SharkBot: a new generation of Android Trojans is targeting banks in Europe

cleafy.com/cleafy-labs/sharkbot-a-new-generation-of-android-trojan-is-targeting-banks-in-europe

Federico Valentini, Francesco Iubatti

Download your PDF guide to TeaBot

Get your free copy to your inbox now

Download PDF Version

Key Points

- At the end of October 2021, a new Android banking trojan was discovered and analyzed by the Cleafy TIR team. Since we didn't find references to any known families, we decided to dub this new family **SharkBot**.
- The main goal of **SharkBot** is to initiate money transfers from the compromised devices via **Automatic Transfer Systems (ATS)** technique bypassing multi-factor authentication mechanisms (e.g., SCA). These mechanisms are used to enforce users' identity verification and authentication, they are usually combined with behavioural detection techniques to identify suspicious money transfers.
- We identified a botnet which is currently targeting the **UK**, **Italy**, and the **US**, including banking applications and cryptocurrency exchanges. Given its modularity architecture we don't exclude the existence of botnets with other configurations and targets.

- Once **SharkBot** is successfully installed in the victim's device, attackers can obtain sensitive banking information through the abuse of Accessibility Services, such as credentials, personal information, current balance, etc., but also to perform gestures on the infected device.
- At the time of writing, **SharkBot** appears to have a very **low detection rate** by antivirus solutions since multiple **anti-analysis techniques** have been implemented: string obfuscation routine, emulator detection and a domain generation algorithm (**DGA**) for its network communication in addition to the fact that the malware has been written from scratch.
- **SharkBot** implements **Overlay attacks** to steal login credentials and credit card information and it also has capabilities to intercept legitimate banking communications sent through SMS.
- At the time of writing, multiple indicators suggest that SharkBot could be at its early stages of development.

Executive Summary

At the end of October 2021, a new Android banking trojan appeared on Cleafy's telemetries. Since the lack of information and the absence of a proper nomenclature of this malware family, we decided to dub it **SharkBot** to better track this family inside our internal Threat Intelligence taxonomy.

SharkBot belongs to a “new” generation of mobile malware, as it is able to perform ATS attacks inside the infected device. This technique has been already seen recently from other banking trojans, such as Gustuff. ATS (Automatic Transfer System) is an advanced attack technique (fairly new on Android) which enables attackers to auto-fill fields in legitimate mobile banking apps and initiate money transfers from the compromised devices. Contrary to TeaBot and Oscorp/UBEL where a live operator is required to insert and authorize a money transfer, with ATS technique Threat Actors can scale up their operations with minimum user intervention. We assume that SharkBot is trying to bypass behavioural detection countermeasures (e.g.,biometrics) put in place by multiple banks and financial services with the abuse of Android Accessibility Services, also bypassing the need of a “new device enrollment”.

Figure 1 – Example of how SharkBot perform an ATS attack

Moreover, SharkBot appears to have all the main features of nowadays Android banking trojan achieved by abusing Accessibility Services[1]such as:

- Ability to perform classic Overlay Attacks against multiple applications to steal login credentials and credit card information
- Ability to intercept/hide SMS messages
- Enabling key-logging functionalities
- Ability to obtain full remote control of an Android device (via Accessibility Services)

At the time of writing, we didn’t notice any samples on Google's official marketplace. The malicious app is installed on the users' devices using both the side-loading technique and social engineering schemes.

Thanks to an in-depth analysis of several samples related to SharkBot, we collected **22 different targets** including international banks from **UK** and **Italy** and **5 different cryptocurrency services**, as shown in the following Figure 2:

