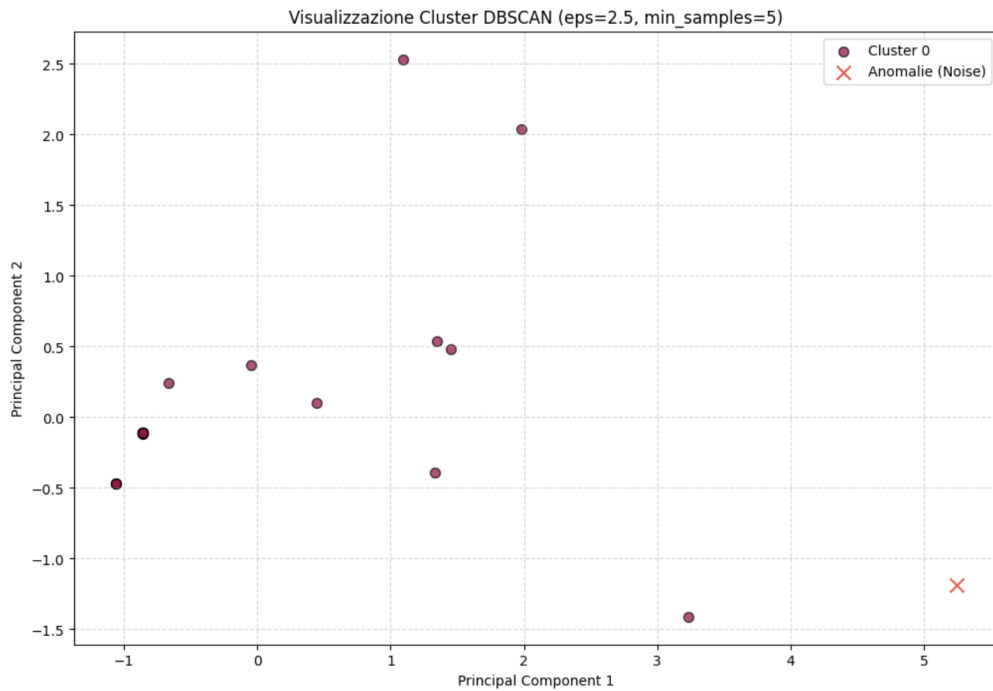


Figure 4.5: DBSCAN cluster plot (eps=2.5, min_samples=5)

```

Total point: 27
Anomalies detected (Noise): 1
      src_ip      dst_ip      server_name  conn_count
26  10.215.173.1  119.8.47.97  push.mobilefonex.com      7
    
```

Figure 4.6: DBSCAN results (eps=2.5, min_samples=5)

As can be noticed from the plot figures 4.3 ($eps = 2$) 4.5 ($eps = 2.5$) the results are deeply dependent on the chosen eps hyperparameter; varying it determines the model's ability to distinguish between legitimate traffic and suspicious activity. By accurately choosing the eps parameter, it is possible to isolate the outliers' connection and target them as anomalies.

While a larger $epsilon$ accounts for the natural variability of standard traffic, it risks over-smoothing the clusters, potentially making obfuscated spyware connections 'invisible' by absorbing them into the legitimate traffic core.

4.2.2 Solutions based on LLMs

Spyware suspicious behaviour often comes out from numerical patterns, such as number of connections, regularity of time intervals, traffic volumes, instead of the semantic meaning of a single log line. Large language models (LLMs) are really useful tools for threat hunting, because they can process large amounts of text, security reports, extracting features and information. For example, a LLMs can be used on alert.json file to analyse and classify the alert, recognising the false positive and understanding the severe ones. In this context, a LLM can be used as an enrichment tool: after the clustering, when suspicious domains emerge these can be passed to a LLM that can provide a human-friendly explanation about them. Here I report some examples of LLM explanations. For the following tests I used Gemini 2.5 Pro and GPT-5 with this prompt: “You are a cybersecurity expert. Your role is to explain the purpose of the given domain, considering that it has been found in the network traffic of a smartphone in the context of a forensic analysis. Please provide a maximum of two lines of explanation. Analyse the following domain:”

“actualite[.]emergence-mada[.]com is the "news" subdomain for Emergence Madagascar, a Malagasy online news publication. Its presence in traffic most likely indicates the user was browsing news articles on this site or an app was pulling news content from this source.” (Gemini 2.5 Pro)

The domain actualite[.]emergence-mada[.]com appears to impersonate or misuse the structure of the legitimate Mauritanian news site on Emergence - “emergence.mr” (which covers news and analyses in Mauritania). Because the sub-domain (actualite) combined with “-mada.com” (instead of the official “.mr”) diverges from the authentic site’s naming scheme, its appearance in smartphone traffic warrants suspicion of phishing, ad-injection or malicious redirect behaviour rather than legitimate news content. (GPT-5)

Actually, this domain is in the IOCs related to Predator spyware and Gemini did not detect it.

“icloudeu[.]com is a malicious typosquatting domain posing as Apple’s legitimate icloud[.]com. Its purpose is almost certainly phishing, attempting to trick the user into entering their Apple ID credentials on a fake login page.”

“ikea-egypt[.]net is not the official IKEA Egypt domain (which is ikea[.]com/eg/) and is highly suspicious. Its presence likely indicates the user visited an impersonation site, potentially for phishing, ad-fraud, or selling counterfeit goods.”

“amazing[.]lab utilizes the .lab TLD, which is typically reserved for internal

development, testing, or research networks. Its presence in external traffic is highly unusual and suggests the device may be communicating with a non-public test environment or a misconfigured/suspicious server.”

“mdm-service[.]qustodio[.]com is a legitimate domain for Qustodio, a parental control and monitoring software. This traffic indicates the device is (or was) actively managed by this app, which uses Mobile Device Management (MDM) services to monitor activity, filter content, and enforce restrictions.”

These cases highlight the bias an LLM can exhibit when judging a domain’s legitimacy based on superficial name similarities. These tests underline the utility of LLMs as an enrichment tool, but confirm that they cannot be considered reliable in the identification phase.

4.2.3 VPNs identifier

The analysis carried out during this work underlines one of the structural limitations of SpyGuard: the system cannot detect or classify the activity of spy applications when all their network traffic is completely encapsulated within a VPN tunnel. In such cases, in fact, the encryption enforced by the VPN protocol hides the destination endpoints, characteristic that SpyGuard normally leverages to detect connection to spyware infrastructures. While this is a limitation inherent to any network-based detection system operating outside the VPN endpoint, nevertheless, situational awareness can be improved by detecting with better accuracy whether the device is making use of a VPN service.

Commercial monitoring applications in some cases use VPN to safely transmit information retrieved and avoid detection. Furthermore, some applications, for example Qustodio, uses the VPN to encapsulate and intercept all the web traffic, applying filters on it.

By indicating the utilization of a VPN system in the report it gives an important benefit for the end user that can verify if the VPN activation corresponds to an international choice. In fact, the system makes it possible to detect situations in which a VPN is enabled without the user’s explicit consent, for example by a monitoring application.

To address this objective, two complementary approaches can be adopted. The first involves the integration of external reputation services, such as vpnguard.io, which provides structured information about IP addresses associated with VPN providers, proxy infrastructures, or commercial hosting platforms. This solution has the advantage of relying on an externally maintained, continuously updated dataset. Furthermore, vpnguard.io has various subscription tiers, including a free plan

that supports up to 1,000 API queries per day. However, immediate dependency on an external API comes with constraints regarding availability, rate limits, and potential costs in the long term.

The second approach involves including a locally-maintained list of known VPN IP ranges directly within the SpyGuard engine. For experiments completed for this thesis, such a list was sourced from a publicly available GitHub repository [114] and integrated into the flow-enrichment phase of the analysis pipeline. This approach does not require any external queries however, it is only as effective as the list itself. The accuracy of the detection depends on the presence of the IPs in the list, and the list needs to be periodically updated, due to the fact that providers evolve or expand their network infrastructures.

In general, the improvement of VPN detection is not a direct solution of the problem of encrypted traffic analysis; instead, it offers an important context that allows drawing conclusions about the communication patterns observed on the device in a more educated manner, hence contributing to a clearer assessment of the conditions under which SpyGuard operates.

```

1 def check_vpn(self, ip_address):
2     """This function implement the effective control that
3     match the IPs with the list"""
4
5     if ip_address in self.ip_reputation_checked:
6         return False
7     self.ip_reputation_checked.append(ip_address)
8
9     try:
10        ip_obj = IPv4Address(ip_address)
11
12        for network in self.vpn_networks:
13            if ip_obj in network:
14                self.alerts.append({
15                    "title": "Suspicious IP Detected (VPN/
16                    Datacenter)",
17                    "description": f"The IP {ip_address} is in
18                    a known VPN range (Local Database Match).",
19                    "host": ip_address,
20                    "level": "Moderate",
21                    "id": "VPN-01"
22                })
23            return True
24
25        except AddressValueError:
26            pass #if for any reason is not an IP
27        except Exception as e:
28            self.errors.append(e)

```

```
26 |  
27 |         return False
```

Here is reported the core function of the VPN checker, that works together with other pieces of code that retrieve the list of IPs and load it in memory.

The code reported basically performs two operations: checks the IP address and adds it in a list to avoid reporting multiple times the same address, and creates a new alert element with id VPN-01 and level “Moderate”.

Both the online VPN detectors and the list used in the test report potentially harmless traffic because are also reported the datacenters and services related for example to Google or Cloudflare.

Natively SpyGuard does not perform any scan for VPN services, but in some cases it reports alerts related to the possible use of VPN, like UDP communications that go outside the local network or SSL connections that use non-standard ports.

A selection of representative test cases is reported below, illustrating the behaviour of the system under different operating conditions and validating the observations discussed in the previous sections.

Tests:

Test 1:

Monitoring APP: IKeyMonitor

Device: A physical Android device (Huawei P20 Lite) running Android 9.

VPN service: NO VPN

The stalkerware was correctly detected by SpyGuard and reported with a “High” severity level.

In addition, two IP addresses were flagged because they matched entries in the VPN/datacenter list.

However, further analysis showed that these IPs belong respectively to Amazon Web Services (AWS) and ZenLayer, both widely used cloud providers.

Therefore, their presence in the network traffic is not directly related to the stalkerware, nor does it indicate the use of an actual VPN infrastructure.

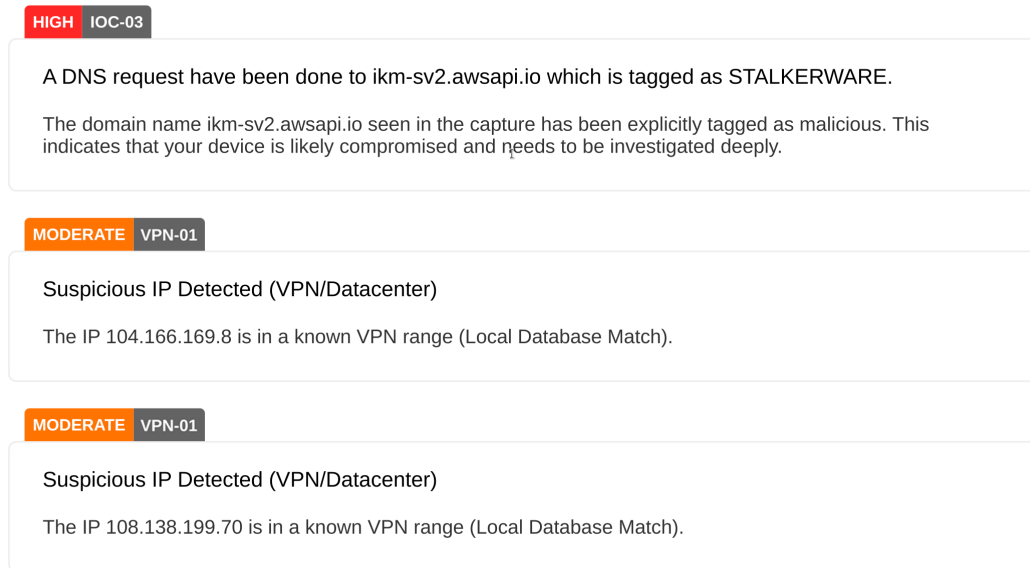


Figure 4.7: IKeyMonitor without VPN

Test 2:

Monitoring APP: IKeyMonitor

Device: A physical Android device (Huawei P20 Lite) running Android 9.

VPN service: PlanetVPN

PlanetVPN is a free VPN service [115] which works with different VPN protocols. During this test I used the StarGuard protocol which is a proprietary protocol based on WireGuard.

In this case SpyGuard wasn't able to detect the stalkerware application due to the encrypted traffic, but reported a suspicious SSL connection done on a non standard port to the IP related to the PlanetVPN service. The same IP was also reported and flagged as VPN by the additional control implemented.

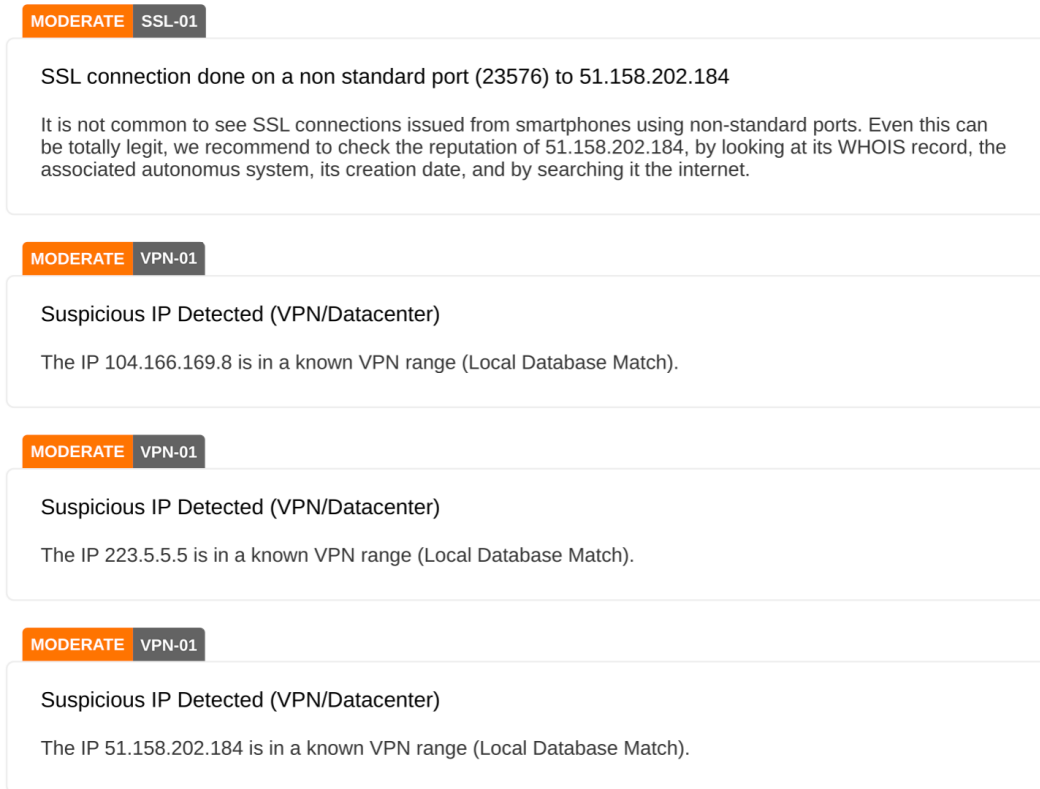


Figure 4.8: IKeyMonitor with PlanetVPN

Test 3:

Monitoring APP: IKeyMonitor

Device: A physical Android device (Huawei P20 Lite) running Android 9.

