BianLian: A New Wave Emerges

fortinet.com/blog/threat-research/new-wave-bianlian-malware.html

July 3, 2019

```
@TargetApi(21)
private void startCapture() {
   Timber.d("ScreencastService -> startCapture()");

   try {
      this.projection = ScreencastComponent.get().getProjectionManager().getMediaProjection(this.resultCode, this.resultData);
      ImageTransmogrifier var1 = new ImageTransmogrifier(this, ScreencastComponent.get().getWindowManager());
      this.it = var1;
      Callback var3 = new Callback() {
         public void onStop() {
            ScreencastService.access$100(ScreencastService.this).release();
         }
      };
      this.mediaProjectionCallback = var3;
      this.vdisplay = this.projection.createVirtualDisplay("andshooter", this.it.getWidth(),
                                                           this.it.getHeight(),
                                                           this.getResources().getDisplayMetrics().densityDpi, 9,
                                                           this.it.getSurface(),
                                                           (android.hardware.display.VirtualDisplay.Callback)null,
                                                           this.handler);
      this.projection.registerCallback(this.mediaProjectionCallback, this.handler);
   } catch (Exception var2) {
      var2.printStackTrace();
      if (var2 instanceof IllegalStateException) {
         this.stopCapture();
         this.startCapture();
      }
   }
```

FortiGuard Labs Breaking Threat Research

Recently, during our daily malware analysis routine, members of the FortiGuard Labs team encountered an Android sample that did not look familiar.

Analysis

At a first look, it seemed clear that the APK was heavily obfuscated, and was possibly packed using some technique we had not seen before. This is not to say that it uses an extremely complicated technique. It seems to mostly rely on generating a variety of random functions to hide the real functionalities of the sample.

During our initial examination, we also spotted some interesting code amongst the rubbish functions being generated, so we decided to run the sample through FortiGuard's in-house APK sandbox analysis system to gather more information from its both static and dynamic analysis.

Sandbox Results

The results obtained from the sandbox helped us understanding the sample. First of all, in the Dex operation section, where a dynamically loaded file should be logged, there are two entries. The first is related to the loading of the main application itself, and a second refers to a file conveniently called "*payload.apk*" – which means that the sample will likely install an additional application during execution. However, most of the interesting calls, like information exfiltration and connection initiations, are executed from the code in the first application.

With this information in hand, and being familiar with recent malware families, we had a hunch as to what malware this could be. However, first things first: we needed to get rid of its very annoying obfuscation.

Obfuscation Analysis

The code base is very messy as it is mostly comprised of randomly generated garbage. Fortunately, the template is very identifiable, and most of the classes are useless, having no real functionality at all. However, when first analyzing this application, the sheer number of different classes can seem discouraging. Fortunately, the template for these junk-classes is very predictable.

Every junk-class sports a name composed of random lowercase and uppercase characters.

Figure 1: Randomly generated useless class

In addition, there is a clear difference between the classes that are useless and the ones that contain used code. Conveniently enough, all of the interesting classes maintain their original name. However, to complicate things further, it looks like every legitimate function or package is accompanied by 7 functions or packages performing useless calculations.

Most of the strings in the code are generated by using functions implementing a XOR decryption of byte arrays – a simple but relatively effective solution. Every string corresponds to a specific function that accepts no arguments and returns a `String` object.

In the screenshots used in the analysis section below, what is shown is the de-obfuscated version of the code. We will not go through the process on how to obtain that in this report, because it is very boring and straight forward. It simply requires the elimination of many unnecessary functions and the execution of the ones yielding the required strings.

At this point we want to give a big shout-out to Max 'Libra' Kersten and his AndroidProjectCreator for making this de-obfuscation process less painful than we anticipated. If you have not heard about this project, go and check it out.

Malware Analysis

As identified in the title of this blogpost, this obfuscated sample belongs to the BianLian malware family, discussed for the first time by ThreatFabric in 2018.

BianLIan started as a dropper for other malware, but developers quickly began to implement their own malicious code that primarily targets Turkish banking applications.

In true Android malware fashion, the first thing the application does is hide its icon and constantly requests permission to abuse Accessibility services functionalities until granted.

