A stealthy threat uncovered: TeaBot on Google Play Store

By Francesco Iubatti, Federico Valentini

Archived: 2026-04-05 23:02:39 UTC

Key Points

- Starting from the second week of February, we have observed via our telemetry data an increase in TeaBot banking trojan infections across **several European countries**.
- Our investigations have revealed that the initial stage of infection originates from an application available on the official **Google Play Store**, with over 100,000 downloads (before being removed from the Google Play Store).
- This application serves as a **dropper**, facilitating the download of a banking trojan of the TeaBot family through multiple stages.
- Before downloading the banking trojan, the dropper performs advanced evasion techniques, including obfuscation and file deletion, alongside multiple checks about the victim countries.
- Upon successful installation, facilitated through a tricky update popup, threat actors (TAs) gain the ability to execute banking fraud via **Account Takeover** (ATO) attacks.

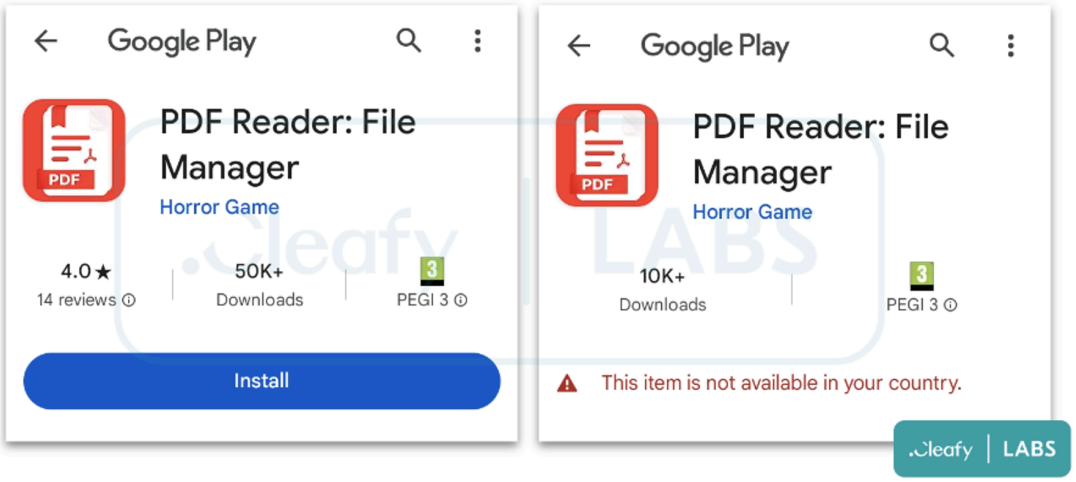

Figure 1 - Requesting the same app, on the Google Play Store, from different countries

Introduction & Background

While **phishing campaigns** have traditionally been the primary method for distributing banking trojans, in recent years, there has been a notable trend where TAs utilise official **app stores** to disseminate their malware. This is achieved by deploying **dropper applications** designed to download malware onto the targeted device. One of the main motivations behind this method is the ability to reach a large **pool of potential victims**, thereby increasing the likelihood of successful fraudulent transactions.

Since the second week of February, we have detected a rise in TeaBot banking trojan infections across multiple European countries through our telemetry data. Our investigations have revealed that the initial stage of infection originates from an application available on the official **Google Play Store**. This application dynamically downloads additional code and configuration files, effectively evading detection.

