SpyNote: Comprehensive Analysis of an Android Remote Access Trojan

By Ireneusz Tarnowski

Published: 2026-01-25 · Archived: 2026-04-05 22:59:26 UTC

21 min read

Jan 23, 2026

Press enter or click to view image in full size

Threat Context and Evolution of SpyNote

In recent years, a steady increase in threats targeting mobile devices has been observed, with the Android ecosystem being particularly affected. Smartphones, which routinely store both personal and corporate data, have become attractive targets for cybercriminals as well as threat actors performing espionage-orientated operations. An example of a long-standing and continuously evolving threat in this domain is the SpyNote malware, also known under related names such as CypherRat and SpyMax.

SpyNote is a Remote Access Trojan (RAT) family that has been present in the mobile threat landscape for several years. Despite the public availability of its builder and its relatively long history, SpyNote continues to be actively leveraged in ongoing campaigns, both large-scale and targeted in nature. In recent periods, its use has also been observed in operations attributed to Advanced Persistent Threat (APT) groups, significantly increasing the overall risk associated with this malware family.

The objective of this report is to provide an overview of the core characteristics, operational model, and distribution methods of SpyNote, followed by a technical analysis of recent selected samples used in current campaigns.

Key characteristics of SpyNote Android RAT

Remote Access Trojan (RAT) malware represents one of the most widespread and persistent threat categories affecting mobile devices. The Android platform, in particular, due to its global prevalence and relatively open application distribution model, remains a prime target for malicious campaigns aimed at data theft, remote device control, and financial abuse. Among active mobile malware families, SpyNote occupies a notable position as a long-standing Android RAT that has served as a foundational codebase for multiple publicly available RAT tools and their subsequent modifications.

SpyNote (also known in underground communities as SpyMax or SpyNote/SpyMax RAT) originally emerged as a commercial or semi-commercial Android RAT offering full remote control over infected devices. A pivotal moment in its evolution occurred in 2020, when the source code of version 6.4 was leaked, leading to widespread proliferation and enabling the development of numerous derivatives (forks) and variants by various cybercriminal groups and third parties. As a result, SpyNote's codebase became the foundation for a range of newer Android RAT tools developed within the underground ecosystem and later distributed under a Malware-as-a-Service (MaaS) model by independent operators.

The technical literature and threat intelligence reporting have also drawn connexions between SpyNote and other tools that exhibit similar operational characteristics. One such example is Craxs RAT, which is described in industry analyses as a derivative or variant of SpyNote/SpyMax, reusing its core codebase while extending it with additional functionality and control mechanisms. In this case, the SpyNote code was further developed by an actor using the alias *EVLF*, who, beginning in 2022, actively developed and distributed Craxs RAT through channels such as Telegram, advertising it as an enhanced Android RAT with expanded device control and surveillance capabilities.

From a threat perspective, SpyNote should not be viewed as a single static tool with a fixed set of features. Multiple variants of this RAT have been identified, often designated using alphabetical or generational naming conventions (e.g., SpyNote.A, SpyNote.B, SpyNote.C). Depending on the campaign, these variants can employ different tactics, techniques, and obfuscation strategies. What unifies them is a broad range of remote control and surveillance capabilities, positioning SpyNote as a highly capable and dangerous tool within the mobile malware ecosystem.

Importantly, the use of SpyNote and its derivatives is not limited to financially motivated cybercrime. Although many campaigns involving these tools are large-scale and focus on credential theft, financial fraud, or application-

level phishing, their functionality also makes them suitable for targeted surveillance operations. Consequently, both cybercriminal groups and potentially Advanced Persistent Threat (APT) actors or other espionage-motivated entities can adapt this malware for their own objectives by extending its modules or integrating it with additional malware components as part of more complex intrusion campaigns. Multiple threat intelligence reports indicate that groups such as OilRig (APT34), APT-C-37 (Pat-Bear), and Kimsuky have included SpyNote in their tooling portfolios during operations targeting high-value assets.

Distribution Model and Operational Use (MaaS)

The distribution of SpyNote and its derivatives, including Craxs RAT, is largely based on the Malware-as-a-Service (MaaS) model, which for years has been one of the primary mechanisms enabling the scalability of cybercrime. This model separates the role of the malware developer from that of the campaign operator: developers provide ready-made malware, builders, and command infrastructure, while end users are responsible for the actual deployment and distribution of the tool.

As noted previously, the leakage of the SpyNote source code enabled unrestricted modification, the development of independent forks, and integration with other malware components. In practice, this led to fragmentation of the SpyNote ecosystem, in which multiple variants coexist, differing in implementation details while preserving a shared architectural foundation and a common set of RAT capabilities characteristic of this malware family.

The tool has been actively advertised through closed and semi-closed communication channels, primarily on platforms such as Telegram. Vendors offered not only the malware itself but also a complete operational backend, including malicious application builders, administrative control panels, and guidance on bypassing Android security mechanisms. This approach significantly reduced the entry barrier for new operators and allowed the rapid launch of infection campaigns without requiring advanced technical expertise.

From a technical standpoint, malware distribution is mainly based on trojanized APK files (Android application installation packages) masquerading as legitimate mobile applications. These are most commonly impersonating web browsers, banking applications, courier services, messaging applications, VPN tools, or applications themed around current social or economic events. Malware APKs are distributed outside the official Google Play Store through phishing websites, direct download links, malicious advertisements, and SMS or email messages containing links to the installers.

A key component of this distribution model is the use of social engineering to persuade users to manually instal the application and grant it extensive system permissions. This mechanism is critical, as SpyNote does not rely on exploit-based infection vectors or remote code execution (RCE). Campaign operators frequently instruct victims to disable Android security features, such as Google Play Protect, or to allow the installation of applications from unknown sources. In some campaigns, more advanced techniques are employed, including device-type-based redirection to customised payloads, link obfuscation through QR codes, or the use of loader applications that download or decrypt the actual SpyNote payload at a later stage.

The MaaS model also promotes the decentralisation of the command-and-control infrastructure. Each operator may deploy their own C2 servers, often hosted between different providers or hidden behind dynamic DNS services. Some variants also support C2 fallback mechanisms and dynamic address rotation using DNS-over-

HTTPS. As a result, a distributed ecosystem of campaigns emerges, which may appear unrelated despite being built on identical or closely related malware codebases. This fragmentation significantly complicates infrastructure-level mitigation efforts and the attribution of activity to specific threat actors.

Operator Infrastructure: Administrative Panel and Builder

An integral component of the SpyNote ecosystem is its publicly accessible administrative panel (C2 panel), offered through the official project website. This panel serves as the central management interface for infected devices and plays a critical role in the operational use of the tool. From a technical and functional perspective, it represents a classic command-and-control interface, providing operators with full visibility into victim activity and the ability to issue commands in real time.

Project Website

The SpyNote administrative panel is marketed as an “advanced Android remote administration tool”; however, the scope of its functionality clearly aligns with that of malicious Remote Access Trojan (RAT) software. Labelling the tool as a “remote admin tool” is a common marketing tactic intended to reduce the legal exposure of its authors. The interface allows operators to monitor and control Android devices in a wide range of capabilities, including passive data collection and active manipulation of the victim’s system. All functions are accessible through a single management console and do not require physical access to the device.

