TeaBot: a new Android malware emerged in Italy, targets banks in Europe

By Federico Valentini, Francesco Iubatti

Archived: 2026-04-05 16:37:36 UTC

Key Points

- At the beginning of January 2021, a new Android banking trojan was discovered and analyzed by our Threat Intelligence and Incident Response (TIR) team. We decided to dub this new family as **TeaBot** since it seems to not be related to any known banking trojan family
- The main goal of **TeaBot** is stealing victim's credentials and SMS messages for enabling frauds scenarios against a predefined list of banks (**more than 60 targeted banks were extracted**)
- Once **TeaBot** is successfully installed in the victim's device, attackers can obtain a live streaming of the device screen (on demand) and also interact with it via Accessibility Services
- On 29th March 2021, we detected for the first time the inclusion of injections against Italian banks
- Also, at the beginning of May 2021, we detected for the first time also the inclusion of injections against Belgium and Netherlands banks
- At the time of writing, **TeaBot** appears to be at its early stages of development according to some irregularities found during our analysis
- For the sake of completeness, after our investigation we noticed that the name 'Anatsa' is also used for tracking this malware family

Executive Summary

At the beginning of January 2021, a new Android banker started appearing and it was discovered and analysed by our Threat Intelligence and Incident Response (TIR) team.

Since lack of information and the absence of a proper nomenclature of this Android banker family, we decide to dub it as **TeaBot** to better track this family inside our internal Threat Intelligence taxonomy.

TeaBot appears to have all the main features of nowadays Android bankers achieved by abusing Accessibility Services such as:

- Ability to perform Overlay Attacks against multiple banks applications to steal login credentials and credit card information
- Ability to send / intercept / hide SMS messages
- Enabling key logging functionalities
- Ability to steal Google Authentication codes
- Ability to obtain full remote control of an Android device (via Accessibility Services and real-time screen-sharing)

Thanks to an in-depth analysis of a new wave of samples detected at the end of March 2021, we found, for the first time, multiple payloads against Italian banks.

Also, **TeaBot** appears to be at its early stages of development according to some irregularities found during our analysis, but developers have already included multi-languages support according to some textual references found (e.g. Spanish, Italian, German, etc.).

We assume that TeaBot, similar to Oscorp, is trying to achieve a real-time interaction with the compromised device combined with the abuse of Android Accessibility Services bypassing the need of a "new device enrollment" to perform an Account Takeover scenario (ATO).

TeaBot – Static Analysis

From the **AndroidManifest** file the following indicators were extracted:

- Initially, the app name used by the malicious app was "**TeaTV**" however during the last month the app name was changed to "**VLC MediaPlayer**", "**Mobdro**", "**DHL**", "**UPS**" and "**bpost**", the same decoy used by the famous banker Flubot/Cabassous
- The main permissions achieved by **TeaBot** allow to:
  - o Send / Intercept SMS messages
  - o Reading phone book and phone state
  - o Use device supported biometric modalities
  - o Modify audio settings (e.g. to mute the device)
  - o Shows a popup on top of all other apps (used during the installation phase to force the user to accept the accessibility service permissions)
  - o Deleting an installed application
  - o Abusing Android Accessibility Services

