

Magdeburger Journal zur Sicherheitsforschung

Gegründet 2011 | ISSN: 2192-4260
Herausgegeben von Stefan Schumacher und Jörg Sambleben
Erschienen im Magdeburger Institut für Sicherheitsforschung

This article appears in the special edition »In Depth Security – Proceedings of the DeepSec Conferences«.
Edited by Stefan Schumacher and René Pfeiffer

Java's SSLSocket: How Bad APIs Compromise Security

Georg Lukas

Internet security is hard. TLS is almost impossible. Implementing TLS correctly in Java is »Nightmare!«. This paper will show how a badly designed security API introduced over 15 years ago, combined with misleading documentation and developers unaware of security challenges, causes modern smartphone applications to be left exposed to Man-in-the-Middle attacks.

Citation: Lukas, G. (2015). Java's SSLSocket: How Bad APIs Compromise Security. *Magdeburger Journal zur Sicherheitsforschung*, *1*, 506–513. Retrieved March 20, 2015, from http://www.sicherheitsforschung-magdeburg.de/publikationen.html

Version 2015/03/20 21:17

1 Abstract

Internet security is hard. TLS[1] is almost impossible. Implementing TLS correctly in Java is *Nightmare!* While the higher-level `HttpsURLConnection`[2] and Apache's `DefaultHttpClient`[3] do it (mostly) right, direct users of Java SSL sockets (`SSLSocket`[4]/ `SSLEngine`[5], `SSLSocketFactory`[6]) are left exposed to Man-in-the-Middle attacks, unless the application manually checks the hostname against the certificate or employs certificate pinning.

The `SSLSocket`[7] documentation claims that the socket provides »Integrity Protection«, »Authentication«, and »Confidentiality«, even against active wiretappers. That impression is underscored by rigorous certificate checking performed when connecting, making it ridiculously hard to run development/test installations. However, these checks turn out to be completely worthless against active MitM attackers, because `SSLSocket` will happily accept *any* valid certificate (like for a domain owned by the attacker). Due to this, many applications using `SSLSocket` can be attacked with little effort.

This problem has[8] been[9] written[10] about[11], but CVE-2014-5075[12] shows that it can not be stressed enough.

1.1 Affected Applications

This problem affects applications that make use of SSL/TLS, but not HTTPS. The best candidates to look for it are therefore clients for application-level protocols like e-mail (POP3/IMAP), instant messaging (XMPP), file transfer (FTP). CVE-2014-5075[13] is the respective vulnerability of the Smack XMPP client library, so this is a good starting point.

1.1.1 XMPP Clients

XMPP clients based on Smack (which was fixed on 2014-07-22[14]):

- ChatSecure[15] (fixed[16] in 13.2.0-beta1)
- GTalkSMS[17] (contacted on 2014-07-28)
- MAXS[18] (tracker issue[19], fixed in 0.0.1.18[20])
- yaxim[21] and Bruno[22] (fixed in 0.8.8[23])
- *undisclosed Android application* (contacted on 2014-07-21)

Other XMPP clients:

- babbler[24] (another XMPP library; fixed on 2014-07-27[25])
- Conversations[26] (Android client, custom XMPP implementation, fixed in version 0.5[27])
- Sawim[28] (Android client, contacted on 2014-07-22)
- Stroke[29] (another XMPP client library, fixed[30] in git)
- Tigase[31] (contacted on 2014-07-27)

1.1.2 Not Vulnerable Applications

The following applications have been checked as well, and contained code to compensate for `SSLSocket`s shortcomings:

- Jitsi[32] (OSS conferencing client)
- K9-Mail[33] (Android e-Mail client)
- Xabber[34] (Based on Smack, but using its own hostname verification)

1 http://tools.ietf.org/html/rfc5246 r. 2014-03-11

2 http://docs.oracle.com/javase/8/docs/api/javax/net/ssl/HttpsURLConnection.html r. 2014-03-11

3 http://hc.apache.org/httpcomponents-client-ga/httpclient/apidocs/org/apache/http/impl/client/DefaultHttpClient.html r. 2014-03-11