Press enter or click to view image in full size

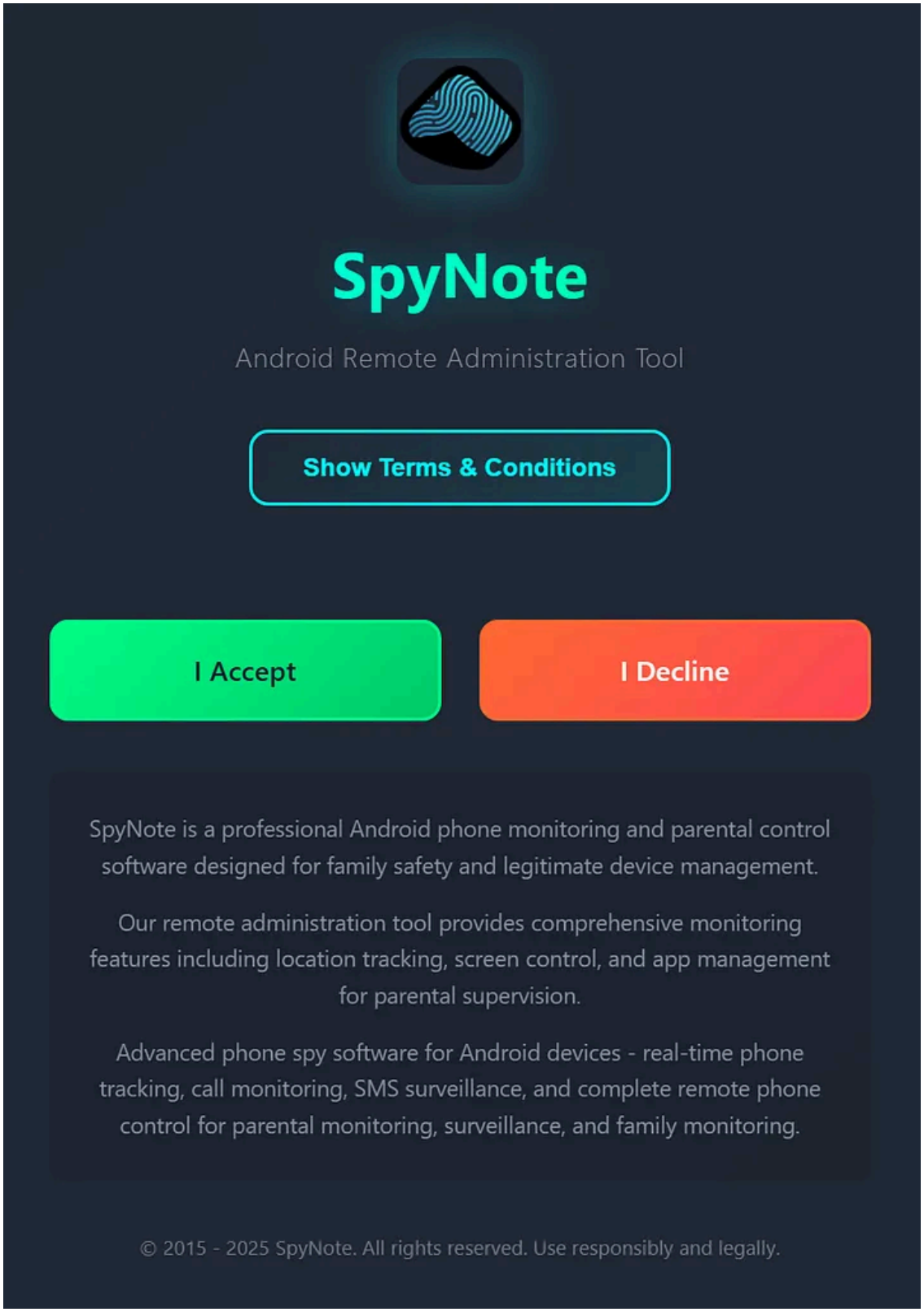

Figure 1. "SpyNote project" service — informational landing page.

Among the core capabilities of the administrative panel is real-time remote screen viewing, combined with the ability to interact with and control the system interface. This functionality enables operators to observe the ongoing activity of the victim, including applications launched, user input, and displayed content. The panel also supports remote screen unlocking, effectively allowing operators to take control of a device even when it is

protected by lock mechanisms. It should be noted that this capability is achieved through Accessibility Service permissions granted rather than by bypassing or breaking Android's native lock-screen security mechanisms.

A particularly significant feature is remote access to device sensors, including a microphone and camera. The panel enables covert audio and video recording without the user's knowledge, as well as real-time camera streaming. When combined with malware activity-hiding mechanisms, these capabilities facilitate long-term, discreet surveillance of the victim.

The administrative panel also provides extensive tools to manage the data stored on the infected device. Operators have access to a file explorer module that supports browsing, downloading, and deleting files, along with dedicated modules to extract SMS messages, call history, contact lists, and stored account credentials. In addition, a built-in keylogging module allows the interception of user input, including potentially sensitive authentication data related to banking or corporate applications.

Press enter or click to view image in full size

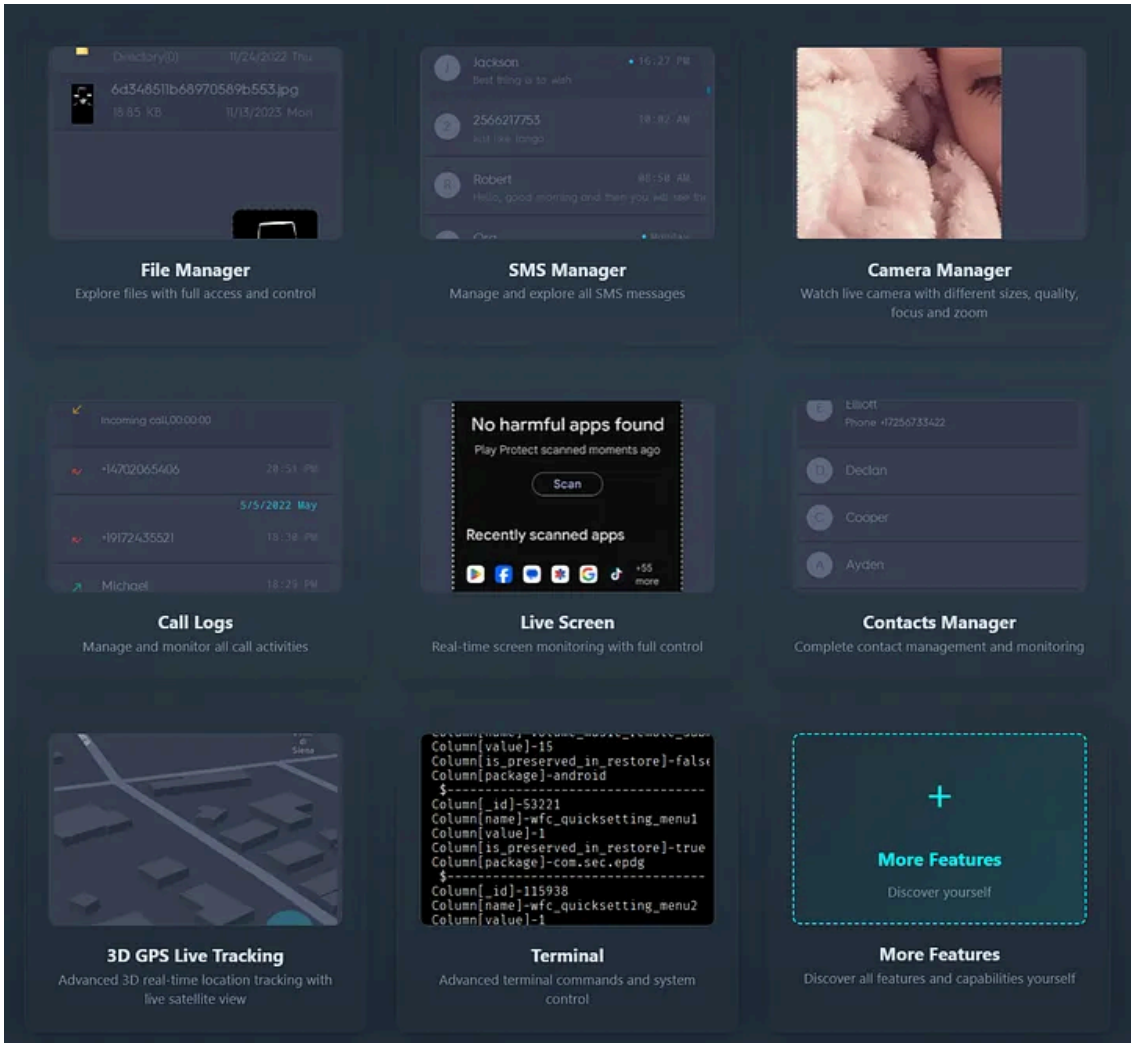

Figure 2. "SpyNote project" service — overview of tool functionality.

One of the more advanced components of the administrative panel is the location tracking module, which provides real-time monitoring of the position of the infected device using GPS data. This functionality is presented through an interactive map with three-dimensional visualisation, allowing for precise tracking of the victim's movements.

When correlated with other modules, it enables the operator to associate user activity with physical location. Persistent location tracking is achieved through a combination of granted location permissions and a background service, resulting in minimal or no system alerts being presented to the user.

The SpyNote panel also includes features typically associated with offensive administrative tools, such as access to a system terminal. This module allows operators to execute commands on the infected device, retrieve detailed system configuration information, and potentially deploy additional components. This functionality significantly increases the operational flexibility of the tool and enables operators to dynamically adapt their actions to a specific victim or scenario.

From a business model perspective, the SpyNote administrative panel is offered under paid licencing schemes, including time-limited subscriptions and lifetime licences. The available options include short-term trial licences, mid-term subscriptions, and one-time purchases granting permanent access to the tool. A notable characteristic of the monetization model is the exclusive acceptance of cryptocurrency payments, such as Bitcoin, Ethereum, or USDT. This approach is commonly observed in ecosystems associated with tools that carry elevated legal and operational risks.