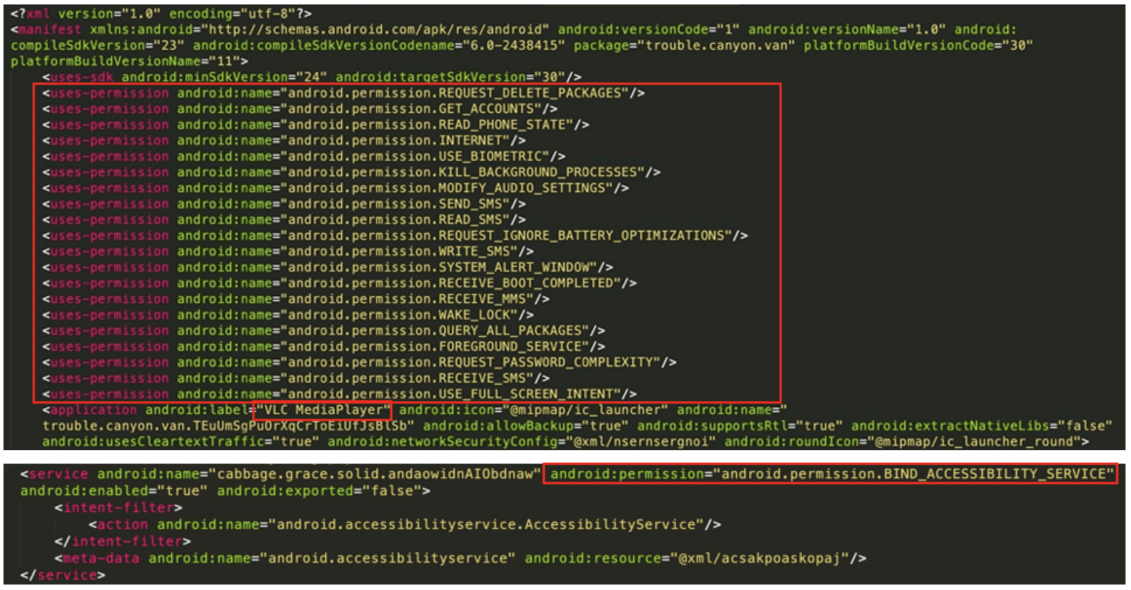

Figure 1 - List of permissions declared in the AndroidManifest.xml

Figure 2 - Main icons app used by TeaBot

TeaBot, like other bankers, uses multiple techniques to slow down analysts, such as:

- The malicious application acts as dropper and dynamically loads a 2nd stage (.dex) where all the malicious code resides
- Usage of “Junk Code”
- Network communications are partially encrypted using XOR algorithm

Furthermore, both the partial network encryption and the presence of some not-working injections and commands (or in some cases a lack of injections for specific targeted banks) suggest to us that **the TeaBot is still under development.**

At the same time, a couple of interesting changes were detected:

- In January 2021, TeaBot was focused only on Spanish banks
- In March 2021, **new samples of TeaBot appeared with also German and Italian banks as targets for the first time**. Also, TeaBot is currently supporting 6 different languages (Spanish, English, Italian, German, French and Dutch):

Figure 3 - Supports of multiple languages found

TeaBot main features

The main features observed during the analysis of the banker are the following.

Keylogging: Through the abuse of the Android Accessibility Services, TeaBot is able to observe and track all the information performed by the user on the targeted applications. We observed similar behavior also in another banker called EventBot, but with the difference that EventBot tracks any apps while **TeaBot tracks only targeted apps**, therefore less traffic is generated between the banker and the C2. TeaBot, during its first communications with the C2, sends the list of installed apps to verify if the infected devices had one or more targeted apps already installed. When TeaBot found one of them, it downloads the specific payload to perform overlay attacks and starts tracking all the activity performed by the user on the targeted app. Those information are sent back to the assigned C2 every 10 seconds.

