Static Analysis of Android APK Package
Lab 2 Part 3

EL Hani Mohammed- Rida

January 8, 2026

Contents

1 Introduction

So here we are again, staring at another suspicious sample, except this time it's not a Windows PE or a Linux ELF, it's an Android APK. And let me tell you, this thing is absolutely screaming "I'm malware!" from the moment you unpack it. The sample weighs in at a hefty 4.43 MiB (4,640,213 bytes to be exact), which is pretty chunky for an Android app that claims to be a simple "Blockchain" application. Spoiler alert: it's not a blockchain app. It's a full-blown Android spyware slash Remote Access Trojan (RAT) that wants to watch everything you do, steal everything you have, and probably send your cat photos to some Russian cybercrime forum.

The SHA256 hash is cc93d01b68b59314a789c5355ac70b8e6965b9f64bb331b0337aac9d2da8aede, and after running it through the usual suspects like apktool, jadx, and Detect It Easy, we discovered that this malicious masterpiece is actually part of the Fantasy Hub malware family. Yes, you read that right. Fantasy Hub. The name sounds like a premium subscription service for nerds, but it's actually a Malware-as-a-Service (MaaS) platform sold on Russian-language cybercrime forums. According to Zimperium researchers, Fantasy Hub comes with full documentation, video tutorials, a subscription model, and even a dropper feature that lets any script kiddie with a few bucks inject their malware into legitimate APKs. It's basically the AWS of Android malware, except instead of hosting websites, you're hosting surveillance operations on people's phones.

The goal of this analysis was simple: figure out what this thing does, how it does it, and how badly it wants to ruin your day. We threw every static analysis tool we could find at it, and what we found was a multi-vector spyware framework that leverages social engineering, permission coercion, and enough obfuscation to make your head spin. This report breaks down the manifest analysis, permission risk assessment, DEX code structure, native library examination, and all the juicy indicators that scream "this is a RAT pretending to be a blockchain app."

2 Sample Properties and Initial Fingerprinting

Let's start with the basics, because even malware has to follow some rules (sort of). Running file on the APK immediately confirms what we already knew: this is an Android package (APK) with an AndroidManifest.xml and an APK Signing Block. The file size is 4,640,213 bytes (4.43 MiB), which is suspiciously large for an app that claims to be a simple utility. Most legitimate Android apps of this type are under 10 MB, but this one is pushing the limits, which suggests that it's packing a lot of functionality (read: malicious capabilities) under the hood.

Using Detect It Easy (DIE), we confirmed that the APK contains DEX (Dalvik Executable) code compiled with dexlib2 using the Android SDK (API 14). The operation system is listed as Android Universal Data Package, and the language is Java, which is standard for Android apps. The APK is compiled for Android 4.0.1-4.0.2 (API level 14), but it targets Android 10 (API level 29) according to the manifest. This discrepancy is interesting because it suggests the malware is trying to maintain compatibility with older Android versions (probably to maximize its infection surface) while also claiming support for modern features.

The apktool.yml file gives us some additional metadata that's worth discussing. The minSdkVersion is 21 which corresponds to Android 5.0 Lollipop, which means the malware can run on devices as old as Android 5.0. The targetSdkVersion is 29 which is Android 10, which is where the malware claims to be targeting. The versionCode is a massive 3,573,638, and the versionName is 5.0.0.0. These version numbers are absurdly high and suggest that this APK was generated by an automated builder or packer, likely part of the Fantasy Hub MaaS infrastructure. Legitimate apps don't have version codes in the millions unless they're doing something very weird with their versioning scheme, and malware authors love to use high version numbers to bypass version-based detection rules.

The forcedPackageId is 127 (0x7f), which is the normal Android resource package ID, so nothing suspicious there. The doNotCompress field lists resources.arsc, png, and jpg, which is standard practice to keep resources readable and fast. The apktool version used for unpacking is 2.7.0-dirty, which is just the version of apktool I used (the dirty tag means the build had uncommitted changes, but that's on my end, not the malware's).

3 Manifest Analysis and the Permission Horror Show

Now we get to the fun part: the AndroidManifest.xml. This is where the malware's true intentions reveal themselves, and let me tell you, this manifest is absolutely terrifying. It's basically a checklist of every dangerous Android permission you can think of, all bundled together in one neat little package of surveillance capitalism. The manifest is so packed with red flags that it might as well be a Soviet military parade.