Figure 2: Permission requests

Once these are obtained, it initiates all of its modules. Compared to its 2018 version, the authors behind this update have added some new functionalities. In addition to extensively abusing Accessibility services, it includes the following modules. This list include modules from the old and wave of BianLian as well as those added in this new version:

- **text:** Module used to send, receive, and log SMS messages
- **ussd**: Module used to run USSD codes and make calls
- **injects**: Module used to run overlay attacks, mostly on banking applications
- **locker**: Module used to lock the screen, rendering the device unusable for a user

In addition to these modules, this BianLian sample included also the following two modules that we will describe in more detail below:

- **screencast**
- **socks5**

Screencast Module

The Screencast Module allows the malware to record the screen of the device. It uses the android package *android.media.projection.MediaProjection* to create a virtual display to screencast.

It first checks if the screen is locked. If it is, it releases the lock and then starts its recording. The recording is started remotely, as with other functionalities, using FCM (Firebase Cloud Messaging).

Figure 3: Screencast Module

Socks5 Module

This module is used to create a functioning SSH server on the device using JSCH (Java Secure Channel), a library that implements SSH2 in pure Java. By using this tool, BianLian can setup a proxy that can run SSH sessions using remote port forwarding on port 34500,

with an implementation similar to 2017's malware MilkyDoor, making communication with the CC harder to detect.

Figure 4: Socks5 Module

Dropped Files

BianLian started its career in the malware industry as a dropper, so it is not surprising to find a payload in this sample. In the older versions of BianLian, this payload was decrypted from the assets of the APK, while in this case it is downloaded from the CC.

The dropped APK is actually far less interesting, and in fact, not actually a malware by itself. The code base is very limited and performs only one function. It checks to see if Google Play Protect is active through the Google SafetyNet API. This code is loaded and used by the main application through Java reflection.

Figure 5: Payload.apk execution

It is worth reemphasizing that this payload is not a different malware, but rather, a tool used by BianLian.

Conclusion

BianLian seems to still be under active development. The added functionalities, even though not completely original, are effective and make this family a potentially dangerous one. Its code base and strategies put it on a par with the other big players in the banking malware space.

In addition, its new obfuscation technique, even though not very complicated, is still capable of tricking string-based detection, and would be very hard to detect with static analysis alone when encountered for the first time.

While looking around for similar malware, we also encountered some Anubis samples using this same obfuscation algorithm. This suggests that the author of this obfuscator is either selling it on hacking forums, or was able to get his hands on the source code of these two families of malware.

FortiGuard Labs has been following this family since it was first detected, and will continue to keep on the lookout for new threats.

-= FortiGuard Lion =-

Solutions

Fortinet customers are protected by the following signatures:

- The BianLian sample analyzed is detected as Android/Agent.AMJ!tr
- The Anubis sample mentioned is detected as Android/Agent.JA!tr

IOC

BianLian: ac32dc236fea345d135bf1ff973900482cdfce489054760601170ef7feec458f
Payload: 75e162dc291e15d13b0f3202a66e0c88ff2db09ec02922ee64818dbddcb78d6d
Anubis: a99eb900d03aa1dd70d7712da7c42cc37ee2f2e21d763acd6ddf71a4027ed504

CCs:
hxxps://tombaba[.]club
hxxps://tomcatdomains[.]page[.]link

Banking applications targeted:
com.akbank.android.apps.akbank_direkt
com.albarakaapp
com.binance.dev
com.btcturk
com.denizbank.mobildeniz
com.finansbank.mobile.cepsube
com.garanti.cepsubesi
com.ingbanktr.ingmobil
com.kuveytturk.mobil
com.magiclick.odeabank
com.mobillium.papara
com.pozitron.iscep
com.teb
com.thanksmister.bitcoin.localtrader
com.tmobtech.halkbank
com.vakifbank.mobile
com.ykb.android
com.ziraat.ziraatmobil
finansbank.enpara
tr.com.hsbc.hsbcturkey
tr.com.sekerbilisim.mbank

Learn more about FortiGuard Labs and the FortiGuard Security Services portfolio. Sign up for our weekly FortiGuard Threat Brief.

Read about the FortiGuard Security Rating Service, which provides security audits and best practices.