The Ultimate Guide To Android SSL Pinning Bypass

Version 1

REDHUNT LABS

DISCOVER. ATTACK. REPEAT.

www.redhuntlabs.com

TABLE OF **CONTENTS**

01 What is SSL Pinning?

02 What is not SSL Pinning?

03 How is SSL Pinning implemented?

04 How to find if an app is SSL pinned?

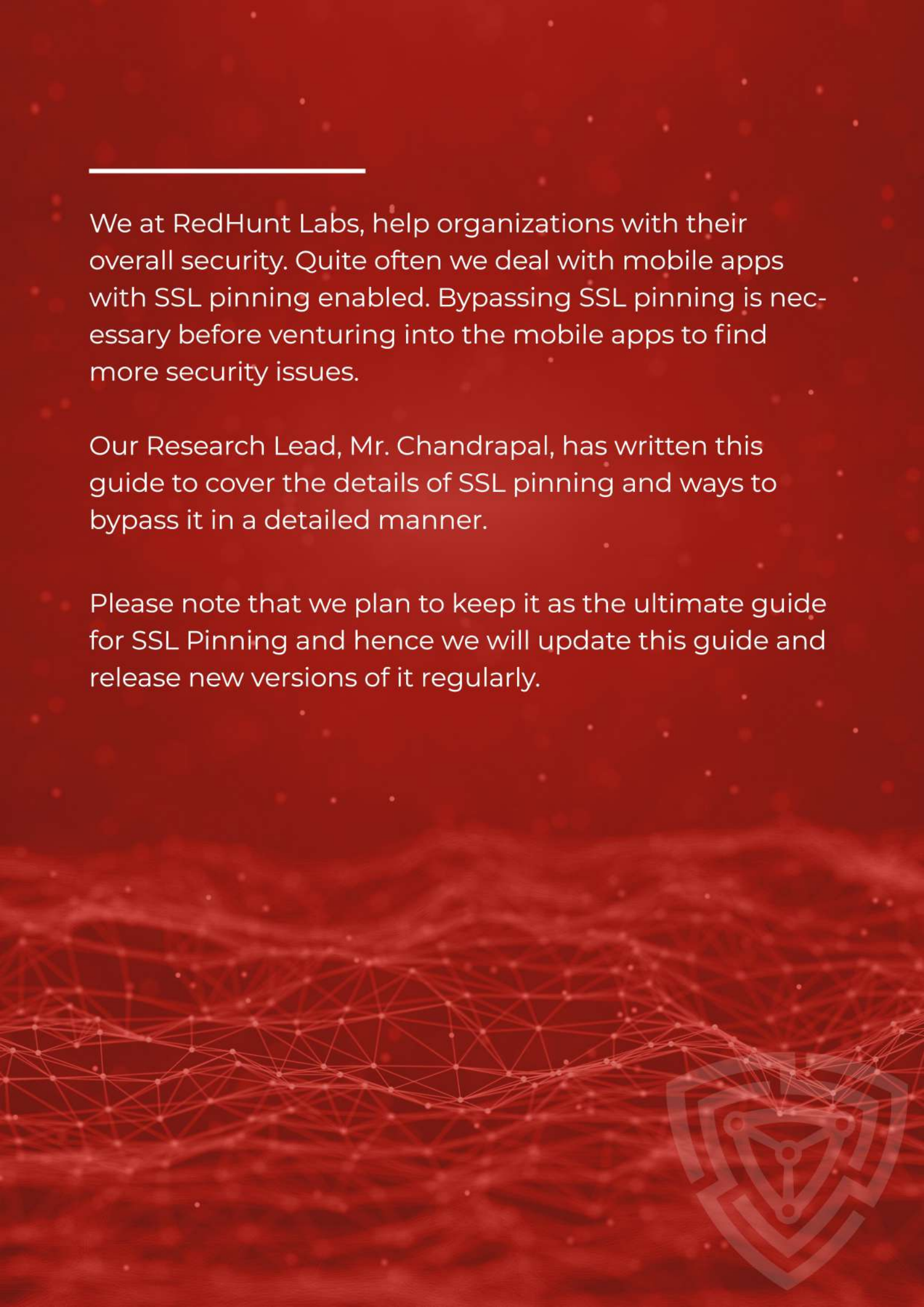
05 Bypassing SSL Pinning

06 Miscellaneous

We at RedHunt Labs, help organizations with their overall security. Quite often we deal with mobile apps with SSL pinning enabled. Bypassing SSL pinning is necessary before venturing into the mobile apps to find more security issues.

