SSL PINNING GUIDE

On Android

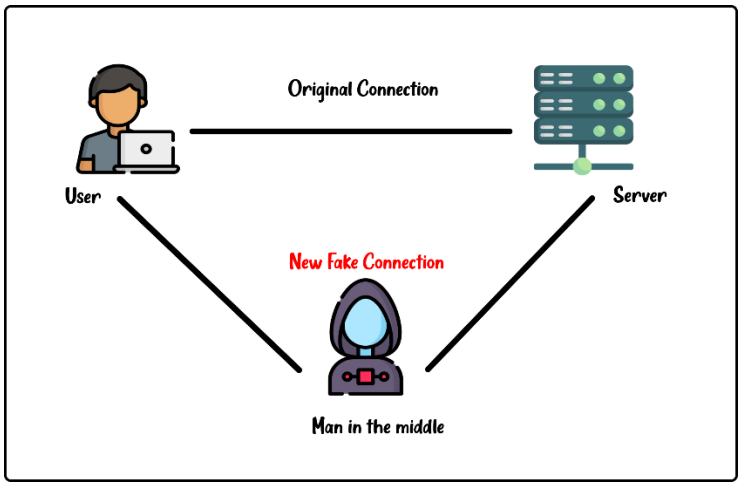
Oleh:

Ifvo Deky Wirawan

1. Pengertian *SSL/Certificate Pinning*

SSL (Secure Socket Layer) / Certificate pinning adalah suatu teknik keamanan aplikasi yang dilakukan untuk memastikan bahwa koneksi SSL yang dilakukan antara aplikasi dengan *server* aman dan sesuai dengan yang diharapkan oleh aplikasi tanpa ada interupsi dari pihak yang tidak berwenang.

Dengan menggunakan *SSL Pinning* maka aplikasi akan memeriksa secara spesifik dengan mencocokkan *hostname* yang tepat sehingga apabila ada seseorang mencoba membuat koneksi baru yang palsu diantara aplikasi dan server, maka aplikasi akan menolaknya. Sehingga dapat mencegah terjadinya *man-in-the-middle-attack*.



Gambar 1. Ilustrasi *man-in-the-middle-attack*

Sebenarnya penggunaan dari *SSL Pinning* belum tentu membuat aplikasi menjadi aman, karena tidak ada aplikasi yang aman. Hanya saja hal ini membuat orang lain menjadi lebih sulit dan membutuhkan effort yang lebih besar untuk membobol aplikasi tersebut.

2. Kelebihan *SSL/Certificate Pinning*

SSL/Certificate Pinning akan membantu dalam keamanan aplikasi, tapi hanya beberapa aplikasi saja yang membutuhkan implementasi metode ini, seperti aplikasi yang berhubungan dengan *mobile banking*, *financial*, *game* dan lainnya.

3. Kekurangan *SSL/Certificate Pinning*

SSL/Certificate Pinning memiliki kekurangan yaitu apabila *server* mengubah *SSL certificate* maka aplikasi harus melakukan *force update*. Karena *SSL certificate* yang berada di *server* dan aplikasi tidak lagi sama.

4. Implementasi *SSL/Certificate Pinning*

Ada beberapa cara yang dapat diterapkan dalam implementasi *SSL/Certificate Pinning* diantaranya yaitu:

1. *Certificate Pinner – Okhttp*
2. *Network Security Config*
3. *TrustManager & SSLSocketFactory*

4.1. Certificate Pinner - Okhttp

Okhttp merupakan Pustaka HTTP yang sangat popular digunakan untuk android, yang biasanya digunakan pada komunikasi dengan REST di android yaitu Retrofit. Cara menggunakannya yaitu dengan menambahkan code berikut di bagian Okhtttp Builder.

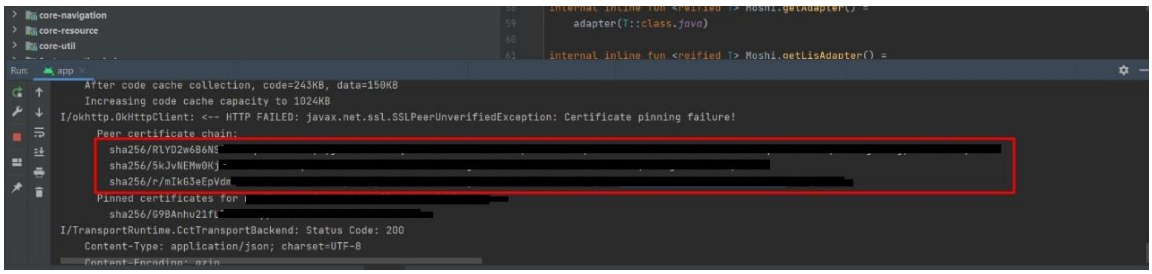
```
1 val certificatePinner = CertificatePinner.Builder()
2       .add(
3             "www.example.com",
4             "sha256/XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
5       ).build()
6
7 val okHttpClient = OkHttpClient.Builder()
8       .certificatePinner(certificatePinner)
9       .build()
```