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <boolean name="kloger:it.                    " value="true" />
    <boolean name="kloger:es.                    " value="true" />
    <boolean name="kloger:it.                    " value="true" />
    <boolean name="kloger:net.                                  " value="true" />
    <boolean name="kloger:com.                   " value="true" />
    <string name="domains">http://185.215.113.31:80/api/|http://178.32.130.170:80/api/</string>
    <boolean name="kloger:es.                    " value="true" />
    <boolean name="lock_device" value="false" />
    <boolean name="kloger:com.                   " value="true" />
    <boolean name="shown_unsup" value="true" />
    <boolean name="kloger:com.                   t" value="true" />
    <string name="captured_injects"></string>
    <boolean name="kloger:com.                   " value="true" />
    <boolean name="kloger:es.                    " value="true" />
    <boolean name="kloger:com.                   " value="true" />
    <string name="def_sms_manager">com.android.messaging</string>
    <boolean name="kloger:com.                   " value="true" />
    <boolean name="kloger:com.                   er" value="true" />
    <boolean name="kloger:es.                    s" value="true" />
    <boolean name="kloger:                                    l" value="true" />
    <boolean name="kloger:es.                    e" value="true" />
    <long name="local_keylogger_versions" value="4" />
    <boolean name="kloger:es.                    l" value="true" />
    <long name="local_injects_versions" value="4" />
    <boolean name="kloger:com.                   " value="true" />
    <boolean name="hide_sms" value="false" />
    <boolean name="kloger:com.                   g" value="true" />
    <boolean name="kloger:com.                   " value="true" />
    <string name="logged_sms"></string>
    <boolean name="kloger:es.                                   " value="true" />
    <boolean name="kloger                                       " value="true" />
</map>
```

Figure 4 - Example of TeaBot's configuration file

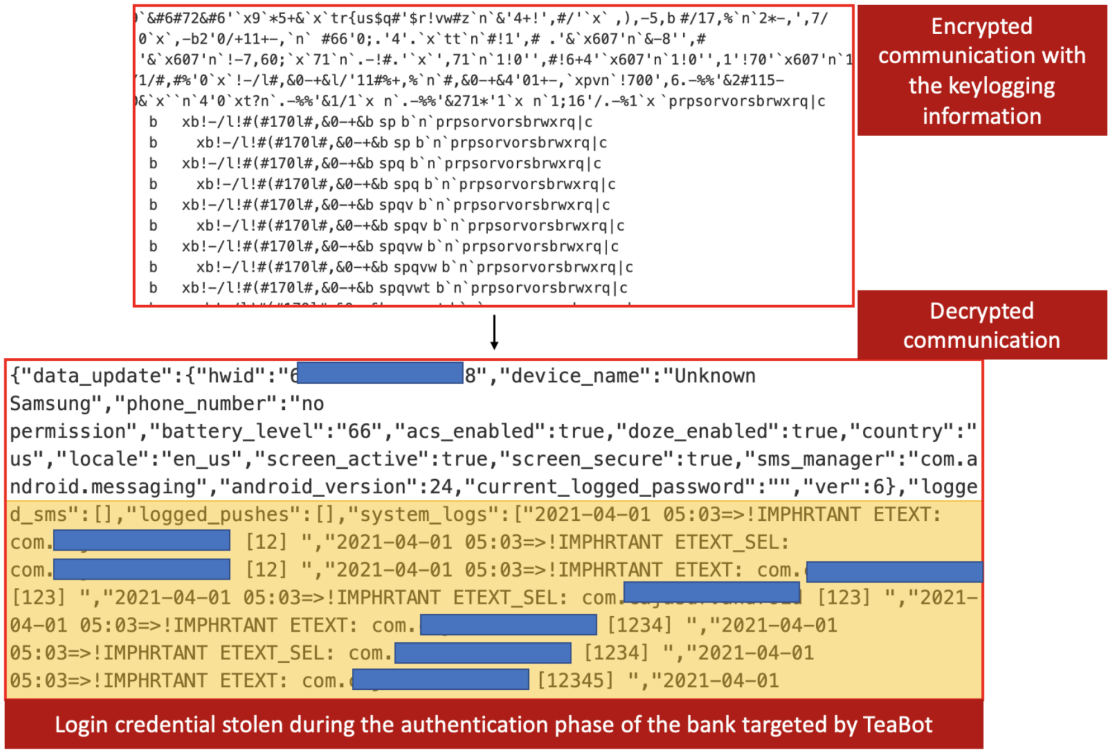

Figure 5 - Example of bank credential stolen by TeaBot

Screenshots: One of the particularities of TeaBot is the capability of taking screenshots to constantly monitor the screen of the compromised device. When the C2 sends the "**start_client**" command with an IP address and PORT, it starts requesting the images and TeaBot starts a loop in which creates a "VirtualScreen" for taking screenshots.