Our Research Lead, Mr. Chandrapal, has written this guide to cover the details of SSL pinning and ways to bypass it in a detailed manner.

Please note that we plan to keep it as the ultimate guide for SSL Pinning and hence we will update this guide and release new versions of it regularly.



Mobile apps are undoubtedly one of the critical assets in an organization's attack surface. Many applications that we pentest have "SSL pinning" enabled. A good number of those applications just have some mediocre SSL pinning logic. But a few apps have a very good implementation of security measures (in which SSL pinning is just one) and bypassing SSL pinning means bypassing a whole lot of other checks.

SSL certificate pinning is a standard security practice for mobile apps, be it Android or iOS apps. It's also called SSL public key pinning or just "pinning" in short. It's a defense-in-depth mechanism where the domain(s) are "pinned" with their SSL public keys/certificates. Simply put, if the mobile app receives any SSL certificate other than what it expects, the app rejects the connection.

Most mobile apps that process payments or PII data have SSL pinning. OWASP Mobile AppSec Verification Standard even recommends it for apps handling highly sensitive data.
This guide will walk you through SSL pinning basics, how it's implemented, and various tools and methods to bypass such protections in Android apps. All the techniques covered will work on both Android emulators and physical devices.

If you don't understand the terms related to the SSL certificate chain like Certificate Authority, Leaf certificate, etc, please read about it here before you continue with this guide.

What is SSL Pinning?

While creating mobile apps, having secure communication is necessary. HTTPS solves this problem by encrypting the traffic from the apps to the server and protecting the data's confidentiality and integrity. Android P (with Network Security Configuration) uses HTTPS by default unless explicitly turned off.

Using HTTPS solves the problem of users communicating over untrusted public WiFi networks. However, it doesn't protect against attacks when SSL certificates are issued by other intermediate CAs that the user's device trusts.

Also, if a user needs to look at the app's HTTPS connections, he/she can explicitly trust a locally created certificate authority and issue a self-signed SSL certificate for the domain the app connects to. This is a common practice for HTTPS proxies like Burp Suite and OWASP ZAP.

SSL pinning is a way for apps to identify if they communicate with the intended server over HTTPS. This is done by verifying some part of the SSL/TLS certificate keychain, usually the `subjectPublicKeyInfo` part. This reduces the attack surface and protects against the attacks mentioned above.

SSL pinning is not restricted to just mobile apps. It was allowed for websites as HTTP Public Key Pinning (HPKP) header and was later deprecated as it caused more problems than those it solved.

What is Not SSL Pinning?

For beginner pentesters, it's easy to get confused about what SSL pinning is and what is not. Adding HTTPS scheme to all the URLs within the app doesn't mean SSL pinned. Adding Burp's root CA certificate to the device alone doesn't bypass SSL pinning always.

This confusion is because the process of bypassing SSL pinning involves two steps. First, adding a custom CA certificate to the mobile device (like Burp CA). Second, tampering / bypassing the certificate pinning logic by making the app trust the CA certificate added in the first step.

Apps without any SSL pinning check would work fine just after the first step. For those that are SSL pinned, they would work only after the second step is successful.

How is SSL Pinning Implemented ?