VPN service: TurboVPN

TurboVPN [116] is a VPN service that offers free connections to a limited list of their servers. As other services, it relies on different protocols.

Also in this test, as expected, SpyGuard wasn't able to discover the stalkerware. In this case an alert was produced on a UDP communication going outside the local network to the IP related to the VPN. As happened in the previous test, the IP was also reported by the additional control.

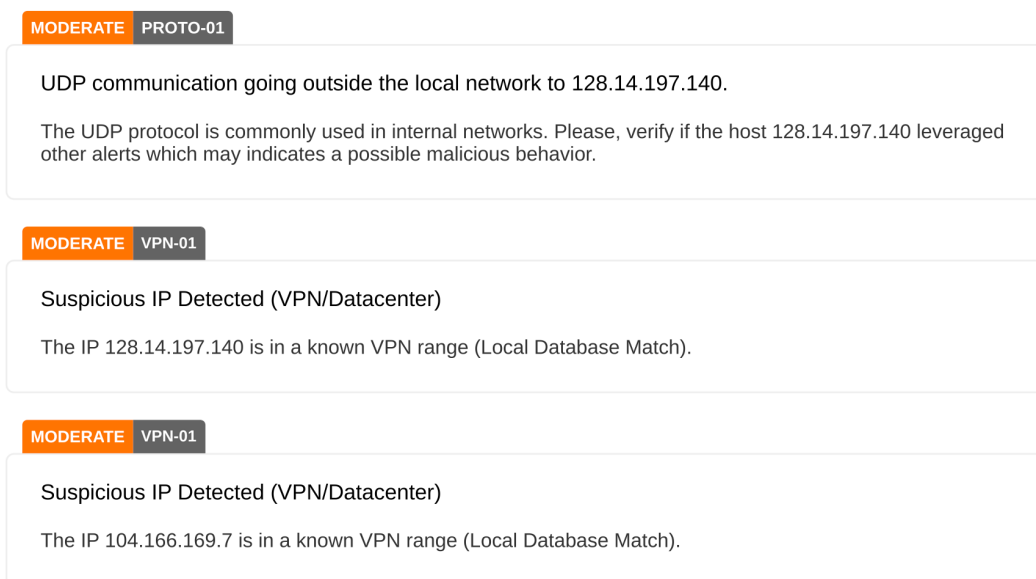


Figure 4.9: IKeyMonitor with TurboVPN

In conclusion, this feature does not directly improve the monitoring app detection capabilities of SpyGuard, as the alerts generated for VPN or datacenters IPs addresses do not provide reliable indicators of stalkerware activity.

Nevertheless, it offers a significant advantage from an analytical perspective, as it allows the user to better understand the nature and destination of the outbound traffic generated by applications on the device.

By highlighting communications toward cloud infrastructures, or undesired VPN infrastructures, this control contributes to increase the awareness about protocols used and can support further manual investigation.

4.2.4 Inducement operations

The last improvement is related to the network traffic analysis, and not strictly to spyguard. As underlines the research “SNDMI: Spyware network traffic detection method based on inducement operations”[117], the spyware and the stalkerware software are often projected to operate in stealth mode, minimizing their network activity. As analysed in this experiment there are two main triggers for network spyware activation: a certain amount of data collected, or a certain amount of software runtime.

A simple passive capture of traffic from an infected device, especially if left idle, may therefore miss crucial indicators of malicious activity, such as communications

with Command and Control (C2) servers or data exfiltration.

To address this problem, inducement techniques can be applied to enhance the probability that spyware generates network traffic.

According to this study, which analyses 100 different spyware sample, the top five user operations focused on by spyware are as follows:

User Operation	Recording Behaviour	Proportion
keyboard tapping	Record the user's keystroke sequence	100%
clipboard manipulation	Record user's clipboard manipulation	41%
password input	Record user's password input, storage	37%
application activity	Record users' application activities	32%
send and receive email	Record users' email activity	27%

Table 4.7: Spyware's main focus on user operations and recording behaviors. Source: [117]

Inducement behaviour	Description
Keyboard tapping	Enter a sequence of virtual keystrokes.
Clipboard manipulation	Manipulate the clipboard to perform user copy, paste, and cut operations.
Application activity	Perform routine software tasks such as Microsoft Word, Skype, etc..
Web browsing	Launch Web browser, click Web page link, and perform related actions.
Screen switching	Switch between screen windows and adjust the window focus accordingly.
File operation	Perform file creation, opening, modification, closing, renaming, moving, and deletion operations.
Email activity	Perform operations to send and receive e-mail messages.
Password input	Enter the user password on the login page of software.

Table 4.8: Description of the inducement behaviors. Source: [117]

This research provides strong evidence that actively performing inducement operations is a highly effective way to overcome the inherent stealth and trigger-based tactics of spyware, making their network traffic far more detectable. For spyware triggered by a specific amount of recorded data, inducement operations cause the spyware to perform its network behaviour faster and more frequently to generate more network traffic. For ones triggered by a certain amount of software runtime, inducement operations cause the spyware to increase the amount of traffic data it transmits when the trigger time is reached.

4.3 Test of MVT on Android

These tests aim to verify how MVT acts in the detection of spyware on mobile devices, verifying its ability to discover and isolate the malicious application through the ADB interface. Moreover the tests aim to evaluate the given commands and the logs generated by MVT, showing the differences with the network analysis. The tests are performed on an Android device on two different applications: Mobile-Tracker-Free and MMGuardian.

MVT has two types of operation that can be performed on Android: check-adb or check-Androidqf. These two produce really similar outputs, with the difference that check-Androidqf performs a static analysis on files as ADB backup or filesystem dump that have been extracted previously, while the check-adb command performs a live analysis on the system connected via USB port using the ADB interface.

Mobile Tracker Free

The analyses have been conducted on the following hardware:

- A physical Android device (Huawei P20 Lite) running Android 9.
- Android Debug Bridge version 1.0.41, running on Linux (aarch64)

Check on ADB

```
1 WARNING [mvt] The analysis of the AndroidQF acquisition produced
   4 detections!
```

Mobile Verification Toolkit analysis correctly reported the application installed:

```
1 -----
2 WARNING [mvt.Android.modules.adb.dumpsys_receivers] Found a known
   suspicious app with ID
3         "com.protect.download" matching indicators from
4         "MobileTrackerFree"
   -----
```

It the console MVT report also security related problems:

```
1 -----
2 WARNING [mvt.Android.modules.adb.settings] Found suspicious "
   secure" setting "install_non_market_apps
3         = 1" (enabled installation of non Google Play
   apps)
4         WARNING [mvt.
   Android.modules.adb.settings] Found suspicious "global" setting
```

```

4           "package_verifier_user_consent = -1" (disabled
      Google Play Protect)
5 WARNING [mvt.Android.modules.Androidqf.getprop] This phone has
      not received security updates for more than six months (last
6           update: 2020-08-01)
7 -----
8           INFO      [mvt.Android.modules.Androidqf.dumpsys_packages]
      Running module DumpsysPackages...
9           INFO      [mvt.Android.modules.Androidqf.dumpsys_packages]
      Found package "com.google.Android.googlequicksearchbox"
      requested 12
10          potentially dangerous permissions
11          INFO      [mvt.Android.modules.Androidqf.dumpsys_packages]
      Found package "Android" requested 20 potentially dangerous
      permissions
12          INFO      [mvt.Android.modules.Androidqf.dumpsys_packages]
      Found package "com.Android.mms" requested 13 potentially
      dangerous
13          permissions
14          INFO      [mvt.Android.modules.Androidqf.dumpsys_packages]
      Found package "mobile.parental2025" requested 14 potentially
      dangerous
15          permissions
16 -----

```

In the logcat.txt file it is possible to find various activities performed by the spyware. All the lines report the timestamp, the process and the event requested.

```

1 11-08 16:52:22.261 1280 1280 I notification_cancel: [1000,1280,
      mobile.parental2025,9537254,NULL,0,0,0,8,NULL]

```

This line is an informative log “I”, where 1280 is the process ID, and tells that a notification which was previously displayed in the status bar has been removed or dismissed from the notification manager. [1000,1280,mobile.parental2025,9537254,NULL,0,0,0,8,NULL] are the parameters passed to the notification cancellation method.

That means that the Android system (UID 1000) deleted a notification belonging to app mobile.parental2025 with the ID 9537254.

```

1 11-08 17:03:37.225 1280 1430 D hw_netstat: total/104/427,mobile.
      parental2025/104/187,com.touchtype.swiftkey/0/240

```

This line is a level “D” log (debug) and reports the network statistics. From this log it is possible to know that mobile.parental2025 sent 104 bytes and received 187 bytes.

```
1 11-08 16:53:22.907 1280 2992 I am_create_service: [0,56162399,.
  ServiceTakePictureRemoteVersion2,10154,10296]
2 11-08 16:53:22.913 1280 1765 I am_finish_activity: [0,88633091,28,
  mobile.parental2025/com.app.activity.picture.
  ActivityTurnONTakePictureRemote,app-request]
3
4 11-08 16:53:22.932 10296 10296 I am_on_create_called: [0,com.app.
  activity.picture.ActivityTurnONTakePictureRemote,performCreate]
5 11-08 16:53:22.949 1280 1280 I notification_cancel: [1000,1280,
  mobile.parental2025,54513,NULL,0,0,0,8,NULL]
6 11-08 16:53:22.956 10296 10296 D ActivityThread: add activity
  client record, r= ActivityRecord{429649f token=Android.os.
  BinderProxy@b725a8 {mobile.parental2025/com.app.activity.
  picture.ActivityTurnONTakePictureRemote}} token= Android.os.
  BinderProxy@b725a8
7 11-08 16:53:22.980 1280 1560 I am_destroy_activity:
  [0,88633091,28,mobile.parental2025/com.app.activity.picture.
  ActivityTurnONTakePictureRemote,finish-imm:
  completedPausedLocked]
8 11-08 16:53:23.369 10296 10296 D ActivityThread: Remove activity
  client record, r= ActivityRecord{429649f token=Android.os.
  BinderProxy@b725a8 {mobile.parental2025/com.app.activity.
  picture.ActivityTurnONTakePictureRemote}} token= Android.os.
  BinderProxy@b725a8
9 11-08 16:53:23.371 1280 1783 I wm_task_removed: [28,removeAppToken
  : last token]
10 11-08 16:53:23.506 10296 10296 I PermissionManager: camera remind
  result:true
11 11-08 16:53:23.518 10296 10296 I BackgroundPermManager: pkgName:
  mobile.parental2025,pid: 10296 ,uidOf3RdApk: 10154 ,permType: 0
  ,permCfg: 1
12 11-08 16:53:23.759 1280 1280 I notification_cancel: [1000,1280,
  mobile.parental2025,9537254,NULL,0,0,0,8,NULL]
13 11-08 16:53:23.759 1280 1280 I notification_cancel: [1000,1280,
  mobile.parental2025,9537254,NULL,0,0,0,8,NULL]
14 11-08 16:53:26.461 1280 1560 I am_destroy_service:
  [0,56162399,10296]
15 11-08 16:53:26.463 1280 1280 I notification_cancel: [1000,1280,
  mobile.parental2025,15042,NULL,0,0,0,8,NULL]
16 11-08 16:53:26.465 10296 10296 I BackgroundPermManager: pkgName:
  mobile.parental2025,pid: 10296 ,uidOf3RdApk: 10154 ,permType: 0
  ,permCfg: 0
17 11-08 16:53:26.479 1280 1302 I am_proc_start: [0,13073,10154,
  mobile.parental2025:uploadpicture,service,mobile.parental2025/
  com.app.service.UploadPictureRemoteService]
```

```
18 | 11-08 16:53:26.501 1280 1560 I am_proc_bound: [0,13073,mobile.
    | parental2025:uploadpicture] 11-08 16:53:26.505 1280 1560 I
    | am_create_service: [0,168918216,.UploadPictureRemoteService
    | ,10154,13073]
```

Between 16:53:22 and 16:53:26, the application mobile.parental2025 executed a remote photo capture and upload sequence.

At 16:53:22, the system initiated the ServiceTakePictureRemoteVersion2 service to perform a remote photo capture, while the associated activity was launched and immediately closed.

By 16:53:23, the system confirmed that the camera and background permissions were granted, and the app cleared related notifications. At 16:53:26, the photo capture service was terminated, background permission was revoked, and the app launched the UploadPictureRemoteService to upload the captured image.

Among the advantages that a static analysis gives, there's the possibility of verifying the presence of suspicious applications that the network analysis wasn't able to identify, or that aren't installed or active on the device anymore. For example, during the tests on Mobile Tracker Free application, MVT has found traces of the installation of MMGuardian, another parental control app used two days before the analysis.

```
1 |          WARNING [mvt.Android.modules.Androidqf.
    | dumphsys_battery_daily] Found a known suspicious app with ID "
    | com.mmguardian.childapp"
2 |          matching indicators from "MMGuardian"
3 |          WARNING [mvt.Android.modules.Androidqf.
    | dumphsys_battery_daily] Found a known suspicious app with ID "
    | com.mmguardian.childapp"
4 |          matching indicators from "MMGuardian"
5 |
6 |          package=com.mmguardian.childapp totalTime="03:34" lastTime="
    | 2025-10-28 17:48:56" appLaunchCount=18
```

APKs download

During this test MVT has not been able to identify and download the APK related to MTF “mobile.parental2025” because on the device used the command throws an exception when reaching a Huawei system application for which is impossible to retrieve the installer. This error is silent and only applications retrieved before it are considered for further analysis and download.

AndroidQF tool has a similar option that executes the code: `out, err :=`

a.Shell("pm", "list", "packages", "-U", "-u", "-i"). This tool manages the error with a second call to pm list package where the utility is performed without the -i flag, so the APK mobile.parental2025 is correctly detected and downloaded.