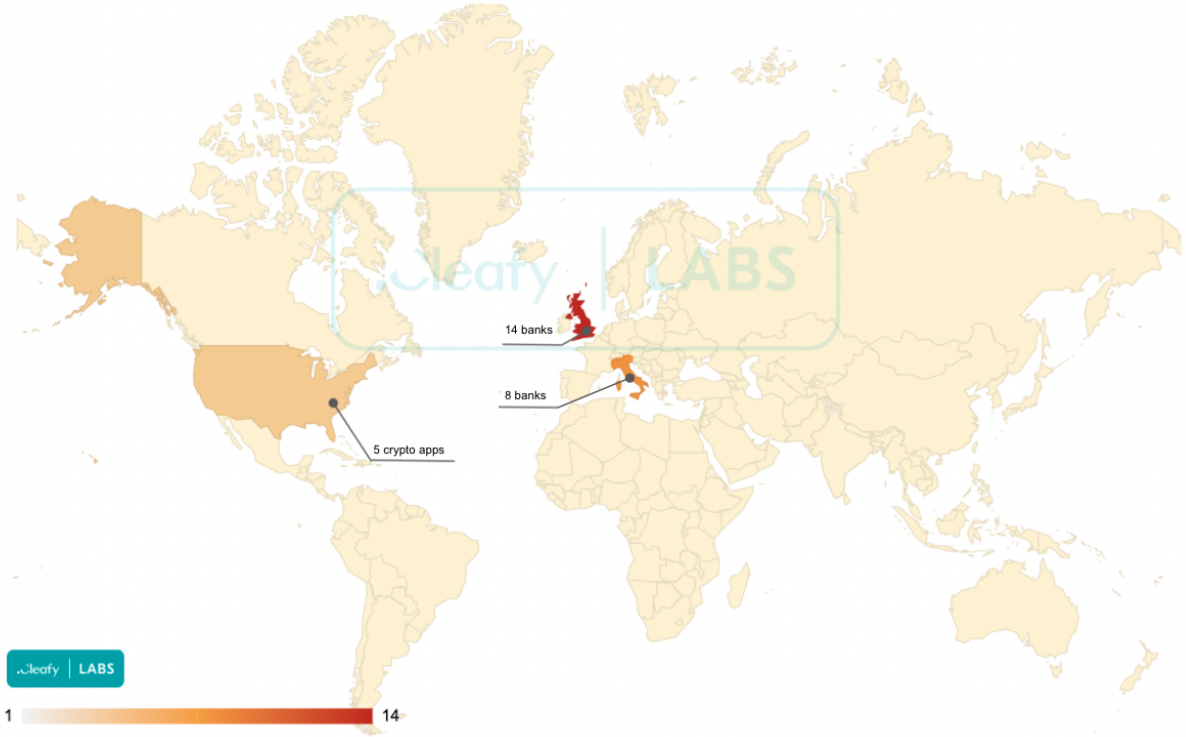

Figure 2 – Geographical distribution of banks currently targeted by SharkBot
[1] https://developer.android.com/reference/android/accessibilityservice/AccessibilityService

Technical Analysis – Overview

SharkBot, is a new generation Android banking trojan, discovered by Cleafy Threat Intelligence team at the end of October 2021. The name "*SharkBot*" comes from multiple strings found in its binaries, which contain the word "*sharked*".

SharkBot hides itself with common names and icons posing as a legitimate application to the victims, as shown in Figure 3.

Figure 3 – Main names/icons used by SharkBot
However, during its installation, SharkBot immediately tries to enable Accessibility Services that keep being requested persistently with fake pop-ups until the victim accepts.