Press enter or click to view image in full size

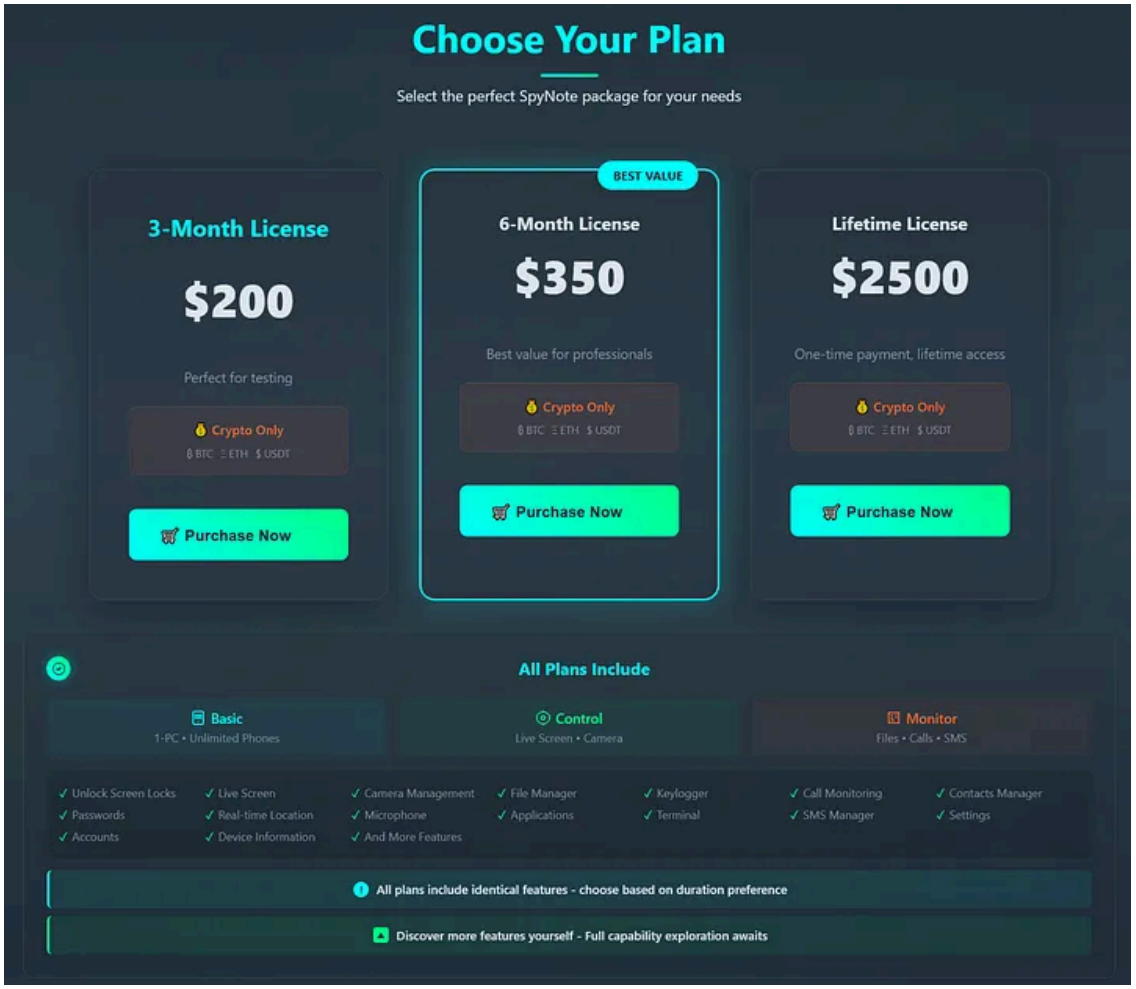

Figure 3. “SpyNote project” service — pricing information.

SpyNote operators also advertise the availability of customised versions of the tool, including modification of the source code and the provision of “special services” on client request. From an analytical standpoint, this indicates a high degree of flexibility and a willingness to tailor malware to specific use cases, including potentially targeted or custom operations.

Command-and-Control (C2) and Builder

The administrative panel described above constitutes the central component of the entire SpyNote malware ecosystem and is responsible for managing the entire infection lifecycle, from the generation of malicious applications to the ongoing control of infected devices. Depending on the SpyNote version and its numerous forks, the panel interface may differ visually and functionally; however, across all observed variants, a common set of core operational mechanisms is preserved. These differences stem from the fact that the tool has been repeatedly modified and adapted by various threat actors over time.

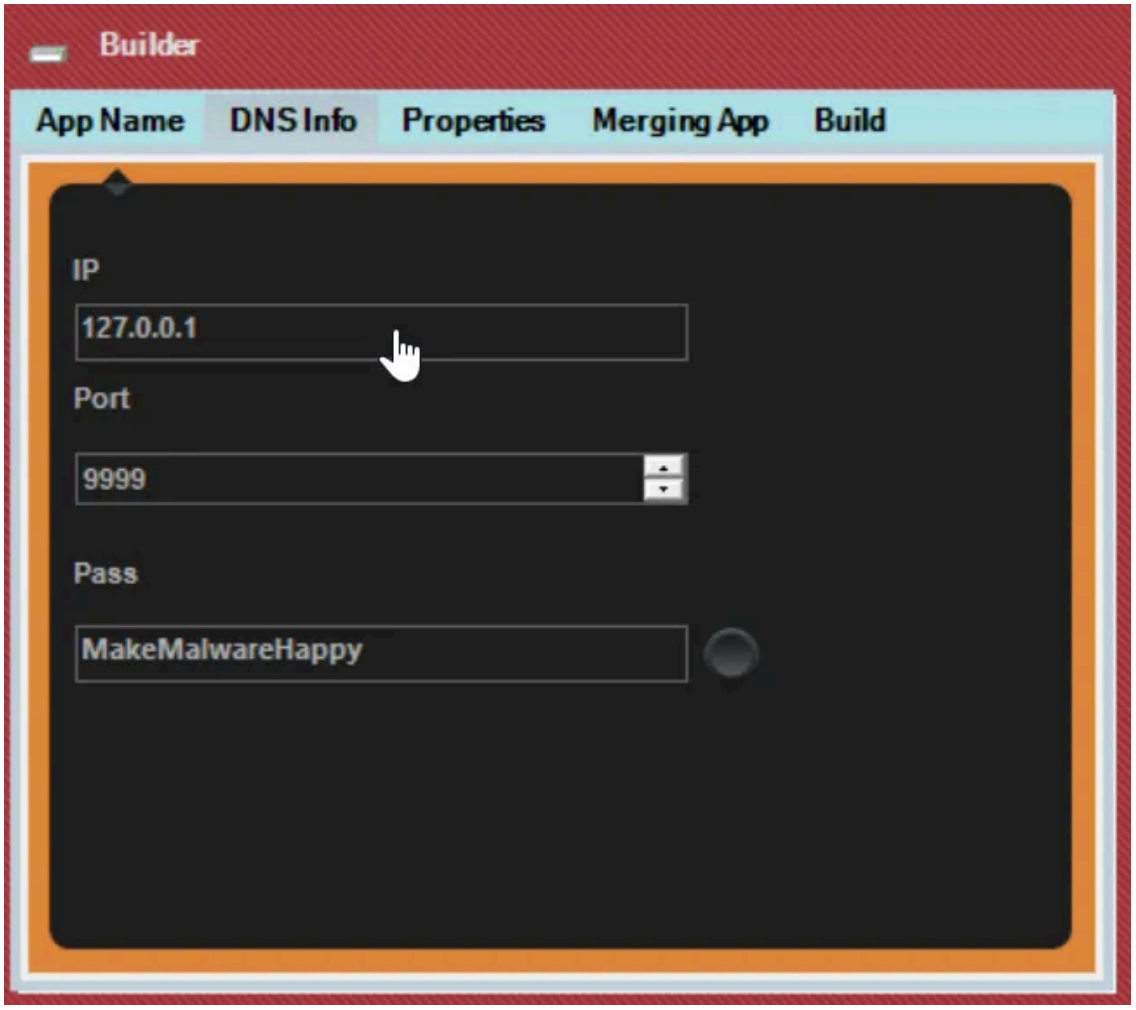

Figure 4. SpyNote administrative panel — infection management.

Regardless of the variant, a key component of the administrative panel is the builder module, which enables the generation of customised and unique malware samples in the form of Android APK files. The builder allows operators to define fundamental parameters of the malicious application, including the application name presented to the user, the package name, and component identifiers, facilitating the masquerading of malware as legitimate software. At this stage, the process name and details of the command-and-control infrastructure are also

configured, notably the IP address or domain of the C2 server and the communication port to which the application will connect once executed on the victim's device. Some builder versions support the generation of multiple variants of the same sample (polymorphism), significantly hindering traditional signature-based detection.