MVT also performs a check on the package downloaded with VirusTotal, by passing via API the hash of the file. Despite, as seen during Spyguard tests, VirusTotal recognises MTF domains, the hash does not provide any report.

```
1 "sha256":  
2 "a9be474958857c0698b4526089412e34173fc1406b00b4b3327b7a83db0b4fbb"
```

MMGuardian

```
1 WARNING [mvt.Android.modules.adb.settings] Found suspicious "  
   secure" setting "accessibility_enabled = 1" (enabled  
   accessibility  
2   services)  
3 WARNING [mvt.Android.modules.adb.dumpsys_receivers] Found a known  
   suspicious app with ID "com.mmguardian.childapp" matching  
4   indicators from "MMGuardian"  
5 INFO [mvt.Android.modules.adb.dumpsys_receivers] Found a  
   receiver to intercept incoming SMS messages:  
6   "com.mmguardian.childapp/com.mmguardian.broadcaster.  
   InComingSMSBlockBR"  
7  
8 WARNING [mvt] The analysis of the Android device produced 40  
   detections!
```

```
1 [  
2   {  
3     "user": "u0_a153",  
4     "pid": 20524,  
5     "ppid": 560,  
6     "virtual_memory_size": 4955148,  
7     "resident_set_size": 156524,  
8     "wchan": "0",  
9     "aprocess": "0",  
10    "stat": "S",  
11    "proc_name": "com.mmguardian.childapp",  
12    "label": "",  
13    "matched_indicator": {  
14      "value": "com.mmguardian.childapp",  
15      "type": "app_ids",  
16      "name": "MMGuardian",
```

```
17 |         "stix2_file_name": "raw.githubusercontent.  
18 |         com_AssocEchap_stalkerware -  
19 |         indicators_master_generated_stalkerware.stix2"  
20 |     }  
    }  
  ]
```

Listing 4.5: processes_detected.json

```
1 "name": "com.mmguardian.childapp"  
2 "sha256":  
3 "23ba4dcecff59ca80199adb89bc9f752d1fc367e99a465551f2e36e3504be864"
```

VirusTotal recognised this package as MMGuardian Child Phone App, and no security vendor flagged this package as malicious.

Even though I do not have access to state-sponsored spyware, Amnesty International reports offers a validated forensic methodology that details all the artifacts left by the spyware. Particularly, the report “Forensic Methodology Report: How to catch NSO Group’s Pegasus” reveals interesting insight about the spyware discovery.[15]

The report confirms that also the most sophisticated spyware leaves traces that can help in detecting the infection.

The report shows that Pegasus started to manipulate the databases and system registers on the infected device in the attempt of hiding its traces. This manipulation is evident by checking the consistency of the residue registers in the SQLite databases DataUsage.sqlite and netusage.sqlite. Pegasus removed the malicious processes names from the table ZPROCESS in the DataUsage database, and it didn’t remove the correspondent entries from the ZLIVEUSAGE table. Amnesty International confirmed that these discrepancies weren’t found on other uninfected iPhones, and so it’s possible to consider them as an index of compromise.

Eventually, from this analysis it’s important to learn that static analysis is not bound to malicious file hunting. Pegasus writes a file com.apple.CrashReporter.plist in the directory /private/var/root/Library/Preferences/, with the purpose of disabling the report of crash logs to Apple. The analysis confirmed the disable of the crash reporter after the successful compromise.

4.4 Test of MobSF

This test of Mobile Security Framework (MobSF) platform for mobile applications aims to analyse the amount of data that can be extracted from the APK package obtained through MVT of ADB interface.

The test shows the ability of this platform to decompile the code and extract useful information to compose the lists of IOCs.

Test of MobSF on theTruthSpy.apk retrieved on MawareBazaar.[118]

- File Name: theTruthSpy.apk
- SHA256: 41752e7b3d8374ec74b94ebc258a0fefd3d41a44bb07a43d85546701afb50e36
- App Name: System Service
- Package Name: com.systemservice

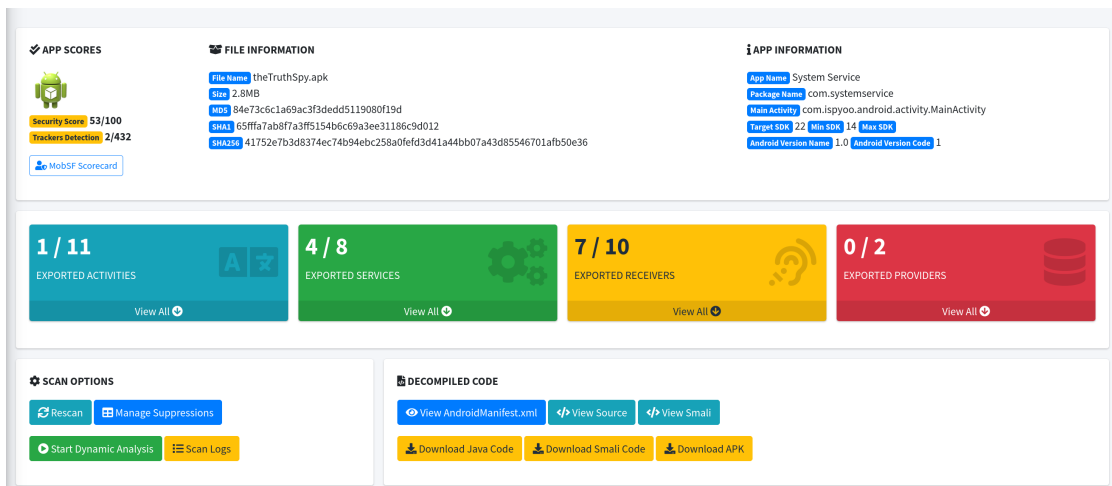


Figure 4.10: Screenshot of the MobSF home page.

From the main screen it is possible to have an overview on what the scan produced. In the decompile section there's the AndroidManifest.xml, the Java source code, and the Smali code.

For example this code creates a separate thread to manage the opening of the camera:

```

1 private void x() {
2     final Semaphore semaphore = new Semaphore(0);
3     final RuntimeException[] runtimeExceptionArr = new
    RuntimeException[1];
4
5     this.B = new Thread(new Runnable() {
6         // from class: net.majorkernelpanic.streaming.f.b.2
7         @Override
8         // java.lang.Runnable
9         public void run() {
10            Looper.prepare();
11            b.this.C = Looper.myLooper();
12            try {
13                try {
14                    b.this.A = Camera.open(b.this.x);
15                } catch (RuntimeException e) {
16                    runtimeExceptionArr[0] = e;
17                }
18            } finally {
19                semaphore.release();
20            }
21            Looper.loop();
22        }
23    });
24
25    this.B.start();
26    semaphore.acquireUninterruptibly();
27
28    if (runtimeExceptionArr[0] != null) {
29        throw new net.majorkernelpanic.streaming.b.a(
    runtimeExceptionArr[0].getMessage());
30    }
31 }

```

Another example is this part of the code that reads the SMS and parses the command received via message and prevents it from being shown by other applications.

```

1 public final class q extends u {
2     /* renamed from: b, reason: collision with root package name
3     */
4     private static final String[] f2923b = {"Android.provider.
    Telephony.SMS_RECEIVED"};
5     private SmsMessage[] a(Intent intent) {
6         SmsMessage[] smsMessageArr;

```

```

6         try {
7             Object[] objArr = (Object[]) intent.getExtras().get("
pdus");
8             smsMessageArr = new SmsMessage[objArr.length];
9             for (int i = 0; i < objArr.length; i++) {
10                try {
11                    smsMessageArr[i] = SmsMessage.createFromPdu((
byte[]) objArr[i]);
12                } catch (Exception e) {
13                    e = e;
14                    Log.d("AndroidSmsCommandWatcher", "Fail to
create an incoming SMS from pdus", e);
15                    return smsMessageArr;
16                }
17            }
18        } catch (Exception e2) {
19            e = e2;
20            smsMessageArr = null;
21        }
22        return smsMessageArr;
23    }
24    @Override // com.ispyoo.common.e.u
25    public void b(com.ispyoo.common.c.a aVar) {
26        boolean a2;
27        com.ispyoo.common.monitor.a.a aVar2 = (com.ispyoo.common.
monitor.a.a) aVar;
28        Context b2 = aVar2.b();
29        SmsMessage[] a3 = a(aVar2.c());
30        for (int i = 0; i < a3.length; i++) {
31            try {
32                String displayMessageBody = a3[i].
getDisplayMessageBody();
33                String originatingAddress = a3[i].
getOriginatingAddress();
34                Log.d("AndroidSmsCommandWatcher", "MESSAGE-CONTENT
: " + displayMessageBody);
35                if (displayMessageBody != null &&
displayMessageBody.length() > 0 && (a2 = com.ispyoo.common.h.a.
a(displayMessageBody))) {
36                    BroadcastReceiver broadcastReceiver = (
BroadcastReceiver) aVar2.a();
37                    broadcastReceiver.abortBroadcast();
38                    broadcastReceiver.setResultData(null);
39                    com.ispyoo.common.h.b.a(b2, displayMessageBody
, originatingAddress, a2);
40                    SharedPreferences.Editor edit = b2.
getApplicationContext().getSharedPreferences("prefs", 0).edit()
;
41                    edit.putBoolean("has_remote_command", true);

```

```

42         edit.commit();
43     }
44     } catch (Exception e) {
45         Log.d("AndroidSmsCommandWatcher", e.getMessage());
46         return;
47     }
48 }
49 }

```

A table summarizes all the permission requested, linking each of them to the code in which is requested or used.

The Android API table groups together all the codes that use a given API. MobSF scans the code searching for vulnerabilities and reports them in the code analysis section. For example 4.9:

IP Address	warning	CWE: CWE-200:	b/a/a/a/c.java
disclosure		Information	b/a/a/a/m.java
		Exposure	net/majorkernelpanic/ streaming/b.java
		OWASP	net/majorkernelpanic/ streaming/rtsp/b.java
		MASVS:	
		MSTG-CODE-2	

Table 4.9

Other useful reports are the Firebase database analysis, which includes the links to the database found and the behavioural analysis that groups the java file by their behaviour.

The analysis also reported the presence of code that prevents the debug and application to run in a virtual environment.

Anti Debug Code:

```

1  Debug.isDebuggerConnected() check

```

Anti-VM Code:

```
1 Build.MODEL check
2 Build.MANUFACTURER check
3 Build.PRODUCT check
4 Build.TAGS check
```

This analysis tool is really useful to deep dive in the application analysis and extract all the possible IOCs that can be used with MVT and SpyGuard.

In this test the following IOCs have been found:

- App Name: System Service
- Package Name: com.systemservice
- Certificate sha1: ff8ccd9816b0524a58fbde1809fb227dbcdfd692
- domains: thetruthspy[.]com, ispyoo[.]com
- c2 domains: https://thetruth-db94a[.]firebaseio[.]com, protocol-a[.]thetruthspy[.]com
- email: support@thetruthspy.com

All the IOCs retrieved in this test (except for the email) are already in the stalkerware indicator list of AssoEchap. [45]

4.5 Spyware Discovery via ADB-Based Forensic Analysis

The primary objective of this test is to evaluate a forensic methodology to identify the presence of a suspicious surveillance software on an Android device. Unlike other detection methods that rely on IOCs lists, this test propose a proactive approach to discovery the threat.

The scenario presented simulates an infection that need to be discovered on a mobile device.

After a first verification with MVT, then all the operations and acquisitions are executed through the ADB interface, without the use of IOCs lists.

This analysis has been conducted on an infected Android mobile, without knowing the nature of the malicious app and without knowing the name of the infected packet.

The following resources were used for this test:

- A physical Android device (Huawei P20 Lite) running Android 9.
- A MacBook pro M2 running macOS Tahoe (version 26.2)
- TiSpy.apk SHA256: fa5d174178e9a3971a2af75529cbd4e7164bfb5b3d11f1c04a1ff2f810bb90bb

The analysed device has to be prepared by unlocking it and activating the USB debugging in the developer options panel.

A first check can be performed with Mobile Verification Toolkit (MTV). Before performing the test, the lists of IOCs must be loaded. To do this, use the *mut-android download-iocs* command.

Then it is possible to perform the actual control on the device using *mut-android check-adb* command.

The results of the analysis did not report any particular packet as spyware, which means there are no packages that match the IOC lists.

```

1 23:59:13 INFO      [mvt.android.modules.adb.packages] Extracted at
   total of 11 installed package names
2      INFO      [mvt.android.modules.adb.packages] The Packages
   module produced no detections!
3      INFO      [mvt.android.modules.adb.logcat] Running module
   Logcat...
4 23:59:14 INFO      [mvt.android.modules.adb.logcat] The Logcat
   module does not support checking for indicators

```

```

5      INFO      [mvt.android.modules.adb.root_binaries] Running
      module RootBinaries...
6      INFO      [mvt.android.modules.adb.root_binaries] The
      RootBinaries module produced no detections!
7      INFO      [mvt.android.modules.adb.files] Running module
      Files...
8      INFO      [mvt.android.modules.adb.files] Found 18 files
      in primary Android tmp and media folders
9      INFO      [mvt.android.modules.adb.files] Processing full
      file listing. This may take a while...
10 23:59:38 INFO  [mvt.android.modules.adb.files] Found 237179
      total files
11 23:59:42 INFO  [mvt.android.modules.adb.files] The Files module
      produced no detections!

```

Anyhow the report produced by MVT provides valuable insights highlighting the enabled accessibility services and disabled Google Play Protect.

```

1 WARNING [mvt.android.modules.adb.settings] Found suspicious "
      secure" setting "accessibility_enabled =
2          1" (enabled accessibility services)
3 WARNING [mvt.android.modules.adb.settings] Found suspicious "
      global" setting
4          "package_verifier_user_consent = -1" (disabled
      Google Play Protect)

```

As largely discussed in the previous chapters, commercial spyware abuse of accessibility services, so to further investigate the warning produced by MVT, ADB shell commands can be used.

The command `adb shell dumpsys accessibility` produces a list of all the services that are using the accessibility services.

```

1 ACCESSIBILITY MANAGER (dumpsys accessibility)
2 User state[attributes:{id=0, currentUser=true,
      touchExplorationEnabled=false, displayMagnificationEnabled=
      false, navbarMagnificationEnabled=false, autoclickEnabled=false
      }
3          services:{Service[label=WiFi Service, feedbackType[
      FEEDBACK_GENERIC], capabilities=33, eventTypes=[
      TYPE_VIEW_TEXT_CHANGED, TYPE_WINDOW_STATE_CHANGED,
      TYPE_NOTIFICATION_STATE_CHANGED, TYPE_WINDOW_CONTENT_CHANGED],
      notificationTimeout=100}}]

```

```
4 | Window[AccessibilityWindowInfo[title=null, id=61, type=TYPE_SYSTEM
   | , layer=0, bounds=Rect(0, 0 - 1080, 2280), focused=true, active
   | =true, pictureInPicture=false, hasParent=false, isAnchored=
   | false, hasChildren=false]]
```

In the response produced it is possible to see a service labeled “WiFi Services”. This service has the capabilities parameter set to 33, which means that it is using two main capabilities. The value 33 is obtained using a bitmask in which each capability is represented with a power of two value. So considering the capability 32 and 1 the service has enabled these functions [119].