Catatan:

Hostname = "www.example.com"

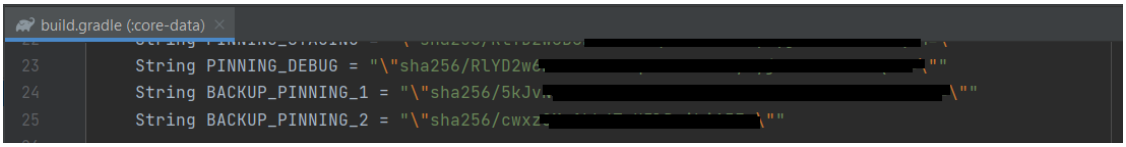
Pin = "sha256/XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"

Pada saat pertama kali menambahkan metode ini dan kita tidak mengetahui pin yang terdapat pada *server*, maka bisa menggunakan *random pin* terlebih dahulu. Setelah itu kita dapat melihat *certificate* yang sebenarnya di log pada android studio.



Gambar 2. *Log* setelah memasang *certificate pinner* di *Okhttp*

Penting: Untuk bagian hostname dan pin saya merekomendasikan untuk menyimpannya kedalam buildconfig Gradle agar lebih aman daripada ditampilkan polos di kotlin/java class seperti dibawah ini.



Gambar 3. Pin pada *gradle*

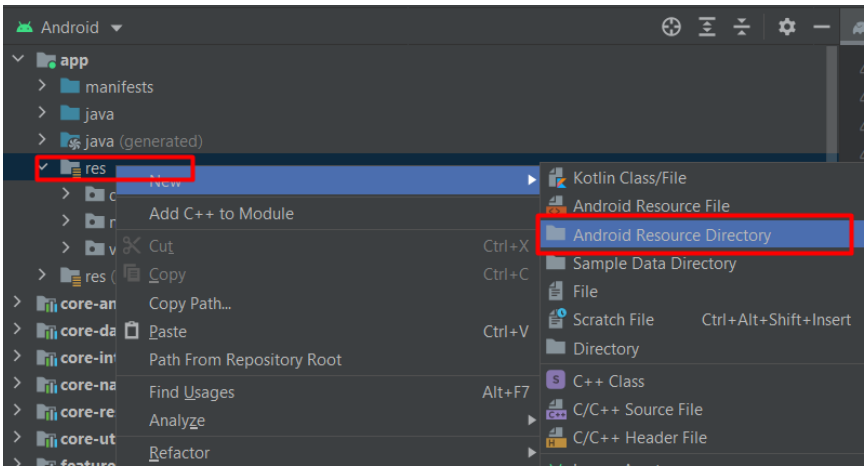
```
debug {
    buildConfigField("int", "schemaDatabaseVersion", schemaDatabaseVersion)
    buildConfigField("int", "schemaDatabaseBeforeVersion", schemaDatabaseBeforeVersion)
    buildConfigField("String", "BASE_URL", URL_DEBUG)
    buildConfigField("String", "HOSTNAME", HOSTNAME_DEBUG)
    buildConfigField("String", "PINNING", PINNING_DEBUG)
}
```

Gambar 4. Pemasangan pin pada *buildConfig*

4.2. Network Security Config

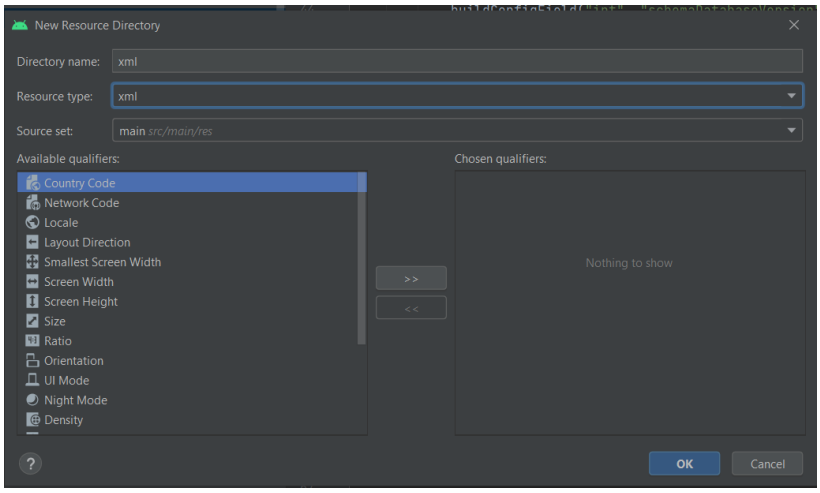
Pada android API level 24 keatas, kita dapat mengimplementasikan *SSL Pinning* menggunakan file XML network security config. Berikut ini merupakan cara untuk mengimplementasikannya.

