Racing for TLS Certificate Validation: A Hijacker's Guide to the Android TLS Galaxy

Sajjad Pourali†[1], Xiufen Yu†[1], Lianying Zhao[2], Mohammad Mannan[1], and Amr Youssef[1]

[1]Concordia University, [2]Carleton University, Canada

August 14, 2024

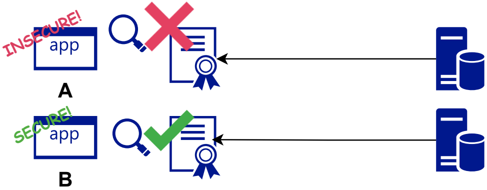
†euqal contribution

Outline

1. Motivation
2. Marvin
3. Validation “Hijacking”
4. Results
5. Reflections

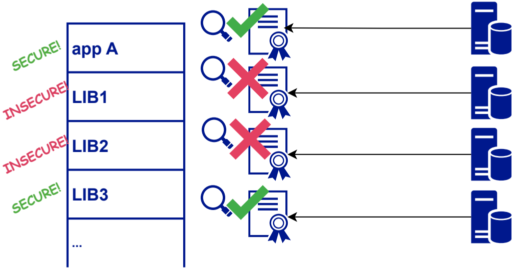
TLS Certificate Validation in Android Apps

- Apps communicate sensitive data, hence the need for TLS
- Security of each TLS connection is anchored in proper certificate validation
- The long-standing old problems:
  - A plethora of certificate validation problems have been identified
  - Findings/observations are attributed to (monolithic) apps



Apps Are No Longer Monolithic (not new)

- Most mobile apps contain code written/provided by multiple parties, aka. SDKs, e.g., Tencent Bugly, Google AdMob, Facebook Analytics and Bytedance SDK
- Fine-grained attribution is necessary for accurate remediation



cf. Android Privacy Sandbox → SDK runtime

Major Contributions

1. Marvin: a tool for **fine-grained attribution** of improper TLS certificate validation
2. Certificate validation **“hijacking”**: Surprisingly, who wrote code leading to insecure connections might not be the party to blame

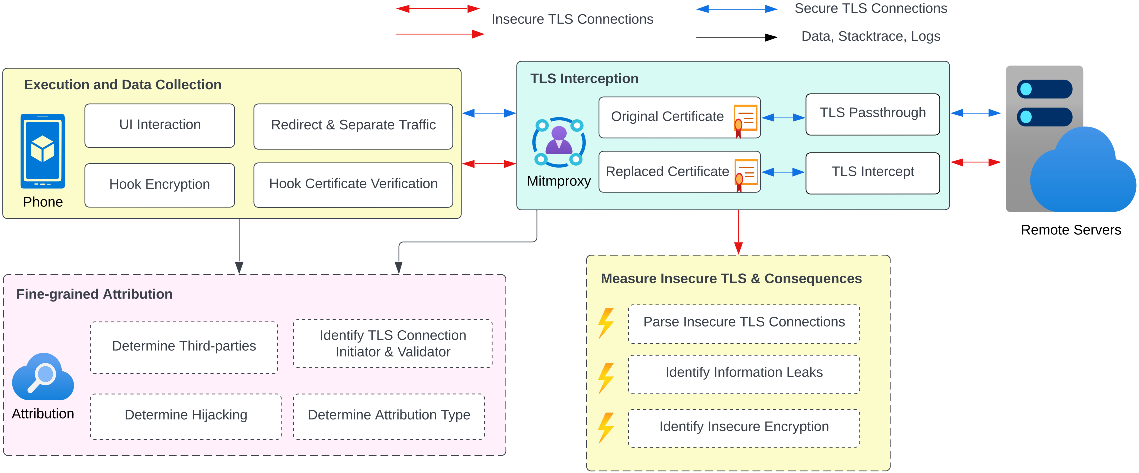
Outline

1. Motivation
2. Marvin
3. Validation “Hijacking”
4. Results
5. Reflections

Certificate Validation Problems Considered

- The 4 validation issues we consider:
  1. Unverified Certificate Signature
  2. Self-signed Certificate
  3. Expired Certificate
  4. Domain Mismatch
- Validation functions involved:
  - `javax.net.ssl.HostnameVerifier` → verify()
  - `javax.net.ssl.X509TrustManager`
    (from javax.net.ssl.SSLSocketFactory) → checkServerTrusted()