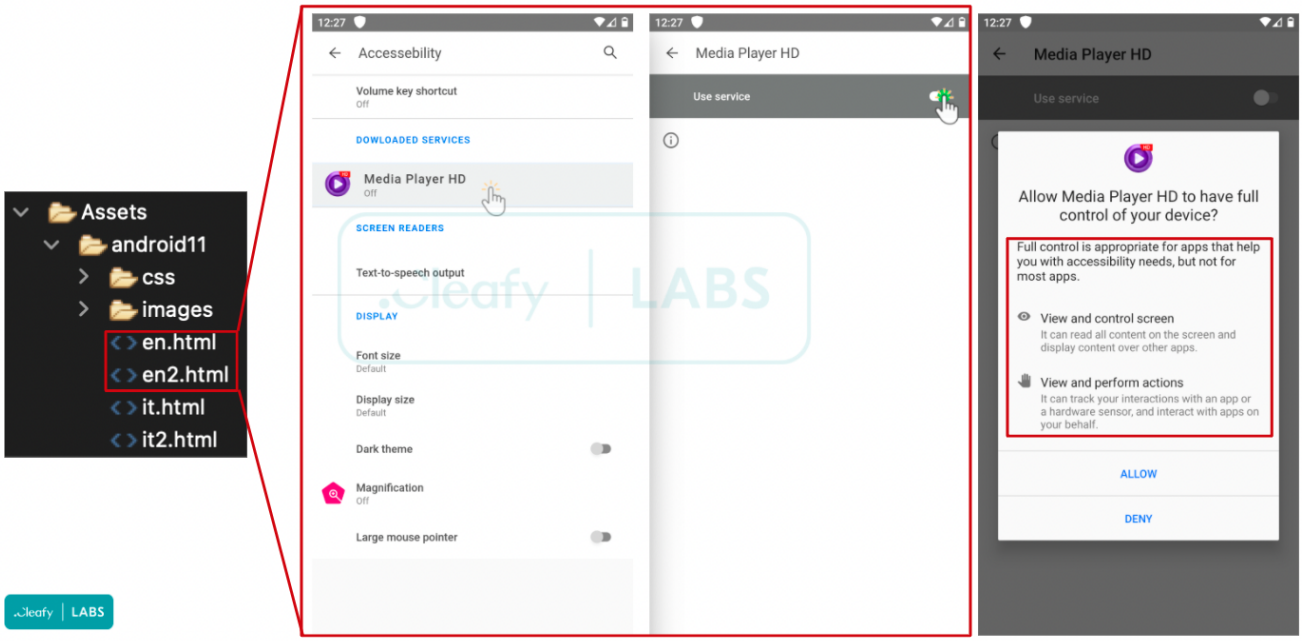

Figure 4 – Installation phases of SharkBot

Once the malicious app has been installed, no icon is displayed on the device and SharkBot is able to get all the permissions needed (declared inside the AndroidManifest file) thanks to the accessibility services enabled. This is done by clicking instantly on the popup shown to the user.

```
<?xml version="1.0" encoding="UTF-8"?>
<manifest android:compileSdkVersion="30" android:compileSdkVersionCodename="11" android:vers
  <uses-sdk android:minSdkVersion="26" android:targetSdkVersion="30"/>
  <uses-permission android:name="android.permission.INTERNET"/>
  <uses-permission android:name="android.permission.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS"/>
  <uses-permission android:name="android.permission.FOREGROUND_SERVICE"/>
  <uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW"/>
  <uses-permission android:name="android.permission.ACTION_MANAGE_OVERLAY_PERMISSION"/>
  <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
  <uses-permission android:name="android.permission.SEND_SMS"/>
  <uses-permission android:name="android.permission.RECEIVE_SMS"/>
  <uses-permission android:name="android.permission.READ_SMS"/>
  <uses-permission android:name="android.permission.WRITE_SMS"/>
  <uses-permission android:name="android.permission.RECEIVE_MMS"/>
  <uses-permission android:name="android.permission.QUERY_ALL_PACKAGES"/>
  <uses-permission android:name="android.permission.REQUEST_DELETE_PACKAGES"/>
  <uses-permission android:name="android.permission.REQUEST_INSTALL_PACKAGES"/>
```

Figure 5 – Android Permissions of SharkBot

With the permissions shown in Figure 5, SharkBot is able to read/send text messages, perform overlay attacks and, with the **REQUEST_IGNORE_BATTERY_OPTIMIZATIONS** permission, it is able to bypass Android's doze component and stay connected to the C2 servers to continue its malicious behavior.

At the time of writing, Sharkbot seems to be still under development as the very first samples tracked down at the end of October use:

- a demo version of the Allatori Java Obfuscation [2] (as shown in Figure 6).
- the word “example” in the package name.

- the words “test1” and “testuk” inside the C2 used during the first exchange of information with malware.
- some functionalities are not yet available.