a. Buka *folder* res → *new* → *Android Resource Directory* → xml



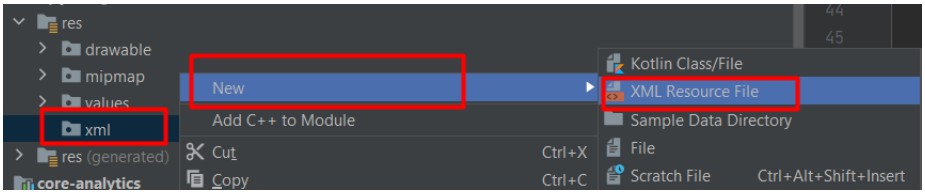
Gambar 5. Buka *Folder Android Resource Directory*

b. Pilih *resource type* berbentuk xml



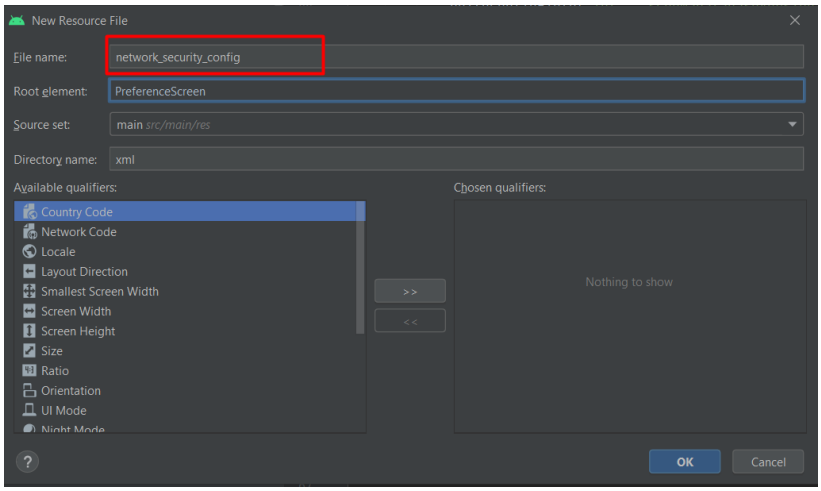
Gambar 6. Pilih *resource type* xml

c. Buat *file* baru di folder xml → *new* → xml *resource* file



Gambar 7. Buat file baru xml *resource file*

d. Beri *file* tersebut dengan nama *network security config*



Gambar 8. Membuat *file network security config*

e. Ubah isi *file* tersebut dengan *code* sebagai berikut:

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

f. Buka file *Android Manifest* dan hapus *code* usesCleartextTraffic true karena apabila kondisinya true semua koneksi akan masuk tanpa di *filter*.

Gambar 9. Menghapus code *usesCleartextTraffic* true

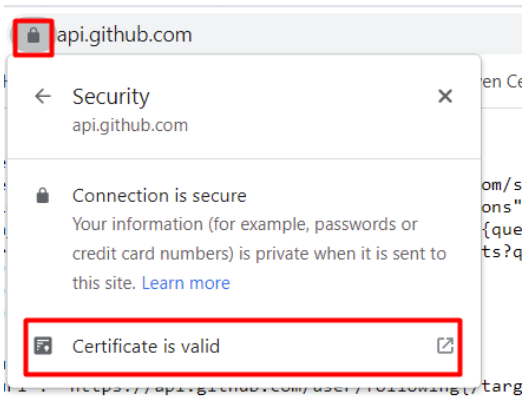
g. Terakhir tambahkan *code* berikut pada *android manifest*

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

4.3. Trust Manager & SSL Socket Factory

Okhttp merupakan cara lama dan bersumber dari package *javax.net.ssl* . Berikut ini adalah cara mengimplementasikannya.

