Bypass SSL Pinning on Android

The McAfee Advanced Threat Research team conducts security research with the aim of staying ahead of the evolving threat landscape to expose and reduce attack surfaces. This series of white papers discusses laboratory security research techniques that are generally known among the professional community of security researchers. The white papers are provided to elevate collaboration and security within the industry and are not to be used for unlawful purposes. Security researchers are responsible for lawfully obtaining equipment and for complying with contracts and licenses for their research.

An increasingly common technique used by mobile application developers to prevent reverse engineering of their internal APIs is to implement SSL pinning. SSL pinning is the process of only accepting a select number of SSL certificates as valid during mobile application network transactions.

A common way to understand how an application talks to either a web service or product is to install a self-signed SSL root certificate. This is possible in both Android and IOS. The idea here is to capture the network traffic while the device uses the self-signed root SSL certificate for all network transfers. Since you know the private key of the self-signed SSL certificate you can then decrypt the SSL network packets from the network capture and inspect what is being sent from your device.

When SSL pinning is in affect you will notice that as soon as you install the self-signed certificate and force the device to use it, you will no longer be able to use the target application. Commonly, you will not be able to get past the login page of the application. If you look at the network capture from a device that has SSL pinning enabled, you will notice that there is no traffic at all. This is because the SSL certificate check happens before any network transactions take place.

There are 3 common ways that Android applications will pin SSL certificates. The first is TrustManager within the Android API from the "java.net.ssl.TrustManager" class. The second is to use the OkHttp library which includes a "CertificatePinner" function. The third is to use the Network Security Configuration to issue a pinned certificate; this only works on Android 7 and above. You can read more about these methods here (https://www.netguru.com/codestories/3-ways-how-to-implement-certificate-pinning-on-android)

The easiest way to check if an application you are analyzing is using SSL pinning is to try to capture some traffic with a self-signed certificate. You can either use Burp Suite (https://portswigger.net/burp) or an Android application like "Packet Capture" (https://play.google.com/store/apps/details?id=app.greyshirts.sslcapture). If you get data in either of these methods, then congratulations, the application you are analyzing is not pinning a certificate. If you do not see any traffic, or the application will not log you in, then continue reading.

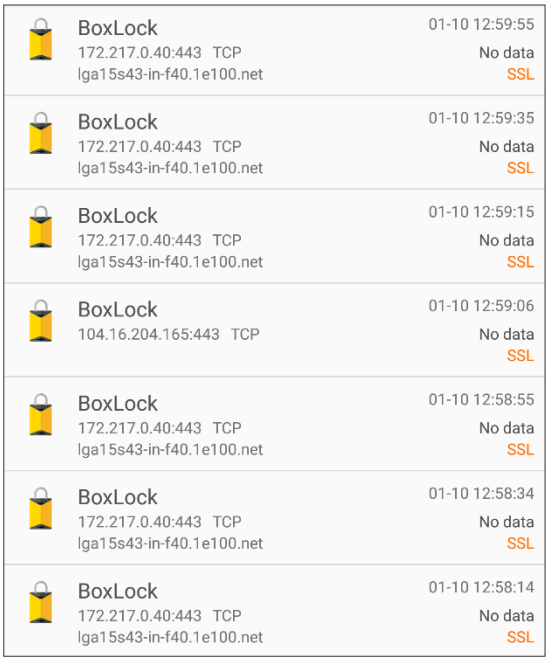


Figure 1. SSL Pinning enabled

The first step is to root your phone or use a phone already rooted. Be aware that the process of rooting your own phone will delete all your data, including your local storage (e.g. photos). I am not going to explain how to root your phone here as there are many tutorials online, e.g. https://www.androidcentral.com/root.

The next few steps will take place on a computer and not the Android device. You will probably need to have the Android SDK platform tools (ADB and Fastboot) on your machine to root your mobile device and they can be found at https://developer.android.com/studio#downloads.