```
private void h() {
    if(jd98awdAWHndoia.f != null) {
        File v0 = this.getExternalFilesDir(null);
        if(v0 != null) {
            jd98awdAWHndoia.k = v0.getAbsolutePath() + "/screenshots/";
            File v0_1 = new File(jd98awdAWHndoia.k);
            if(!v0_1.exists() && !v0_1.mkdirs()) {
            }
        }
    }
}

@SuppressLint({"WrongConstant"})
private void i() {
    DisplayMetrics v0 = this.getResources().getDisplayMetrics();
    this.b = v0.densityDpi;
    int v1 = v0.widthPixels;
    this.c = v1;
    int v0_1 = v0.heightPixels;
    this.d = v0_1;
    ImageReader v0_2 = ImageReader.newInstance(v1, v0_1, 1, 2);
    jd98awdAWHndoia.i = v0_2;
    jd98awdAWHndoia.h = jd98awdAWHndoia.f.createVirtualDisplay("DEMO", this.c, this.d, this.b, 16, v0_2.getSurface()
    jd98awdAWHndoia.i.setOnImageAvailableListener(new c(this, null), jd98awdAWHndoia.g);
    jd98awdAWHndoia.m.set(true);
}
```

Figure 6 - Snippet of TeaBot's code for taking screenshots of the compromised devices

Overlay attack: "*The Overlay attack is a well-known technique implemented on modern Android banking trojans (e.g. Anubis, Cerberus/Alien) which consist of a malicious application/user somehow able to perform actions on behalf of the victim. This usually takes the form of an imitation app or a WebView launched "on-top" of a legitimate application (such as a banking app).*"

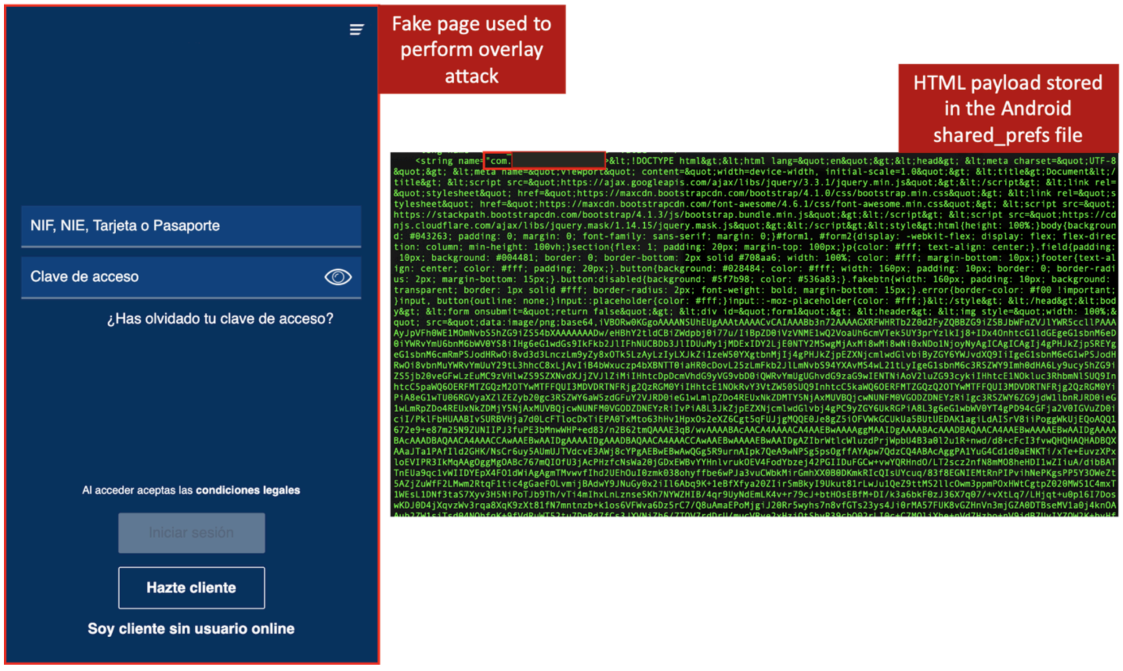

Figure 7 - Example of injection used to perform overlay attack

See **Appendix 1 - Geographical distribution of banks currently targeted by TeaBot** for an overview of targeted apps.

