Take a note of SpyNote malware

By Amit Tambe

Published: 2025-02-23 · Archived: 2026-04-05 16:54:18 UTC

The Android threat landscape is fraught with diverse types of malware, each bringing its own ingenuity to the field. Although each piece of Android malware has its own malicious agenda, the typical objective in most cases is to steal user data, especially personal data that can be used for nefarious purposes or even sold later. Based on the agenda alone, certain malware can be classified as spyware, because their sole objective is to spy on users by stealing as much user data as possible, and for as long as possible while staying hidden. This is slightly different than the objective of infostealers — which is, typically, to steal user credentials and credit card information for resale.

Among noteworthy spyware, one that has been in the limelight recently is SpyNote. This spyware app spreads via smishing (i.e. malicious SMS messages) by urging the victims to install the app from provided links. Naturally, the hosting and downloading happen outside of the official Play Store app, to prevent the security evaluation done by Google Play Store from thwarting the spread of this spyware.

In this article we describe some prominent features of this malware app based on our recent analysis. These features are typical for spyware (including stalkerware, which has different spreading mechanisms — a topic for a separate article).

SpyNote features

We analysed a SpyNote sample with SHA256 bad77dca600dc7569db4de97806a66fa969b55b77c24e3a7eb2c49e009c1f216 (pkg name Glasgow.pl.dimensional).

Manifest File

As is typical with Android malware analysis, we start the analysis by looking at the AndroidManifest.xml file. Fig. 1 shows a snapshot of the Manifest file. Although the list of permissions requested by SpyNote is not very exhaustive, we do see some suspicious permissions being requested. These include READ_SMS, PROCESS_OUTGOING_CALLS, CAMERA, RECORD_AUDIO, WRITE_EXTERNAL_STORAGE, and BIND_ACCESSIBILITY_SERVICE. Even though a request for these permissions doesn't always indicate malicious behavior, these are typical permissions that spyware requests.


A screenshot of a section of the AndroidManifest.xml file of SpyNote opened in a code editor.
Figure 1. Suspicious permissions requested by SpyNote.

Entry point

As per (one piece of) Android literature, there are multiple ways to create an entry point for your application:

- Launcher activity that handles intents of MAIN and LAUNCHER types.
- Service running in the background.
- Broadcast receiver that gets called when the system sends one of the broadcasts that the app expects.
- The declaration of attachBaseContext method defined in the Application

For our analysis, the next logical step to identify the entry point for this suspected spyware app and finding which method SpyNote uses for that. Upon first inspection, we see several methods used in AndroidManifest.xml that can qualify as “entry points”. To gain more understanding, we install SpyNote on an Android device and observe its behavior. This can help to correlate the observed behaviour on the phone with the code.

Upon installation, we note that the app is nowhere to be seen in the App launcher. A simple check in the Phone Settings → Apps menu, however, shows that the app has been installed successfully. We can infer that the app is hiding to avoid detection.

For spyware, the reason behind hiding is to avoid detection and carry on with its objective of stealing user data as long as possible. A couple of methods may be used by spyware to achieve this hiding functionality:

- Use of setComponentEnabledSetting to hide the app at runtime
- Providing no launcher activity category (CATEGORY_LAUNCHER) in AndroidManifest.xml (corresponding to the intent containing ACTION_MAIN)

As shown in Fig. 2 the launcher activity in SpyNote has CATEGORY_LAUNCHER category missing. This renders the app hidden on the phone. Fig. 2 also shows the class name of the main activity.


A screenshot of another section of the AndroidManifest.xml file of SpyNote, where CATEGORY_LAUNCHER is missing.
Figure 2. Missing “Launcher” category renders app hidden.

Exclude from “Recents”

Apart from hiding itself, SpyNote also takes the extra step of hiding its activities from showing up in the “Recents” screen (the screen that displays recently used apps). It achieves this by defining the attribute android:excludeFromRecents in Androidmanifest.xml and setting it to “true”.