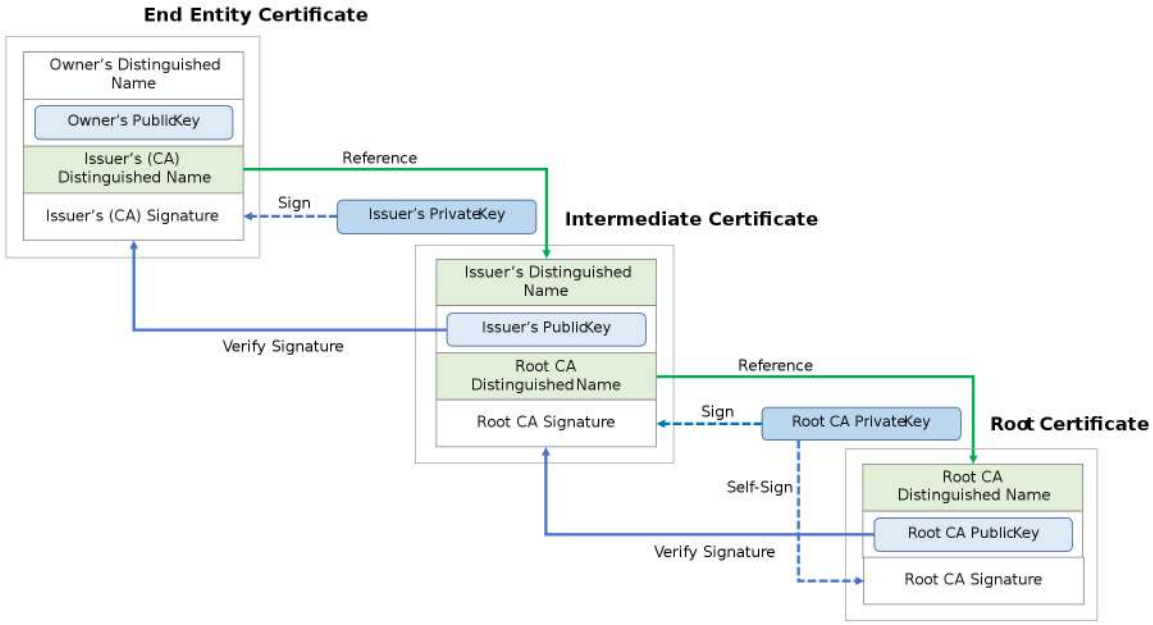
Understanding different SSL pinning implementations allow you to get a deeper insight into what can go wrong and how one can bypass it. If you are not interested in it, feel free to skip to the next section on "*How to find if an app is SSL pinned*."

There are multiple ways to implement SSL pinning in Android. One can use different parts of the SSL certificate chain for pinning or implement the pinning logic only for certain parts of the app/libraries used (like payment gateways, etc.) in the app.

(To keep this guide simple, we assume that SSL pinning is implemented for all the network connections the app makes. Even if the pinning is only for certain parts of the app/libraries, the detection and bypass process remains the same.)

Pinning With Different Components of the Certificate Chain

The SSL Certificate chain (or Chain of Trust) of a domain generally consists of a leaf certificate (aka end-entity certificate), an intermediate certificate, and a root CA certificate. This root CA certificate is what the mobile device's trust.



(Source: https://en.wikipedia.org/wiki/Public_key_certificate#/media/File:Chain_Of_Trust.svg)

One could choose to pin using either the leaf, the intermediate, or the root certificate. Pinning the leaf certificate gives the utmost security but hurts badly when the pinned SSL certificate expires.

As Matthew Dolan suggests, the choice of the certificate to pin impacts the level of protection you achieve in the app, decreasing as you reach the root certificate.

Pinning in Android N

The Network Security Configuration in Android N (API 24) and above makes SSL pinning easy. Adding the certificate hashes under "`pin digest`" in `res/xml/network_security_config.xml` gets the job done.