CAPABILITY_CAN_RETRIEVE_WINDOW_CONTENT (value 1)

CAPABILITY_CAN_PERFORM_GESTURES (value 32)

The line eventTypes describes the listeners that the service has activated. This service will be notified when one element of the text, window state, notification state, or window content changes.

Due to the fact that it is very suspicious that a wifi service needs to access all these resources and contents the investigation should continue.

To retrieve the list of all third-party packets installed on the device `adb shell pm list packages -f -3` where `-f` flag shows packages associated file and `-3` filters for third-party apps.

Here a suspicious packet appears in the output.

It’s named “pkg=com.qshmetryg.okrcqynr”, the use of random characters is common practice for malware to avoid the IOCs and blocking list.

The application code path “codePath=/data/app/com.qshmetryg.okrcqynr-cdNoOANmL6hPezXXEQHDQ=” confirms that the application was installed by the user and is not a system application. Due to the fact that commercial spyware applications make large abuse of permissions the following step investigates the permission requested by the suspicious packet discovered during the analysis.

To analyse the permission and perform an in-depth exploration of this suspicious application I used the `adb shell dumpsys package com.qshmetryg.okrcqynr` command. The response includes the table of all the app’s receivers declared in the *AndroidManifest.xml* file. In addition are reported the permission of the application divided in: installed permissions, requested permissions and runtime permissions.

```

1 Activity Resolver Table:
2   Non-Data Actions:
3     android.intent.action.MAIN:
4       60a1fc6 com.qshmetryg.okrcqynr/.PYmWIjdtlE filter 5215ea1
5         Action: "android.intent.action.MAIN"
6         Category: "android.intent.category.LAUNCHER"
7       1220b87 com.qshmetryg.okrcqynr/.jtMXJJJoZuT filter b9088c6
8         Action: "android.intent.action.MAIN"
9       c6c3ab4 com.qshmetryg.okrcqynr/.vwlIcQTQVz filter 55ac087
10        Action: "android.intent.action.MAIN"
11        Category: "android.intent.category.LAUNCHER"
12 Receiver Resolver Table:
13   Schemes:
14     package:
15       6555510 com.qshmetryg.okrcqynr/.core.CiPsSLOxAd filter af8ab69
16         Action: "android.intent.action.PACKAGE_REMOVED"
17         Action: "android.intent.action.PACKAGE_ADDED"
18         Scheme: "package"
19   Non-Data Actions:
20     com.google.android.c2dm.intent.RETRY:
21       7ef8534 com.qshmetryg.okrcqynr/.HflKfIUcsZ filter 8fb751c
22         Action: "com.google.android.c2dm.intent.RETRY"
23         Category: "com.qshmetryg.okrcqynr"
24     android.intent.action.PHONE_STATE:
25       6555510 com.qshmetryg.okrcqynr/.core.CiPsSLOxAd filter 10b0233
26         Action: "android.intent.action.PHONE_STATE"
27     android.net.conn.CONNECTIVITY_CHANGE:
28       6555510 com.qshmetryg.okrcqynr/.core.CiPsSLOxAd filter f007984
29         Action: "android.intent.action.NEW_OUTGOING_CALL"
30         Action: "android.provider.Telephony.SMS_RECEIVED"
31         Action: "android.net.conn.CONNECTIVITY_CHANGE"
32         mPriority=2147483647, mOrder=0, mHasPartialTypes=false
33     com.qshmetryg.okrcqynr.core.YjGQkEsgEp.TIMER_ALARM:
34       bd11604 com.qshmetryg.okrcqynr/.core.YjGQkEsgEp filter e026e16
35         Action: "com.qshmetryg.okrcqynr.core.YjGQkEsgEp.TIMER_ALARM"
36         mPriority=1000, mOrder=0, mHasPartialTypes=false
37     android.app.action.DEVICE_ADMIN_ENABLED:
38       361c01 com.qshmetryg.okrcqynr/.core.QrvmwKEJGt filter 4d8c708
39         Action: "android.app.action.DEVICE_ADMIN_ENABLED"
40     com.qshmetryg.okrcqynr.core.uwCauSsust.START:
41       54b5a8e com.qshmetryg.okrcqynr/.core.uwCauSsust filter f97c397
42         Action: "com.qshmetryg.okrcqynr.core.uwCauSsust.START"
43         Action: "com.qshmetryg.okrcqynr.core.uwCauSsust.STOP"
44         mPriority=1000, mOrder=0, mHasPartialTypes=false
45     android.intent.action.SIM_STATE_CHANGED:
46       6555510 com.qshmetryg.okrcqynr/.core.CiPsSLOxAd filter 2cda9f0
47         Action: "android.intent.action.SIM_STATE_CHANGED"
48     com.google.android.c2dm.intent.RECEIVE:

```

```

49       7ef8534 com.qshmetryg.okrcqynr/.Hf1KfIUcsZ filter 5698e8f
50         Action: "com.google.android.c2dm.intent.RECEIVE"
51         Category: "com.qshmetryg.okrcqynr"
52       2f78122 com.qshmetryg.okrcqynr/com.google.firebase.iid.
FirebaseInstanceIdReceiver filter b29f6fa
53         Action: "com.google.android.c2dm.intent.RECEIVE"
54         Action: "com.google.android.c2dm.intent.REGISTRATION"
55         Category: "com.qshmetryg.okrcqynr"
56       2f78122 com.qshmetryg.okrcqynr/com.google.firebase.iid.
FirebaseInstanceIdReceiver filter 97a84ab
57         Action: "com.google.android.c2dm.intent.RECEIVE"
58         Category: "com.qshmetryg.okrcqynr"
59       com.qshmetryg.okrcqynr.core.uwCauSsust.STOP:
60       54b5a8e com.qshmetryg.okrcqynr/.core.uwCauSsust filter f97c397
61         Action: "com.qshmetryg.okrcqynr.core.uwCauSsust.START"
62         Action: "com.qshmetryg.okrcqynr.core.uwCauSsust.STOP"
63         mPriority=1000, mOrder=0, mHasPartialTypes=false
64       android.intent.action.BOOT_COMPLETED:
65       6555510 com.qshmetryg.okrcqynr/.core.CiPsSLOxAd filter 1af55a2
66         Action: "android.intent.action.BOOT_COMPLETED"
67         Action: "android.intent.action.ACTION_SHUTDOWN"
68       android.intent.action.USER_PRESENT:
69       6555510 com.qshmetryg.okrcqynr/.core.CiPsSLOxAd filter 852256d
70         Action: "android.intent.action.USER_PRESENT"
71       android.provider.Telephony.SMS_RECEIVED:
72       6555510 com.qshmetryg.okrcqynr/.core.CiPsSLOxAd filter f007984
73         Action: "android.intent.action.NEW_OUTGOING_CALL"
74         Action: "android.provider.Telephony.SMS_RECEIVED"
75         Action: "android.net.conn.CONNECTIVITY_CHANGE"
76         mPriority=2147483647, mOrder=0, mHasPartialTypes=false
77       com.qshmetryg.okrcqynr.advance.ACTION_REPLY:
78       49a0ef3 com.qshmetryg.okrcqynr/.advance.NotificationReceiver filter
679525
79         Action: "com.qshmetryg.okrcqynr.advance.ACTION_REPLY"
80       com.google.android.c2dm.intent.REGISTRATION:
81       7ef8534 com.qshmetryg.okrcqynr/.Hf1KfIUcsZ filter c5129ee
82         Action: "com.google.android.c2dm.intent.REGISTRATION"
83         Category: "com.qshmetryg.okrcqynr"
84       2f78122 com.qshmetryg.okrcqynr/com.google.firebase.iid.
FirebaseInstanceIdReceiver filter b29f6fa
85         Action: "com.google.android.c2dm.intent.RECEIVE"
86         Action: "com.google.android.c2dm.intent.REGISTRATION"
87         Category: "com.qshmetryg.okrcqynr"
88       android.intent.action.NEW_OUTGOING_CALL:
89       6555510 com.qshmetryg.okrcqynr/.core.CiPsSLOxAd filter f007984
90         Action: "android.intent.action.NEW_OUTGOING_CALL"
91         Action: "android.provider.Telephony.SMS_RECEIVED"
92         Action: "android.net.conn.CONNECTIVITY_CHANGE"
93         mPriority=2147483647, mOrder=0, mHasPartialTypes=false

```

```

94     android.intent.action.ACTION_SHUTDOWN:
95         6555510 com.qshmetryg.okrcqynr/.core.CiPsSLOxAd filter 1af55a2
96             Action: "android.intent.action.BOOT_COMPLETED"
97             Action: "android.intent.action.ACTION_SHUTDOWN"
98 Service Resolver Table:
99     Non-Data Actions:
100         com.google.firebase.INSTANCE_ID_EVENT:
101             fca442a com.qshmetryg.okrcqynr/com.google.firebase.iid.
102             FirebaseInstanceIdService filter 11f8ec
103                 Action: "com.google.firebase.INSTANCE_ID_EVENT"
104                 mPriority=-500, mOrder=0, mHasPartialTypes=false
105             c60021b com.qshmetryg.okrcqynr/.yKiKnxRCAq filter 851274a
106                 Action: "com.google.firebase.INSTANCE_ID_EVENT"
107         com.google.firebase.MESSAGING_EVENT:
108             12a3a5 com.qshmetryg.okrcqynr/.vubfeRoSoC filter 40714b5
109                 Action: "com.google.firebase.MESSAGING_EVENT"
110             c6a237a com.qshmetryg.okrcqynr/com.google.firebase.messaging.
111             FirebaseMessagingService filter 614731
112                 Action: "com.google.firebase.MESSAGING_EVENT"
113                 mPriority=-500, mOrder=0, mHasPartialTypes=false
114         android.service.notification.NotificationListenerService:
115             961014a com.qshmetryg.okrcqynr/.core.listener.tbizLNpj1A filter f9bb7d8
116             permission android.permission.BIND_NOTIFICATION_LISTENER_SERVICE
117                 Action: "android.service.notification.NotificationListenerService"
118         android.accessibilityservice.AccessibilityService:
119             81e080a com.qshmetryg.okrcqynr/.core.listener.dqym0bcrjY filter 1c276bb
120             permission android.permission.BIND_ACCESSIBILITY_SERVICE
121                 Action: "android.accessibilityservice.AccessibilityService"
122 Permissions:
123     Permission [com.qshmetryg.okrcqynr.permission.C2D_MESSAGE] (64ca3f3):
124         sourcePackage=com.qshmetryg.okrcqynr
125         uid=10154 gids=null type=0 prot=signature
126         perm=Permission{98808b0 com.qshmetryg.okrcqynr.permission.C2D_MESSAGE}
127         packageSetting=PackageSetting{9b18329 com.qshmetryg.okrcqynr/10154}
128 Registered ContentProviders:
129     com.qshmetryg.okrcqynr/com.google.firebase.provider.FirebaseInitProvider:
130         Provider{278ee21 com.qshmetryg.okrcqynr/com.google.firebase.provider.
131         FirebaseInitProvider}
132 ContentProvider Authorities:
133     [com.qshmetryg.okrcqynr.firebaseinitprovider]:
134         Provider{278ee21 com.qshmetryg.okrcqynr/com.google.firebase.provider.
135         FirebaseInitProvider}
136         applicationInfo=ApplicationInfo{87b9bf3 com.qshmetryg.okrcqynr}
137 Key Set Manager:
138     [com.qshmetryg.okrcqynr]
139         Signing KeySets: 39
140 Packages:
141     Package [com.qshmetryg.okrcqynr] (9b18329):
142         userId=10154