4 http://docs.oracle.com/javase/8/docs/api/javax/net/ssl/SSLSocket.html r. 2014-03-11

5 http://docs.oracle.com/javase/8/docs/api/javax/net/ssl/SSLEngine.html r. 2014-03-11

6 http://docs.oracle.com/javase/8/docs/api/javax/net/ssl/SSLSocketFactory.html#createSocket-java.net.Socket-java.lang.String-int-boolean- r. 2014-03-11

7 http://docs.oracle.com/javase/8/docs/api/javax/net/ssl/SSLSocket.html r. 2014-03-11

8 http://kevinlocke.name/bits/2012/10/03/ssl-certificate-verification-in-dispatch-and-asynchttpclient/ r. 2014-03-11

9 https://developer.android.com/training/articles/security-ssl.html#WarningsSslSocket r. 2014-03-11

10 http://tersesystems.com/2014/03/23/fixing-hostname-verification/ r. 2014-03-11

11 http://stackoverflow.com/a/18174689/539443 r. 2014-03-11

12 http://op-co.de/CVE-2014-5075.html r. 2014-03-11

13 http://op-co.de/CVE-2014-5075.html r. 2014-03-11

14 https://github.com/igniterealtime/Smack/commit/d35fd16a21e2aa942a0a815762f52bf473cd5eff r. 2014-03-11

15 https://guardianproject.info/apps/chatsecure r. 2014-03-11

16 https://github.com/guardianproject/ChatSecureAndroid/commit/3f150daded7461255b9d51bfc59ff91f8a77ed81 r. 2014-03-11

17 http://code.google.com/p/gtalksms/ r. 2014-03-11

18 http://projectmaxs.org/homepage/ r. 2014-03-11

19 https://projectmaxs.atlassian.net/browse/MAXS-25 r. 2014-03-11

20 https://social.geekplace.eu/notice/2139 r. 2014-03-11

21 https://yaxim.org/ r. 2014-03-11

22 http://yaxim.org/bruno r. 2014-03-11

23 https://yaxim.org/blog/2014/08/04/yaxim-0-dot-8-8-important-security-update/ r. 2014-03-11

24 http://babbler-xmpp.blogspot.de/ r. 2014-03-11

25 https://bitbucket.org/sco0ter/babbler/commits/359b1275c6bec8f0502320e57a2489dba9b177ab r. 2014-03-11

26 https://github.com/siacs/Conversations r. 2014-03-11

27 https://github.com/siacs/Conversations/commit/4607e2c546fec78d7ae0ca8ce779a2267e6edbe2 r. 2014-03-11

28 http://sawim.ru/ r. 2014-03-11

29 http://swift.im/stroke/ r. 2014-03-11

30 http://swift.im/git/stroke/commit/?id=77959428b7f4150569dda9fac35becf7e10b96c7 r. 2014-03-11

31 http://www.tigase.org/ r. 2014-03-11

32 https://jitsi.org/ r. 2014-03-11

33 http://code.google.com/p/k9mail/ r. 2014-03-11

34 https://github.com/redsolution/xabber-android r. 2014-03-11

1.2 Background: Security APIs in Java

The amount of vulnerable applications can be easily explained after a deep dive into the security APIs provided by Java (and its offsprings). Therefore, this section will handle the dirty details of trust (mis)management in the most important implementations: old Java, new Java, Android and in Apache's HttpClient.

1.2.1 Java SE up to and including 1.6

When network security was added into Java 1.4 with the JSSE[35] (and we all know how well security-as-an-afterthought works), two distinct APIs have been created for certificate verification[36] and for hostname verification[37]. The rationale for that decision was probably that the TLS/SSL handshake happens at the socket layer, whereas the hostname verification depends on the application-level protocol (HTTPS[38] at that time). Therefore, the `X509TrustManager`[39] class for certificate trust checks was integrated into the low-level `SSLSocket` and `SSLEngine` classes, whereas the `HostnameVerifier`[40] API was only incorporated into the `HttpsURLConnection`[41].