Marvin: Fine-grained Attribution Analysis



Insecure TLS Connections

Secure TLS Connections

Data, Stacktrace, Logs

Execution and Data Collection

Phone

UI Interaction

Redirect & Separate Traffic

Hook Encryption

Hook Certificate Verification

TLS Interception

Mitmproxy

Original Certificate

Replaced Certificate

TLS Passthrough

TLS Intercept

Remote Servers

Fine-grained Attribution

Attribution

Determine Third-parties

Identify TLS Connection Initiator & Validator

Determine Hijacking

Determine Attribution Type

Measure Insecure TLS & Consequences

Parse Insecure TLS Connections

Identify Information Leaks

Identify Insecure Encryption

We correlate local API stack traces with network traffic during certificate validation

Datasets: Google Play vs. Chinese App Stores

Rationale: Research has identified various distinctions between Google Play apps and Apps from Chinese stores, e.g., permissions, installation sources, policies/regulations

- Google Play apps
  - Based on APKPure ranking
    5,061 successfully analyzed in total
- Apps from Chinese stores
  - 360 Mobile Assistant — Qihoo 360 AppStore
    2,765 successfully analyzed in total
- Analyzed on two Pixel 7 and one Pixel 6 devices with Android 13
  12–24 minutes per app

Outline

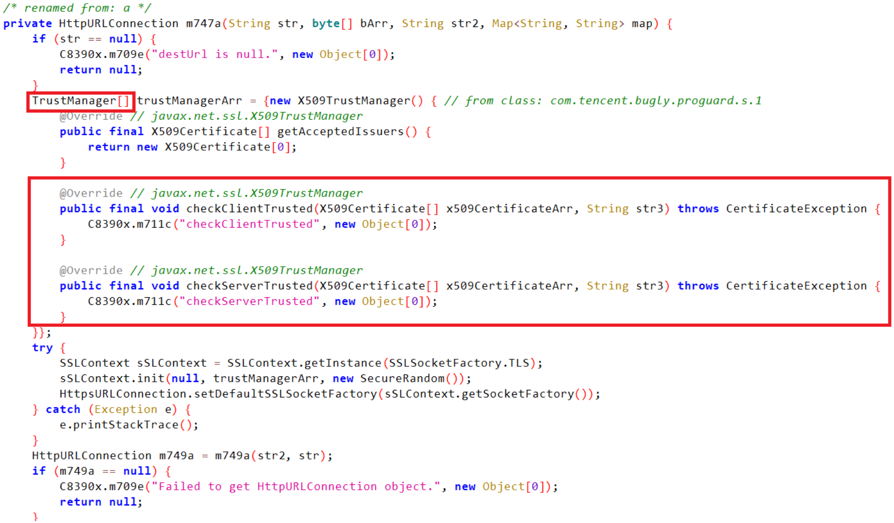
1. Motivation
2. Marvin
3. Validation “Hijacking”
4. Results
5. Reflections

What Enables the “Hijacking”?

- Despite the various HTTP client implementations in Android, most support methods to set global default values: `setDefaultSSLSocketFactory()` and `setDefaultHostnameVerifier()`
- Which then allows the new instance creator to override the validation functions (as interface methods)
- Any code within the app will be able to do it, affecting the rest of the app

Hijacking Is Bad as the Name Implies

The standard functions were either overridden with insecure implementations or just skipped



```
/* renamed from: a */
private HttpURLConnection m747a(String str, byte[] bArr, String str2, Map<String, String> map) {
    if (str == null) {
        C8390x.m709e("destUrl is null.", new Object[0]);
        return null;
    }
    TrustManager[] trustManagerArr = {new X509TrustManager() { // from class: com.tencent.bugly.proguard.s.1
        @Override // javax.net.ssl.X509TrustManager
        public final X509Certificate[] getAcceptedIssuers() {
            return new X509Certificate[0];
        }

        @Override // javax.net.ssl.X509TrustManager
        public final void checkClientTrusted(X509Certificate[] x509CertificateArr, String str3) throws CertificateException {
            C8390x.m711c("checkClientTrusted", new Object[0]);
        }

        @Override // javax.net.ssl.X509TrustManager
        public final void checkServerTrusted(X509Certificate[] x509CertificateArr, String str3) throws CertificateException {
            C8390x.m711c("checkServerTrusted", new Object[0]);
        }
    }};
    try {
        SSLContext sSLContext = SSLContext.getInstance(SSLSocketFactory.TLS);
        sSLContext.init(null, trustManagerArr, new SecureRandom());
        HttpsURLConnection.setDefaultSSLSocketFactory(sSLContext.getSocketFactory());
    } catch (Exception e) {
        e.printStackTrace();
    }
    HttpURLConnection m749a = m749a(str2, str);
    if (m749a == null) {
        C8390x.m709e("Failed to get HttpURLConnection object.", new Object[0]);
        return null;
    }
```