First, let's talk about the app's identity. The package name is obfuscated with a long random string (typical for malware generated by builders), and the app label is listed as "Blockchain" with a version name of 5.0.0.0. The app is targeting SDK version 29 (Android 10) and requires a minimum SDK of 21 (Android 5.0). So far, so normal. But then we get to the permissions section, and that's where things go completely off the rails.

3.1 Permission Risk Assessment

The manifest requests a staggering number of dangerous permissions. Let me categorize them by risk level, because there are a lot of them and they all deserve individual attention.

Starting with the HIGH RISK Permissions related to surveillance and data theft, we have CAMERA which gives full access to the device camera for photo and video capture. Then there's RECORD AUDIO for microphone access allowing audio recording and surveillance. The malware also requests ACCESS FINE LOCATION and ACCESS COARSE LOCATION for GPS and network-based location tracking. It wants READ SMS and SEND SMS which is the classic combination for OTP interception and SMS relay attacks. READ CALL LOG gives access to call history for profiling and tracking purposes. READ CONTACTS allows the malware to steal the user's entire contact list. GET ACCOUNTS provides access to device accounts like Google and email accounts for potential account takeover scenarios. Finally READ PHONE STATE gives access to phone number, IMSI, and device identifiers.

Now let's talk about the CRITICAL Permissions that give full device control. The big one is BIND ACCESSIBILITY SERVICE with priority set to 999. This is the crown jewel of Android malware. Accessibility services can read everything on screen, simulate touches, grant permissions on behalf of the user, and basically take full control of the device UI. Then we have BIND DEVICE ADMIN which grants device administrator privileges, making the app extremely difficult to uninstall. SYSTEM ALERT WINDOW allows the malware to draw overlays on top of other apps, which is the classic phishing and UI hijacking technique. USE FULL SCREEN INTENT lets it display full-screen notifications and prompts, used to coerce users into granting permissions. BIND VPN SERVICE creates a local VPN connection to intercept and monitor all network traffic, and it's cleverly labeled as "Firewall Services" to appear legitimate. Finally BIND INPUT METHOD allows the malware to register as a custom keyboard, essentially an IME keylogger disguised as "Simple Keyboard".

The MEDIUM RISK Permissions are focused on persistence and installation capabilities. RECEIVE BOOT COMPLETED enables auto-start after device reboot. REQUEST IGNORE BATTERY OPTIMIZATIONS allows the malware to bypass battery optimization to run in background indefinitely. REQUEST INSTALL PACKAGES and REQUEST DELETE PACKAGES give the ability to install and uninstall apps, which is classic dropper behavior. WRITE EXTERNAL STORAGE and READ EXTERNAL STORAGE provide full file system access

for data exfiltration.

The permission combination here is absolutely devastating. The BIND ACCESSIBILITY SERVICE alone is enough to compromise a device completely, because it allows the malware to read every password, OTP code, and piece of sensitive information displayed on screen. Combined with SYSTEM ALERT WINDOW for overlays and BIND DEVICE ADMIN for persistence, this malware can grant itself additional permissions, prevent uninstallation, and overlay fake login screens on top of legitimate apps to steal credentials. The BIND VPN SERVICE permission is particularly nasty because it allows the malware to intercept all network traffic by routing it through a local VPN. This is MITM-style traffic inspection without needing root access.

3.2 Component Analysis

The manifest declares multiple components that reveal the malware's modular architecture. Let's break down the key ones because they tell us exactly how this thing operates.

Under Activities, we find multiple activities with names like RequestAdmin, RequestScreenCap, RequestKeyboard, and RequestVPN. These are all social engineering flows designed to trick the user into granting dangerous permissions. There are also activities labeled "Blockchain", "Play Store", and "Settings" that use excludeFromRecents and translucent themes to masquerade as legitimate system apps and appear trustworthy. We also see FloatingView and Over activities which are UI overlays specifically designed for phishing and credential theft. Many activities have weird obfuscated class names which is classic malware obfuscation trying to make reverse engineering more annoying.