The API design was not very future-proof either: `X509TrustManager`'s `checkClientTrusted()`[42] and `checkServerTrusted()`[43] methods are only passed the certificate and authentication type parameters. There is no reference to the actual SSL connection or its peer name. The only workaround to allow hostname verification through this API is by creating a custom `TrustManager` for each connection, and storing the peer's hostname in it. This is neither elegant nor does it scale well with multiple connections.

The `HostnameVerifier` on the other hand has access to both the hostname and the session, making a full verification possible. However, only `HttpsURLConnection` is making use of a `HostnameVerifier` (and is only asking it if it determines a mismatch between the peer and its certificate, so the default `HostnameVerifier` always fails).

Besides of the default `HostnameVerifier` being unusable due to always failing, the API has another subtle surprise: while the `TrustManager` methods fail by throwing a `CertificateException`[44], `HostnameVerifier.verify()` simply returns `false` if verification fails.

As the API designers realized that users of the raw `SSLSocket` might fall into a certificate verification trap set up by their API, they added a well-buried warning into the JSSE reference guide for Java 5[45]:

> **IMPORTANT NOTE:** When using raw `SSLSockets/SSLEngines` you should always check the peer's credentials before sending any data. The `SSLSocket/SSLEngine` classes do not automatically verify, for example, that the hostname in a URL matches the hostname in the peer's credentials. An application could be exploited with URL spoofing if the hostname is not verified.

Of course, URLs are only a thing in HTTPS, but the point remains... well hidden. The `SSLSocket`[46] reference article on the other hand does not contain any warnings, it implies that the application developer is doing the right thing.

And even if the hidden warning reaches the developer, there is no hint about *how* to implement the peer credentials checks. There is no API class that would perform this tedious and error-prone task, and implementing it correctly requires a Ph.D. degree in rocket surgery, as well as deep knowledge of some[47] related[48] Internet[49] standards[50].

1.2.2 Apache HttpClient

The Apache HttpClient library[51] is a full-featured HTTP client written in pure Java, adding flexibility and functionality in comparison to the default HTTP implementation.

The Apache library developers came up with their own API interface for hostname verification, `X509HostnameVerifier`[52], that also happens to incorporate Java's `HostnameVerifier` interface. The new methods added by Apache are expected to throw

35 http://en.wikipedia.org/wiki/Java_Secure_Socket_Extension r. 2014-03-11

36 http://docs.oracle.com/javase/8/docs/api/javax/net/ssl/X509TrustManager.html r. 2014-03-11

37 http://docs.oracle.com/javase/8/docs/api/javax/net/ssl/HostnameVerifier.html r. 2014-03-11

38 http://tools.ietf.org/html/rfc2818 r. 2014-03-11

39 http://docs.oracle.com/javase/8/docs/api/javax/net/ssl/X509TrustManager.html r. 2014-03-11

40 http://docs.oracle.com/javase/8/docs/api/javax/net/ssl/HostnameVerifier.html r. 2014-03-11

41 http://docs.oracle.com/javase/8/docs/api/javax/net/ssl/HttpsURLConnection.html r. 2014-03-11

42 http://docs.oracle.com/javase/8/docs/api/javax/net/ssl/X509TrustManager.html#checkClientTrusted-java.security.cert.X509Certificate:A-java.lang.String- r. 2014-03-11

43 http://docs.oracle.com/javase/8/docs/api/javax/net/ssl/X509TrustManager.html#checkServerTrusted-java.security.cert.X509Certificate:A-java.lang.String- r. 2014-03-11

44 http://docs.oracle.com/javase/8/docs/api/java/security/cert/CertificateException.html r. 2014-03-11

45 http://docs.oracle.com/javase/1.5.0/docs/guide/security/jsse/JSSERefGuide.html#ClassRelationship r. 2014-03-11

46 http://docs.oracle.com/javase/8/docs/api/javax/net/ssl/SSLSocket.html r. 2014-03-11

47 http://tools.ietf.org/html/rfc2459 r. 2014-03-11

48 http://tools.ietf.org/html/rfc2818 r. 2014-03-11

49 http://tools.ietf.org/html/rfc5246 r. 2014-03-11

50 http://tools.ietf.org/html/rfc6125 r. 2014-03-11

51 http://hc.apache.org/httpcomponents-client-ga/ r. 2014-03-11

52 http://hc.apache.org/httpcomponents-client-ga/httpclient/apidocs/org/apache/http/conn/ssl/X509HostnameVerifier.html r. 2014-03-11

`SSLException` when verification fails, while the old method still returns `true` or `false`, of course. It is hard to tell if this interface mixing is adding confusion, or reducing it. One way or the other, it results in the appropriate glue code, see Fig. 1.