```

```
137 pkg=Package{d9b7f9f com.qshmetryg.okrcqynr}
138 codePath=/data/app/com.qshmetryg.okrcqynr-ccdNo0ANmL6hPezXXEQHDQ==
139 resourcePath=/data/app/com.qshmetryg.okrcqynr-ccdNo0ANmL6hPezXXEQHDQ==
140 legacyNativeLibraryDir=/data/app/com.qshmetryg.okrcqynr-
    ccdNo0ANmL6hPezXXEQHDQ==/lib
141 primaryCpuAbi=arm64-v8a
142 secondaryCpuAbi=null
143 versionCode=1 minSdk=21 targetSdk=28
144 versionName=1.0
145 splits=[base]
146 apkSigningVersion=3
147 applicationInfo=ApplicationInfo{87b9bf3 com.qshmetryg.okrcqynr}
148 flags=[ HAS_CODE ALLOW_CLEAR_USER_DATA ]
149 hwflags=[ ]
150 privateFlags=[
151 PRIVATE_FLAG_ACTIVITIES_RESIZE_MODE_RESIZEABLE_VIA_SDK_VERSION ]
152 dataDir=/data/user/0/com.qshmetryg.okrcqynr
153 supportsScreens=[small, medium, large, xlarge, resizeable, anyDensity]
154 usesOptionalLibraries:
155     org.apache.http.legacy
156 usesLibraryFiles:
157     /system/framework/org.apache.http.legacy.boot.jar
158 timeStamp=2026-01-23 19:16:31
159 firstInstallTime=2026-01-23 19:16:31
160 lastUpdateTime=2026-01-23 19:16:31
161 installerPackageName=com.google.android.packageinstaller
162 signatures=PackageSignatures{121e96e version:3, signatures:[fe1eb996], past
    signatures: []}
163 installPermissionsFixed=true
164 pkgFlags=[ HAS_CODE ALLOW_CLEAR_USER_DATA ]
165 declared permissions:
166     com.qshmetryg.okrcqynr.permission.C2D_MESSAGE: prot=signature, INSTALLED
167 requested permissions:
168     com.qshmetryg.okrcqynr.permission.C2D_MESSAGE
169     android.permission.READ_CONTACTS
170     android.permission.READ_SMS
171     android.permission.CALL_PHONE
172     android.permission.READ_PHONE_STATE
173     android.permission.ANSWER_PHONE_CALLS
174     android.permission.PROCESS_OUTGOING_CALLS
175     android.permission.READ_LOGS
176     android.permission.READ_CALL_LOG
177     android.permission.WRITE_EXTERNAL_STORAGE
178     android.permission.SDCARD_WRITE
179     android.permission.VIBRATE
180     android.permission.INTERNET
181     android.permission.ACCESS_COARSE_LOCATION
182     android.permission.ACCESS_FINE_LOCATION
183     android.permission.ACCESS_NETWORK_STATE
```

```
183 android.permission.READ_CALENDAR
184 android.permission.MODIFY_AUDIO_SETTINGS
185 android.permission.RECORD_AUDIO
186 android.permission.CAMERA
187 android.permission.WAKE_LOCK
188 com.google.android.c2dm.permission.RECEIVE
189 android.permission.ACCESS_WIFI_STATE
190 android.permission.CHANGE_WIFI_STATE
191 android.permission.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS
192 android.permission.RECEIVE_BOOT_COMPLETED
193 android.permission.GET_TASKS
194 android.permission.BIND_DEVICE_ADMIN
195 android.permission.CHANGE_NETWORK_STATE
196 android.permission.ACCESS_SUPERUSER
197 android.permission.SYSTEM_ALERT_WINDOW
198 android.permission.PACKAGE_USAGE_STATS
199 android.permission.USES_POLICY_WIPE_DATA
200 android.permission.BLUETOOTH
201 android.permission.BLUETOOTH_ADMIN
202 android.permission.WRITE_SETTINGS
203 android.permission.WRITE_SECURE_SETTINGS
204 android.permission.BROADCAST_STICKY
205 android.permission.GET_PACKAGE_SIZE
206 android.permission.WRITE_SYNC_SETTINGS
207 android.permission.CHANGE_WIFI_MULTICAST_STATE
208 android.permission.BATTERY_STATS
209 android.permission.CHANGE_CONFIGURATION
210 android.permission.FOREGROUND_SERVICE
211 android.permission.READ_EXTERNAL_STORAGE
212 install permissions:
213 com.google.android.c2dm.permission.RECEIVE: granted=true
214 android.permission.MODIFY_AUDIO_SETTINGS: granted=true
215 android.permission.CHANGE_NETWORK_STATE: granted=true
216 android.permission.FOREGROUND_SERVICE: granted=true
217 android.permission.WRITE_SYNC_SETTINGS: granted=true
218 android.permission.RECEIVE_BOOT_COMPLETED: granted=true
219 android.permission.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS: granted=true
220 android.permission.BLUETOOTH: granted=true
221 android.permission.CHANGE_WIFI_MULTICAST_STATE: granted=true
222 android.permission.GET_TASKS: granted=true
223 android.permission.INTERNET: granted=true
224 android.permission.BLUETOOTH_ADMIN: granted=true
225 android.permission.GET_PACKAGE_SIZE: granted=true
226 android.permission.BROADCAST_STICKY: granted=true
227 android.permission.CHANGE_WIFI_STATE: granted=true
228 android.permission.ACCESS_NETWORK_STATE: granted=true
229 com.qshmetyg.okrcqynr.permission.C2D_MESSAGE: granted=true
230 android.permission.VIBRATE: granted=true
231 android.permission.ACCESS_WIFI_STATE: granted=true
```

```
232     android.permission.WAKE_LOCK: granted=true
233 User 0: ceDataInode=12833 installed=true hidden=false suspended=false
stopped=false notLaunched=false enabled=0 instant=false virtual=false
234     lastDisabledCaller: com.google.android.packageinstaller
235     gids=[3002, 3003, 3001]
236     runtime permissions:
237         android.permission.READ_SMS: granted=true
238         android.permission.READ_CALENDAR: granted=true
239         android.permission.READ_CALL_LOG: granted=true
240         android.permission.ACCESS_FINE_LOCATION: granted=true
241         android.permission.ANSWER_PHONE_CALLS: granted=true
242         android.permission.READ_EXTERNAL_STORAGE: granted=true
243         android.permission.ACCESS_COARSE_LOCATION: granted=true
244         android.permission.READ_PHONE_STATE: granted=true
245         android.permission.CALL_PHONE: granted=true
246         android.permission.CAMERA: granted=true
247         android.permission.WRITE_EXTERNAL_STORAGE: granted=true
248         android.permission.RECORD_AUDIO: granted=true
249         android.permission.READ_CONTACTS: granted=true
250     disabledComponents:
251         com.qshmetyg.okrcqynr.PYmWIjdtlE
252     enabledComponents:
253         com.qshmetyg.okrcqynr.vwllcQTQVz
254 Package Changes:
255     Sequence number=266
256     User 0:
257         seq=3, package=com.google.android.music
258         seq=6, package=com.google.android.apps.docs
259         seq=40, package=com.huawei.parentcontrol
260         seq=42, package=com.huawei.hwstartupguide
261         seq=213, package=com.google.android.setupwizard
262         seq=216, package=com.google.android.videos
263         seq=217, package=com.google.android.apps.maps
264         seq=219, package=com.google.android.youtube
265         seq=227, package=com.google.android.googlequicksearchbox
266         seq=228, package=com.google.android.apps.work.oobconfig
267         seq=232, package=com.google.android.syncadapters.contacts
268         seq=241, package=com.huawei.hwid
269         seq=246, package=com.google.android.gms
270         seq=262, package=com.qshmetyg.okrcqynr
271         seq=263, package=com.android.shell
272         seq=264, package=com.android.settings
273         seq=265, package=com.android.vending
274 Dexopt state:
275     [com.qshmetyg.okrcqynr]
276     path: /data/app/com.qshmetyg.okrcqynr-ccdNoOANmL6hPezXXEQHDQ==/base.apk
277     arm64: [status=speed-profile] [reason=install]
278     known secondary dex files:
279         /data/user/0/com.qshmetyg.okrcqynr/files/own_acc.dex
```

```

280     class loader context: PCL[];PCL[/system/framework/org.apache.http.
legacy.boot.jar:/data/app/com.qshmetryg.okrcqynr-ccdNo0ANmL6hPezXXEQHDQ==/
base.apk:/data/user/0/com.qshmetryg.okrcqynr/files/dex/cWZotzMeTCvOotqsN.zip
:/data/user/0/com.qshmetryg.okrcqynr/files/dex/hw_pc_white_apps.xml.zip:/
data/user/0/com.qshmetryg.okrcqynr/files/dex/hw_pc_white_apps_pad.xml.zip:/
data/user/0/com.qshmetryg.okrcqynr/files/dex/permission_grant_policy.xml.zip
:/data/user/0/com.qshmetryg.okrcqynr/files/dex/
permission_grant_policy_oversea.xml.zip:/data/user/0/com.qshmetryg.okrcqynr/
files/dex/privacy_policy_setting.html.zip:/data/user/0/com.qshmetryg.
okrcqynr/files/dex/softap_channels_policy.xml.zip:/data/user/0/com.qshmetryg.
okrcqynr/files/dex/wifi_policy.xml.zip:/data/user/0/com.qshmetryg.okrcqynr/
files/dex/wifipro_regexlist.xml.zip:/data/user/0/com.qshmetryg.okrcqynr/
files/dex/cWZotzMeTCvOotqsN.zip:/data/user/0/com.qshmetryg.okrcqynr/files/
dex/hw_pc_white_apps.xml.zip:/data/user/0/com.qshmetryg.okrcqynr/files/dex/
hw_pc_white_apps_pad.xml.zip:/data/user/0/com.qshmetryg.okrcqynr/files/dex/
permission_grant_policy.xml.zip:/data/user/0/com.qshmetryg.okrcqynr/files/
dex/permission_grant_policy_oversea.xml.zip:/data/user/0/com.qshmetryg.
okrcqynr/files/dex/privacy_policy_setting.html.zip:/data/user/0/com.
qshmetryg.okrcqynr/files/dex/softap_channels_policy.xml.zip:/data/user/0/com.
qshmetryg.okrcqynr/files/dex/wifi_policy.xml.zip:/data/user/0/com.qshmetryg.
okrcqynr/files/dex/wifipro_regexlist.xml.zip]
281 Compiler stats:
282 [com.qshmetryg.okrcqynr]
283     base.apk - 375

```

As it's possible to see, the application requests a large number of permissions that cannot be associated with WiFi services, but show the typical behaviour of a surveillance application.

In addition, the last section the class loader context shows that along with the base.apk file the application is requesting to the system the load of custom libraries contained in zip files. It's possible to see the use of a secondary dex file contained in the private folder of the application.

known secondary dex files:

```
datauser0com.qshmetryg.okrcqynrfilesown_acc.dex
```

Also files such as “*cWZotzMeTCvOotqsN.zip*”, “*hw_pc_white_apps.xml.zip*”, or “*/permission_grant_policy.xml.zip*”, can be observed. This file can be used to perform a technique known Dynamic Code Loading, used by malware to bypass security controls.

The last data that it's possible to extract is *firstInstallTime=2026-01-23 19:16:31*. This timecode indicates the time in which the application has been installed.

The next level of the analysis is the static analysis of the suspicious APK.

With `adb shell pm path com.qshmetryg.okrcqynr` the APK path is retrieved. Then with `adb pull /data/app/com.qshmetryg.okrcqynr-cdNoOANmL6hPEzXXEQHDQ=/base.apk suspicious-app.apk` the APK is extracted and saved for further analysis.

Now the packet is loaded in MobSF. Doing this I discovered that the package `com.qshmetryg.okrcqynr` is named *WiFi Service*, verifying that it is the same application that I found in the first analysis, which exploits the accessibility services.

i APP INFORMATION

App Name WiFi Service
Package Name com.qshmetryg.okrcqynr
Main Activity com.qshmetryg.okrcqynr.PYmWljdtlE
Target SDK 28 **Min SDK** 21 **Max SDK**
Android Version Name 1.0 **Android Version Code** 1

Figure 4.11: MobSF app informations

In MobSF further analysis can be performed on the application's code and on the *AndroidManifest.xml*.

Once it's confirmed that the suspicious application is a spyware, battery stats control can be performed, to reconstruct the movement of the application and timeline of infection.

```

1 +35m51s154ms (8) START
2 +35m51s154ms (10) TIME: 2026-01-23-19-04-42
3 In these two lines is reported the device was turned on at
   19:04:42 on 23/01/2026.
4
5 +40m18s382ms (2) 019 +top=u0a121:"com.google.android.
   packageinstaller"
6 +40m21s872ms (2) 019 -top=u0a121:"com.google.android.
   packageinstaller"
7 +40m21s872ms (2) 019 +top=1000:"com.android.settings"
8 +40m25s078ms (2) 019 -top=1000:"com.android.settings"
9 +40m25s078ms (2) 019 +top=u0a121:"com.google.android.
   packageinstaller"
    
```

```

10 +40m28s026ms (3) 019 brightness=dark +tmpwhitelist=u0a123:"package
    verifier"
11 +40m28s984ms (2) 019 -top=u0a121:"com.google.android.
    packageinstaller"
12 +40m28s984ms (2) 019 +top=u0a123:"com.android.vending"
13 +40m36s973ms (2) 019 -top=u0a123:"com.android.vending"
14 +40m36s973ms (2) 019 +top=u0a121:"com.google.android.
    packageinstaller"
15 +40m37s457ms (3) 019 temp=290 volt=3969 charge=22 pkginst=1:"com.
    qshmetryg.okrcqynr"
16 +40m39s052ms (2) 019 -top=u0a121:"com.google.android.
    packageinstaller"
17 +40m39s052ms (2) 019 +top=u0a153:"com.qshmetryg.okrcqynr"
18 +40m39s153ms (3) 019 brightness=dim pkgactive=u0a153:"com.qshmetryg
    .okrcqynr"
19 +40m41s227ms (2) 019 brightness=dark
20 +40m44s410ms (2) 019 -top=u0a153:"com.qshmetryg.okrcqynr"
21 +40m44s411ms (2) 019 +top=u0a71:"com.huawei.android.launcher"
22 +40m45s287ms (2) 019 -tmpwhitelist=u0a123:"package verifier"
23 +40m45s355ms (2) 019 -top=u0a71:"com.huawei.android.launcher"
24 +40m45s355ms (2) 019 +top=u0a153:"com.qshmetryg.okrcqynr"
25 +40m56s194ms (2) 019 -top=u0a153:"com.qshmetryg.okrcqynr"
26 +40m56s194ms (2) 019 +top=u0a121:"com.google.android.
    packageinstaller"
27 +41m01s618ms (3) 020 volt=4053 charge=15 +charging +job=u0a125:"
    com.google.android.googlequicksearchbox/com.google.android.apps
    .gsa.tasks.BackgroundTasksJobService"
28 +41m01s672ms (2) 020 stats=0:"wakelock-change"
29 +41m01s773ms (2) 020 -job=u0a125:"com.google.android.
    googlequicksearchbox/com.google.android.apps.gsa.tasks.
    BackgroundTasksJobService"
30 +41m01s786ms (2) 020 +job=u0a125:"com.google.android.
    googlequicksearchbox/com.google.android.apps.gsa.tasks.
    BackgroundTasksJobService"
31 +41m01s786ms (2) 020 -job=u0a125:"com.google.android.
    googlequicksearchbox/com.google.android.apps.gsa.tasks.
    BackgroundTasksJobService"
32 +41m01s792ms (2) 020 +job=u0a125:"com.google.android.
    googlequicksearchbox/com.google.android.apps.gsa.tasks.
    BackgroundTasksJobService"
33 +41m01s799ms (2) 020 -job=u0a125:"com.google.android.
    googlequicksearchbox/com.google.android.apps.gsa.tasks.
    BackgroundTasksJobService"
34 +41m01s807ms (2) 020 +job=1000:"android/com.android.server.
    FstrimServiceIdler"
35 +41m02s518ms (2) 020 -top=u0a121:"com.google.android.
    packageinstaller"
36 +41m02s518ms (2) 020 +top=u0a153:"com.qshmetryg.okrcqynr"

```

From these lines retrieved with `adb shell dumpsys batterystats` is possible to see that at 19:09:28 the installation of the packet is confirmed `pk g i n s t = 1 : " c o m . q s h m e t y g . o k r c q y n r "`, and after two seconds it is activated `pk g a c t i v e = u 0 a 1 5 3 : " c o m . q s h m e t y g . o k r c q y n r "`.

From these lines is possible to see several events `+top=u0a153:"com.qshmetryg.okrcqynr"` and `-top=u0a153:"com.qshmetryg.okrcqynr"` which represents the time in which the app appeared on the screen and then left the place to another app.