Press enter or click to view image in full size

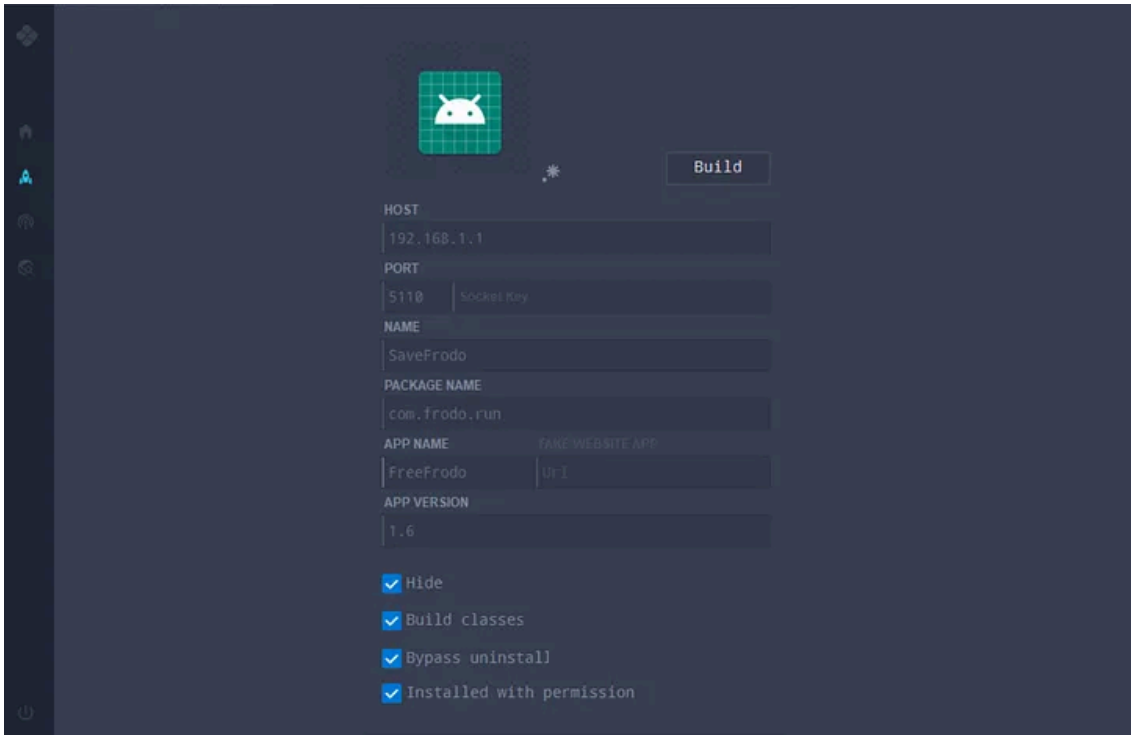

Figure 5. SpyNote administrative panel — builder configuration.

A critical aspect of the building process is the ability to selectively enable specific functionalities within the generated sample. The panel provides operators with a set of toggles corresponding to individual operational modules, such as access to the camera, microphone, location data, SMS messages, contacts, and files. Additionally, the builder allows for the enforcement of requests for special permissions, including the ability to draw screen overlays, ignore battery optimisation mechanisms, and suppress system notifications. Of particular importance is the option to automatically obtain extended privileges following the activation of the Accessibility Service, which in practice enables further escalation of control over the operating system without direct user interaction.

Press enter or click to view image in full size

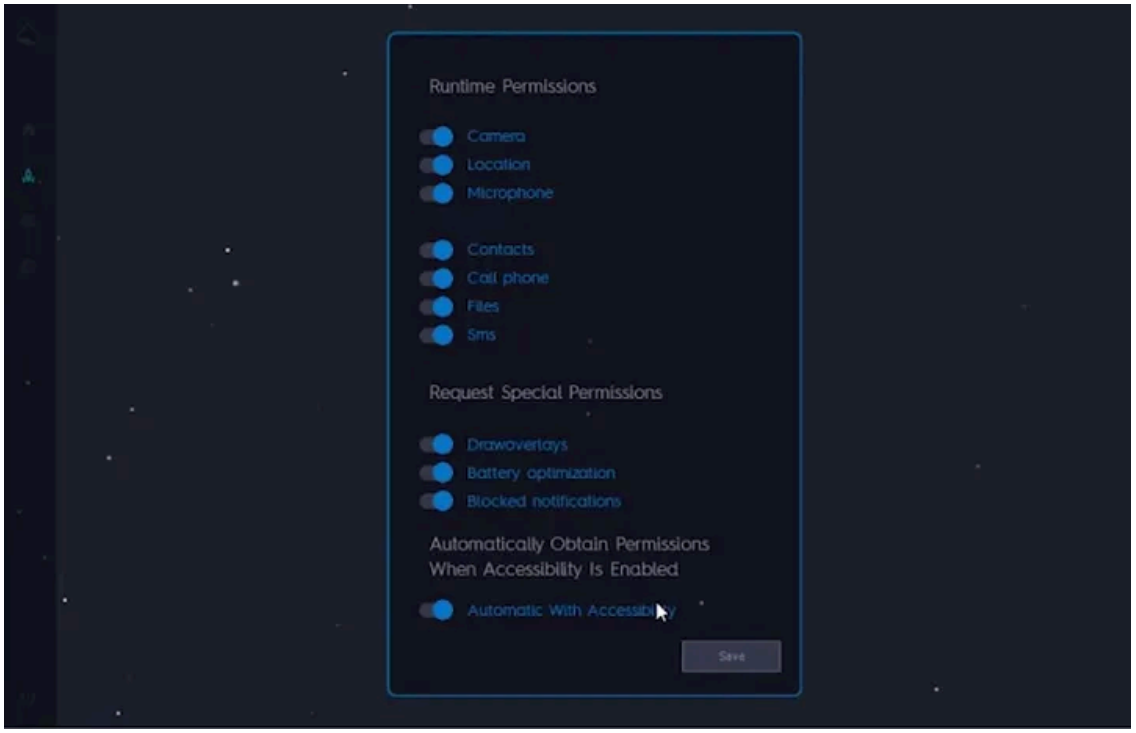

Figure 6. SpyNote administrative panel — builder configuration.

From a technical perspective, the SpyNote administrative panel is a desktop application developed using the .NET framework. The APK build process relies on external tools, most notably apktool, which is used for decompiling, modifying, and recompiling Android application packages. The integration of the builder with apktool allows operators to generate new variants without programming knowledge, automating the injection of configuration data and the embedding or obfuscation of RAT code.

Press enter or click to view image in full size

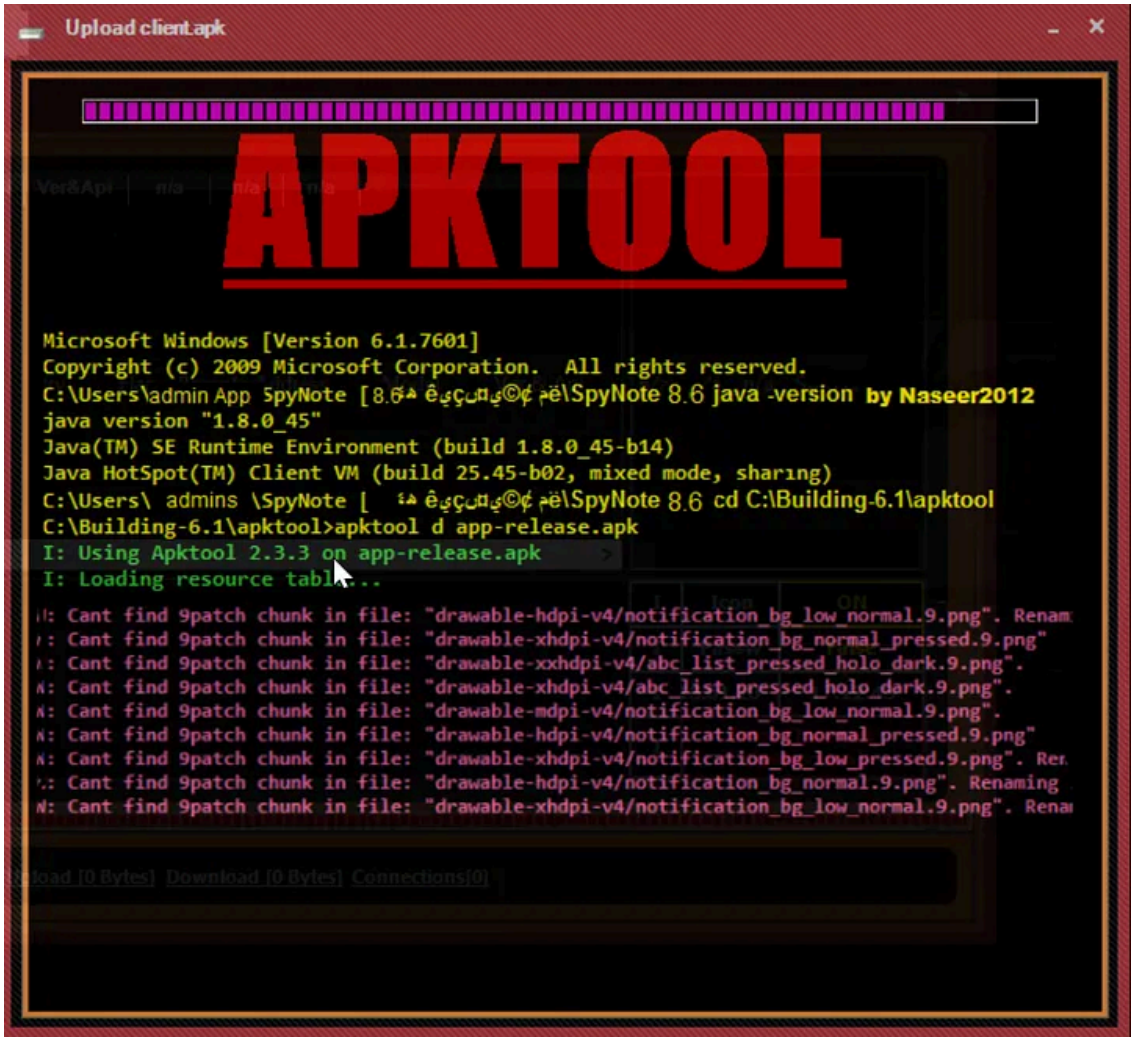

Figure 7. SpyNote administrative panel — APK compilation process.