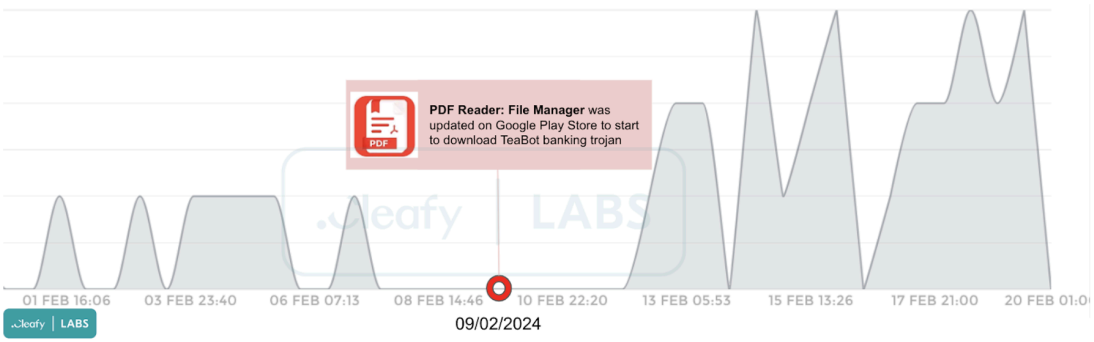

Figure 2 - TeaBot's infectionst elemetry from Cleafy data

In the next section of the article, we will describe the critical points of each stage of infection, which can be summarised as follows:

- Stage 1: The initial phase of infection starts with an application found on the official Google Play Store. This application is exclusively available to users in specific countries and disguises itself as a common utility application. It acts as a dropper, facilitating the download of additional code from a designated server.
- Stage 2: The downloaded code, in the form of a .dex file, is utilised to conduct further assessments of the user device. Specifically, it verifies whether the device is located within one of the specific countries and determines if it is operating on an actual device or within an emulator or sandbox.
- Stage 3: Once the checks are completed, an additional APK is downloaded and installed on the user's device. The APK is responsible for unpacking a file containing malicious code within itself, thus enabling TAs to carry out fraudulent operations.

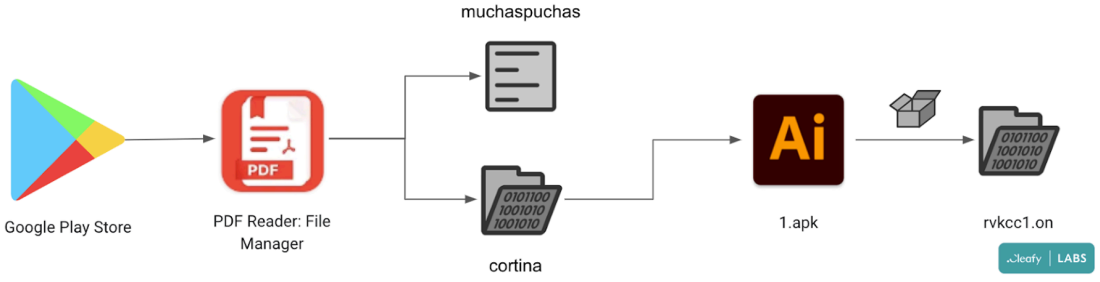

Figure 3 - Infection chain overview

Technical Analysis

First Stage: Dropper Application from Google Play

The "PDF Reader: File Manager" application appears as a genuine **PDF file manager** and is exclusively accessible for download in targeted countries. This application was released on the Google Play Store on January 31, 2024, and updated on February 9, 2024. Upon our discovery, and a few days after its update, the application had already been downloaded over 10,000 times. However, it reached over 100,000 downloads before being removed from the store.