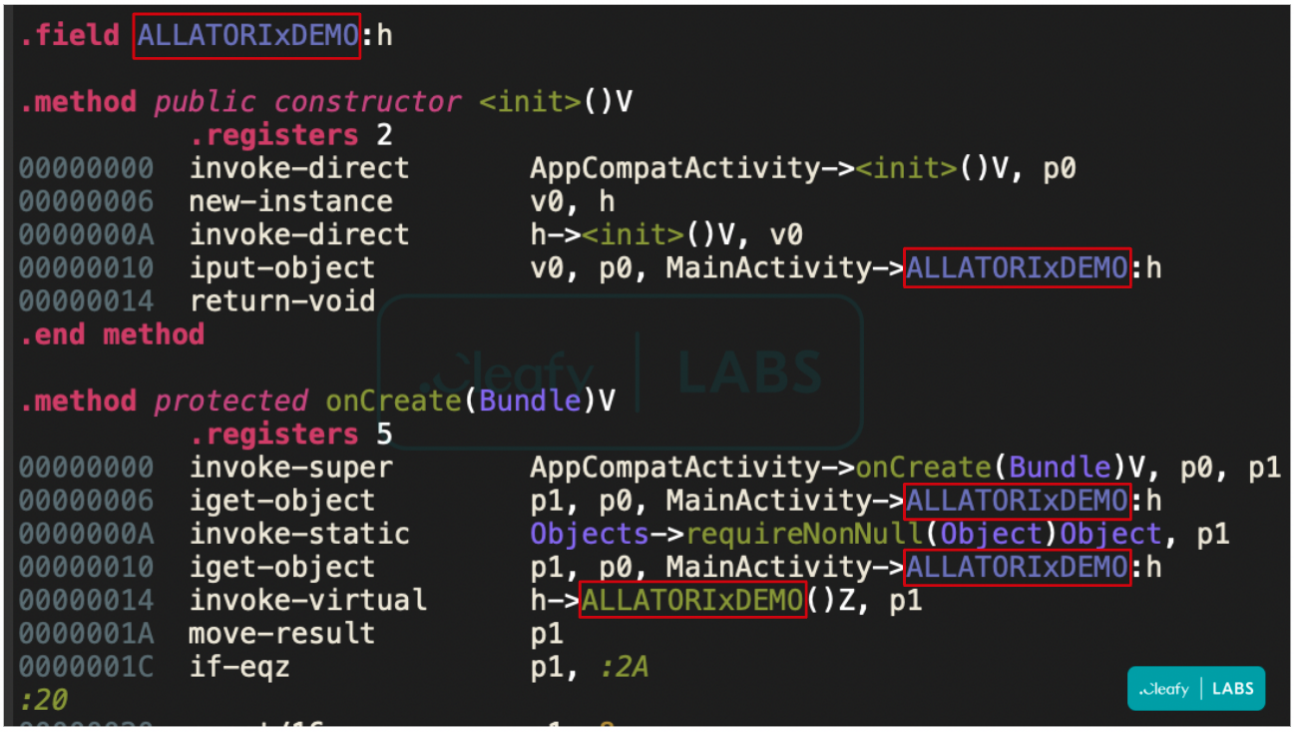

Figure 6 – Use of a demo version of the Allatori Java Obfuscation

So far, SharkBot has a very **low detection rate** by antivirus solutions (**only 3/62**), as shown by Figure 7. This means that the malware has been written from scratch, in addition to the fact that it uses an external module, downloaded from the C2, containing the ATS core functionalities and anti-detections technique used to slow down the static and dynamic analysis.

Analysing the underground hacking forums, we didn’t find any references to this malware. This makes us think that SharkBot is still a private botnet.

Figure 7 – Detection of SharkBot by antivirus solutions

```
JSONObject v0 = new JSONObject();
Objects.requireNonNull(this.J);
v0.put("botnetID", "mediaplayer");
Objects.requireNonNull(this.J);
v0.put("ownerID", "traff");
Objects.requireNonNull(this.J);
v0.put("versionBot", "1.18.0");
v0.put("botID", this.J.d);
v0.put("model", Build.MANUFACTURER);
v0.put("vendor", Build.MODEL);
v0.put("os", Build.VERSION.RELEASE);
v0.put("sdk", Build.VERSION.SDK_INT);
v0.put("lang", "it");
v0.put("appsList", this.H());
v0.put("myName", this.h.getResources().getString(0x7F0E001E));  // string:app_name "Media Player HD"
v0.put("myNameLabel", this.h.getResources().getString(0x7F0E001F));  // string:app_name_a "Media Player HD"
v0.put("myPackage", this.h.getPackageName());
String[] v1 = new StringBuilder().insert(0, this.f).append(this.a()).toString().split(",");
v4 = 0;
```