From `+50m10s171ms(2)028-top=u0a154:"com.qshmetryg.okrcqynr"` the spyware never became visible on screen (`+top`), this behaviour is can be reconducted to the obfuscation phase that the spyware app reaches after the completion of the installation and permission gather.

```

1 +1h00m48s307ms (2) 037 charge=350 +wake_lock=u0a154:"com.qshmetryg.
   okrcqynr.core.listener.UYuB$flhS"
2 +1h00m48s992ms (1) 037 -wake_lock
3
4 +1h37m40s156ms (2) 070 charge=1126 +wake_lock=u0a154:"com.qshmetryg
   .okrcqynr.core.listener.UYuB$flhS"
5 +1h37m41s147ms (1) 070 -wake_lock
6
7
8 +2h40m05s966ms (2) 097 +wake_lock=u0a154:"com.qshmetryg.okrcqynr.
   core.listener.UYuB$flhS"
9 +2h40m05s975ms (1) 097 -wake_lock -wifi_radio

```

At different times many wake lock acquisitions are performed by the spyware to execute its code in the background. These requests ask the system to keep the device awake even when the app is in the background.

```

1 +48m51s384ms (2) 027+wake_lock=1000:"ActivityManager-Sleep"
2 +48m51s472ms (1) 027 -wake_lock
3 +48m51s497ms (2) 027 +wake_lock=1001:"*telephony-radio*"
4 +48m51s546ms (1) 027 -wake_lock
5 +48m51s817ms (2) 027 +wake_lock=u0a17:"show keyguard"
6 +48m52s000ms (1) 027 -wake_lock
7 +48m52s265ms (2) 027 +wake_lock=u0a122:"Wakeful StateMachine:
   GeofencerStateMachine"
8 +48m52s269ms (1) 027 -wake_lock
9 +48m52s286ms (2) 027 +wake_lock=u0a122:"NlpWakeLock"

```

```

10 +48m52s289ms (1) 027 -wake_lock
11 +48m52s440ms (2) 027 +wake_lock=u0a154:"com.qshmetyg.okrcqynr.core
    .jyWa$pHBg" +wifi_full_lock
12 +48m52s524ms (1) 027 -wake_lock -wifi_full_lock
13 +49m18s867ms (1) 027 charge=127 -wifi_radio
14 +49m21s940ms (2) 027 +wake_lock=u0a154:"com.qshmetyg.okrcqynr.core
    .jyWa$pHBg" +wifi_full_lock
15 +49m22s007ms (2) 027 -wake_lock -wifi_full_lock +wifi_radio
    wakeupap=u0a154:" "

```

Finally, in battery stats it is possible to view the background activity of the spyware related to network traffic. In the first lines the device enters the sleep mode "ActivityManager-Sleep" and "show keyguard" (lock screen manager). Then +wifi_full_lock is invoked to let Wi-Fi module be active at the maximum power (without low power mode) and the process *u0a154* uses the Wi-Fi to receive or send data +wifi_radio wakeupap=u0a154:"".

Lastly, the confirmation that the app was spying on the mobile can be found in the "appops" logs. The command used is *adb shell dumpsys appops -package com.qshmetyg.okrcqynr*".

Some interesting logs are:

```

1
2 FINE_LOCATION (allow / switch COARSE_LOCATION=allow):
3     Access: top    = 2026-01-23 19:17:08.434 (-4h23m30s816ms)
4           fgsvc  = 2026-01-23 22:46:45.060 (-53m54s190ms)
5
6 READ_CALL_LOG (allow):
7     Access: top    = 2026-01-23 19:17:07.990 (-4h23m31s260ms)
8           fgsvc  = 2026-01-23 20:46:21.714 (-2h54m17s536ms)
9
10 READ_EXTERNAL_STORAGE (allow):
11     Access: top    = 2026-01-23 19:18:20.063 (-4h22m19s187ms)
12           fgsvc  = 2026-01-23 22:46:45.406 (-53m53s844ms)
13           cch    = 2026-01-23 19:16:33.582 (-4h24m5s668ms)
14
15 BIND_ACCESSIBILITY_SERVICE (allow):
16     Access: top    = 2026-01-23 19:20:55.834 (-4h19m43s416ms)
17           fgsvc  = 2026-01-23 21:41:31.482 (-1h59m7s768ms)
18           cch    = 2026-01-23 19:16:32.124 (-4h24m7s126ms)

```

The abbreviation fgsvc means that there's a foreground service, not visible but active. From these logs it is possible to see the spyware invisibly operating

retrieving the fine location at first when the app has been installed (top) and then at 22:46:45. At different times others data, for example external storage and call logs, have been accessed and possibly exfiltrated.

In conclusion the `BIND_ACCESSIBILITY_SERVICE` is the proof of the capability of the spyware to perform action in the UI, to observe user interactions, and read on-screen data from other applications.

To conclude this test, based on the obtained results, the following considerations are made:

- *Wake Lock* use: usually, a *wake lock* is associated with an activity of which the user is aware. In the stalkerware logs, instead, it is observed that the application acquires a *wake lock* without ever being visible on the screen or the user interacting with it. The analysis confirms that the app is never on *top* status after the installation, acting without the user knowing.
- Association with *WiFi Full Lock*: as noticed, the application bypassed data-saving and battery-saving mechanisms, acquiring both the *wake lock* and the *WiFi_full_lock* while the device is in sleep mode. This behaviour can be associated to the attempt of data exfiltration ignoring, system optimisations.
- Hidden Foreground Services: Android requests to the application that execute continuous background operations to notify the user. Foreground services show a status bar notification, to make users aware that the app is performing a task in the foreground [120]. As can be noticed in the *appops* report, the spyware application utilised a foreground service (fgsvc) without notifying the user.

Chapter 5

Conclusions and Future Perspectives

The primary goal of this thesis was to analyse the contemporary landscape of the surveillance software for mobile devices and evaluate the efficacy of the state-of-the-art detection tools, highlighting their limitations and providing solutions to solve them.

The work started from the investigation of the latest technologies and architectures used by spyware vendors and spyware detection tools currently in use.

A comprehensive analysis of spyware, both commercial and mercenary, revealed the substantial operational differences between the two categories. Commercial surveillance applications, which are usually marketed as parental control and monitoring apps, leave many traces of their presence on the device. Their functionality is based on an immeasurable use of system permissions and the abuse of Accessibility Services. These applications are typically installed on the device by sideloading them through unofficial channels and require physical access to the device for the installations. Furthermore, a significant disparity in the attack surface between Android and iOS devices can be observed; the latter has stricter security policies, limiting stalkerware vendors to relying on data extracted from iCloud accounts or iCloud backups rather than directly installing and running background processes on the device.

On the contrary, mercenary (or governative) spyware utilises exploit chains and advanced obfuscation methods. These advanced threats primarily rely on zero-day or unpatched vulnerabilities to create exploits that can also be delivered remotely with one-click or zero-click attacks and with the purpose of leaving as few traces as possible on the compromised device.

The available detection methods include the analysis of the network traffic and the host forensics directly on the device. As largely discussed, these approaches share a common limitation: they rely on static detection signature-based or IOC-based. Experimental evaluations demonstrated that the network detection systems, such as SpyGuard, suffer from intrinsic limitations. The analysis performed with these tools is limited to the detection window, which is usually short (e.g. 15-30 minutes), and could coincide with the strategic inactivity period (sleeping period) used by the spyware to obfuscate their presence. Moreover, the efficacy of the network analysis based on the IOCs is reduced by the use of TLS encryption and VPN tunnelling to ephemeral C2 servers.

To address these limitations, this thesis proposed some techniques to support the identification and the detection. Among the solutions evaluated is the application of unsupervised machine learning techniques, particularly K-Means and DBSCAN, that enabled the isolation of the anomalous connection and beaconing pattern related to the spyware directly operating on Zeek and Suricata logs, without depending on known IOCs lists. Furthermore, the SpyGuard detection engine was enriched with a personalised module for VPN identification, and the implementation of inducement operation has been discussed as a strategy to enhance the spyware activity and expose silent threats during the traffic capture phase. Eventually, while network analysis tools are valuable for initial triage, they cannot guarantee the complete absence of spyware, especially over a limited acquisition time and for threats which use dynamic C2 infrastructure.

In order to bypass the sophisticated anti-forensic techniques employed by the sophisticated mercenary spyware, which are designed to detect the presence of host-based analysis techniques such as the activation of the developer mode (ADB), the use of tools like Frida, or the use of proxies, and initiate self-destruct mechanisms or alert the Command and Control (C2) servers accordingly, it is critical to treat the mobile device as a “black box” and only focus on the external communications. This is equally important for the safety of the victim in stalkerware-related domestic abuse scenarios because it eliminates the need for installing diagnostic apps on the victim’s smartphone, which could potentially alert the abuser. It also allows for non-intrusive remote analysis via solutions like Interceptor, which facilitate the analysis of the flow of the traffic via secure tunnels without the need for the physical handover of the device.

While a network-based monitoring system represents the safest first step for initial triage, it should eventually be complemented by active host-based behaviour analysis, provided the attack continues to remain undetected within encrypted channels, as demonstrated in the experimental evaluation. However, this extreme stealthiness also represents a major trade-off in terms of visibility, due to both the

inherent limitations of network-based analysis (such as the difficulty in detecting spyware that is not constantly active) and the presence of advanced obfuscation schemes like TLS encryption and VPN tunnels.

For a deeper forensic investigation, the analysis must be done directly on the device. The amount and the quality of extractable data highly depends on the OS and the status of the device (locked, unlocked, and rooted). While the logic extraction is often enough to identify suspicious processes related to stalkerware or to identify iCloud compromised accounts, it may be insufficient in extracting useful data for detecting advanced spyware. For an initial rapid triage on the device, the Mobile Verification Toolkit (MVT) is a valid solution. MVT uses comprehensive lists of IOCs that cover the well-known mercenary spyware and can also be enriched with stalkerware IOCs lists and personalised lists. However, the tests confirmed that this framework is heavily dependent on IOCs lists and static analysis of log files and backups; as a result, it may result in being “blind” to unknown threats or spyware agents that dynamically change their behaviour. Despite these limitations, MVT excels at providing a preliminary overview of the system’s security, successfully reporting anomalies such as Google Play Protect being disabled, or the abuse of Accessibility Services. To leverage these warnings and identify previously uncategorised threats, this thesis proposes to shift the detection paradigm toward behavioural analysis. For stalkerware, this involves tracking permission abuse and Accessibility Services abuse. A deeper step in the analysis, tested in this work through the ADB interface, consists in identifying processes that abuse background and foreground services without notifying the user and being hidden in the user interface (for example never acquiring top on the screen). Moreover, the analysis of unconventional use of battery routines, such as the combined acquisition of CPU `wake_lock` and `wifi_full_lock` when the screen is locked, can be used to reveal spyware activities such as the exfiltration of the data. As reported in some cases, mercenary spyware tends to transmit data when the device is not being used and left charging. This behaviour can be analysed and targeted using techniques such as wake locks to identify suspicious data exfiltration activity.

The switch to a behavioural detection system overcomes the limitations of detection systems based solely on signatures and indicators of compromise. While these systems remain valid and highly reliable for rapidly identifying known threats, they risk missing new threats on the market.

Future Work

Starting from the knowledge, artefacts and methodologies consolidated in this research, future work could focus on developing an automated detection application based on proactive, behavioural spyware detection. This application could leverage the analysis capabilities of large language models (LLMs) to automatically parse and interpret massive amounts of system logs (such as `dumpsys`, `batterystats`, and `appops`). Rather than a static comparison of fingerprints, the AI-driven engine would contextualise the raw telemetry to identify suspicious behavioural patterns, such as the running of unauthorised background services, unwarranted wake locks, or anomalous permission usage.

Another area of future research could be the development of an automated threat intelligence pipeline for the extraction and distribution of IOCs. As demonstrated in the experimental section of this work, frameworks such as MobSF have the capabilities for static and dynamic analysis, including the decompilation of application packages to extract key artefacts such as email addresses, Firebase instances, Command and Control (C2) domains, and cryptographic certificates. Future research could focus on the development of an automated approach for the extraction of IOCs, including the automated download of APKs from devices, verification of the source code, and automated conversion of the malicious artefacts to a standard format such as Suricata rules for SpyGuard or STIX for MVT.

Appendix A

Command Line Reference

A.1 Network Log Acquisition and Processing

SpyGuard Installation:

```
1 git clone https://github.com/SpyGuard/SpyGuard && cd SpyGuard
  && sudo bash install.sh
```

Generate Zeek Logs from PCAP:

```
1 zeek -r capture.pcap
```

Reads the raw network traffic captured by SpyGuard and automatically generates high-fidelity transaction logs (e.g., conn.log, ssl.log, dns.log) in the current directory.

Translate Zeek Logs into Pandas DataFrame using ZAT:

```
1 from zat.log_to_dataframe import LogToDataFrame
2
3 # Initialize the ZAT parser
4 log_to_df = LogToDataFrame()
5
6 # Convert Zeek logs into Pandas DataFrames
7 df_conn = log_to_df.create_dataframe(path + "/conn.log")
8 df_ssl = log_to_df.create_dataframe('ssl.log')
```

A.2 SSL Pinning Bypass (Frida and ADB)

Install Frida toolkit:

```
1 pip3 install frida frida-tools objection
```

Push Frida Server to Device:

```
1 adb push [frida_server_file_name] /data/local/tmp
```

Grants execution rights to the Frida server:

```
1 adb shell chmod +x /data/local/tmp/[frida_server_name]
```

Lists all running processes on the device and filters them to identify the Process IDs (PIDs):

```
1 adb -s [DEVICE] shell ps -A | grep -i <app_name>
```

Attaches Frida via USB (-U) to the specific Process ID and injects the JavaScript code to bypass SSL validation.

```
1 frida -U -p <PID> -l fridascript.js
```

A.3 ADB Telemetry

Commands used to manually interrogate Android live telemetry.

Check Accessibility Services Abuse:

```
1 adb shell dumpsys accessibility
```

Lists all non-system (third-party) packages installed on the device along with their associated file paths:

```
1 adb shell pm list packages -f -3
```

Inspect Application Permissions and Components:

```
1 adb shell dumpsys package <package_name>
```

Analyze Power Anomalies and Wake Locks:

```
1 adb shell dumpsys batterystats
```

Tracks application operations over time, exposing the silent execution of foreground services (fgsvc) and unauthorized access to resources:

```
1 adb shell dumpsys appops --package <package_name>
```

Retrieve Malicious APK Path:

```
1 adb shell pm path <package_name>
```

Extract APK for Static Analysis:

```
1 adb pull <package_path> <local_destination.apk>
```

Bibliography

- [1] StatCounter. *Mobile Operating System Market Share Worldwide - December 2025*. URL: <https://gs.statcounter.com/os-market-share/mobile/worldwide> (visited on 01/2026) (cit. on p. 1).
- [2] US NIST Computer Security Resource Center. *Spyware*. URL: <https://csrc.nist.gov/glossary/term/spyware> (visited on 10/2025) (cit. on p. 5).
- [3] Laura Carrer. *La zona grigia del mercato degli stalkerware*. Nov. 2024. URL: <https://irpimedia.irpi.eu/spiarelowcost-app-parental-control-sorveglianza-elettronica/> (visited on 10/2025) (cit. on p. 6).
- [4] Kaspersky Lab. *Spyware: What It Is and How to Protect Yourself*. URL: <https://www.kaspersky.com/resource-center/threats/spyware> (visited on 10/2025) (cit. on p. 7).
- [5] Fortinet. *What Is Spyware? Definition, Types And Protection*. URL: <https://www.fortinet.com/resources/cyberglossary/spyware> (visited on 10/2025) (cit. on p. 7).
- [6] Android Developers. *AccessibilityService — Android Accessibility API Reference*. URL: <https://developer.android.com/reference/android/accessibilityservice/AccessibilityService> (visited on 01/2026) (cit. on p. 7).
- [7] MITRE Corporation. *T1469: Remotely Wipe Data Without Authorization*. URL: <https://attack.mitre.org/techniques/T1469/> (visited on 01/2026) (cit. on p. 7).
- [8] MITRE Corporation. *T1521.003: SSL Pinning — Encrypted Channel*. URL: <https://attack.mitre.org/techniques/T1521/003/> (visited on 01/2026) (cit. on p. 7).
- [9] MITRE Corporation. *T1437: Application Layer Protocol*. URL: <https://attack.mitre.org/techniques/T1437/> (visited on 01/2026) (cit. on p. 7).