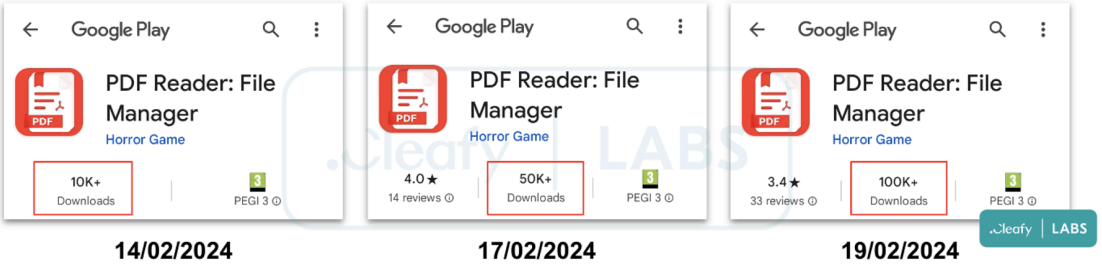

Figure 4 - Number of downloads of the dropper application from Google Play Store

The dropper application uses a classical behaviour, requesting REQUEST_INSTALL_PACKAGE and WRITE_EXTERNAL_STORAGE permissions to manage the additional code needed. However, unlike some earlier campaigns, where the banking trojan was downloaded directly from a server or GitHub repositories, in this campaign, several steps and checks are performed before downloading TeaBot.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android" android:versionCode="2" android:versionName="1.0.1" android:
compileSdkVersion="34" android:compileSdkVersionCodename="14" package="com.tragisoap.fileandpdfmanager" platformBuildVersionCode="34"
platformBuildVersionName="14" xmlns:ns1="http://schemas.android.com/apk/distribution" ns1:requiredSplitTypes="base__abi,base__density" xmlns
:ns2="http://schemas.android.com/apk/distribution" ns2:splitTypes="">
    <uses-sdk android:minSdkVersion="30" android:targetSdkVersion="34"/>
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" android:maxSdkVersion="28"/>
    <uses-permission android:name="android.permission.MANAGE_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.REQUEST_INSTALL_PACKAGES"/>
    <queries>
        <provider android:authorities="com.ixhailes.trast"/>
    </queries>
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <permission android:name="com.tragisoap.fileandpdfmanager.DYNAMIC_RECEIVER_NOT_EXPORTED_PERMISSION" android:protectionLevel="signature"
    />
    <uses-permission android:name="com.tragisoap.fileandpdfmanager.DYNAMIC_RECEIVER_NOT_EXPORTED_PERMISSION"/>
    <application android:theme="@style/Theme.FileAndPDFManager" android:label="@string/app_name" android:icon="@mipmap/ic_launch
```

Figure 5 - Snippet of the AndroidManifest.xml file

Once the application is downloaded and installed, a service component within the dropper application performs two HTTP GET requests to retrieve the following files:

1. **muchaspuchas:** a string containing multiple Java methods and keywords, delimited by the "|" character, leveraged by the dropper to load the additional dex file at run-time;
2. **cortina:** a .dex file containing additional code imported by the application.

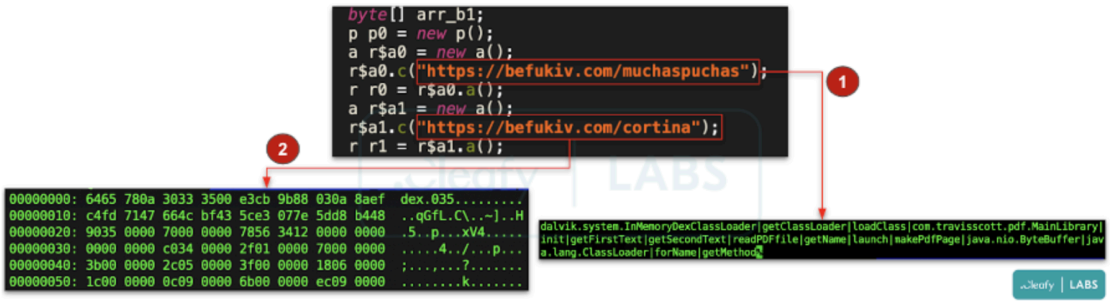

Figure 6 - HTTP GET requests for downloading and executing the additional .dex file

In the next paragraph, we will move to the second stage of the infection chain by describing some of the features in the dex file.

Second Stage: dex file

TeaBot dropper employs sophisticated evasion techniques to avoid detection. One such technique involves the dynamic downloading of a secondary executable as a dex file from a dedicated C2 server. By adopting this modular approach, TAs can dynamically rotate the payload contained within the dex file, enhancing their ability to evade detection by security measures. In fact, during this campaign, from February 9th to 19th, 2024, the malware downloaded multiple dex with minor modifications.

The purpose of the additional code within the dex file is to conduct further checks on the user's device, specifically:

- Verify if the device is an emulator by examining various information such as model, manufacturer, etc.
- Checking whether the ISO-3166-1 country code of the current registered operator corresponds to specific countries, including:

  es: Spain
  sk: Slovakia
  cz: Czech Republic
  ru: Russia
  hr: Croatia
  si: Slovenia
  sl: Sierra Leone
  bg: Bulgaria
  ee: Estonia
  fi: Finland
  ie: Ireland
  gb: United Kingdom (Great Britain)

An interesting aspect concerning the targeted countries is that Russia and CIS regions are typically excluded during the installation phases in multiple malware campaigns. However, in the case of this TeaBot campaign, Russia stands out as one of the targets

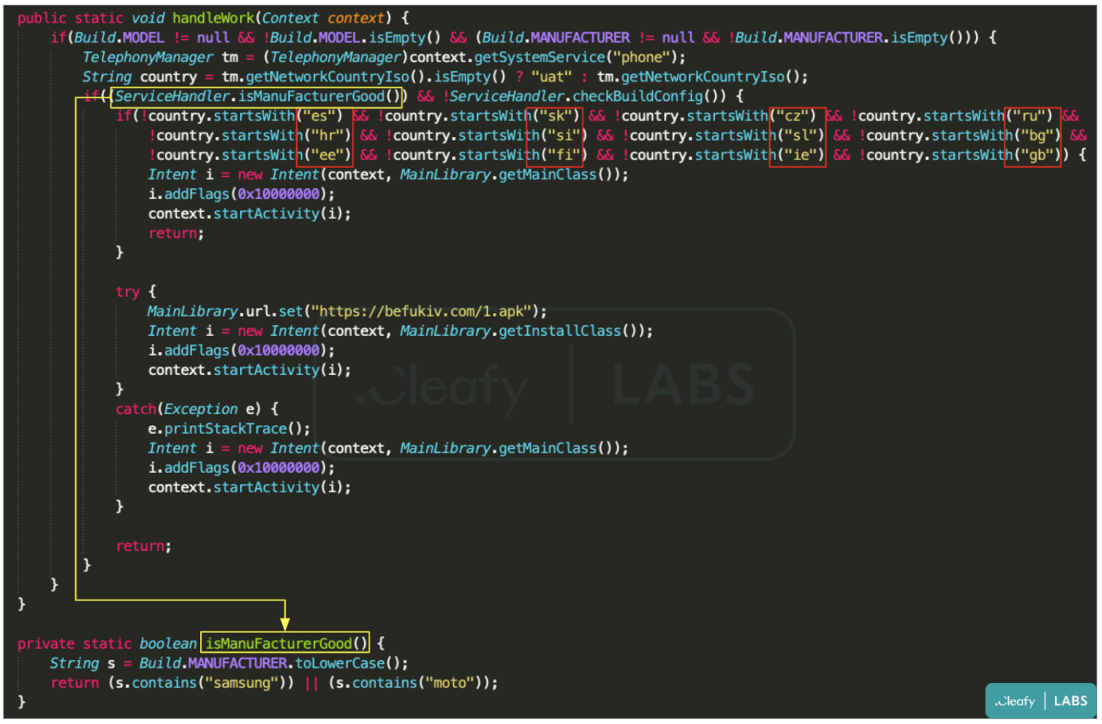

Figure 7 - Multiple checks used before downloading the TeaBot banking trojan

If the user device passes the checks mentioned above, a new application (APK file) is downloaded onto the device. To accomplish this, TAs employ a well-established strategy that shows a fake request update, deceiving the user:

```
switch(Locale.getDefault().getLanguage()) {
    case "bg": {
        return "PDF Reader: File Manager\n Трябва да се актуализира";
    }
    case "cs":
    case "cz": {
        return "PDF Reader: File Manager\n Je třeba aktualizovat";
    }
    case "de": {
        return "PDF Reader: File Manager\n Muss aktualisiert werden";
    }
    case "ee":
    case "et": {
        return "PDF Reader: File Manager\n Vajab värskendamist";
    }
    case "fi": {
        return "PDF Reader: File Manager\n On päivitettävä";
    }
    case "hr": {
        return "PDF Reader: File Manager\n Treba ga ažurirati";
    }
    case "lt": {
        return "PDF Reader: File Manager\n Reikia atnaujinti";
    }
    case "nb":
    case "no": {
        return "PDF Reader: File Manager\n Må oppdateres";
    }
    case "si":
    case "sl": {
        return "PDF Reader: File Manager\n Treba je posodobiti";
    }
    case "sk": {
        return "PDF Reader: File Manager\n Je potrebné aktualizovať";
    }
    default: {
        return "PDF Reader: File Manager\n Needs to be updated";
```

Figure 8 - Fake update messages used by the dropper to download TeaBot

Third Stage: APK file

The last step in the infection is to download the actual TeaBot APK (1.apk). However, although there are several suspicious permissions within the AndroidManifest.xml file, including the notorious AccessibilityServices, there are no accessibility abuse functions in the code. This is due to the fact that the app unpacked a file in the asset directory (called ‘rvkcc1.on’) containing all the malicious features of the banking trojan TeaBot.