You will also need to set up Burp Suite for the network capture and SSL decrypting. A great tutorial can be found at https://support.portswigger.net/customer/portal/articles/1841101-configuring-an-android-device-to-work-with-burp) and, if you need help installing the Burp Suite SSL Certificate on an Android device, you can follow the instructions found at https://support.portswigger.net/customer/portal/articles/1841102-Mobile%20Set-up_Android%20Device%20-%20Installing%20CA%20Certificate.html.

At this point you should have set up the Android device with root and installed our custom SSL certificate. You will also have set up the software on the computer to be used as a proxy for the Android phone, allowing us to capture all the network traffic from the device. Our next step will be to install and run Frida on the Android device and hook the system calls that are enforcing the SSL pinning to take place.

You can download Frida from GitHub here (https://github.com/frida/frida/releases), ensuring you select the correct Frida-server binary for your Android device and the correct Frida-core for your computer. If you have a newer Android device it is probably amd64, but you should always double check.

Once you have the Frida-server binary downloaded you will need to move it onto the Android device in a folder with executable permissions, which requires root. Please make sure you have “USB debugging” enabled on the Android device or ADB will not be able to connect.

```
#Push the frida-server file to the Android device
adb push /path/to/frida-server /data/local/tmp
#Push the Burp Suite SSL certificate to the device
adb push /path/to/burpca-cert-der.crt /data/local/tmp/cert-der.crt
#Now we will need to make the server executable
adb shell "chmod 755 /data/local/tmp/frida-server"
#With the frida-server and certificate in place we need to execute it.
adb shell # this will log you into the Linux underneath Android.
#Once you have a shell switch to the root user of the device.
su # this will prompt you on the phone to accept the command to use root, say yes
#Lastly, we will move to the correct folder and execute frida-server
cd /data/local/tmp
./frida-server
```

1. Push the frida-server file to the Android device
    a. adb push /path/to/frida-server /data/local/tmp
2. Push the Burp Suite SSL certificate to the device
    a. adb push /path/to/burpca-cert-der.crt /data/local/tmp/cert-der.crt
3. Now we will need to make the server executable
    a. adb shell "chmod 755 /data/local/tmp/frida-server"
4. With the frida-server and certificate in place we need to execute it.
    a. adb shell # this will log you into the Linux underneath Android.
5. Once you have a shell switch to the root user of the device.
    a. su # this will prompt you on the phone to accept the command to use root, say yes
6. Lastly, we will move to the correct folder and execute frida-server
    a. cd /data/local/tmp
    b. ./frida-server

With the Android device running the Frida-server, and all network traffic passing through the Burp Suite proxy, we will need to execute the Frida universal SSL unpinning script from the computer.

> frida –U –f <full name of Android application> --codeshare pcipolloni/universal-android-ssl-pinning-bypass-with-frida --no-pause

-U – Tells Frida to use the Android connected over USB debugging.