Looking at the Services section, we find the AccessibilityService labeled "Touch Helper" with priority 999. This is the crown jewel of Android malware because this service can do literally anything the user can do, automatically. There's a KeyboardService labeled "Simple Keyboard" which is an IME keylogger for capturing every keystroke. The VpnService is labeled "Firewall Services" for traffic interception and monitoring. Multiple foreground services are declared with FOREGROUND SERVICE permission to ensure the malware stays running in the background.

The Receivers section shows us the persistence mechanisms. There's a BootReceiver with RECEIVE BOOT COMPLETED for auto-start on reboot. A ScreenReceiver triggers on screen on and screen off events. PackagesReceiver monitors for package added, removed, and replaced to track app installations. Finally DeviceAdminReceiver handles device admin enabled for device admin persistence.

The manifest also includes queries tags that target specific social media and messaging apps including WhatsApp, Instagram, Facebook Messenger, and their Lite variants. This suggests the malware is specifically designed to target users of these popular apps, likely for OTP interception, credential theft, or message monitoring.

4 DEX Code Analysis

After unpacking the APK with apktool, we dove into the smali code to see what horrors await us. The package structure is heavily obfuscated with random-looking package names, but the functionality is crystal clear once you know what you're looking for. This is a modular spyware framework with separate components for each capability.

4.1 Code Structure and Obfuscation

The main package structure consists of multiple obfuscated packages, each containing specific functionality. The obfuscation uses short, random-looking names that don't follow normal Java naming conventions, which is a classic indicator of automated obfuscation or builder-generated

code. However, some class names are still descriptive enough to reveal their purpose, which tells us the malware authors weren't quite as careful as they should have been.

We found RequestAdmin.smali which prompts the user to grant Device Admin privileges with social engineering text. There's RequestVPN.smali that prompts for VPN service permissions under the guise of "Firewall Services". RequestKeyboard.smali and KeyboardService.smali implement the IME keylogger masquerading as system keyboard. RequestScreenCap.smali and HandleScreenCap.smali handle screen capture functionality for surveillance purposes.

The accessibility abuse is implemented in AccessService.smali, PerformGesture.smali, and ExecClick.smali for UI automation and auto-clicking capabilities. There are device-specific files like ClickUnlock.smali and OppoUnlock.smali that target specific manufacturers with custom unlock automation. FloatingView.smali implements the overlay UI for phishing and credential harvesting. RecordPayPass.smali is particularly damning because the name literally says it all, it's for payment PIN and password capture. RecordScreen.smali handles screen recording for extended surveillance.

The persistence mechanisms are in BootReceiver.smali, ScreenReceiver.smali, and PackagesReceiver.smali for event-based triggers that keep the malware running and responsive to system events.

But wait, there's more! We also found smali files for fake cryptocurrency exchange apps. There's Binance.smali, Okex.smali, and Trust.smali which are fake exchange wallet interfaces. MyWebView.smali, AndroidInterface.smali, and MyWebChromeClient.smali implement WebView-based phishing with JavaScript bridge for stealing inputs in real-time.

This confirms that the malware not only masquerades as a "Blockchain" app, but it also includes fake WebView interfaces designed to look like legitimate cryptocurrency exchanges. Users who think they're logging into Binance or Trust Wallet are actually entering their credentials into a phishing WebView controlled by the malware.

4.2 Suspicious Methods and Malicious Behavior

Let me highlight three particularly egregious examples of malicious code that really showcase the sophistication of this malware.

First, let's talk about the Accessibility Service Abuse. The AccessibilityService is configured with a priority of 999 and has event types set to capture ALL UI events. This means the service receives every single touch, click, scroll, and text input on the device. The service includes methods for performing gestures, executing clicks, and device-specific unlock methods. This allows the malware to automatically click through permission prompts, essentially granting itself additional permissions without user consent. It can read OTP codes from SMS and messaging apps in real-time. It can extract credentials from login screens by reading the text fields. It can simulate user gestures to bypass security measures and anti-automation checks. And it can unlock the device automatically using manufacturer-specific techniques.

Second, we have the IME Keylogger. The KeyboardService registers as a legitimate input method which means it pretends to be a keyboard app. It's labeled "Simple Keyboard" or "System Keyboard" to appear trustworthy and system-level. Once enabled by the user, it captures every keystroke the user types across all apps. The captured data is likely sent to a remote server or stored locally for later exfiltration. The use of an IME keylogger is particularly insidious because users often don't realize that third-party keyboards can log everything they type, including passwords, credit card numbers, and private messages.