Once launched, the administrative panel opens a listening port and awaits incoming connexions from infected devices. When a generated application is installed and executed on a victim's device, it initiates a connection to the C2 server defined during the build process. Upon successful communication, the device is registered within the panel as a new victim, granting the operator full visibility into its status and the ability to remotely manage its functionalities.

The panel enables the dynamic activation of individual malware modules in response to the operator's immediate requirements. This includes, among other actions, initiating screen monitoring, capturing images from the camera, recording audio, tracking location, browsing files, and monitoring communications. These capabilities can be activated in real time, allowing flexible surveillance operations and adaptive behaviour based on victim activity.

Analysis of generated APK samples indicates that the build process involves more than simply embedding a static C2 address. It also includes the selective enable or disablement of manifest components and background services. As a result, different SpyNote samples may exhibit significantly different permission requests and runtime behaviours, even when generated from the same administrative panel. This mechanism complicates signature-based detection and facilitates the proliferation of a large number of distinct malware variants.

Press enter or click to view image in full size

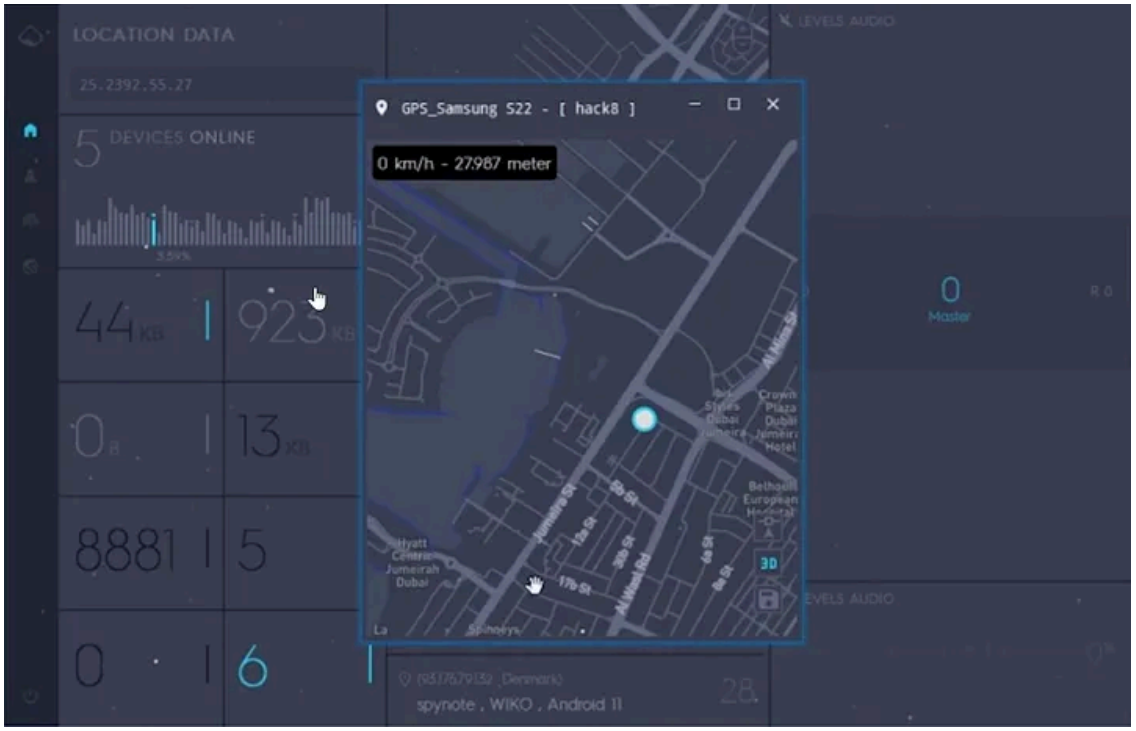

Figure 8. SpyNote administrative panel — infected device management.

From an operational standpoint, the SpyNote administrative panel simultaneously serves as a malware generation tool, a command-and-control server, and an operator console. This consolidation of functionality, combined with ease of use and extensive configuration capabilities, makes SpyNote an attractive tool for both cybercriminals and threat actors conducting more targeted operations, including those of a potential espionage nature.

Technical Analysis of SpyNote Samples

The technical analysis was conducted on selected SpyNote samples obtained from ongoing campaigns as well as from publicly available malware sample repositories. The objective of the analysis was to identify current infection mechanisms, persistence techniques, command-and-control (C2) communication, and the scope of functionality executed on infected Android devices.

Get Ireneusz Tarnowski’s stories in your inbox

Join Medium for free to get updates from this writer.

Remember me for faster sign in

The SpyNote execution flow begins at the application installation stage, delivered in the form of an Android APK package. At this stage, the AndroidManifest.xml file already declares a broad set of permissions; however, their actual granting and activation occur progressively during subsequent infection phases, in accordance with the Android runtime permission model.

Upon first launch, the application initialises its primary activity, responsible for preparing the malware’s operational environment. During this phase, components are activated to collect key device information, including

Android OS version, hardware model, system identifiers, and security patch level. This information is subsequently used to determine the appropriate privilege escalation techniques and operational paths.

Press enter or click to view image in full size

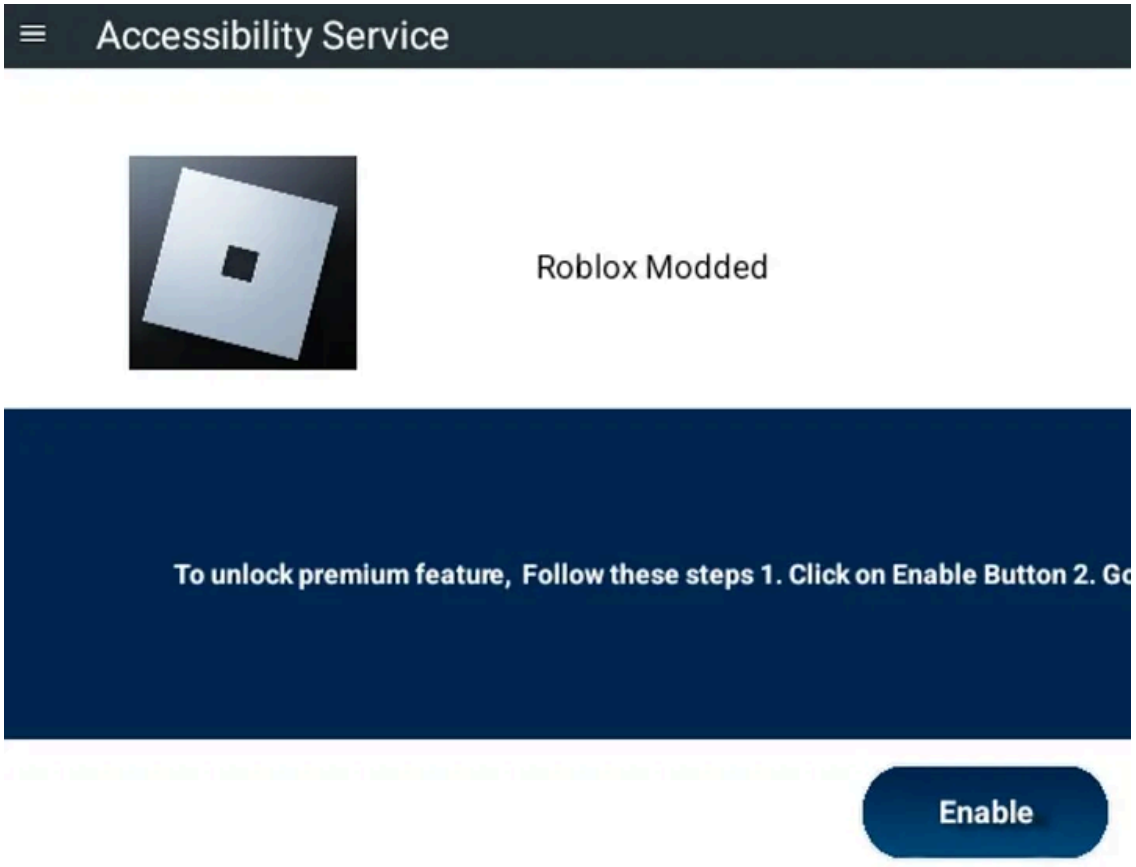

Figure 9. Example screen — Accessibility Service permission request (illustrated using a trojanized sample "Roblox Modded.apk").