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <domain-config>
        <domain includeSubdomains="true">example.com</domain>
        <pin-set expiration="2018-01-01">
            <pin digest="SHA-256">7HIpactkIAq2Y49orFOOQKurWxmmSFZhBCoQYcRhJ3Y=</pin>
            <!-- backup pin -->
            <pin digest="SHA-256">fwza0LRMXouZHRC8Ei+4PyuldPDcf3UKgO/04cDM1oE=</pin>
        </pin-set>
    </domain-config>
</network-security-config>
```

(https://developer.android.com/training/articles/security-config#CertificatePinning)

Pinning using 3rd Party Libraries

SSL pinning can be achieved on Android apps using but not limited to the following libraries:

- OkHttp
- Retrofit
- Picasso
- Volley
- Apache HttpClient

To see code examples of SSL pinning using the above libraries check out Matthew Dolan's extensive article.

Various apps implement SSL pinning using different methods. There is room for strengthening SSL pinning in case of weak implementation, or it is broken, but in this guide, we will skip it, and we may cover it in the next part.

How to Find If An App is SSL Pinned?

You can try to bypass a security measure like SSL pinning only when the security measure is in place. If you are starting an Android app pentest, the initial step is to understand if the app has SSL pinning.

If you already know that the target app is using SSL pinning, then feel free to skip over to the next section "*Bypassing SSL Pinning*".

There are various ways to determine if an app is SSL pinned and can broadly be categorized under static and dynamic analysis.

Static Analysis:

Static analysis involves decompiling the app and analyzing the code and logic of the app by code. We recommend using jadx instead of jd-gui as jadx is well maintained, has good community support, and is regularly updated.

Once the app is decompiled, use the following techniques to find if the app is SSL pinned.

1. Checking for Pin Digest in network_security_config.xml file

Android 7.0 (API 24) and above allows adding certificate pins in the network security configuration file. To find if the Android app has a network security config file, check for "`android:networkSecurityConfig`" in the Android Manifest file.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
    <application android:networkSecurityConfig="@xml/network_security_config"
                    ... >
        ...
    </application>
</manifest>
```

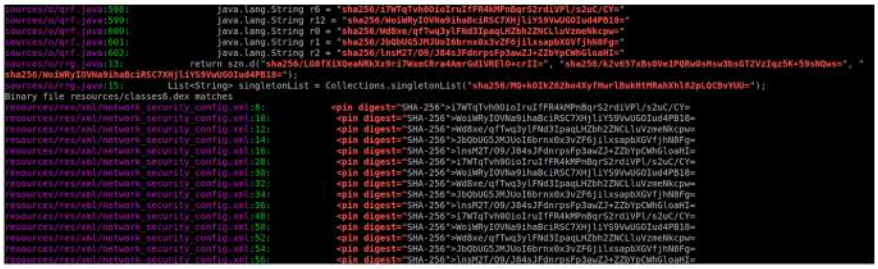
If it exists, then check for "`pin-set`" or "`pin digest=`" in the network security config file (which usually exists at `res/xml/network_security_config.xml`)

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <domain-config>
        <domain includeSubdomains="true">example.com</domain>
        <pin-set expiration="2018-01-01">
            <pin digest="SHA-256">7HIpactkIAq2Y49orFOOQKurWxmmSFZhBCoQYcRhJ3Y=</pin>
            <!-- backup pin -->
            <pin digest="SHA-256">fwza0LRMXouZHRC8Ei+4PyuldPDcf3UKgO/04cDM1oE=</pin>
        </pin-set>
    </domain-config>
</network-security-config>
```

2. Grepping the Decompiled Source Code

If it's a native app and can be decompiled to source code, there is another way to test SSL pinning. Using a simple grep over the source code will suggest to you if there's any type of SSL pinning added to the app.