Figure 8 – Some information about SharkBot botnet
[2] http://www.allatori.com/ (*Allatori is a legitimate software)

Evasion techniques

SharkBot uses different anti-analysis and detection techniques, in particular:

> Strings obfuscation, to slow down the static analysis and “hide” all the commands and important information used by the malware, as shown in Figure 9.

```
000000EE  const-string         v15, "t[iZNPm[dJ"
000000F2  invoke-static        x->H(Object)String, v15  # DECRYPTED_STRING: "sendInject" (0x1)
000000F8  move-result-object   v15
000000FA  invoke-virtual       String->equals(Object)Z, v13, v15
00000100  move-result          v13
00000102  if-eqz               v13, :19A
:106
00000106  const/4              v13, 5
00000108  goto                 :1A2
:10A
0000010A  const-string         v15, "dVfP`[TSt\u007FcSnP"
0000010E  invoke-static        x->H(Object)String, v15  # DECRYPTED_STRING: "changeSmsAdmin" (0x1)
00000114  move-result-object   v15
00000116  invoke-virtual       String->equals(Object)Z, v13, v15
0000011C  move-result          v13
0000011E  if-eqz               v13, :19A
:122
00000122  const/4              v13, 4
00000124  goto                 :1A2
:126
00000126  const-string         v15, "rPnPtJfRk\u007FwN"
0000012A  invoke-static        x->H(Object)String, v15  # DECRYPTED_STRING: "uninstallApp" (0x1)
00000130  move-result-object   v15
00000132  invoke-virtual       String->equals(Object)Z, v13, v15
00000138  move-result          v13
0000013A  if-eqz               v13, :19A
:13E
0000013E  const/4              v13, 0
00000140  goto                 :1A2
:142
00000142  const-string         v15, "rNc_s[DQiXnY"
00000146  invoke-static        x->H(Object)String, v15  # DECRYPTED_STRING: "updateConfig" (0x1)
0000014C  move-result-object   v15
```

Figure 9 – Example of strings obfuscation

- **Anti-Emulator**. When the malicious application is installed on the device, it checks if the device is an emulator or a real phone. This technique is usually used to bypass sandboxes or common emulators used by researchers during the dynamic analysis.
- **External ATS module**. Once installed, the malware downloads an additional module from the C2. The external module is a ".jar" file that contains all the functionality used to perform the ATS attacks. We analyze this module in the paragraph "SharkBot - ATS (Automatic Transfer System) module".
- **Hide the icon app**. Once installed, SharkBot hides the icon of the app from the device screen.
- **Anti-delete**. Like other malware, SharkBot uses the Accessibility Services to avoid that the user uninstalls the malicious application from the settings options.
- **Encrypted communication**. All the communication between the malware and C2 are encrypted and encoded with Base64. In addition to this, SharkBot uses a Domain Generator Algorithm (DGA).

```
private String a() {
    String v6;
    int v5;
    Calendar v2;
    StringBuilder v1_1;
    try {
        v1_1 = new StringBuilder();
        v2 = Calendar.getInstance();
        String[] v3 = ".top,.xyz,.cc,.info,.com,.ru,.info,.net".split(",");
        v5 = 0;
        while(true) {
        label_11:
            if(v5 >= v3.length) {
                return v1_1.toString().toLowerCase();
            }

            v6 = v3[v5];
            break;
        }
    }
    catch(Exception v1) {
        this.J(new StringBuilder().insert(0, "domenDGA: ").append(v1.toString()).toString());
        return "";
    }
```

Figure 10 – SharkBot DGA method

SharkBot "classical" features

Although SharkBot has an ATS module, it also has some common features present in other banking trojan, in particular:

- The capability to read and hide SMS received from the infected user (sending them to the threat actor C2 server). This feature is mostly used by threat actors to get the 2FA sent by the bank via text messages.
- The "now famous" overlay attack used to steal login credentials and credit card data. This feature is used by SharkBot to obtain the login credentials of the targeted banks/crypto app, to perform the ATS attack to the next step.