The next stage of malware execution involves gradual privilege escalation, implemented through a sequence of dedicated activities designed to coerce the user into granting additional high-impact permissions. This mechanism relies heavily on social engineering techniques and the use of system overlay windows. Initially, the user is asked to grant access to the Accessibility Service, which represents a critical turning point in the SpyNote operational chain. Once obtained, this permission allows malware to monitor user interactions and simulate input events, enabling further automated privilege escalation.

With Accessibility Service permissions in place, SpyNote proceeds to additional system takeover steps. These include semi-automated enforcement of Device Administrator privileges, activation of a custom system keyboard (a common mobile keylogging technique), and authorisation to operate unrestricted in the background by bypassing battery optimisation mechanisms. In certain variants, attempts have also been made to obtain permissions to instal and uninstall application packages, enabling further malware expansion or the deployment of additional components.

In parallel with privilege escalation, persistence mechanisms are deployed. SpyNote registers BroadcastReceiver components configured to handle selected system events, including device reboot (BOOT_COMPLETED), screen state changes, power connection events, and USER_PRESENT events indicating device unlock. As a result, malicious services are automatically restarted after the system reboots or are activated upon user interaction, increasing operational reliability while reducing the observable indicators of malicious activity.

After acquiring the required permissions, the malware initialises its full set of operational capabilities. Services are launched for location tracking, audio and video capture, keystroke logging, and application activity monitoring. Depending on the configuration, some of these capabilities operate in a passive mode, awaiting commands from the C2 server, while others may be triggered automatically in response to predefined system events.

Press enter or click to view image in full size

```
public static final int PERMISSION_GRANTED = 0;
public static final String PHONE_STATE = "android.intent.action.PHONE_STATE";
public static final String PORT = "PORT";
public static final int REQUEST_MEDIA_PROJECTION = 1;
public static final int REQUEST_PERMISSIONS = 100;
public static final String RESULT_CODE = "RESULT_CODE";
public static final String USER_PRESENT = "android.intent.action.USER_PRESENT";
static final String ping = "ping";
static final Class<?>[] arrayClasses = {iMGpqTLQfPSqrU.class, NTgmPEDNVJh.class, BkEUhlW.class};
static volatile Method remoteJobs = null;
private static final String[] formatSpecifiers = {".%s", "%s/%s"};
static final String[] getAccessibilitySettingsActivity = {"com.android.settings.Settings$AccessibilitySettingsActivity", "com.android.setti
public static final Object SharedSync = new Object();
public static final Object SharedSync_pass = new Object();
private static Intent screen_permission_data = null;
private static Intent screen_Intent = null;
public static final String[] permissions = {"android.permission.CAMERA", "android.permission.READ_SMS", "android.permission.RECORD_AUDIO",

public static final class SocketInfo {
    public static final String multiTasking = "m_multi_t_tasking";
    public static final int VERSION = Build.VERSION.SDK_INT;
    public static final String NAME = "SaveFrodo";
    public static final String HOST = "[redacted]";
    public static final String PORT = "5110";
    public static final String AccessibilityName = "FreeFrodo";
    public static final String SOK_EY = "SOK_EY_DEFAULT";
    public static final boolean IS_BYPASS_UNINSTALL = Boolean.parseBoolean("true");
    public static final String LOAD_URL_WEB_VIEW = "";
}
```

Figure 10. Decompiled and decoded SpyNote class: network configuration.

The final stage of the execution chain involves establishing communication with the C2 infrastructure of the campaign operator. SpyNote initiates outbound network connexions to the configured C2 server, transmitting previously collected device identification data and signalling readiness to receive commands. Communication is performed either periodically or in an event-driven manner, depending on the configuration, and serves both for exfiltrating data from the compromised device and for receiving control instructions governing further malware behaviour through a shared data and management channel. In more advanced variants, this communication may also be concealed through encryption mechanisms or VPN tunnelling, complicating detection at the network layer.

In general, the SpyNote operational model reflects a deliberate multi-stage architecture designed to progressively assume control over the victim’s device while minimising the risk of detection. The combination of social engineering, the abuse of the mechanisms of the Android system, and centralised C2-based control makes SpyNote a tool well suited for long-term covert surveillance operations.

Command-and-Control (C2) Communication

The component analysed <package_name>.run.Socket is responsible for handling the complete network communication between the infected device and the SpyNote command-and-control server (C2). This implementation represents the central element of the C2 communication architecture and is designed to support

stable and long-lived bidirectional communication while maintaining flexibility in the transmission of commands and data. The use of persistent TCP connections distinguishes SpyNote from the majority of mobile RATs, which typically rely on short-lived, pull-based communication models.

On the client side, communication is implemented using asynchronous network channels (AsyncChannel) built on Java NIO (Non-blocking I/O), leveraging TCP sockets. During connexion initialisation, malware configures key socket options such as TCP_NODELAY and SO_KEEPALIVE, indicating an intentional effort to minimise latency and maintain long-lived sessions. The C2 server address and port are dynamically retrieved from the Support.SocketInfo class, confirming that these values are injected during the sample build process via the administrative panel.

The connection establishment process is implemented in the Client.Connect class, which performs repeated connection attempts with built-in exception handling. In the event of a failure, the malware introduces a delay before retrying, allowing it to survive temporary C2 infrastructure outages or victim-side network disruptions. Upon successful connection, a timestamp is recorded and subsequently used for session state monitoring.

Immediately after establishing the connection, the client transmits an initialisation packet containing an extensive set of identification and telemetry data. This data set includes a unique device identifier, malware version, application package name, key class names, battery status, location data, screen lock state, and detailed information about the operating system and vendor-specific interface. This packet serves as the registration mechanism for a newly compromised device within the administrative panel and enables the operator to rapidly assess the operational value of the victim.

Inbound data from the C2 server is handled by the ReceiveData class, which implements a proprietary communication protocol. Data are transmitted as a byte stream in which individual segment lengths are separated by a null byte. Once a complete frame is received, the data are decoded and optionally decompressed using the GZIP mechanism before being passed on for further processing. The decoded packets are placed into a shared Socket.packets queue, enabling asynchronous handling by other execution threads.

Command processing is performed by the IncomingPackets class, which cyclically retrieves packets from the queue and forwards them to the Client.Data method. At this stage, incoming instructions are interpreted based on logical command keys, triggering the execution of corresponding malware functions.

A notable architectural feature is the handling of operational tasks. SpyNote employs parallel task execution for activities such as file system traversal, directory content enumeration, and operations on multimedia files, including screen capture with configurable parameters. The use of thread pools allows efficient utilisation of device resources while enabling these activities to be performed in the background without generating obvious indicators of activity visible to the user.

The persistence of C2 communication is maintained through a dedicated connection health monitoring mechanism implemented in the CheckConnection class. This component periodically sends “ping” packets and tracks idle session time. In the absence of responses or upon detection of connectivity issues, malware autonomously initiates disconnection and reconnection procedures. This mechanism ensures the continuity of the campaign even under unstable mobile network conditions. In addition, up-to-date information is periodically transmitted regarding battery level, charging state, location, and screen lock status, allowing the operator to maintain real-time

situational awareness of the victim device. Such telemetry may also function as operational triggers — for example, to activate audio or video recording when the device is connected to a power source to minimise battery drain and reduce the likelihood of user detection.

All control logic governing the operation of individual communication components is centralised within the Controller class. This class coordinates the initialisation, termination, and reinitialization of tasks responsible for connection handling, data reception, command processing, and session monitoring. This design provides high fault tolerance and enables automatic recovery from communication disruptions, a characteristic commonly associated with mature malware families. In some variants, additional communication concealment mechanisms were observed, including custom headers, obfuscated session identifiers, or tunnelling over HTTPS, significantly complicating network traffic monitoring and event correlation.

Analysis of the communication component clearly indicates that SpyNote employs a custom C2 protocol based on persistent TCP sessions, supporting compression, dynamic command handling, and multithreaded processing. This architecture is optimised for long-term control over infected devices, flexible functional expansion, and minimisation of session loss risk, reinforcing SpyNote's classification as a mature and opeRAT.onally robust Android RAT.

Obfuscation Techniques

SpyNote employs a broad range of obfuscation and code protection techniques that have evolved from basic methods to sophisticated mechanisms designed to evade modern security controls. Although early versions and publicly available builders often lacked any meaningful protection, contemporary variants (such as SpyNote.C or V7) exhibit a significantly higher level of complexity.