As per Android documentation, “the Recents screen, is a system-level UI that lists recently accessed activities and tasks. The user can navigate through the list, select a task to resume, or remove a task from the list by swiping it away”. This is typically accessed by tapping on the hamburger menu at the bottom of the screen or swiping the screen from the bottom.

There are several ways that spyware apps may use for launching hidden apps:

- Attacker sends a command via SMS. The spyware app receives the broadcast for “SMS received” and then unhides the app icon.

- Launching apps upon receiving an external trigger:
    - Receiving broadcast for an outgoing phone call and trigger the app
    - Adding an intent filter for a specific URL in the manifest file. Whenever the victim browses to that URL, the malware will be launched. The victim can be convinced to visit that URL by sending SMS messages, for example.
- Using a separate launcher app that only sends the desired intent to the malware app, which is launched upon receiving this intent.

 An Android phone with the “Recents” view visible. SpyNote hides itself from this view.
Figure 3. The “Recents” screen.

It is essential for spyware that it stays hidden. However, this raises the question, “How will the victim launch the spyware app so that it can steal data”?

The SpyNote malware app can be launched via an external trigger. We created a minimalistic “Hello World”-style Android app (as shown in Fig. 4), that only sends the necessary intent (an “intention” to perform an action). Upon receiving the intent, the malware app launches the main activity.

 An example piece of code that would trigger the main activity of the hidden malware.
Figure 4. External trigger app that launches hidden malware.

Permissions

Upon launch, SpyNote primarily requests for the BIND_ACCESSIBILITY_SERVICE permission and once the victim grants it, the malware grants itself multiple other required permissions (android.permission.PROCESS_OUTGOING_CALLS, android.permission.RECORD_AUDIO, android.permission.WRITE_EXTRENAL, etc. as per Fig. 1)

 A screenshot of the lines of SpyNote code where more permissions are granted.
Figure 5. Self-granting permissions using Accessibility Services

Figure 5 shows the steps after victim grants BIND_ACCESSIBILITY_SERVICE permission:

1. the spyware app requests all permissions mentioned in the manifest file one by one, and
2. instead of waiting for the victim to grant these permissions, it generates a “tap” stroke to replicate a tap by the user, thereby self-granting all requested permissions.

Diehard services

After getting SpyNote to launch, we can verify from the phone settings that it runs two services using obfuscated service names. Based on AndroidManifest.xml, however, we can confirm that these are implemented in two classes called C71 and C38.

Further investigation of the code reveals that these services are what we can call “diehard services”. The main goal of diehard services is to make shutting down the malware app very difficult — both by victims or by the Android system itself.

SpyNote achieves this functionality by registering a broadcast receiver (an Android component that allows you to register for system or application events), called “RestartSensor”. This broadcast receiver is a unique one, because no other receiver on the system will be able to process the specific broadcast that SpyNote generates when it is about to be shut down.

Whenever SpyNote services are about to be shut down, that service’s onDestroy method gets called. onDestroy method then itself generates a “RestartSensor” broadcast. This is a broadcast specific to this app, and cannot be generated by any other app on the system. Upon receiving this broadcast, the broadcast handler (which is the previously registered unique receiver) restarts the services. Fig.6 shows this process.


A flowchart of how SpyNore is able to restart itself after getting killed by Android.
Figure 6. Diehard service restarts itself.

Figure 7. shows this process via code screenshots. The spyware app first declares a broadcast receiver in the AndroidManifest.xml file. Whenever a shutdown attempt is made to destroy the malicious services, the onDestroy method generates the “RestartSensor” broadcast. The broadcast receiver then restarts the service.


A collection of screenshots of the lines of code handling the auto-restarting of SpyNote.
Figure 7. Steps taken by SpyNote to prevent the killing of its services.

C2 Communcation

As is typical of spyware, the stolen data is useful only if it is sent back to the threat actor. This exfiltration happens by setting up connections with the attacker’s Command and Control (C2) server. Identifying such a C2 server and blocking communications with it can help prevent any personal data exfiltration.