Other features: TeaBot has other features quite common to other known Android bankers such as:

- disabling Google Protect
- sending / intercepting / hiding SMS messages
- stealing other accounts from the Android Settings and Google Authentication 2FA codes
- simulating gestures and clicks on the screen (via Accessibility Services).

Dynamic Analysis

When the malicious app has been downloaded on the device, it tries to be **installed as an "Android Service",**which is an application component that can perform long-running operations in the background.

This feature is abused by TeaBot to silently hide itself from the user, once installed, preventing also detection and ensuring its persistence.

Furthermore, during the installation phases, TeaBot starts communicating with its C2 server in the background.

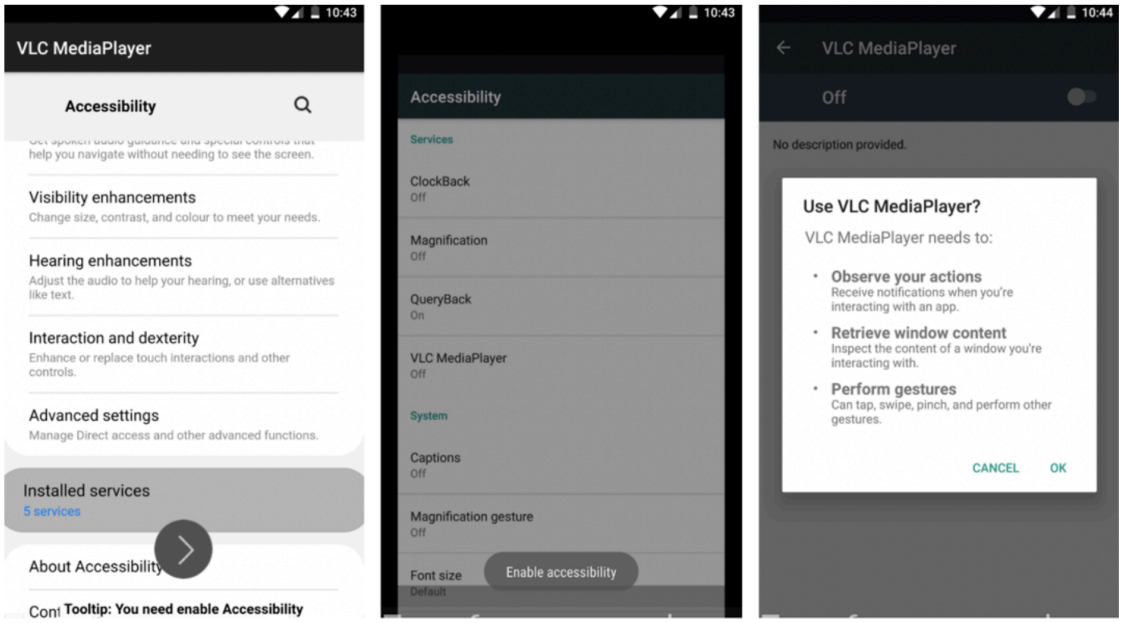

Figure 8 - Screenshots taken during the installation phase of TeaBot

After the installation TeaBot will request the following Android permissions, which are mandatory to perform its malicious behavior:

- **Observe your actions**
  Used to intercept and observe the user action
- **Retrieve window content**
  Used to retrieve sensitive information such as login credentials, SMS, 2FA codes from authentication apps, etc.
- **Perform arbitrary gestures**
  TeaBot uses this feature to accept different kinds of permissions, immediately after the installation phase, for example the REQUEST_IGNORE_BATTERY_OPTIMIZATIONS permission popup.

Once the requested permissions have been accepted, the malicious application will remove its icon from the device.