```
grep -rni "<pin digest=\"\|CertificatePinner\.Builder(\|\.
certificatePinner(\|sha256/[A-Za-z0-9\/+=]\{43,45\}"
```

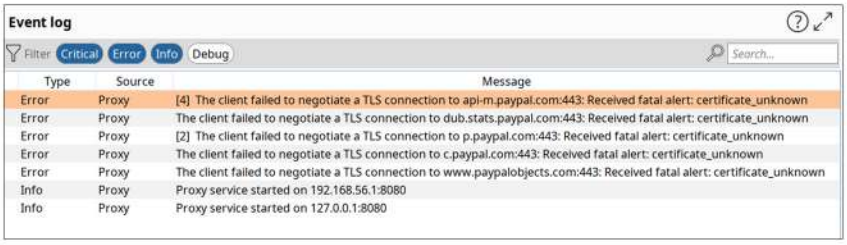


Dynamic Analysis

Dynamic analysis involves running the Android app and then analyzing the behavior of the app, network communications, etc. When the app is running, use the following techniques to find if the app is SSL pinned.

1. Proxy Error Logs

After you successfully set up HTTP(S) proxy with the mobile device, if you encounter a TLS negotiation error when using the app, then in most cases, it's an indication of SSL pinning.



2. Logcat Logs

It's common for developers to log any errors (including SSL errors) in logs. In most cases, the logs indicate some kind of SSL pinning with the app.

Bypassing SSL Pinning

Once you find that an app has SSL pinning, the next step is to bypass it. The effort to bypass SSL pinning differs with each app. If SSL pinning is the only security measure, the bypass may be a bit easier. Else other security measures like anti-tamper detection, root detection, etc will make the SSL pinning bypass tougher.

We will follow a process that begins with the easiest bypass technique and move to increasingly difficult ones if that bypass technique doesn't work. Here's our process:

1. Using a Lower Version of Android

Works on	N/A
Effort to setup & Bypass	★☆☆☆☆
Dependencies	N/A

Suppose the Android app uses SSL pinning verification that works above a particular Android version. In that case, installing the app on a lower Android version may help bypass the protection.

One typical example is apps using Network Security Config file to pin SSL certificate hashes but allows the app to be run on Android versions lower than Android N.

Since this pinning method works for Android N and higher versions, installing it in Android M (API level 23) and lower allows to bypass SSL pinning.

(**Note:** This technique doesn't work if the app also uses TrustKit library)

2. Modification of Network Security Config File

Works on	N/A
Effort to setup & Bypass	★★★★☆
Dependencies	apktool, keytool, jarsigner, zipalign, aapt

If you have read the above "Pinning in Android N" section, you will be aware that SSL pinning can easily be achieved by adding "`pin digest`" in the Network Security Config file.

Removing the "`pin digest`" and recompiling the apk might bypass the SSL pinning if the app just uses the Network Security Config file for SSL pinning logic. There are quite a lot of dependencies for this technique, however, it's a one-time effort to set up.

3. JustTrustMe - Xposed Module

Works on	Rooted devices
Effort to setup & Bypass	★★☆☆☆
Dependencies	Xposed Framework

JustTrustMe is an Xposed module created to bypass certificate pinning by disabling SSL certificate checks. Even though it sounds good, there's "no guarantee it might work always". It's worth a try.

Install Xposed framework on your rooted device. Install JustTrustMe APK and enable it in Xposed Installer. Restart the device to enable it fully.

JustTrustMe app hooks functions that implement SSL pinning in popular Android libraries and replace their functionality to allow any SSL certificate.

4. Objection

Works on	Non-rooted & Rooted devices
Effort to setup & Bypass	★★★★☆
Dependencies	adb, apktool, aapt, keytool, jarsigner, zipalign

Objection is an open-source runtime mobile exploration toolkit that's powered by Frida. It can be easily installed using `pip3 install objection`. However, it has quite a few dependencies (like adb, apktool, aapt, and more) that need to be set up at first. This tool comes with multiple functionalities in which SSL pinning bypass is just one.

To bypass SSL pinning, you need to patch the target Android apk file with Frida Gadget. Patching involves decompiling the apk file, modifying the code to add Frida Gadget, repackaging the modified code, and signing it. Objection tool does all of it for you with the following command:

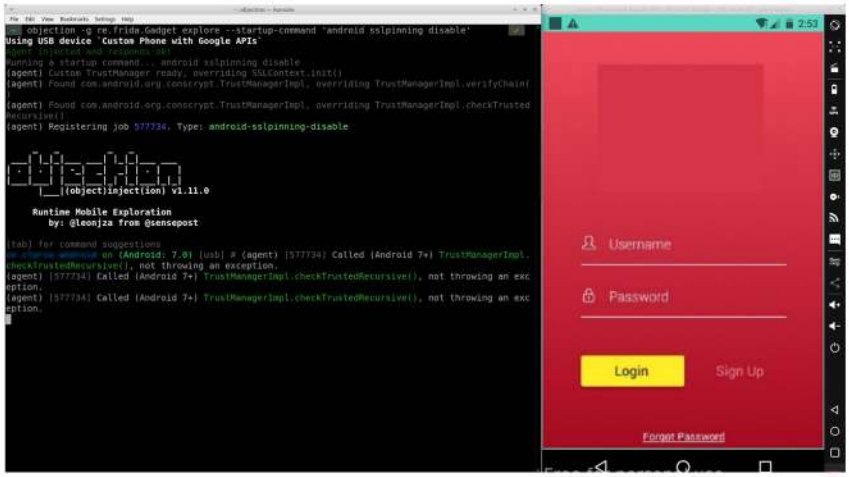
```
objection patchapk -s package.apk
```

There are a good number of chances that repackaging might not work. If you face that situation, always check out their wiki or issues section to find the fix.

If repackaging the file is successful, objection creates a new apk file that ends with the name `.objection.apk`. For the above example, the output will be `package.objection.apk`.

Install the patched apk file on an Android device. Once installed, open the app. The app waits till the objection tool connects to the Frida gadget. Execute the following to bypass SSL pinning:

```
objection explore --startup-command 'android sslpinning disable'
```



5. Frida Gadget

Works on	Non-rooted & Rooted devices
Effort to setup & Bypass	★★★☆☆
Dependencies	Frida, objection (optional), adb, apktool, aapt, keytool, jarsigner, zipalign

Frida is a dynamic instrumentation framework that allows you to hook and change the mobile app's logic at runtime. Frida is so powerful that it "requires its own ultimate" guide to list all its features.

Frida has 3 modes of operation in which "injected" and "embedded" are the most commonly used modes for Android SSL bypassing.

Injected mode is achieved by running the Frida server only on a rooted device. Apps that have additional security measures like root detection, malicious packages detection, etc make this injected mode a bit difficult.

As the first step, we use the embedded mode with the help of a shared library - Frida Gadget. This shared library needs to be packed along with the target app. So the high-level process looks like decompile the application, add Frida gadget to the source code, and recompile the source code.

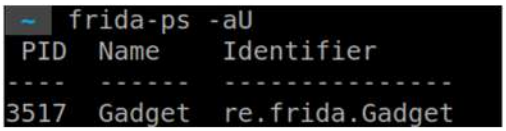
You can automate this by using the above Objection command:

```
objection patchapk -s package.apk
```

Else, you can follow the manual way of patching the apk described in this guide.

Once the APK is patched, install Frida tools on the attacker machine using `pip3 install frida-tools`. After installing, you will see programs like frida, frida-ps, frida-ls-devices on your system.

Install the patched APK on an Android device and open it. The app waits till Frida connects to the Frida gadget. The output of



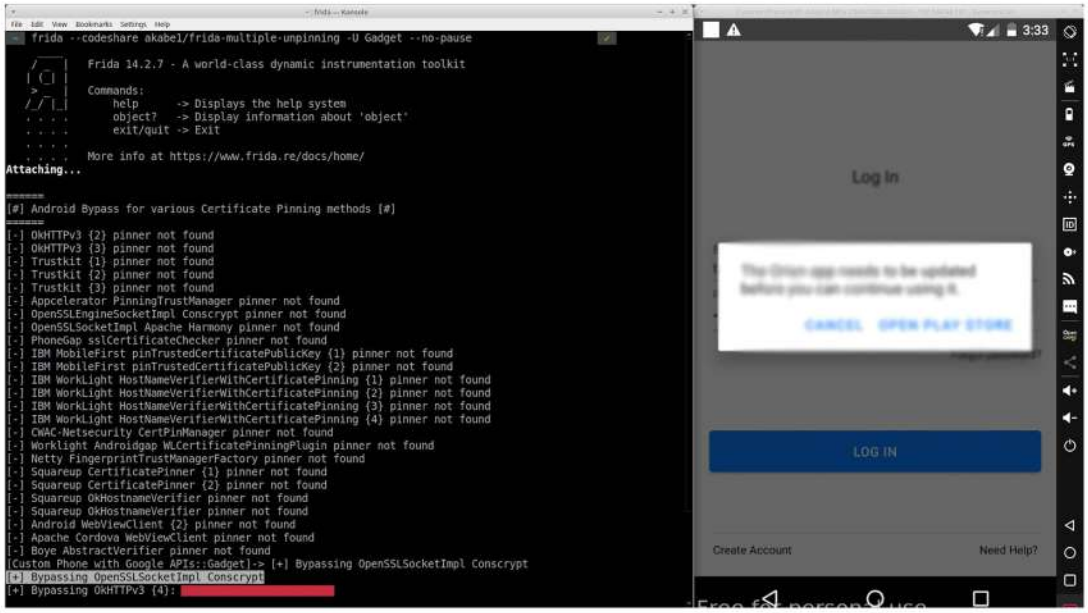
Frida community has few scripts that try to bypass all possible SSL pinning implementations.

Frida Multiple Unpinning

This Frida script bypasses most of the SSL pinning implemented using popular libraries. This includes the bypass techniques from popular Frida scripts like Universal Android SSL Pinning Bypass. What makes this script unique is that it's straightforward. Just set up the Frida server on the mobile device, fetch the package name, and execute the following command.

You can execute frida-multiple-unpinning using:

```
frida --codeshare akabe1/frida-multiple-unpinning -U Gadget
```



6. Frida

Works on	Rooted devices
Effort to setup & Bypass	★★★☆☆
Dependencies	Frida

In the above technique, we used recompiling the app with the Frida gadget and bypass SSL pinning. For apps that have anti-tamper checks / are released as Android App Bundles, then patching it might be tricky.

In such cases, using Frida's injected mode is the better option. This injected mode works only on rooted devices.

Run Frida's standalone server within rooted mobile devices and hook the target application in run time.

```
frida --codeshare akabe1/frida-multiple-unpinning -U -f
<com.android.package> --no-pause
```

7. Taint Analysis Using Frida

Works on	Non-rooted & Rooted devices
Effort to setup & Bypass	★★★★★
Dependencies	Frida, jadx-gui

This last technique is a time-consuming process and should be the last resort. If all the previous techniques failed, it most probably means that the target Android app is using its custom SSL pinning logic.

This technique requires you to understand decompiled Java code logic and involves lots of trial and error.

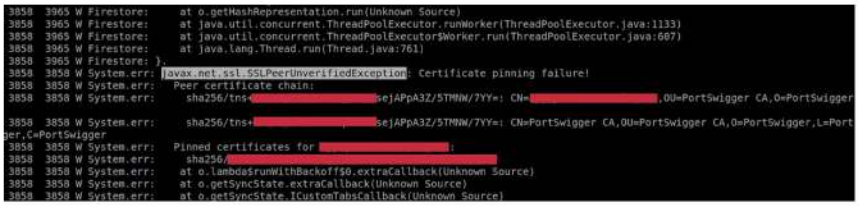
As the first step, using jadx-gui we manually get a list of interesting classes of the target app. This process can be automated to some extent using Frida's EnumerateLoadedClasses function but if the app uses proguard / dexguard for obfuscation, this automated technique fails.

Once you have the list of classes, you can hook all the functions of those classes and continue using the app. Then wait till there's an SSL error in the Burp logs/Android logcat output.

When there's an SSL error, the Frida script would have logged all the methods that were invoked before the error. Then you manually analyze each method to see if it implements some kind of SSL pinning logic and write a custom Frida script to bypass the logic.

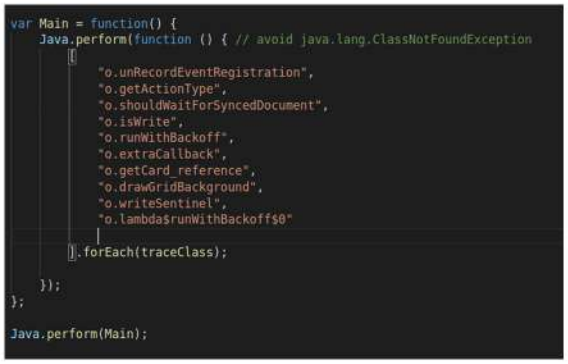
For example, in an Android app pentest, we found the app's SSL pinning was not bypassed by any of the above techniques. The app had some level of dexguard obfuscation.

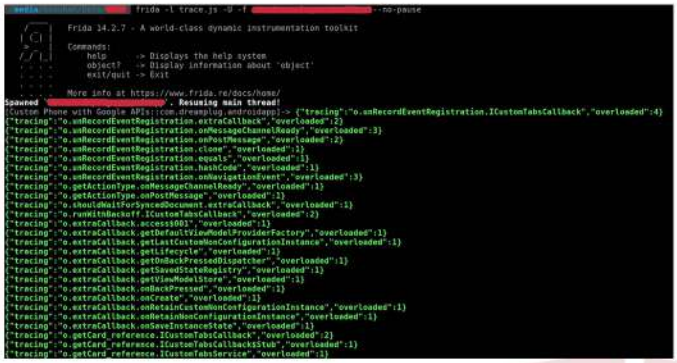
By manually analyzing logcat errors and the decompiled Java code in jd-gui, we found some classes that have were related to SSL pinning:



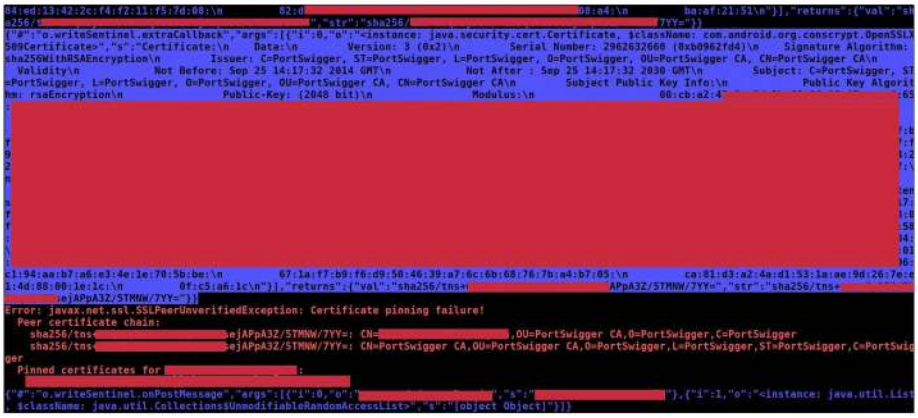
We hooked all the functions of those classes using Frida script using the command:

`frida -l trace.js -U -f com.package.app --no-pause`





We found that the `o.writeSentinel.extraCallback` function is getting called just before the SSL exception occurred.



Digging deeper into the code logic of `o.writeSentinal.extraCallback` and other functions it invoked showed that the app implements its own SSL verification checks. The function that raises `SSLPeerUnverifiedException` is a void function and its only task was to terminate the connection in case of SSL certificate hash mismatch.

In order to bypass this logic, we coded a Frida script that hooks the `onPostMessage` function and just returns nothing. As this was a void function and had no other logic apart from SSL pinning, the script was much simpler.