Press enter or click to view image in full size

```
public class MySettings {
    public static final String BLACKSCREEN = "BLACKSCREEN";
    public static final String PERFINISHALL = "PerFinishAll";
    public static final String UN_LOCK_OPPO_AND_OTHER = "UN_LOCK_OPPO_AND_OTHER";
    private static SharedPreferences mSharedPref;
    public static String ScreenWidth = utilities.stuoxctjtdaoeuivbicuhkpiflkolbbmlmpdazwmgcmmbmmdif53("SCkrgcqosbwnxdhmzmtxnqbdhqnsmmwxjswojjqugetwga
    public static String ScreenHight = utilities.stuoxctjtdaoeuivbicuhkpiflkolbbmlmpdazwmgcmmbmmdif53("SCkrgcqosbwnxdhmzmtxnqbdhqnsmmwxjswojjqugetwga
    public static String RecordName = utilities.stuoxctjtdaoeuivbicuhkpiflkolbbmlmpdazwmgcmmbmmdif53("RCkrgcqosbwnxdhmzmtxnqbdhqnsmmwxjswojjqugetwgag
    public static String Screenshotsdir = utilities.stuoxctjtdaoeuivbicuhkpiflkolbbmlmpdazwmgcmmbmmdif53("SCkrgcqosbwnxdhmzmtxnqbdhqnsmmwxjswojjquget
    public static String LASTKEYBOARD = utilities.stuoxctjtdaoeuivbicuhkpiflkolbbmlmpdazwmgcmmbmmdif53("Lkrgcqosbwnxdhmzmtxnqbdhqnsmmwxjswojjqugetwga
    public static String isblocked = utilities.stuoxctjtdaoeuivbicuhkpiflkolbbmlmpdazwmgcmmbmmdif53("Bkrgcqosbwnxdhmzmtxnqbdhqnsmmwxjswojjqugetwgaggg
    public static String SendSkilton = utilities.stuoxctjtdaoeuivbicuhkpiflkolbbmlmpdazwmgcmmbmmdif53("SKkrgcqosbwnxdhmzmtxnqbdhqnsmmwxjswojjqugetwga
    public static String AutoStartOn = utilities.stuoxctjtdaoeuivbicuhkpiflkolbbmlmpdazwmgcmmbmmdif53("Akrgcqosbwnxdhmzmtxnqbdhqnsmmwxjswojjqugetwgag
    public static String Cantgo = utilities.stuoxctjtdaoeuivbicuhkpiflkolbbmlmpdazwmgcmmbmmdif53("cankrgcqosbwnxdhmzmtxnqbdhqnsmmwxjswojjqugetwgagggj
    public static String skipbttryoppo = utilities.stuoxctjtdaoeuivbicuhkpiflkolbbmlmpdazwmgcmmbmmdif53("skkrgcqosbwnxdhmzmtxnqbdhqnsmmwxjswojjqugetw
    public static String skipmiuibty = utilities.stuoxctjtdaoeuivbicuhkpiflkolbbmlmpdazwmgcmmbmmdif53("skkrgcqosbwnxdhmzmtxnqbdhqnsmmwxjswojjqugetwga
    public static String bindoncedone = utilities.stuoxctjtdaoeuivbicuhkpiflkolbbmlmpdazwmgcmmbmmdif53("bnkrgcqosbwnxdhmzmtxnqbdhqnsmmwxjswojjqugetwg
    public static String Crash = utilities.stuoxctjtdaoeuivbicuhkpiflkolbbmlmpdazwmgcmmbmmdif53("crkrgcqosbwnxdhmzmtxnqbdhqnsmmwxjswojjqugetwgagggjjg
    public static String T2 = "krgcqosbwnxdhmzmtxnqbdhqnsmmwxjswojjqugetwgagggjjgvesqubnghmzrrkkqsnacmunvgfubvshpqhjtlaxsqvznwwjywgciufunbugdiwafsacu
    public static String T3 = "krgcqosbwnxdhmzmtxnqbdhqnsmmwxjswojjqugetwgagggjjgvesqubnghmzrrkkqsnacmunvgfubvshpqhjtlaxsqvznwwjywgciufunbugdiwafsacu
    public static String T4 = "krgcqosbwnxdhmzmtxnqbdhqnsmmwxjswojjqugetwgagggjjgvesqubnghmzrrkkqsnacmunvgfubvshpqhjtlaxsqvznwwjywgciufunbugdiwafsacu
    public static String T5 = "krgcqosbwnxdhmzmtxnqbdhqnsmmwxjswojjqugetwgagggjjgvesqubnghmzrrkkqsnacmunvgfubvshpqhjtlaxsqvznwwjywgciufunbugdiwafsacu
    public static Boolean will_ask_battary_2 = false;

    private static String getmet2(int i) {
        return "krgcqosbwnxdhmzmtxnqbdhqnsmmwxjswojjqugetwgagggjjgvesqubnghmzrrkkqsnacmunvgfubvshpqhjtlaxsqvznwwjywgciufunbugdiwafsacuzrqatoddyqerhwr
    }
```

Figure 11. Configuration observed in a decompiled file (encoded version).

Some variants operate as droppers, where the initial APK functions solely to deploy a secondary payload concealed within a DEX file embedded in the application's resources. Connection parameters for the C2 infrastructure are frequently embedded within these additional DEX files, which introduces another layer of concealment. Such techniques have been observed in campaigns attributed to APT groups (e.g., APT43), where the actual SpyNote code is hidden in the /assets directory under benign-looking filenames such as security.dat or

search.db. Decryption of this payload is performed using a dedicated native library, commonly referred to as SILENTKEY (e.g., libnative-lib.so), which exposes a decryptFile function. The shift from simple Java-based XOR encryption to native-library-based decryption significantly reduces the effectiveness of static analysis using standard DEX decompilation tools.

Malware extensively leverages encoding mechanisms to conceal critical configuration data, including C2 IP addresses, port numbers, and communication keys. Exfiltrated victim data — such as captured keystrokes — is likewise stored and transmitted in encoded or, in some cases, encrypted form. In newer variants, multi-stage decoding chains are employed, with strings being decrypted only at the moment of use, further complicating static function mapping and string-based detection.

Recent SpyNote versions make use of commercial packers and advanced string obfuscation techniques, rendering the application code largely unreadable to conventional decompilers and hindering the identification of malicious functionality. Obfuscated class and service names (e.g., classes labelled C71 or C38) are commonly used to blend malicious components into the landscape of legitimate system services. Many samples also include artificial code padding, dead code, fake classes, and deeply nested try/catch blocks, all of which are intended to disrupt control-flow analysis and delay reverse engineering efforts.