Outline

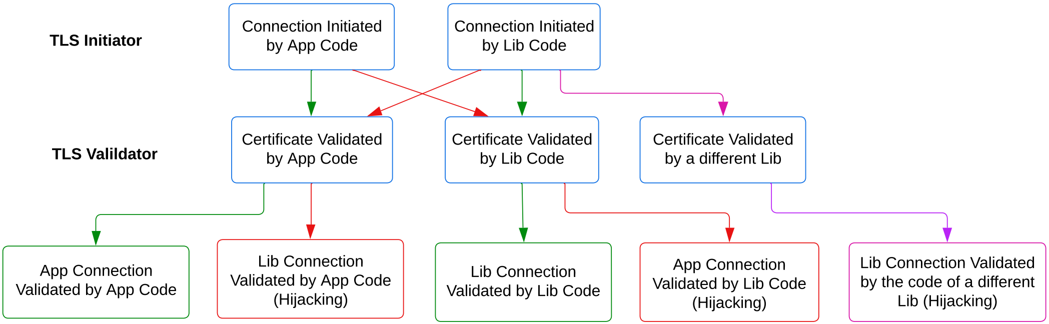
1. Motivation
2. Marvin
3. Validation “Hijacking”
4. Results
5. Reflections

Statistics

- 1851/7826 apps with at least one certificate validation issue, leading to insecure connections
  - 1529/2765 (55.3%) for Chinese apps
  - 322/5061 (6.4%) for Google Play apps
- 592/1851 apps with validation function override (hijacking) — 32% of the insecure ones
  - 524/1529 (34.3%) for Chinese apps
  - 68/322 (21.1%) for Google Play apps

Attribution Cases

Among the apps with insecure connections:



TLS Initiator

Connection Initiated by App Code

Connection Initiated by Lib Code

TLS Valildator

Certificate Validated by App Code

Certificate Validated by Lib Code

Certificate Validated by a different Lib

App Connection Validated by App Code

Lib Connection Validated by App Code (Hijacking)

Lib Connection Validated by Lib Code

App Connection Validated by Lib Code (Hijacking)

Lib Connection Validated by the code of a different Lib (Hijacking)

Google apps	99 (30.7%)	50 (15.5%)	28 (8.7%)	12 (3.7%)	23 (7.1%)
Chinese apps	361 (23.6%)	194 (12.7%)	747 (48.9%)	102 (6.7%)	360 (23.5%)

One Example for Each Case

App connection validated by app code

```
at com.datayes.common.net.interceptor.ssl.OkHttpSSLSocketFactory$1.checkServerTrusted(Native Method)
at com.android.org.conscrypt.Platform.checkServerTrusted(Platform.java:260)
at com.android.org.conscrypt.ConscryptEngine.verifyCertificateChain(ConscryptEngine.java:1638)
...
at com.datayes.common_cloud.net.interceptor.TokenInterceptor.intercept(TokenInterceptor.java:97)
```

Library connection validated by library code

```
at cn.jiguang.net.DefaultHostVerifier.verify(Native Method)
at com.android.okhttp.internal.io.RealConnection.connectTls(RealConnection.java:200)
at com.android.okhttp.internal.io.RealConnection.connectSocket(RealConnection.java:153)
...
at cn.jiguang.net.HttpUtils.a(Unknown Source:196)
at cn.jiguang.net.HttpUtils.httpPost(Unknown Source:1)
```

App connection validated by library code (hijacking)

```
at com.tencent.bugly.proguard.s.checkServerTrusted(Native Method)
at com.android.org.conscrypt.Platform.checkServerTrusted(Platform.java:260)
at com.android.org.conscrypt.ConscryptEngine.verifyCertificateChain(ConscryptEngine.java:1638)
...
at com.dnurse.main.ui.FlashActivity.downLoadImage(FlashActivity.java:11)(SourceFile:341)
at com.dnurse.main.ui.FlashActivity$a.doInBackground(FlashActivity.java:1)
```

Library connection validated by app code (hijacking)

```
at rich.y$a.verify(Native Method)
at com.android.okhttp.internal.io.RealConnection.connectTls(RealConnection.java:200)
at com.android.okhttp.internal.io.RealConnection.connectSocket(RealConnection.java:153)
...
at com.growingio.android.sdk.data.net.HttpService.performRequest(HttpService.java:132)
at com.growingio.android.sdk.data.net.HttpService.performRequest(HttpService.java:81)
```

One Example for Each Case (CONT’D)

