Title

No SSL Pinning Verification Present

Description (Problem): -

It's very common for developers to implement communication over HTTPs, but not in a proper way. This improper implementation is reduced to replacing the protocol name in the URL from http to https, e.g. https://www.example.com. Indeed, such an implementation will enable TLS/SSL encryption (if the backend server supports it). However, it will not ensure a good enough security level.

The certificates will then allow the hacker to intercept encrypted communication. This kind of attack is called Man-In-The-Middle. It is the main reason why you should spend a bit more time and effort to implement HTTPs configuration correctly.

Solution

To avoid this exploit, developers should implement Certificate Pinning. It's a method that depends on server certificate verification on the client side. This verification requires the server certificate or its fingerprint to be previously known to the mobile app. When establishing a connection with the server, the app should compare the fingerprint with a certificate from the remote server. If the fingerprints are identical, then the connection is valid and the data transfer can proceed. If the fingerprints are not identical, then the app should reject the connection immediately, as it's compromised. The following 3 methods are the most popular ways to implement Certificate Pinning in Android apps.

Impact

If certificate pinning verification not present then it can lead to man in middle attack on app

Latest Network Security Configuration to implement after Android 7.0

The Android platform provides a new, easy tool to handle network configuration - Network Security Configuration (NSC). It has been available since Android 7.0. With NSC, you can declare communication methods, including Certificate Pinning, using XML files. To enable the configuration, you need to bind a configuration file with the Manifest. To bind it, use the networkSecurityConfig attribute in the Application tag. Here is a short snippet showing how to handle it:

Step1: - Create a network security config file under

res/xml/network_security_config.xml

Step2: - Add the android:networkSecurityConfig attribute to the application tag.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
  xmlns:android="http://schemas.android.com/apk/res/android"
  package="co.netguru.demoapp">
  <application
    android:networkSecurityConfig="@xml/network_security_config">
    ...
  </application>
</manifest>
```

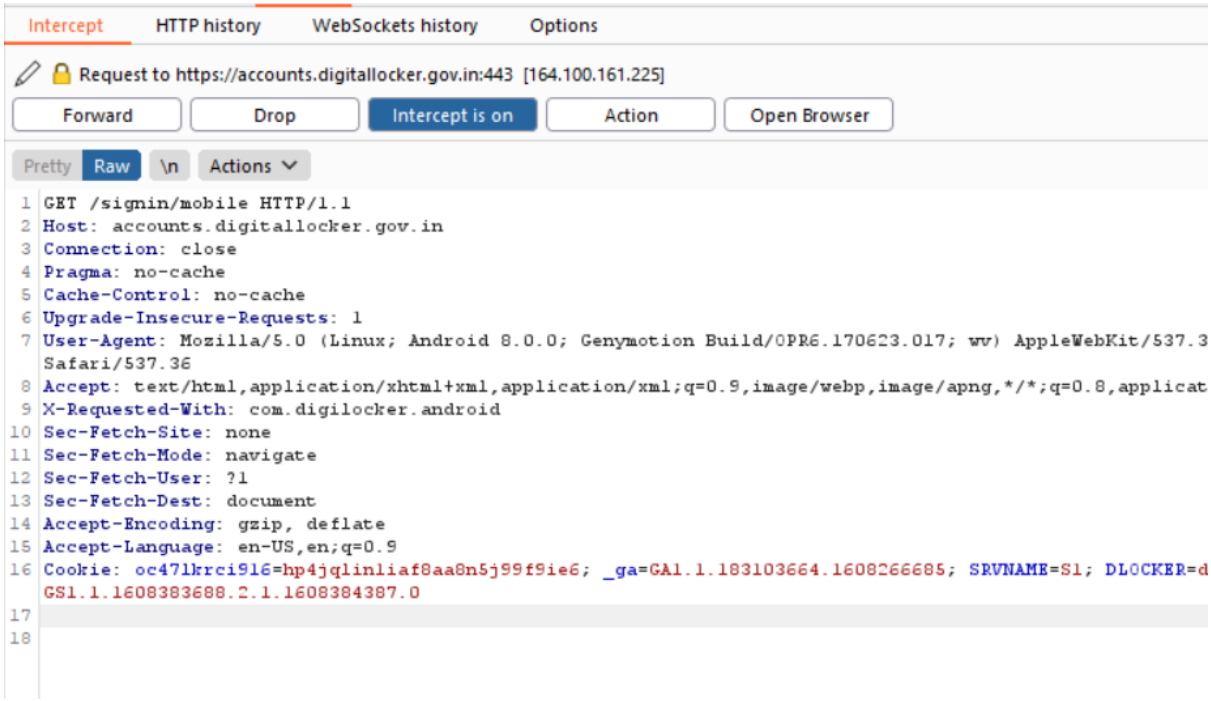
Step3: - Set up the configuration file and add fingerprints.

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
  <domain-config>
    <domain includeSubdomains="true">example.com</domain>
    <pin-set>
      <pin digest="SHA-256">ZC3lTYTDBJQVf1P2V7+fibTqbIsWNR/X7CWNVW+CEEA=</pin>
      <pin digest="SHA-256">GUAL5bejH7czkXcAeJ0vCiRxwMnVBsDlBMBsFtfLF8A=</pin>
    </pin-set>
  </domain-config>
</network-security-config>
```

Recommendation

I've presented three ways of implementing Certificate Pinning. Personally, I think that the most flexible option is to use CertificatePinner. This method is both short and universal - it works on all Android API levels out of the box. You must use OkHttp as the HTTP client, but the library is very easy to use and pretty much a standard on Android at this point, so it's recommended anyway.

Proof of Concept



Here, there is not restriction for request intercepting. I can directly intercept the request of application.