In the SpyNote sample that we are analyzing, C2 communication is established immediately as part of initialization of spyware services. A simple search for “connect” call, reveals the existence of C2 communication. The C2 IP and port are base64 encoded as shown in Figure 8.


A screenshot of the lines of code where connection with SpyNote’s command-and-control server is set up.
Figure 8. C2 communication.

Phone call recording

SpyNote takes spying one step further, and even records incoming phone calls to the victim as .wav files and sends the files to the C2 server. It does this by first granting itself the “READ_CALL_LOG” permission, which is possible thanks to the previously granted ACCESSIBILITY_SERVICE permission, and additionally defining a broadcast receiver for the system broadcast intent “PHONE_STATE”.

When the victim gets an incoming call, the phone state changes, and the broadcast receiver is triggered. The code in the broadcast receiver checks if the victim has answered the call, and once confirmed, starts recording the audio. Figure 9 shows code snippets performing this malicious activity.


Screenshots of SpyNote code snippets performing phone call recording.

Figure 9. Phone call recording

Image capturing/screenshot

The spyware uses MediaProjection API to capture images of the victim's phone.

MediaProjection API lets an application capture device contents that can be recorded or cast to other devices such as TVs. The content is always captured from a real display device. This captured content is then rendered to an intermediate "virtual display" which is the centerpiece of media projection. Finally, the Surface is a consumer of the images captured. It gets these images from the "virtual display" and renders those.
To achieve this, SpyNote registers an onImageAvailableListener, and this listener is called whenever a new image is available. The malware app then directly sends this captured image data as a JPG file to the C2 server. All this is shown in Figure 10.


Screenshots of SpyNote code snippets related to capturing screenshots on the compromised device.

Figure 10. Spyware taking screenshot using ImageReader.

Logging and stealing of data

Spying, logging, and exfiltration of user data are some of the core "features" of any spyware. SpyNote is no different. After a successful launch, it creates a new log file, and starts logging variety of actions performed by the victim.

Keylogging is another core feature of SpyNote, and all keys typed by the victim are logged as Base64 strings by the malware app in its own log file. Due to its own log file, it becomes straightforward for the spyware app to save the snooped user data, such as credentials. For example, we checked SpyNote's log file after unlocking the device screen that required us to type a password. Figure 11 shows the log file created by SpyNote and the screen unlock password captured in it.


Screenshots of the log file created by SpyNote, including entries captured via keylogging.

Figure 11. Spyware infologger.

Some other examples of captured actions include screen on/off, screen unlock password, list of apps shown on the home screen, movement gestures, and so on.

Difficult uninstallation

The goal of all spyware is to maximize its stay on the victim device, and extract as much information as possible. SpyNote which starts its activities as a hidden app, also ensures longer persistence by making the uninstallation process difficult.

As the app is hidden, victims cannot “long press” the app icon and uninstall it. The next option for victims to uninstall apps would be to go to Settings à Apps and uninstall.

However, SpyNote prevents this by closing the menu screen whenever the victim navigates to the app through Settings. This is possible because of the BIND_ACCESSIBILITY_SERVICE permission acquired by the spyware.

The victim cannot stop services of this spyware via developer options either, because the malware has “diehard services” running on the device, as described earlier. The victim is eventually left only with the option of performing a factory reset, losing all data, thereby, in the process.

Summary

The SpyNote sample is spyware that logs and steals a variety of information, including key strokes, call logs, information on installed applications and so on. It stays hidden on the victim’s device making it challenging to notice. It also makes uninstallation extremely tricky. The only option that the victim is left with is performing factory reset to remove the malware.

F-Secure detects this sample as “Malware.Android/Bankbot.FLJI.Gen”.

IOC

- bad77dca600dc7569db4de97806a66fa969b55b77c24e3a7eb2c49e009c1f216
- 37[.]120[.]141[.]140:7775