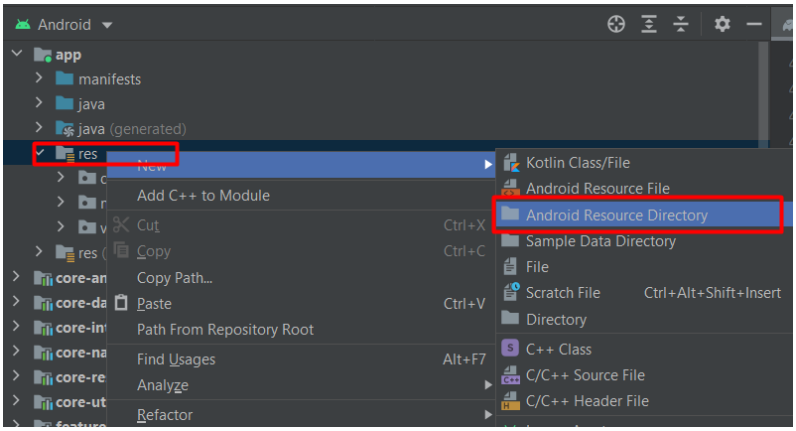
a. Buka *browser* dan sebagai contoh masukkan url api.github.com



b. Kemudian klik *certificate* (valid) → Pilih tab ***Details*** → ***Copy* to *File***. Maka akan muncul *Certificate Export Wizard* → Klik *Next* → Pilih ***Base-64 encoded X.59* (.CER)** → Klik *Next* → Pilih tempat penyimpanan (beri nama **mycertificate**) → Klik *Next* → Klik *Finish*.

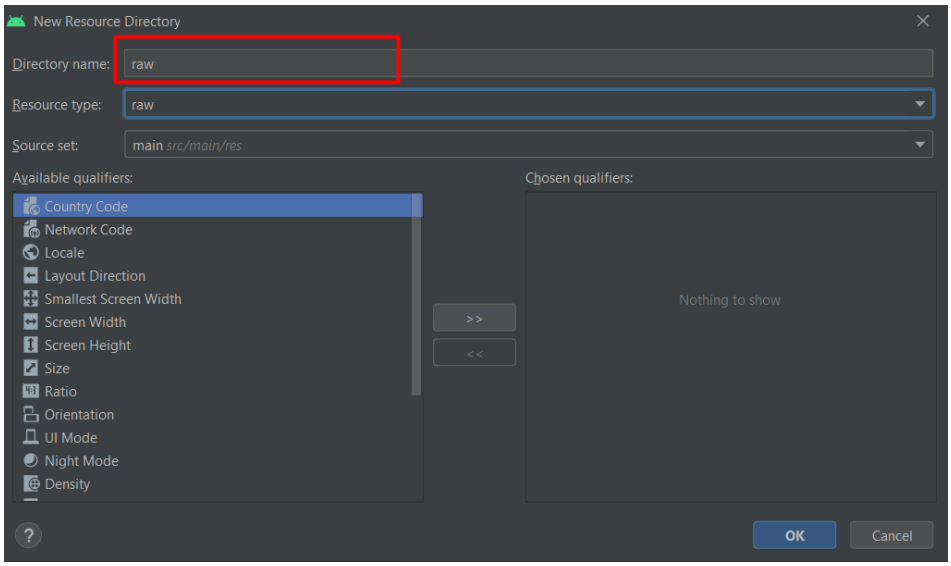
c. Anda akan mendapatkan *certificate* seperti dibawah ini

d. Buka *folder* res → new→ *Android Resource Directory*->raw



Gambar 10. Buka *Folder Android Resource Directory*

e. Pilih *resource type* berbentuk raw



Gambar 11. Pilih *resource type* xml

f. Tambahkan *file certificate* kedalam *folder* raw

Gambar 12. Pemasangan pin pada *buildConfig*

g. Ambil *KeyStore* dengan *file certificate* pada *resource* (sebagai *inputStream*)

```
1 val resourceStream = resources.openRawResource(R.raw.mycertificate)
2 val keyStoreType = KeyStore.getDefaultType()
3 val keyStore = KeyStore.getInstance(keyStoreType)
4
5 keyStore.load(resourceStream, null)
```

h. Dapatkan *TrustManagerFactory* dan mulai dengan *KeyStore*.

```
1
2 val trustManagerAlgorithm = TrustManagerFactory.getDefaultAlgorithm()
3 val trustManagerFactory = TrustManagerFactory.getInstance(trustManagerAlgorithm)

4 trustManagerFactory.init(keyStore)
```

i. Dapatkan instance SSLContext, ikat dengan TrustManager, dan buat sslContext dengan koneksi URL.

```
1 val sslContext = SSLContext.getInstance("TLS")
2 sslContext.init(null, trustManagerFactory.trustManagers, null)
3 val url = URL("http://www.example.com/")
4 val urlConnection = url.openConnection() as HttpsURLConnection
5 urlConnection.sslSocketFactory = sslContext.socketFactory
```

Referensi:

https://developer.android.com/training/articles/security-ssl?hl=id

https://www.netguru.com/blog/3-ways-how-to-implement-certificate-pinning-on-android