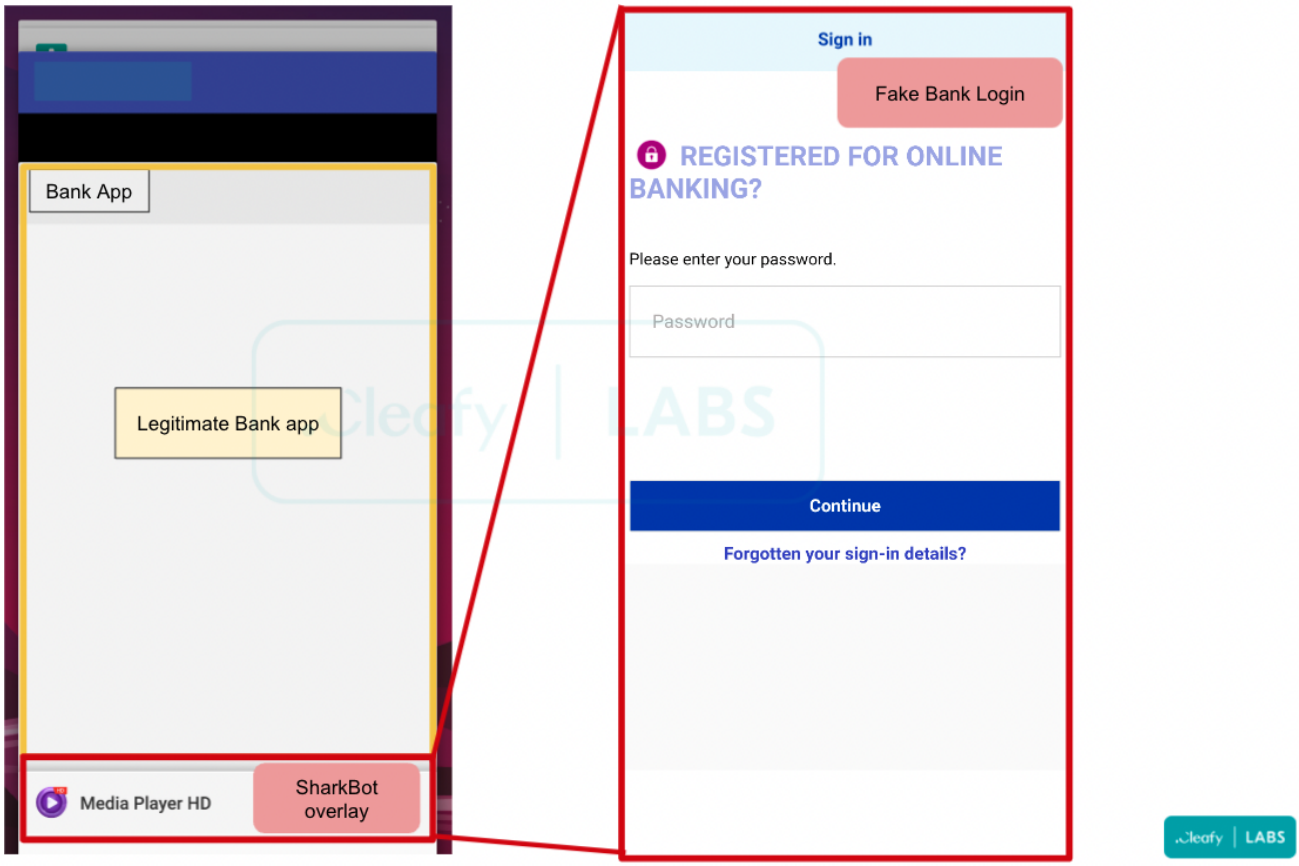

Figure 11 – Example of overlay attack performed by SharkBot

SharkBot – ATS (Automatic Transfer System) module

Android's Accessibility Service has been historically abused by multiple banking trojans (e.g., TeaBot, Oscorp/UBEL) for conducting multiple malicious actions in the infected device. SharkBot, similar to Gustuff, is able to abuse Accessibility Service enabling **ATS** attacks inside the infected device.