Library connection validated by another library (hijacking)

```
at com.kuaishou.weapon.p0.q2$a.checkServerTrusted(Native Method)
at com.android.org.conscrypt.Platform.checkServerTrusted(Platform.java:260)
at com.android.org.conscrypt.ConscryptEngine.verifyCertificateChain(ConscryptEngine.java:1638)
...
at com.umeng.commonsdk.statistics.internal.c.a(Unknown Source:170)
at com.umeng.commonsdk.statistics.internal.c.a(Unknown Source:57)
```

Multiple hijacking actors (race condition)

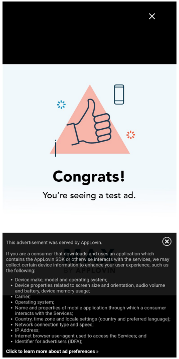
```
/* ———— (1) Baidu is hijacked by Bugly ———— */
at com.tencent.bugly.proguard.s$1.checkServerTrusted(Native Method)
at com.android.org.conscrypt.Platform.checkServerTrusted(Platform.java:260)
at com.android.org.conscrypt.ConscryptEngine.verifyCertificateChain(ConscryptEngine.java:1638)
...
at com.baidu.lbsapi.auth.g.a(Unknown Source:47)
at com.baidu.lbsapi.auth.g.a(Unknown Source:30)
/* ———— (2) Baidu is validated by Baidu again ———— */
at com.baidu.location.h.p.checkServerTrusted(Native Method)
at com.android.org.conscrypt.Platform.checkServerTrusted(Platform.java:260)
at com.android.org.conscrypt.ConscryptEngine.verifyCertificateChain(ConscryptEngine.java:1638)
...
at com.baidu.location.h.l.run(Unknown Source:171)
```

Information Leakage and Various Attacks

Among the apps with certificate validation issues:

- Apps from Chinese stores: 1358/1529 (88.8%) transmit sensitive information using insecure TLS connections
- Google play apps: the percentage is 278/322 (86.3%)

A PoC phishing attack on **a real app**:



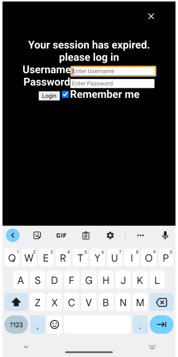
This advertisement was served by AppLovin.

If you are a consumer that downloads and uses an application which contains the AppLovin SDK or otherwise interacts with the services, we may collect certain device information to enhance your user experience, such as the following:

- Device make, model and operating system;
- Device properties related to screen size and orientation, audio volume and battery, device memory usage;
- Carrier;
- Operating system;
- Name and properties of mobile application through which a consumer interacts with the Services;
- Country, time zone and locale settings (country and preferred language);
- Network connection type and speed;
- IP Address;
- Internet browser user-agent used to access the Services; and
- Identifier for advertisers (IDFA);

Click to learn more about ad preferences »

Before



Your session has expired.
please log in
Username Enter Username
Password Enter Password
Login ☑ **Remember me**

After

Outline

1 Motivation

2 Marvin

3 Validation “Hijacking”

4 Results

5 Reflections

Observations

- When/whether the (updated) default values are retrieved, before each HTTPS call
- Most implementations (e.g., Apache HttpClient, Volley, and Square OkHttp) do not, except Google's fork of OkHttp
- Other HTTP clients are potentially as vulnerable as per our manual analysis

Mitigation?

- Is this working as designed?
  - Is such flexibility needed?
  - The threat model shift
- The possibility of introducing warnings/errors in Android Studio (as per Google)
  - To place the burden on the developer
- If Privacy Sandbox (SDK runtime) could be adopted and enforced

Additional Devices/OSes Tested

- Huawei Mate20 Pro, EMUI 10.1.0 (Android 10): Vulnerable
- LG G8 ThinQ, Android 12: Vulnerable
- Amazon Fire HD 8 (12th Gen), Fire OS 8.3.2.4 (Android 11): Vulnerable
- Samsung Galaxy A10e, Android 11: Vulnerable
- Honor Magic4 Pro, Android 14: Vulnerable
- Huawei P40, HarmonyOS 4.2.0: Vulnerable

Thank you!!

Lianying Zhao: lianying.zhao@carleton.ca

Sajjad Pourali: s_poural@cisse.concordia.ca **Xiufen Yu**: xiufen.yu@mail.concordia.ca

Paper highlights

Fine-grained attribution for TLS certificate validation issues

Certificate validation hijacking, leading to insecure connections

The tricky cause and implications of certificate validation hijacking

Marvin: `https://github.com/Madiba-Research/Marvin/`