Four ways to bypass Android SSL. Verification and Certificate Pinning

Mykhailo Antonishyn

NAS G.E.Pukhov Institute for Modelling in Energy Engineering of Ukraine
General Naumov Str. 15, Kyiv, Ukraine, 03164
antonishin.mihail@gmail.com, orcid.org/0000-0002-2665-0066

INTRODUCTION

Gone are the days when mobile applications stoically ignore all manners of SSL errors and allow you to intercept and modify their traffic at will. Instead, most modern applications at least check the presented certificate chains to a valid, trusted certificate authority (CA). All pentesters like to convince the app that our certificate is valid and trusted so we can man-in-the-middle (MITM) it and modify its traffic.

PURPOSE

The report looks at SSL-pinning technology which should protect the mobile application against MITM attack. The most important is SSL pinning testing. The report describes four ways of bypass SSL-pinning.

MAIN PART

SSL pinning also is known as Public Key Pinning is an attempt to solve these issues, ensuring that the certificate chain used is the one your app expects by checking a particular public key or certificate appears in the chain. Four general ways SSL-pinning to bypass [2, 3]:

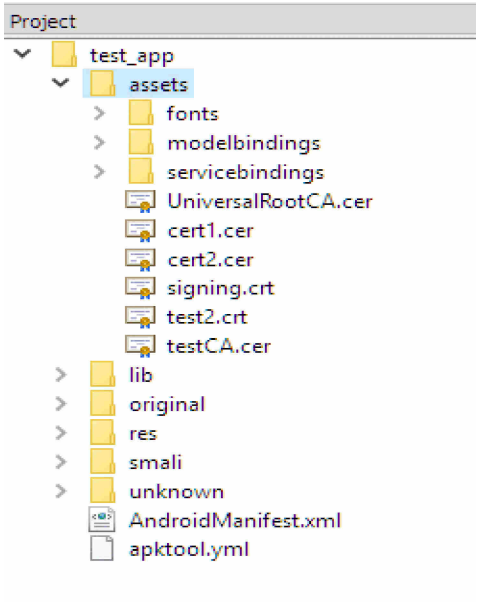
- Adding a custom CA to the trusted certificate store;
- Overwriting a packaged CA cert with a custom CA cert;
- Objections;
- Xposed Modules.

Technique 1 – Adding a Custom CA to the User Certificate Store

The simplest way to avoid SSL errors is to have a valid, trusted certificate (Fig.1). This is relatively easy if security specialist can install new, trusted CAs to the device – if the operating system trusts custom CA, it will trust a certificate signed by custom CA. Android has two built-in certificate stores that keep track of which CAs are trusted by the operating system – the system store (holding pre-installed CAs) and the user store (holding user-installed CAs). When the application is repackaged with this updated manifest, it will trust the user-added CA store. Alternatively, if running on a specific platform version is required, we can define specific trust anchors in the '/res/xml/network_security_config.xml' configuration file of the APK. For example, the following file defines a new trusted CA that needs to be stored at /res/raw/my_ca.xml.