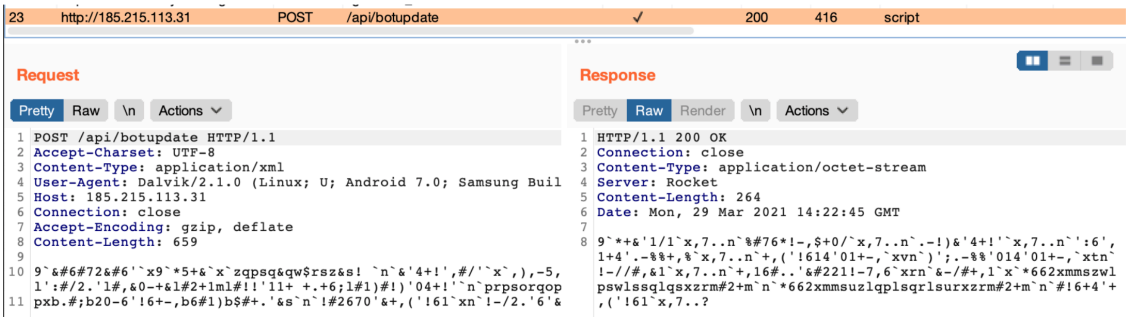

Figure 9 - Example of encrypted communication

During its first communications, TeaBot sends the list of installed apps to verify if the infected devices had one or more targeted apps already installed. When one or more targeted applications are found, the C2 sends the specific

payloads to the device.

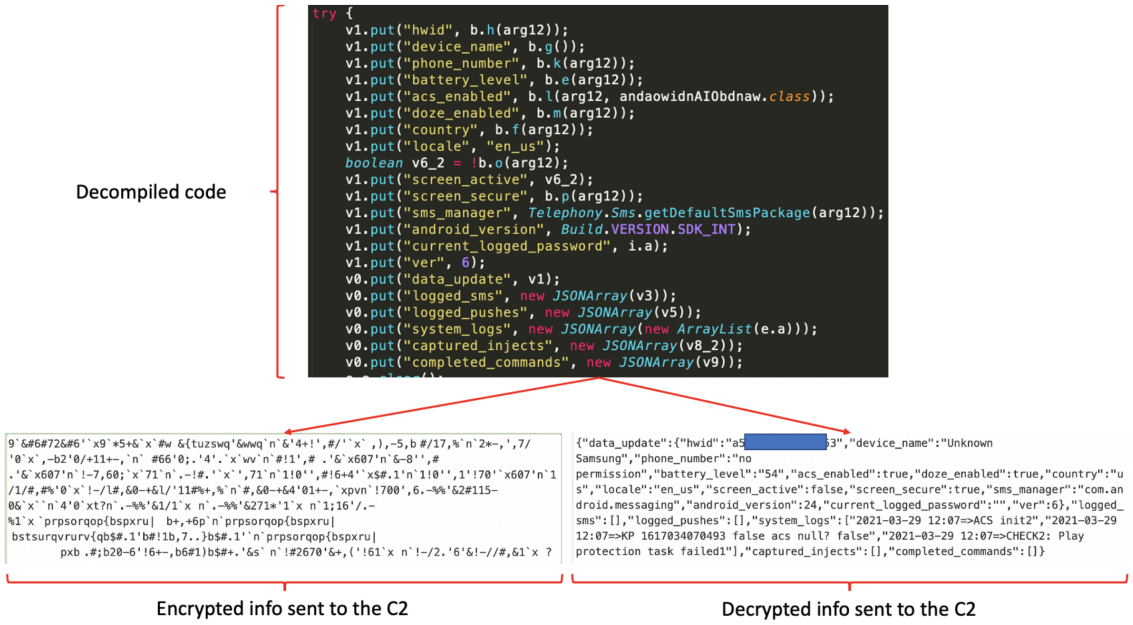

Figure 10 - Example of encrypted and decrypted communication

By analyzing TeaBot network communications, it was possible to group them into the following three main types:

- **[C2-URL]/api/botupdate:**
  every 10 seconds TeaBot sends a POST request with all the information about the compromised device (Figure 8) (e.g. name of the SMS manager app installed, captured injects, passwords found etc.). **Those communications are the only one encrypted with the XOR algorithm using the same key across multiple TeaBot samples** (“66”). The response is typically composed by a configuration update (e.g. C2 addresses, command launched, etc.)
- **[C2-URL]/api/getkeyloggers:**
  every 10 seconds TeaBot performs a GET request to retrieve the list of the apps targeted by the key logger functionality
- **[C2-URL]/api/getbotinjects**:
  a POST request is made by TeaBot during its first stage of infection with a JSON file (not encrypted) containing all the package name installed on the compromised device. With this information, TeaBot is able to know if there is one or more targeted apps and download the related injection(s).

Appendix 1: Geographical distribution of banks currently targeted by TeaBot