Press enter or click to view image in full size

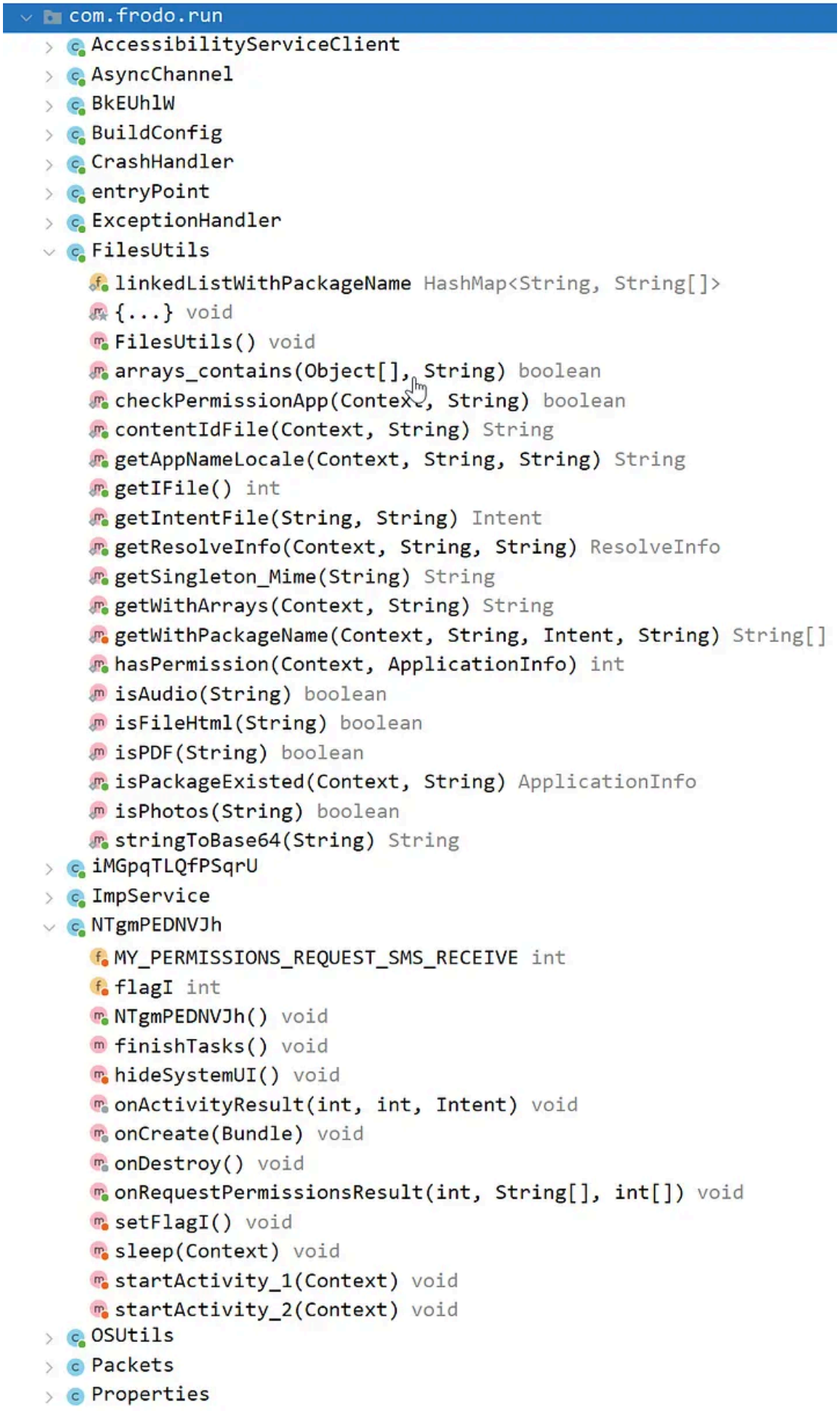
com.frodo.run
AccessibilityServiceClient
AsyncChannel
BkEUhlW
BuildConfig
CrashHandler
entryPoint
ExceptionHandler
FilesUtils
linkedListWithPackageName HashMap<String, String[]>
{...} void
FilesUtils() void
arrays_contains(Object[], String) boolean
checkPermissionApp(Context, String) boolean
contentIdFile(Context, String) String
getAppNameLocale(Context, String, String) String
getIFile() int
getIntentFile(String, String) Intent
getResolveInfo(Context, String, String) ResolveInfo
getSingleton_Mime(String) String
getWithArrays(Context, String) String
getWithPackageName(Context, String, Intent, String) String[]
hasPermission(Context, ApplicationInfo) int
isAudio(String) boolean
isFileHtml(String) boolean
isPDF(String) boolean
isPackageExisted(Context, String) ApplicationInfo
isPhotos(String) boolean
stringToBase64(String) String
iMGpqTLQfPSqrU
ImpService
NTgmPEDNVJh
MY_PERMISSIONS_REQUEST_SMS_RECEIVE int
flagI int
NTgmPEDNVJh() void
finishTasks() void
hideSystemUI() void
onActivityResult(int, int, Intent) void
onCreate(Bundle) void
onDestroy() void
onRequestPermissionsResult(int, String[], int[]) void
setFlagI() void
sleep(Context) void
startActivity_1(Context) void
startActivity_2(Context) void
OSUtils
Packets
Properties

Figure 12. SpyNote class and method tree — unobfuscated version.

In more advanced cases, the malware code is decrypted only at runtime and loaded directly into memory (in-memory execution). As a result, malicious modules are not present in plain text form on the disc, effectively bypassing file-based scanning mechanisms such as Google Play Protect. This technique is primarily observed in variants used in targeted campaigns or developed by advanced operators and is not a standard feature of the publicly available SpyNote builder.

The cumulative application of these techniques significantly impedes both static analysis and behavioural detection of SpyNote, underscoring the increasing sophistication of this malware family and its continued adaptation to evolving defensive measures.

Summary

SpyNote represents a mature, multi-component Android RAT designed as a cohesive ecosystem encompassing C2 infrastructure, an operator control panel, a personalised sample generation mechanism, and a modular client-side architecture. This design enables centralised campaign management and precise tailoring of individual malware instances to specific operational requirements.

From a functional perspective, SpyNote combines passive surveillance capabilities with active device control. Its feature set includes credential harvesting, user activity monitoring, access to system resources, and execution of remote commands. A core architectural element is the systematic abuse of Android Accessibility Services, which enables logical privilege escalation and automation of user interface interactions without requiring root access. In practice, this makes SpyNote particularly effective on modern Android versions, as it bypasses many security restrictions imposed on non-privileged applications. Another distinguishing characteristic is its ability to maintain persistent TCP sessions, setting SpyNote apart from most mobile RATs and complicating network traffic analysis.

The SpyNote operational model shifts the configuration logic to the sample build stage. Communication parameters, functional scope, and post-installation behaviour are defined within the operator panel, resulting in malware variants that differ significantly in runtime behaviour and technical artefacts. This approach substantially reduces the effectiveness of signature-based detection and facilitates long-term persistence on compromised devices.

From a detection and response standpoint, SpyNote represents a class of threats characterised by deep integration with the Android system mechanisms, resilience to device reboots, and the ability to operate stealthily in the background based on system events. The use of a proprietary C2 protocol and dynamic command processing further complicates infrastructure identification and network-level blocking efforts.

Consequently, SpyNote should not be viewed as a single malware family in a narrow sense, but rather as a flexible operator toolkit capable of adapting to changes in the Android security model. In its current form, the term “SpyNote” effectively encompasses a broad family of forks and variants with varying levels of sophistication, some of which diverge significantly from the original 6.4 release — an important consideration for both technical analysis and campaign attribution. Its presence within the mobile threat landscape should therefore be assessed

from a long-term perspective, with emphasis on behavioural detection, event correlation, and monitoring for abuse of the core mechanisms of the Android system.

Indicators of Compromise (IoC)

```
Analysed Samples

sonapk.apk
SHA256: 3cf8bd9828f8fe52867fcb09f3caa59f5ce0aa76ff20cf644807ae76b57c2c86
C2: tcp://103.61.224[.]102:2323

Gmail.apk
SHA256: 8f09663836bef9fad23b34560dbcb0848e99d66e40787c37d498e7647792d5b6
C2: tcp://103.61.224[.]102:2323

inpost.apk
SHA256: 40d31617f45e8317e9d8fa6e42e67d587bdc546b50fd2197b26ac27b51d037de
C2: tcp://103.61.224[.]102:3333

Roblox Modded.apk
SHA256: acf2d29c8c65ee2fe57445e672fbee01fa240b0039b66ea507f110468c6c8210
C2: tcp://144.31.30[.]235:7771

Other (latest) samples

SHA256: 6fd37bbd31b52c6312a0c7972d7fe7242dade45f3d8faa2fc548bef2e3400ecd
C2: tcp://20.82.176[.]195:7771

 SHA256: 231b21251d16d17e564a2014765d1de553eb821abd92781b18c94889650a3bf7
C2: tcp://91.92.251[.]105:12004

 SHA256: b29b8bd2d47254de3d7bf21d7610209d3cc4db49cd3e7ba2fd1ea040f49cb6db
C2: tcp://185.87.254[.]82:2305

SHA256: d9c47a7d7e42402c3ce2dd191ea09e9f7e29b1ee8d78d9aec0a47ed7b4bcdb80
C2: tcp://mm-includes.gl.at.ply[.]gg:33004

SHA256: 5d4aa3800788f80d2a0b0460574ee3d3403c642b19e294941cd9e59b37aebae5
C2: tcp://193.161.193[.]99:40920

SHA256: d14fb879a81e6c415146092d2aab8f8c69991828dbebc0ec27363248f9b260c0
C2: tcp://83.217.209[.]142:1333

SHA256: e21f8722ab3d3557e7b0dda0faca39c517bbf0afd84bf4bbdc92687c9bd58aae
C2: tcp://tcp.cloudpub[.]ru:48683
```

Sources and Reference Materials

1. https://www.mobile-hacker.com/2025/06/05/analysis-of-spyware-that-helped-to-compromise-a-syrian-army-from-within/
2. https://www.group-ib.com/blog/craxs-rat-malware/
3. https://malpedia.caad.fkie.fraunhofer.de/details/apk.spynote
4. https://hunt.io/malware-families/spynote
5. ThreatLabz 2025 Mobile, IoT & OT Threat Report
6. https://app.apkdetect.com/search/?malware=SpyNote

END OF ANALYSIS — 21 January 2026 (IR3k)

The following analysis does not aim to provide an exhaustive description of all capabilities and variants of the SpyNote malware family. It focuses on selected technical aspects and representative samples observed in recent campaigns, with particular emphasis on architectural design, privilege abuse mechanisms, persistence, and command-and-control communication. While SpyNote implements a wide range of additional features and variants, only those deemed most relevant from a defensive and analytical perspective are discussed.

The purpose of this report is strictly educational and analytical, intended to highlight techniques employed by contemporary Android malware and to support detection, analysis, and threat awareness efforts. Any inaccuracies or omissions are unintentional and do not affect the overall analytical conclusions.

This analysis has been published on the CERT Orange Polska portal at:
https://cert.orange.pl/aktualnosci/operacyjna-analiza-kanalow-komunikacyjnych-mobilnego-rcsa/