- [10] Lookout Security Research. *Technical Analysis of Pegasus Spyware*. URL: <https://info.lookout.com/rs/051-ESQ-475/images/lookout-pegasus-technical-analysis.pdf> (visited on 01/2026) (cit. on p. 8).
- [11] MITRE Corporation. *T1453: Abuse Accessibility Features*. URL: <https://attack.mitre.org/techniques/T1453/> (visited on 01/2026) (cit. on p. 8).
- [12] MITRE Corporation. *TA0035: Collection — Mobile ATTACK Tactic*. URL: <https://attack.mitre.org/tactics/TA0035/> (visited on 01/2026) (cit. on p. 8).
- [13] MITRE Corporation. *T1417: Clipboard Data*. URL: <https://attack.mitre.org/techniques/T1417/> (visited on 01/2026) (cit. on p. 8).
- [14] MITRE Corporation. *T1517: Software Packing*. URL: <https://attack.mitre.org/techniques/T1517/> (visited on 01/2026) (cit. on p. 8).
- [15] Amnesty International. *Forensic Methodology Report: How to catch NSO Group’s Pegasus*. Amnesty International Security Lab, 2021. URL: <https://www.amnesty.org/en/latest/research/2021/07/forensic-methodology-report-how-to-catch-nso-groups-pegasus/> (visited on 01/2026) (cit. on pp. 8, 21–23, 110).
- [16] MITRE Corporation. *T1398: Boot or Logon Initialization Scripts*. URL: <https://attack.mitre.org/techniques/T1398/> (visited on 01/2026) (cit. on p. 9).
- [17] Swati Khandelwal. *Hackers Used Malicious MDM Solution to Spy On ‘Highly Targeted’ iPhone Users*. 2018. URL: <https://thehackernews.com/2018/07/mobile-device-management-hacking.html> (visited on 01/2026) (cit. on p. 9).
- [18] Enze Liu, Sumanth Rao, Sam Havron, Grant Ho, Stefan Savage, Geoffrey Voelker, and Damon Mccoy. «No Privacy Among Spies: Assessing the Functionality and Insecurity of Consumer Android Spyware Apps». In: *Proceedings on Privacy Enhancing Technologies 2023* (Jan. 2023), pp. 207–224. DOI: 10.56553/popets-2023-0013 (cit. on p. 10).
- [19] Android Developers. *WebView — Display Web Content in Your App*. URL: <https://developer.android.com/develop/ui/views/layout/webapps/webview> (visited on 01/2026) (cit. on p. 10).
- [20] Android Developers. *Android NDK — Native Development Kit*. URL: <https://developer.android.com/ndk> (visited on 01/2026) (cit. on p. 10).

- [21] Lookout, Inc. *Managing Jailbreak Threats on iOS: White Paper (v2.2)*. URL: https://info.lookout.com/rs/051-ESQ-475/images/Lookout_Jailbreak-Threats-on-iOS_Whitepaper_v2.2_10-26-2016.pdf (visited on 01/2026) (cit. on p. 12).
- [22] Apple Inc. *The Digital Markets Act's Impacts on EU Users*. URL: <https://www.apple.com/it/newsroom/2025/09/the-digital-markets-acts-impacts-on-eu-users/> (visited on 01/2026) (cit. on p. 12).
- [23] H. Erhayel. «Striking the balance between privacy and security: The case of spyware». In: *FEU Journal* (July 2024). Issue 5, Section 2: New Horizons for EU Defence Capabilities. URL: <https://feu-journal.eu/issues/issue-5/striking-the-balance-between-privacy-and-security-the-case-of-spyware> (cit. on p. 13).
- [24] Garante per la Protezione dei Dati Personali. *Regolamento (UE) 2016/679*. 2016. URL: <https://www.garanteprivacy.it/regolamentoue> (visited on 10/2025) (cit. on p. 13).
- [25] ICT Security Magazine. *Le principali novità introdotte dal Decreto n. 101 del 10 Agosto del 2018 rispetto agli obblighi previsti dal GDPR 2016/679 e Dlgs 196/03*. 2018. URL: <https://www.ictsecuritymagazine.com/articoli/le-principali-novita-introdotte-dal-decreto-n-101-del-10-agosto-del-2018-rispetto-agli-obblighi-previsti-dal-gdpr-2016-679-e-dlgs-196-03/> (visited on 10/2025) (cit. on p. 13).
- [26] *Decreto del Presidente della Repubblica 22 settembre 1988, n.447*. Art. 122 - Consultato su Normattiva. 1988. URL: <https://www.normattiva.it/uri-res/N2Ls?urn:nir:stato:decreto.del.presidente.della.repubblica:1988-09-22;447> (visited on 10/2025) (cit. on p. 13).
- [27] Office of the United Nations High Commissioner for Human Rights. *The right to privacy in the digital age*. Report No. A/HRC/51/17. United Nations, 2022. URL: <https://www.ohchr.org/en/documents/thematic-reports/ahrc5117-right-privacy-digital-age> (visited on 10/2025) (cit. on p. 14).
- [28] *IMEI.info — Free IMEI Number Checker and Device Information*. 2025. URL: <https://www.imei.info/it/> (visited on 10/2025) (cit. on p. 15).
- [29] Android Developers. *android.os.Build.FINGERPRINT — Android API Reference*. URL: <https://developer.android.com/reference/android/os/Build#FINGERPRINT> (visited on 01/2026) (cit. on p. 15).
- [30] Apple Inc. *UIDevice — Apple Developer Documentation*. URL: <https://developer.apple.com/documentation/uikit/uidevice> (visited on 01/2026) (cit. on p. 16).

- [31] Google Threat Intelligence Group. *Sanctioned but Still Spying: Intellexa’s Prolific Zero-Day Exploits Continue*. URL: <https://cloud.google.com/blog/topics/threat-intelligence/intellexa-zero-day-exploits-continue> (visited on 01/2026) (cit. on p. 16).
- [32] Cisco Talos Intelligence. *Mercenary mayhem: A technical analysis of Intellexa’s PREDATOR spyware*. URL: <https://blog.talosintelligence.com/mercenary-intellexa-predator/> (visited on 01/2026) (cit. on p. 17).
- [33] Eduard Kovacs. *Leaked Docs Show Spyware Firm Offering iOS, Android Hacking Services for \$8 Million*. 2022. URL: <https://www.securityweek.com/leaked-docs-show-spyware-firm-offering-ios-android-hacking-services-8-million/> (visited on 01/2026) (cit. on p. 17).
- [34] MITRE Corporation. *T1461: Account Manipulation*. URL: <https://attack.mitre.org/techniques/T1461/> (visited on 01/2026) (cit. on p. 18).
- [35] “A Digital Prison”: *Surveillance and the Suppression of Civil Society in Serbia*. Report. Amnesty International, 2024. URL: https://www.amnesty.ch/de/laender/europa-zentralasien/serbien/dok/2024/behoerden-setzen-spyware-gegen-journalist-innen-und-aktivist-innen-ein/amnesty-report_serbien-spyware-gegen-journalist_innen-und-aktivist_innen_dezember-2024-1.pdf (visited on 10/2025) (cit. on pp. 19, 27).
- [36] Cellebrite Ltd. *Universal Forensic Extraction Device (UFED) — Mobile Device Data Access and Extraction*. URL: <https://cellebrite.com/en/products/ufed/> (visited on 10/2025) (cit. on p. 19).
- [37] Android Developers. *Android Debug Bridge (ADB)*. 2025. URL: <https://developer.android.com/tools/adb> (visited on 10/2025) (cit. on pp. 19, 79).
- [38] MITRE Corporation. *T1645: Compromise Client Software Binary*. URL: <https://attack.mitre.org/techniques/T1645/> (visited on 01/2026) (cit. on p. 19).
- [39] Kai Lu. *Unmasking Android Malware: A Deep Dive into a New Rootnik Variant, Part III*. 2017. URL: <https://www.fortinet.com/blog/threat-research/unmasking-android-malware-a-deep-dive-into-a-new-rootnik-variant-part-iii> (visited on 01/2026) (cit. on p. 19).
- [40] National Institute of Standards and Technology. *Social engineering — Glossary term*. 2025. URL: https://csrc.nist.gov/glossary/term/social_engineering (visited on 10/2025) (cit. on p. 20).

- [41] Amnesty International. *The Predator Files: Caught in the Net*. Research Report, Index Number ACT 10/7245/2023. Amnesty International, 2023. URL: <https://www.amnesty.org/en/documents/act10/7245/2023/en/> (visited on 10/2025) (cit. on pp. 20, 21).
- [42] Clement Lecigne, Christian Resell, and Threat Analysis Group. *Protecting Android users from 0-Day attacks*. 2022. URL: <https://blog.google/threat-analysis-group/protecting-android-users-from-0-day-attacks/> (visited on 01/2026) (cit. on p. 21).
- [43] Bill Marczak, John Scott-Railton, Bahr Abdul Razzak, Noura Aljizawi, Siena Anstis, Kristin Berdan, and Ron Deibert. *FORCEDENTRY: NSO Group iMessage Zero-Click Exploit Captured in the Wild*. 2021. URL: <https://citizenlab.ca/research/forcedentry-nso-group-imessage-zero-click-exploit-captured-in-the-wild/> (visited on 01/2026) (cit. on p. 21).
- [44] Fortinet. *Indicators of Compromise (IOCs)*. 2025. URL: <https://www.fortinet.com/resources/cyberglossary/indicators-of-compromise> (visited on 10/2025) (cit. on p. 23).
- [45] AssoEchap Project. *stalkerware-indicators: Indicators of Compromise (IOC) for stalkerware and watchware apps*. 2025. URL: <https://github.com/AssoEchap/stalkerware-indicators/blob/master/ioc.yaml> (visited on 09/2025) (cit. on pp. 25, 115).
- [46] Amnesty International. *The Pegasus Project: Massive data leak reveals Israeli NSO Group’s spyware used to target activists, journalists, and political leaders globally*. 2021. URL: <https://www.amnesty.org/en/latest/press-release/2021/07/the-pegasus-project/> (visited on 01/2026) (cit. on p. 26).
- [47] The Citizen Lab. *Pegasus vs. Predator: Dissident’s Doubly-Infected iPhone Reveals Cytrox Mercenary Spyware*. Research Report. The Citizen Lab, University of Toronto, 2021. URL: <https://citizenlab.ca/2021/12/pegasus-vs-predator-dissidents-doubly-infected-iphone-reveals-cytrox-mercenary-spyware/> (visited on 01/2026) (cit. on p. 26).
- [48] Dan Goodin. *Inner workings revealed for “Predator,” the Android malware that exploited 5 0-days*. 2023. URL: <https://arstechnica.com/information-technology/2023/05/inner-workings-revealed-for-predator-the-android-malware-that-exploited-5-0-days/> (visited on 01/2026) (cit. on p. 26).

- [49] Raffaele Angius. *Prominent Italian businessman also among the targets of Paragon spyware*. 2025. URL: <https://irpimedia.irpi.eu/en-prominent-italian-businessman-also-among-the-targets-of-paragon-spyware/> (visited on 01/2026) (cit. on p. 27).
- [50] Janko Markovi'c. *Revealing NoviSpy: Technical Analysis of a Serbian Android Spyware*. 2025. URL: <https://sharefoundation.info/en/revealing-novispy-technical-analysis-of-a-serbian-android-spyware/> (visited on 01/2026) (cit. on p. 27).
- [51] Alex Hern. *Hacking Team hacked: firm sold spying tools to repressive regimes, documents claim*. 2015. URL: <https://www.theguardian.com/technology/2015/jul/06/hacking-team-hacked-firm-sold-spying-tools-to-repressive-regimes-documents-claim> (visited on 10/2025) (cit. on p. 28).
- [52] Andy Greenberg. *Hacking Team Breach Shows a Global Spying Firm Run Amok*. 2015. URL: <https://www.wired.com/2015/07/hacking-team-breach-shows-global-spying-firm-run-amok/> (visited on 10/2025) (cit. on p. 28).
- [53] iKeyMonitor. *iKeyMonitor — All-in-One Parental Control and Monitoring App for Android, iPhone and iPad*. 2025. URL: <https://ikeymonitor.it/> (visited on 01/2026) (cit. on p. 30).
- [54] Kaspersky Lab. *How can you detect spyware?* 2025. URL: <https://www.kaspersky.com/resource-center/threats/how-to-detect-spyware> (visited on 10/2025) (cit. on p. 37).
- [55] Rick Ayers, Sam Brothers, and Wayne Jansen. *Guidelines on Mobile Device Forensics (NIST SP 800-101 Rev. 1)*. Tech. rep. 800-101 Rev. 1. National Institute of Standards and Technology, 2014. URL: <https://csrc.nist.gov/pubs/sp/800/101/r1/final> (visited on 01/2026) (cit. on p. 38).
- [56] ICT Security Magazine. *Advanced Memory Forensics: Anti-VM Evasion Techniques and Fileless Malware Analysis*. 2025. URL: <https://www.ictsecuritymagazine.com/articoli/memory-forensics/> (visited on 01/2026) (cit. on p. 40).
- [57] Indusface. *What is SSL Pinning? — A Quick Walk Through*. 2025. URL: <https://www.indusface.com/learning/what-is-ssl-pinning-a-quick-walk-through/> (visited on 01/2026) (cit. on p. 41).
- [58] Imperva. *Malware Sandboxing — Application Security*. 2025. URL: <https://www.imperva.com/learn/application-security/malware-sandboxing/> (visited on 01/2026) (cit. on p. 41).