ATS (Automatic Transfer System) attacks enable TA to auto-fill fields in legitimate mobile banking apps and initiate money transfers from the compromised devices to a money mule network controlled by TA or other affiliates. This makes it possible to scale up their operations with minimum user intervention.

For a bank perspective, mobile ATS attacks are very hard to identify and handle since typically:

- They don't require a "new device enrollment" phase which drastically reduces their footprint.
- They are able to bypass any 2FA mechanism used by banking applications (e.g. SMS-based, push-based, etc.).
- As all the actions are performed by the trusted user, ATS attacks are able to bypass Behavioral detection mechanisms, including Behavioral biometrics.

- Illegitimate wire transfers are inserted and authorized on the victim device itself, which typically is considered “trusted” by banks.

Once a victim has granted accessibility permissions, all the contents shown in the device screen can be intercepted and manipulated by SharkBot. Those capabilities are gained through Android AccessibilityEvents which are events that are sent by Android OS when something notable happens in the user interface. In fact, the main purpose of an accessibility event is to communicate changes in the UI to an AccessibilityService.

SharkBot appears to have interest only on a specific subset of accessibility events, which are the following:

Figure 12 – AccessibilityEvent types intercepted by SharkBot

We can group all the accessibility events intercepted by SharkBot as follows:

TYPE_VIEW_CLICKED **TYPE_VIEW_SELECTED** **TYPE_VIEW_TEXT_CHANGE** **TYPE_VIEW_TEXT_SELECTION_CHANGE**	**fired when a button is clicked, an item is selected or when text changes are detected.**

TYPE_VIEW_CLICKED **TYPE_VIEW_SELECTED** **TYPE_VIEW_TEXT_CHANGE** **TYPE_VIEW_TEXT_SELECTION_CHANGE**	**fired when a button is clicked, an item is selected or when text changes are detected.**
TYPE_WINDOW_STATE_CHANGED TYPE_WINDOW_CONTENT_CHANGED CONTENT_CHANGE_TYPE_TEXT	fired when a visually distinct section of the user interface is detected, for example when a new Activity has been launch (e.g navigating to a different page of the same application, or switching applications).
TYPE_NOTIFICATION_STATE_CHANGED TYPE_ANNOUNCEMENT	fired when a new notification appears on the device or when an application makes announcements.

SharkBot has already implemented various functions which are been used for parsing all the data extracted from the UI, save them into a JSON format and exfiltrate them to the designed C2 server:

```
private JSONObject saveNodeInfo(String arg8, String arg9, AccessibilityNodeInfo arg10) {
}

private JSONObject saveNodeInfoRecursive(AccessibilityNodeInfo arg7, String arg8, int arg9) {
}

private AccessibilityNodeInfo searchNode(AccessibilityNodeInfo arg6, JSONObject arg7) {
}

private List searchNodeByOtherParam(AccessibilityNodeInfo arg5, JSONObject arg6, List arg7) {
}
```

Figure 13 – Retrieve and save data extracted from the intercepted Accessibility events

TA can also passively logs all the exfiltrated information from each infected device and enriching them with detailed information useful for a further ATS attack, such as account balance(s), enabled 2FA/SCA/MFA mechanisms, cash-out availability (e.g. SEPA, Instant payments), etc.

Once the ATS attack is remotely requested by TA, SharkBot will start interacting with the infected device and auto-fill fields in legitimate mobile banking apps and initiate money transfers. During this phase TA can also interact with the targeted application simulating gestures and clicks, if required.