-f – Application to spawn on the device

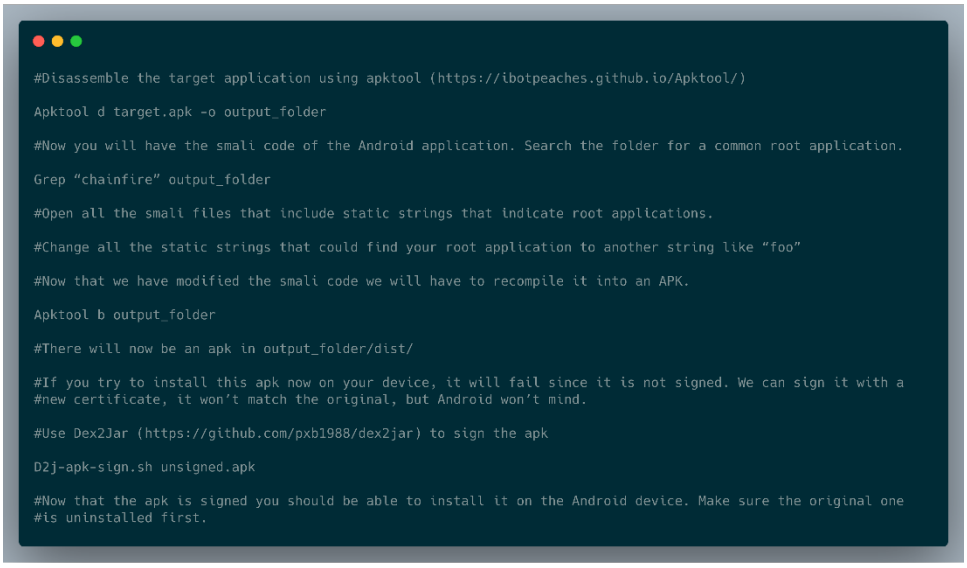
--codeshare – Will pull down the script from Frida's codeshare repository (https://codeshare.frida.re/).

--no-pause – Will automatically start the main thread after startup

At this point, if everything worked correctly, you should be able to login to the application, bypassing its SSL pinning requirement. You should also start to see traffic in Burp Suite with the packets decrypted.

If you are still not able to login to the application, and you have verified that you have followed the above steps correctly, the application may be using a Root checker and denying your rooted device from even attempting an SSL connection. As it is known that SSL pinning can be bypassed with a rooted device some Android application developers will include a root checking library that can find SU binaries or other artifacts of root applications on your device and deny access.

The best way to check if your target Android application is checking your phone for root is to decompile the application and look for libraries and strings. One library that I have run into before is Scottyab's Rootbeer library (https://github.com/scottyab/rootbeer). The way you can bypass these types of checks is to use something like a root cloaker (Magisk system-less root has this built-in) but the problem with them is that Frida cannot find the SU binary either, so it fails.



```
#Disassemble the target application using apktool (https://ibotpeaches.github.io/Apktool/)

Apktool d target.apk -o output_folder

#Now you will have the smali code of the Android application. Search the folder for a common root application.

Grep “chainfire” output_folder

#Open all the smali files that include static strings that indicate root applications.

#Change all the static strings that could find your root application to another string like “foo”

#Now that we have modified the smali code we will have to recompile it into an APK.

Apktool b output_folder

#There will now be an apk in output_folder/dist/

#If you try to install this apk now on your device, it will fail since it is not signed. We can sign it with a
#new certificate, it won’t match the original, but Android won’t mind.

#Use Dex2Jar (https://github.com/pxb1988/dex2jar) to sign the apk

D2j-apk-sign.sh unsigned.apk

#Now that the apk is signed you should be able to install it on the Android device. Make sure the original one
#is uninstalled first.
```

The steps needed to bypass a root checker are:

1. Disassemble the target application using apktool (https://ibotpeaches.github.io/Apktool/)
    a. apktool d target.apk -o output_folder
2. Now you will have the smali code of the Android application. Search the folder for a common root application.
    a. grep “chainfire” output_folder
3. Open all the smali files that include static strings that indicate root applications.
4. Change all the static strings that could find your root application to another string like “foo”
5. Now that we have modified the smali code we will have to recompile it into an APK.
    a. apktool b output_folder
6. There will now be an apk in output_folder/dist/
7. If you try to install this apk now on your device, it will fail since it is not signed. We can sign it with a new certificate, it won’t match the original, but Android won’t mind.
8. Use Dex2Jar (https://github.com/pxb1988/dex2jar) to sign the apk
    a. d2j-apk-sign.sh unsigned.apk
9. Now that the apk is signed you should be able to install it on the Android device. Make sure the original one is uninstalled first.

If you successfully remove all the root checking strings and rerun the Frida script you should be able to unpin the SSL certificate correctly and have access to the network traffic of the target application.