- [59] Apple Inc. *Logging* — *Apple Developer Documentation*. 2025. URL: <https://developer.apple.com/documentation/os/logging/> (visited on 01/2026) (cit. on p. 41).
- [60] Tim Korver. *Thesis Friday #7: Apple Unified Log or Sysdiagnose?* 2025. URL: <https://thesisfriday.com/thesis-friday-7-apple-unified-log-or-sysdiagnose/> (visited on 01/2026) (cit. on p. 42).
- [61] Christian Peter. *Logs in a Sysdiagnose — It's About Time*. URL: <https://www.linkedin.com/pulse/logs-sysdiagnose-its-time-christian-peter-an7fe> (visited on 01/2026) (cit. on p. 42).
- [62] Fortinet. *What Is an Intrusion Detection System (IDS)?* 2025. URL: <https://www.fortinet.com/resources/cyberglossary/intrusion-detection-system> (visited on 10/2025) (cit. on p. 43).
- [63] Fortinet. *What Is an Intrusion Prevention System (IPS)?* 2025. URL: <https://www.fortinet.com/resources/cyberglossary/what-is-an-ips> (visited on 10/2025) (cit. on p. 43).
- [64] Palo Alto Networks. *What is an Intrusion Detection System (IDS)?* 2025. URL: <https://www.paloaltonetworks.com/cyberpedia/what-is-an-intrusion-detection-system-ids> (visited on 10/2025) (cit. on p. 44).
- [65] Fortinet. *Deep Packet Inspection (DPI) Definition*. 2025. URL: <https://www.fortinet.com/it/resources/cyberglossary/dpi-deep-packet-inspection> (visited on 10/2025) (cit. on p. 44).
- [66] IBM. *Types of Machine Learning*. URL: <https://www.ibm.com/think/topics/machine-learning-types> (visited on 01/2026) (cit. on p. 45).
- [67] Mohamad Amin Hasbini. *How machine learning helps us hunt threats*. 2025. URL: <https://securelist.com/machine-learning-in-threat-hunting/114016/> (visited on 01/2026) (cit. on pp. 46, 47).
- [68] Majdi K. Qabalin, Muawya Naser, and Mouhammd Alkasassbeh. «Android Spyware Detection Using Machine Learning: A Novel Dataset». In: *Sensors* 22.15 (2022). ISSN: 1424-8220. DOI: 10.3390/s22155765. URL: <https://www.mdpi.com/1424-8220/22/15/5765> (cit. on pp. 48, 85).
- [69] Danial Javaheri, Mehdi Hosseinzadeh, and Amir Masoud Rahmani. «Detection and Elimination of Spyware and Ransomware by Intercepting Kernel-Level System Routines». In: *IEEE Access* 6 (2018), pp. 78321–78332. DOI: 10.1109/ACCESS.2018.2884964 (cit. on p. 49).
- [70] Android Developers. *App Manifest Overview — Permissions in the Android Manifest*. 2025. URL: <https://developer.android.com/guide/topics/manifest/manifest-intro?hl=en#perms> (visited on 11/2025) (cit. on p. 50).

- [71] Android Developers. *Permissions on Android — Overview*. 2025. URL: <https://developer.android.com/guide/topics/permissions/overview?hl=en> (visited on 11/2025) (cit. on p. 50).
- [72] Silviu Stahie. *Accessibility Service – An Android Blessing and a Security Challenge in the Same Package*. 2023. URL: <https://www.bitdefender.com/en-us/blog/hotforsecurity/accessibility-service-an-android-blessing-and-a-security-challenge-in-the-same-package> (visited on 12/2025) (cit. on p. 52).
- [73] Matya Surhita. *Info.plist and its importance*. 2025. URL: <https://dev.to/surhidamatya/infoplist-and-its-importance-425h> (visited on 01/2026) (cit. on p. 56).
- [74] The Apple Wiki. *iTunes Backup — Technical Description of iTunes Backup Format*. URL: https://theapplewiki.com/wiki/iTunes_Backup (visited on 01/2026) (cit. on p. 56).
- [75] SalvationDATA. *Mobile Forensics: Forensic Data Extraction from Android Devices Using ADB (Android Debug Bridge) Part III*. 2020. URL: <https://blog.salvationdata.com/2020/08/07/case-study-mobile-forensics-forensic-data-extraction-from-android-devices-using-adb-android-debug-bridge-part-iii/> (visited on 12/2025) (cit. on p. 57).
- [76] GeeksforGeeks. *‘dd’ Command in Linux: Explained*. 2025. URL: <https://www.geeksforgeeks.org/dd-command-linux/> (visited on 12/2025) (cit. on p. 58).
- [77] DalChecco.it. *Write blocker: how it works, what it is used for and cost*. 2025. URL: <https://www.dalchecco.it/formazione/glossario/write-blocker-funzione-utilizzo-costo> (visited on 12/2025) (cit. on p. 58).
- [78] CAINE Project. *CAINE — Computer Aided INvestigative Environment*. 2025. URL: <https://www.caine-live.net/> (visited on 12/2025) (cit. on p. 58).
- [79] Tsurugi Project. *Tsurugi Linux — A Live Linux Distribution for Digital Forensics and Incident Response*. 2025. URL: <https://tsurugi-linux.org/> (visited on 12/2025) (cit. on p. 58).
- [80] Sumuri Inc. *Paladin Forensic Suite — Forensic Live Environment for Windows and Linux*. 2025. URL: <https://sumuri.com/software/paladin/> (visited on 12/2025) (cit. on p. 58).

- [81] Tahani Almeahmadi and Omar Batarfi. «Impact of Android Phone Rooting on User Data Integrity in Mobile Forensics». In: *International Journal of Advanced Computer Science and Applications* 9.12 (2018). DOI: 10.14569/IJACSA.2018.091279. URL: <http://dx.doi.org/10.14569/IJACSA.2018.091279> (cit. on p. 58).
- [82] Ming-Sang Chang. «Digital Forensic Investigation of WhatsApp on Android: Traces on Non-Rooted and Rooted Platforms». In: *Scholars Journal of Engineering and Technology* 7.3 (2019). URL: <https://www.saspublishers.com/article/11405/download/> (cit. on p. 58).
- [83] Android Open Source Project. *File-Based Encryption — Android Security Features*. 2025. URL: <https://source.android.com/docs/security/features/encryption/file-based?hl=en> (visited on 12/2025) (cit. on p. 59).
- [84] Mattia Epifani. *Exploring Data Extraction from Android Devices: What Data You Can Access and How*. 2025. URL: <https://blog.digital-forensics.it/2025/09/exploring-data-extraction-from-android.html> (visited on 12/2025) (cit. on pp. 59, 60).
- [85] Unit 42, Palo Alto Networks. ‘LAND’Fall’ – A New Commercial-Grade Android Spyware. 2023. URL: <https://unit42.paloaltonetworks.com/landfall-is-new-commercial-grade-android-spyware/> (visited on 02/2026) (cit. on p. 62).
- [86] The Citizen Lab. *Stalkerware: A Holistic View*. 2025. URL: <https://citizenlab.ca/wp-content/uploads/2025/10/stalkerware-holistic-1.pdf> (visited on 02/2026) (cit. on p. 62).
- [87] MVT Project. *Androidqf — Forensic Query Framework for Android Evidence Extraction (GitHub Repository)*. 2025. URL: <https://github.com/mvt-project/Androidqf> (visited on 11/2025) (cit. on p. 63).
- [88] Mobile Security Framework (MobSF) Project. *MobSF — Mobile Security Framework for Automated Mobile Application Security Testing*. 2025. URL: <https://mobsf.live/> (visited on 11/2025) (cit. on p. 64).
- [89] SpyGuard Project. *SpyGuard — A forked and enhanced version of TinyCheck for detecting signs of compromise*. 2025. URL: <https://github.com/SpyGuard> (visited on 10/2025) (cit. on pp. 65, 69).
- [90] Open Information Security Foundation. *Suricata — Open Source Threat Detection Engine*. 2025. URL: <https://suricata.io/> (visited on 10/2025) (cit. on p. 65).
- [91] MISP Project. *MISP — Open Source Threat Intelligence Platform*. 2025. URL: <https://www.misp-project.org/> (visited on 10/2025) (cit. on p. 67).

- [92] FIRST.org. *Traffic Light Protocol (TLP)*. 2025. URL: <https://www.first.org/tlp/> (visited on 10/2025) (cit. on p. 67).
- [93] Interceptor Project. *Storia — Interceptor*. 2025. URL: <https://interceptor.info/storia-interceptor/> (visited on 10/2025) (cit. on p. 70).
- [94] Zeek Project. *Zeek — The Open Source Network Security Monitoring Platform*. 2025. URL: <https://zeek.org/> (visited on 10/2025) (cit. on pp. 70, 86).
- [95] MG Extreme. *Image from article hosted on MG Extreme*. 2024. URL: <https://www.mgextreme.com/m2-bridge/> (visited on 10/2025) (cit. on p. 71).
- [96] MG Extreme. *M2 Bridge New — Advanced Network Traffic Analysis for Spyware Detection*. 2025. URL: <https://www.mgextreme.com/m2-bridge/> (visited on 01/2026) (cit. on p. 71).
- [97] *Manuale M2B-01 — M2 Bridge Procedures and Forensic Reporting*. 2024. URL: <https://www.selcomintelligence.com/wp-content/uploads/2024/09/Manuale-M2B-01.pdf> (visited on 01/2026) (cit. on p. 72).
- [98] Mobile Verification Toolkit Project. *MVT Documentation — Mobile Verification Toolkit*. 2025. URL: <https://docs.mvt.re/en/latest/> (visited on 11/2025) (cit. on p. 72).
- [99] Android Developers. *dumpsys — Android Command-line Diagnostic Tool*. 2025. URL: <https://developer.android.com/tools/dumpsys?hl=en> (visited on 11/2025) (cit. on p. 73).
- [100] Android Developers. *logcat — Android Command-line Log Tool*. 2025. URL: <https://developer.android.com/tools/logcat?hl=en> (visited on 11/2025) (cit. on p. 73).
- [101] ADB Shell. *adb shell pm list packages — ADB Shell Command Reference*. 2025. URL: <https://adbshell.com/commands/adb-shell-pm-list-packages> (visited on 11/2025) (cit. on p. 74).
- [102] KasperskyLab. *iShutdown — Python scripts for extracting and analyzing iOS shutdown logs from sysdiagnose archives*. 2025. URL: <https://github.com/KasperskyLab/iShutdown> (visited on 12/2025) (cit. on p. 74).
- [103] Matthias Frielingsdorf. *Key IOCs for Pegasus and Predator Spyware Cleaned With iOS 26 Update*. 2025. URL: <https://iverify.io/blog/key-iocs-for-pegasus-and-predator-spyware-cleaned-with-ios-26-update> (visited on 01/2026) (cit. on p. 75).
- [104] Frielingsdorf, Matthias. *With iOS 26.2 Release, Apple Reverses Course on shutdown.log*. 2026. URL: <https://iverify.io/blog/with-ios-26.2-release-apple-reverses-course-on-shutdown.log> (visited on 02/2026) (cit. on p. 75).

- [105] Indusface. *What is SSL Pinning? - A Quick Walk Through*. 2025. URL: <https://www.indusface.com/learning/what-is-ssl-pinning-a-quick-walk-through/> (visited on 10/2025) (cit. on pp. 78, 79).
- [106] NewBit. *rootAVD: Script to obtain root privileges on Android Studio*. 2023. URL: <https://github.com/newbit1/rootAVD> (visited on 10/2025) (cit. on p. 79).
- [107] Frida Project. *Frida: A dynamic instrumentation toolkit for developers, reverse engineers, and security researchers*. 2025. URL: <https://github.com/frida/frida/releases> (visited on 10/2025) (cit. on p. 79).
- [108] akabe1. *Frida Multiple Unpinning Script*. 2025. URL: <https://codeshare.frida.re/@akabe1/frida-multiple-unpinning/> (visited on 10/2025) (cit. on p. 80).
- [109] Qustodio. *Why Qustodio — Parental Control Features and Benefits*. 2025. URL: <https://www.qustodio.com/en/why-qustodio> (visited on 10/2025) (cit. on p. 83).
- [110] SuperCowPowers. *ZAT: Zeek Analysis Tools — Processing and analysis of Zeek network data with Pandas, scikit-learn, Kafka and Spark*. 2025. URL: <https://github.com/SuperCowPowers/zat> (visited on 10/2025) (cit. on p. 86).
- [111] Google Developers. *Clustering overview — Machine Learning*. 2025. URL: <https://developers.google.com/machine-learning/clustering/overview> (visited on 10/2025) (cit. on p. 86).
- [112] Eng Majdi AL-Qabalin. *Android Spyware Detection Using Machine Learning: A Novel Dataset*. 2022. DOI: 10.17632/mhvgtywrx1.1. URL: <https://data.mendeley.com/datasets/mhvgtywrx1/1> (visited on 10/2025) (cit. on pp. 87, 91).
- [113] VirusTotal. *VirusTotal — Free Online Malware and Threat Analysis Service*. 2025. URL: <https://www.virustotal.com/gui/home/> (visited on 11/2025) (cit. on p. 88).
- [114] Dan-Duran. *Known VPN IP List — vpn-list.txt (GitHub) for VPN Checker*. List of known VPN IP ranges used for detection. 2025. URL: <https://raw.githubusercontent.com/Dan-Duran/vpn-checker/main/known-vpn/vpn-list.txt> (visited on 11/2025) (cit. on p. 98).
- [115] FreeVPN Planet. *FreeVPN Planet — Free VPN Service Without Registration or Logs*. 2025. URL: <https://freevpnplanet.com/it/> (visited on 11/2025) (cit. on p. 100).

- [116] Turbo VPN. *Turbo VPN — Fast and Secure VPN Service for Privacy and Access*. 2025. URL: <https://turbovpn.com/it/> (visited on 11/2025) (cit. on p. 101).
- [117] Jiaxin Peng, Chun Guo, Yuan Ping, Yunhe Cui, Yi Chen, and Guowei Shen. «SNDMI: Spyware Network Traffic Detection Method Based on Inducement Operations». In: *Computers & Security* 140 (2024), p. 103806. ISSN: 0167-4048. DOI: 10.1016/S0167-4048(24)00107-X. URL: <https://www.sciencedirect.com/science/article/pii/S016740482400107X> (cit. on pp. 102–104).
- [118] MalwareBazaar / abuse.ch. *MalwareBazaar sample record for TheTruthSpy*. 2025. URL: <https://bazaar.abuse.ch/sample/41752e7b3d8374ec74b94ebc258a0fef3d41a44bb07a43d85546701afb50e36/> (visited on 11/2025) (cit. on p. 111).
- [119] Google - Android Developers. *AccessibilityServiceInfo | API reference*. URL: <https://developer.android.com/reference/android/accessibilityservice/AccessibilityServiceInfo> (visited on 01/2026) (cit. on p. 118).
- [120] Android Developers. *Foreground Services — Background Work on Android*. URL: <https://developer.android.com/develop/background-work/services/fgs> (visited on 02/2026) (cit. on p. 130).