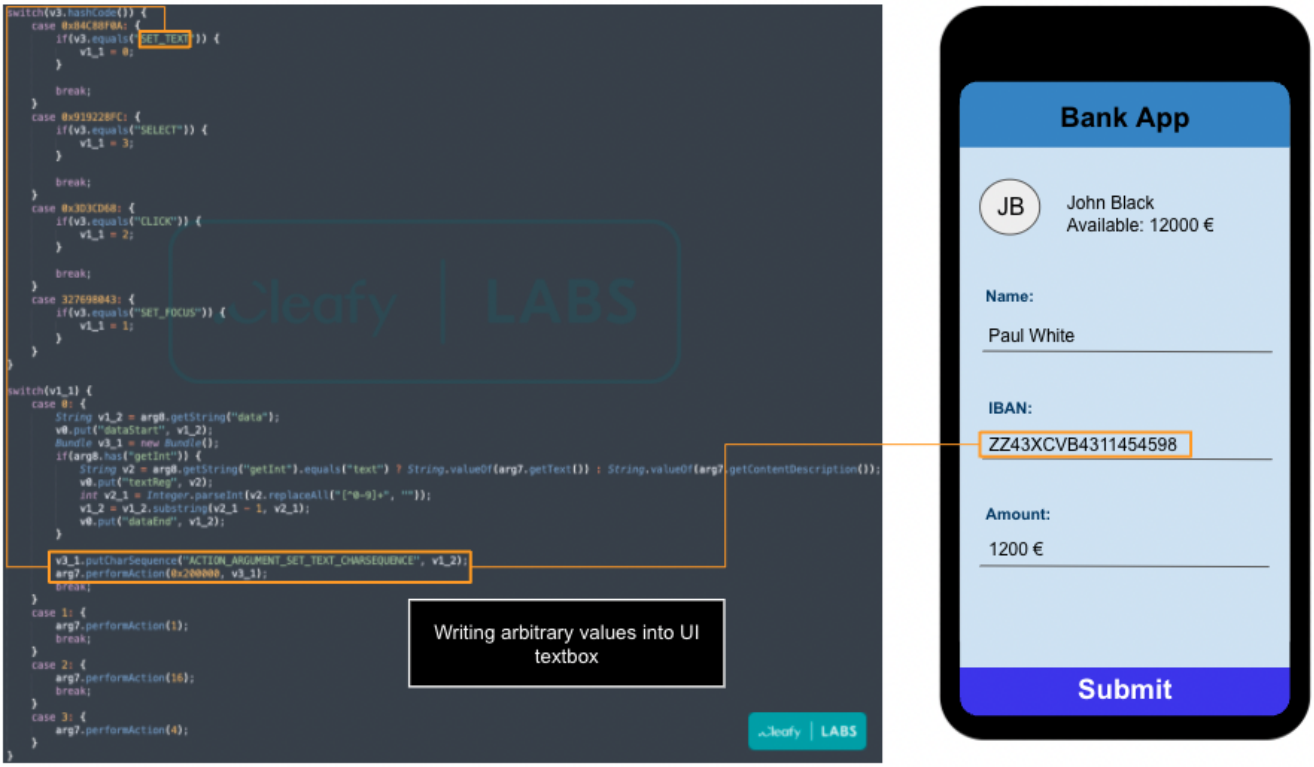

Figure 14 – Auto-fill fields in legitimate mobile banking app during ATS attack

Conclusion

With the discover of **SharkBot** we have shown new evidence about how mobile malwares are quickly finding new ways to perform fraud, trying to bypass behavioural detection countermeasures put in place by multiple banks and financial services during the last years.

Like the evolution of workstation malwares occurred in the past years, also in the mobile field we are seeing a rapid evolution towards more sophisticated patterns like ATS attacks.

Appendix 1: SharkBot commands

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

Command	Description
updateLib	Not implemented
updateSQL	Update configuration data stored on a local database
updateConfig	Update the configuration file, containing the C2 url and the targets
uninstallApp	Delete an app installed on the infected device
changeSmsAdmin	Change the default SMS app manager

Command	Description
sendInject	Receive Overlay attacks payloads from C2
updateTimeKnock	Update timestamp bot
sendPush	Not implemented
unlockPhone	Set a specific variable during ATS attack
ats	Enable ATS attacks
overlay	Enable Overlay attacks
enableKeyLogger	Get keylogging steps during ATS attack
unlockPhone1	Check if the device has a PIN, pattern or password setted up.
overlay2	Enable Overlay attacks
openPackage	Open an arbitrary Android application
doze	Bypass Android “doze” feature for enabling network communication in the background
stopAll	Reset ATS routine

Appendix 2: IOCs

App Name	Media Player HD
Package Name	com.pycdvgljmfgh3hgp8jo72giu.omflsx1q2g
MD5	f7dfd4eb1b1c6ba338d56761b3975618
C2	sharkedtest1[.]xyz
sharkedtestuk[.]xyz	