my_ca.xml

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
<base-config>
<trust-anchors>
<certificates src="@raw/my_ca"/>
</trust-anchors>
</base-config>
</network-security-config>
```



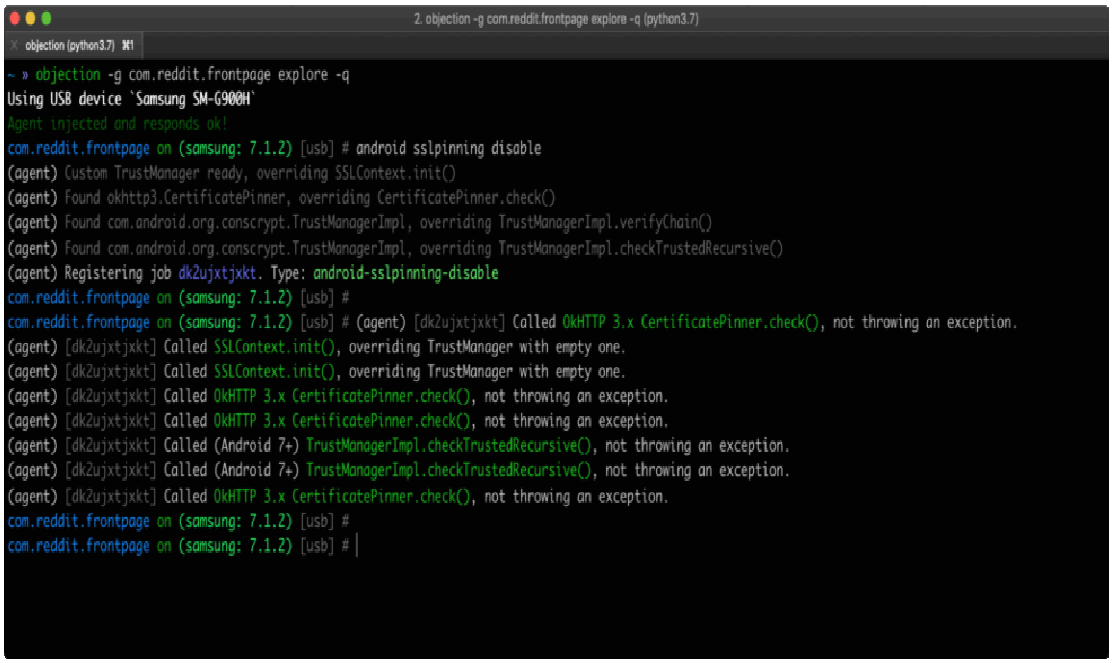
Trusted certs

If the application is only validating that the presented certificate is valid, this technique should allow you to establish a successful MITM condition.

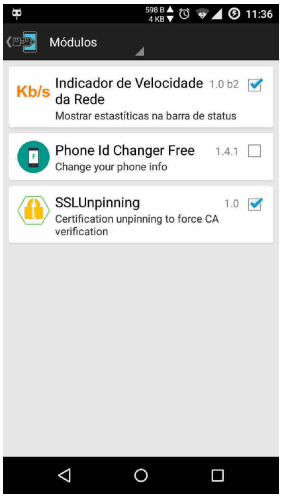
Technique 2 – Overwrite Packaged CA Certificate with Custom CA Certificate

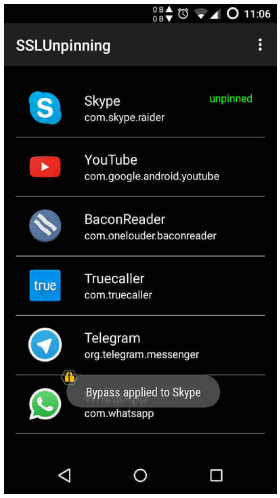
If a security specialist successfully installs his/her own certificate to the user-added CA store, the application is targeting Android 6.0 The developers may have taken additional steps to restrict the set of CAs trusted by the application. Recalling from technique 1 we defined a custom trust anchor and provided a path to a CA certificate – this is intended functionality that may be used by developers to attempt to protect their application from SSL interception. If a custom certificate chain is being distributed with an application, extracting the APK and overwriting the provided CA with our custom CA should be enough to cause our intercepting certificate to be trusted. Note that in some cases, additional verification of the trust chain may be happening, so this method may yield mixed results.

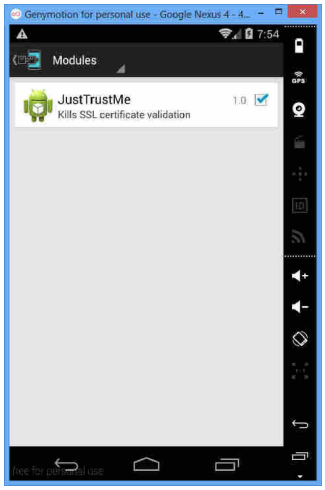
Opening the APK with a tool such as APK Studio makes the presence of certificates bundled with the deployed application obvious. In the image above, the certificates are located under the 'assets' directory. Overwriting the aptly named 'UniversalRootCA' certificate with our custom CA should allow us to trick the application into accepting our certificate [1, 2, 6].



Objection SSL-pinning bypass







Xposed SSLUnpinning

Xposed JustTrustMe

Technique 3 – Objections

If installing custom CA isn't enough to successfully proxy SSL traffic, it's possible that the application is performing some kind of SSL pinning or additional SSL validation. Typically to bypass this type of validation it needs to hook the application's code and interfere with the validation process itself. This type of interference uses to be restricted to rooted/jailbroken phones, but with the help of Objections, it's now possible to instrument an Android application and gain access to the full suite of Objections functionality without rooting a device. Typically, Objections will run on the operating system as a stand-alone program – but that requires rooting a device. Objections contains of the functionality of Frida but encapsulated in a dynamic library that gets loaded by the target app at runtime, allowing you to instrument and modify the target app's code. To load Objections, we need to extract the APK, insert the dynamic library, edit some smali code so our dynamic library is the first thing that gets called at application startup, then re-package the APK and install it. To save time, however, there's yet another tool we can use – Objection. Objection automates this entire process and requires only the target APK to be provided on the command line [6].

Technique 4 – Xposed Modules

Xposed Module SSLUnpinning is a first tool for SSL pinning bypass. It can help security specialist to intercept the traffic from an app which uses certificate pinning, with a tool like Burp Proxy. The SSLUnpinning through Xposed Framework, makes several hooks in SSL classes to bypass the certificate verifications for one specific app, then you can intercept all your traffic [4, 5].

Xposed Module JustTrustMe is a second tool for SSL pinning bypass. It can help security specialist to intercept the traffic from an app which uses certificate pinning, with a tool like Burp Proxy. The JustTrustMe through Xposed Framework makes several hooks in SSL classes to bypass the certificate verifications for one specific app, then you can intercept all your traffic [5].

CONCLUSIONS

According to the results of the study, there are 4 general ways to bypass SSL-pinning, as a result of bypassing SSL-pinning, the information security specialist gets access to testing the web application server. After that, it is necessary to test the vulnerability of the software application server and check the level of protection of the information processed in it [4].

Keywords: SSL-pinning bypass, android application, android application security assessment.

REFERENCES

1. **Four Ways** to Bypass Android SSL Verification and Certificate Pinning, **2020** [Online]. Available: https://blog.netspi.com/four-ways-bypass-android-ssl-verification-certificate-pinning/. Accessed on: May 19, 2020.
2. **All about** SSL pinning bypass, **2020** [Online]. Available: https://ninadmathpati.com/all-about-ssl-pinning-bypass/. Accessed on: May 19, 2020.
3. **SSL PINNING**: Mobile banking protection on android with ssl certificate, **2020** [Online]. Available: https://www.emaro-ssl.ru/blog/ssl-pinning-for-android/. Accessed on: May 19, 2020.
4. **Xposed Module**: Just Trust Me, **2020** [Online]. Available: https://github.com/Fuzion24/JustTrustMe. Accessed on: May 19, 2020.
5. **Xposed Module**: SSLUnpinning, **2020** [Online]. Available: https://github.com/ac-pm/SSLUnpinning_Xposed. Accessed on: May 19, 2020.
6. **Android-ssl-bypass, 2020** [Online]. Available: https://github.com/iSECPartners/android-ssl-bypass. Accessed on: May 19, 2020.