Third, there's the WebView Phishing mechanism. The malware includes fake cryptocurrency exchange interfaces built with WebView components. These interfaces use a JavaScript bridge to intercept user input in real-time. When a user enters their exchange login credentials or wallet seed phrase into the fake WebView, the JavaScript bridge captures the data and passes it to the malware's code for exfiltration. The WebView chrome client handles file uploads and other events to make the phishing page feel more realistic and interactive.

5 Resource Analysis

The res directory contains the usual Android resources like layouts, drawables, and strings, but there are a few suspicious elements worth noting. The res/raw directory contains an empty file named discountc69, which is typical of placeholder resources used by malware builders. The name follows the same obfuscation pattern as other resource files where a random word is followed by a random number.

The purpose of an empty raw resource is usually to serve as a trigger or placeholder. The malware might check for the presence of this file, and if it's missing or has zero length, the malware switches to a fallback behavior like fetching a payload from a remote server. Alternatively, it could just be noise added by the builder to confuse analysts.

The string resources contain a lot of Turkish language text. Strings suggest either Turkish authorship or Turkish-speaking victims as the primary target. The informal tone of these strings indicates the work of a less professional threat actor, probably someone who bought access to the Fantasy Hub platform and customized it for their specific campaign.

6 Attribution and Classification

Based on the static analysis, we can confidently attribute this sample to the Fantasy Hub malware family. The attribution is supported by multiple indicators.

The string "spywarefantasydums272" found in the resources directly references Fantasy Hub. The modular architecture with separate Request flows for each permission type matches the Fantasy Hub MaaS model perfectly. The social engineering focus is consistent with Fantasy Hub's documented behavior. The inclusion of fake exchange WebViews and the "Blockchain" masquerade aligns with Fantasy Hub's templates. The Turkish language strings suggest a Fantasy Hub customer targeting Turkish-speaking victims.

The malware classification is Android Spyware and Remote Access Trojan from the Fantasy Hub MaaS platform. Primary capabilities include device surveillance, data theft, UI hijacking, persistence, traffic interception, and keylogging. Distribution method is social engineering via fake app stores. Target audience is Turkish-speaking Android users with focus on cryptocurrency enthusiasts.

7 Indicators of Compromise

Key IOCs for detection include the SHA256 hash cc93d01b68b59314a789c5355ac70b8e6965b9f64bb331b0337aac and MD5 hash e776598b0acc34d893c85b7adbe442e9. The app label is "Blockchain" with version code 3573638. Service names include "Touch Helper", "Simple Keyboard", and "Firewall Services". The dangerous permission combination is BIND ACCESSIBILITY SERVICE plus BIND DEVICE ADMIN plus SYSTEM ALERT WINDOW plus BIND VPN SERVICE. The string marker "spywarefantasydums272" can be used for detection. Behavioral IOCs include auto-granting permissions via Accessibility, screen overlay on banking apps, and VPN traffic interception.

8 Conclusions and Recommendations

This APK is a textbook example of modern Android spyware. It's modular, heavily obfuscated, uses social engineering to coerce permissions, and leverages Android's accessibility and admin APIs for full device control. The fake cryptocurrency exchange interfaces target crypto users specifically.

The attribution to Fantasy Hub is solid based on string markers, behavioral patterns, and architectural similarities. The MaaS model explains the high version code, modular permission

flows, and Java-only implementation. This is mass-produced commodity spyware sold to low-skill cybercriminals.

Detection recommendations include monitoring for the deadly permission combination, flagging high version codes, detecting obfuscated packages with generic labels, and scanning for Fantasy Hub string markers.

For end users, never grant accessibility permissions to non-accessibility apps, never grant device admin to non-MDM apps, never install APKs from outside Play Store, and be suspicious of apps requesting multiple dangerous permissions.

Static analysis limitations mean we couldn't determine the C2 protocol, exfiltration frequency, root exploits, configuration data, or full Accessibility automation. Dynamic analysis would be needed for these.

This is a Fantasy Hub spyware sample masquerading as a Blockchain app, targeting Turkish users with focus on cryptocurrency theft and device surveillance. The MaaS distribution model means many more variants will appear. Stay safe and say no to accessibility permissions.