```
static {
    c.b = b.b(StringPool.YwgoTC(), 2);  // 思怵恕 // .on
    c.c = b.b(StringPool.tmyJRvb(), 2);  // 恁恅惴恐恐 // rvkcc
    c.d = b.b(StringPool.FhzqF(), 2);  // 恔恕恩恞惛恚恼 // gfemdiy
    c.e = b.b(StringPool.H(), 2);  // 忿态怀怪怆怅怄怅恨怄怅怆怅怀态忿态怀怪怆怅怄怅怀态怀怪怆怅怄怅恨怄怅怄恨怀恨怀 // 12345679876543212345
    c.f = b.b(StringPool.UZxV(), 2);  // $DATA
    c.h = b.b(StringPool.Imn(), 2);  // 惨恩恐怵恕惛恒恁协柚惛恩恋恩惨 // secondary-dexes
    c.i = b.b(StringPool.mA(), 2);  // 恇恆恟恇恚忿恩恋思恟怵恐惴 // MultiDex.lock
    c.j = b.b(StringPool.PHiIfT(), 2);  // 思恐恟恒惨惨恩惨 // .classes
    c.k = b.b(StringPool.rkDGGuzc(), 2);  // 思悁恚恃 // .zip
    c.l = b.b(StringPool.hbSp(), 2);  // 恐怵惛恩恬恐恒恐恛恩 // code_cache
}
```

Figure 9 - Some of the deobfuscated strings

Once the rvkcc1.on file is unpacked to a dex file, the code is imported into the app, and the dex is removed from the app directory. At this point, the malware is installed, and as usual, it requests the Accessibility permissions during the installation phases to obtain complete control of the infected device.

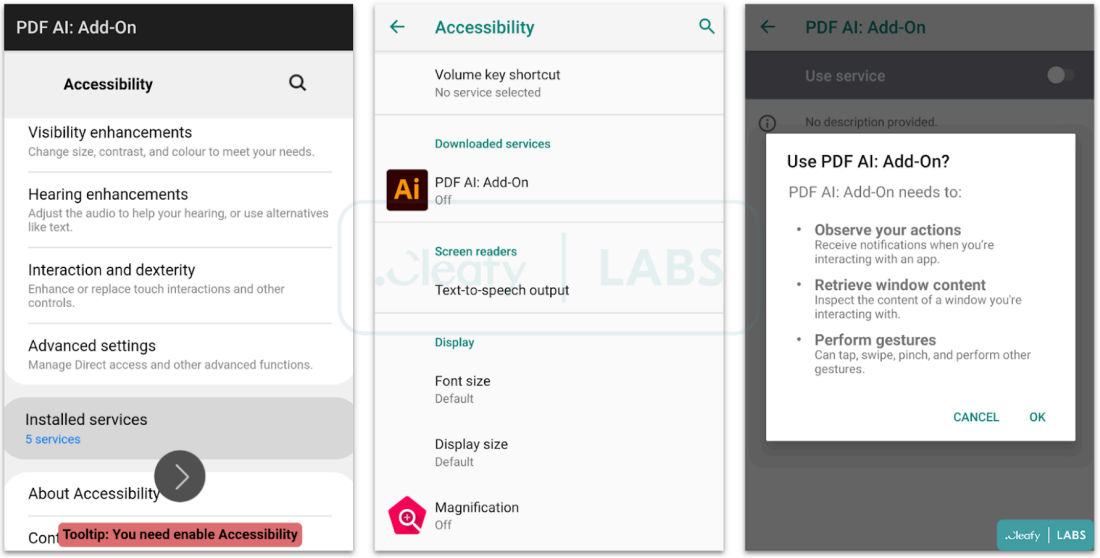

Figure 10 - Installation phases of TeaBot

An interesting fact about this recent sample of TeaBot is the introduction of code dedicated to the Revolut banking application. Specifically, if the victim has installed the Revolut app, the malware can use accessibility features to retrieve the Revolut app and the victim's balance, as shown:

```
CharSequence charSequence0 = accessibilityEvent0.getPackageName();
if(charSequence0 != null && (charSequence0.equals("com.revolut.revolut"))) {
    List list0 = accessibilityNodeInfo0.findAccessibilityNodeInfosByViewId("com.revolut.revolut:id/endLabel");
    StringBuilder stringBuilder0 = new StringBuilder("Revolut total VAL: ");
    for(Object object0: list0) {
        stringBuilder0.append(((AccessibilityNodeInfo)object0).getText());
        stringBuilder0.append(" ,");
    }

    List list1 = accessibilityNodeInfo0.findAccessibilityNodeInfosByViewId("com.revolut.revolut:id/row_valuePrimary");
    StringBuilder stringBuilder1 = new StringBuilder("Revolut per currency VAL: ");
    for(Object object1: list1) {
        stringBuilder1.append(((AccessibilityNodeInfo)object1).getText());
        stringBuilder1.append(" ,");
    }

    f.a(stringBuilder0.toString());
    f.a(stringBuilder1.toString());
    System.out.println(stringBuilder0.toString());
    System.out.println(stringBuilder1.toString());
}
```

Figure 11 - Snippet of code used to retrieve Revolut app information

Conclusions

TeaBot was discovered back in January 2021 by the Cleafy Threat Intelligence and Incident Response team as an emerging threat since it appears to be in its early stages of development, according to some irregularities found during our initial analysis.

During the last few years, we noticed constant updates behind this threat, including new targets, new evasion techniques, and supporting new languages, including Russian, Slovak, and Mandarin Chinese.

This last investigation aims to show how the TAs behind TeaBot are still improving their techniques to remain undetected and maximise their efficiency, refining advanced evasion techniques and leveraging the official Google Play store as the favourite distribution method.

Additionally, it is noteworthy to mention a peculiar aspect uncovered during our investigations: the potential targeting of **Russian citizens** in the TeaBot campaign. This finding is based on the configurations extracted from this last research. It is worth noting that, in banking fraud, botnets and within much of the underground forum community, countries within the Commonwealth of Independent States (CIS) are typically excluded a priori. This exclusion is often enforced through specific techniques embedded within the malware's source code. Including Russian targets in this TeaBot campaign deviates from this norm, suggesting a potentially expanded scope of operation or a strategic shift in targeting priorities by the TAs behind the threat.

Related works

TeaBot: a new Android malware emerged in Italy, targets banks in Europe (May 2021) - link

TeaBot is now spreading across the globe (March 2022) - link

Appendix 1: list of IoCs

IoC	Description
a325ba7810b0791d2c6c4757ae4fe074	Md5 - Dropper
com.tragisoap.fileandpdfmanager	Package name - Dropper
6a108e97eb659eded211f25eb6649989	MD5 - dex file (cortina)
https://befukiv.]com	Dropper server
17deac9f87fedb65c323f13beaa6a1ce	MD5 - TeaBot (1.apk)
185.215.113.]31	TeaBot C2 server
91.215.85.]55	TeaBot C2 server