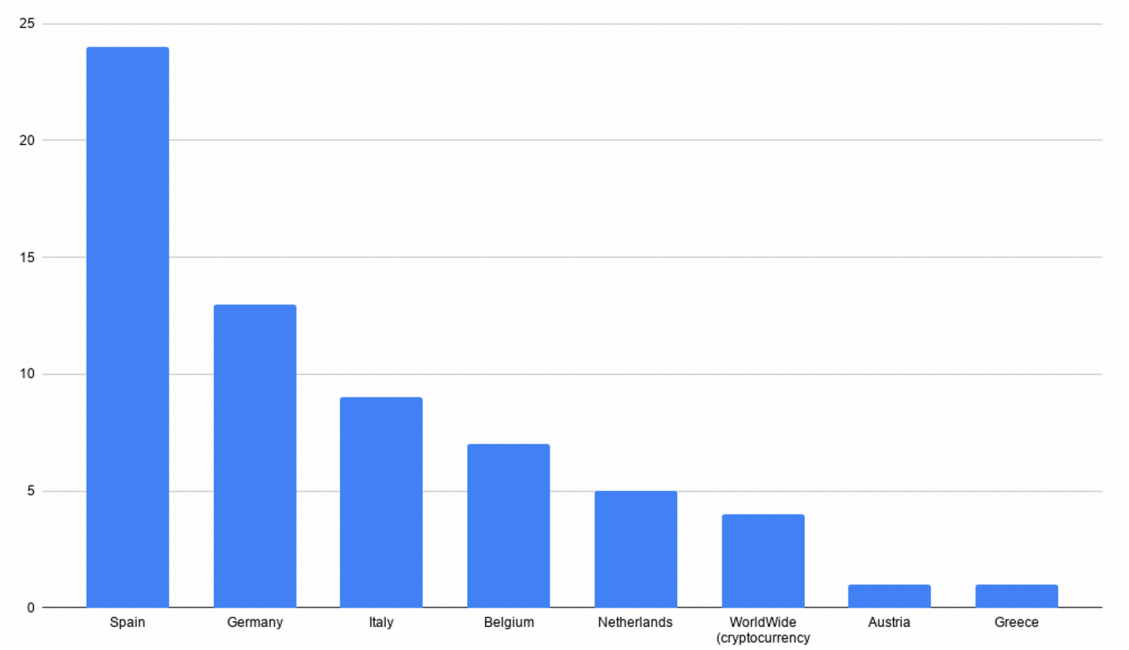

Figure 11 - Geographical distribution of banks currently targeted by TeaBot

Appendix 2: TeaBot commands

The following table will summarize the list of all the commands found in TeaBot during the technical analysis:

- **app_delete** : Delete an application from the package name
- **ask_syspass** : Show a biometric authorization popup
- **ask_perms** : Request permissions to the users
- **change_pass** : Show a toast message (small popup) that inform the user to update the password (lock pattern)
- **get_accounts** : Get the accounts in Android settings
- **kill_bot** : Remove itself
- **mute_phone** : Mute the device
- **open_activity** : Open an application from the package name
- **open_inject** : Perform the overlay attack, opening the injection (html payload)
- **reset_pass** : Under development
- **start_client** : Define an IP and PORT used to observe the compromised device through screenshots
- **swipe_down** : Used to perform gesture like swipe on the screen
- **grab_google_auth** : Open and get the codes in Google Auth app
- **activate_screen** : Enable the screen. TeaBot has the ability to control the device's screen (e.g. The banker is able to keep screen from dimming)

Appendix3: IOCs

- **App Name** : "VLC MediaPlayer" | "TeaTV"
- **SHA256** : 89e5746d0903777ef68582733c777b9ee53c42dc4d64187398e1131cccfc0599 | 7f5b870ed1f286d8a08a1860d38ef4966d4e9754b2d42bf41d7 511e1856cc990

- **C2** : 185.215.113[.]31 | kopozkapalo[.]xyz | sepoloskotop[.]xyz | 178.32.130[.]170
- **XOR Key**: 66

Source: https://www.cleafy.com/cleafy-labs/teabot