```
var Main = function() {
    Java.perform(function () {
 Java.use('o.writeSentinel').onPostMessage.implementation = function () {
            console.log('[+] App triggered writeSentinel function');
            return
        }
    });
};

Java.perform(Main);
```

Using the taint analysis technique, we were able to successfully bypass SSL pinning in the app. The SSL pinning implementations could differ but the process to figure out how the SSL logic is implemented stays the same. In this example, the Frida script was easy as the app only had SSL pinning security measures (along with obfuscation). If it had other security measures or had SSL pinning checks in shared libraries, the bypass logic would have been a bit tougher.

Miscellaneous:

When it comes to bypassing SSL pinning with tools, a few tools are still recommended but are outdated.

Android SSL TrustKiller

(https://github.com/iSECPartners/Android-SSL-TrustKiller)

According to the tool's description: "This tool leverages Cydia Substrate to hook various methods to bypass certificate pinning by accepting any SSL certificate." However, Cydia Substrate is not maintained and supports Android versions 2.3 through 4.3, making this tool not usable.

Inspeckage

(https://github.com/ac-pm/Inspeckage)

Inspeckage is a tool created for dynamic analysis of Android apps with the Xposed framework's help. It offers SSL pinning bypass but is limited to those apps using JSSE, Apache, or okhttp3 library. Also, the tool seems to be abandoned, and another contributor is trying to maintain the tool by adding bug fixes.

SSL Unpinning

(https://github.com/ac-pm/SSLUnpinning_Xposed)

SSL Unpinning is another tool from the author of Inspeckage, which is not maintained anymore. It is a subset of Inspeckage where the only functionality is SSL pinning bypass, and the tool just supports three libraries.

Find your Exposed Attack Surface.
Take action, and reduce your Attack Surface.

Discover
Attack
Repeat

We are a cybersecurity company specializing in Attack Surface Management (ASM) and security consultation to help SMEs and Large Enterprises track and secure their attack surface.
Learn more at: www.redhuntlabs.com

Get the latest cybersecurity research and tips at
www.redhuntlabs.com/blog

REDHUNT LABS
DISCOVER. ATTACK. REPEAT.

For more info follow us