Based on that interface, `AllowAllHostnameVerifier`[53], `BrowserCompatHostnameVerifier`[54], and `StrictHostnameVerifier`[55] were created, which can actually be plugged into anything expecting a plain `HostnameVerifier`. The latter two also actually perform hostname verification, as opposed to the default verifier in Java, so they can be used wherever appropriate. Their difference is:

> The only difference between BROWSER_COMPATIBLE and STRICT is that a wildcard (such as »*.foo.com«) with BROWSER_COMPATIBLE matches all subdomains, including "a.b.foo.com".

If you can make use of Apache's HttpClient library, just plug in one of these verifiers as follows to ensure hostname verification:

```
1  sslSocket = ...;
2  sslSocket.startHandshake();
3  HostnameVerifier verifier = new
4      StrictHostnameVerifier();
5  if (!verifier.verify(serviceName,
6      sslSocket.getSession())) {
7          throw new CertificateException
8     ("Server failed to authenticate as"
9                          + serviceName);
10 }
11 // NOW you can send and receive data!
```

1.2.3 Android

Android's designers must have been well aware of the shortcomings of the Java implementation, and the problems that an application developer might encounter when testing and debugging. They created the `SSLCertificateSocketFactory`[56] class, which makes a developer's life really easy:

1. It is available on all Android devices, starting with API level 1.
2. It comes with appropriate warnings about its security parameters and limitations:

   > **Most `SSLSocketFactory` implementations do not verify the server's identity, allowing man-in-the-middle attacks.** This implementation does check the server's certificate hostname, but only for createSocket variants that specify a hostname. When using methods that use `InetAddress` or which return an unconnected socket, you MUST verify the server's identity yourself to ensure a secure connection.

3. It provides developers with two easy ways to disable all security checks for testing purposes: a) a static `getInsecure()` method (as of API level 8), and b)

   > On development devices, `setprop socket.relaxsslcheck yes` bypasses all SSL certificate and hostname checks for testing purposes. This setting requires root access.

4. Uses of the insecure instance are logged via adb:

   `Bypassing SSL security checks at caller's request`

   Or, when the system property is set:

   `*** BYPASSING SSL SECURITY CHECKS (socket.relaxsslcheck=yes) ***`

Some time in 2013, a training article[57] about Security with HTTPS and SSL was added, which also features its own section for »Warnings About Using SSLSocket Directly«, once again explicitly warning the developer:

> **Caution: SSLSocket does not** perform hostname verification. It is up the your app to do its own hostname verification, preferably by calling `getDefaultHostnameVerifier()` with the expected hostname. Further beware that `HostnameVerifier.verify()` doesn't throw an exception on error but instead returns a boolean result that you must explicitly check.

Typos aside, this is very true advice. The article also covers other common SSL/TLS related problems like certificate chaining, self-signed certs and SNI, making it a must read. The fact that it does not mention the `SSLCertificateSocketFactory` is only a little snag.

1.2.4 Java 1.7+

As of Java 1.7, there is a new abstract class `X509ExtendedTrustManager`[58] that finally unifies the two sides of certificate verification:

> Extensions to the X509TrustManager interface to support SSL/TLS connection sensitive trust management.
>
> To prevent man-in-the-middle attacks, hostname checks can be done to verify that the hostname in an end-entity certificate

53 http://hc.apache.org/httpcomponents-client-ga/httpclient/apidocs/org/apache/http/conn/ssl/AllowAllHostnameVerifier.html r. 2014-03-11

54 http://hc.apache.org/httpcomponents-client-ga/httpclient/apidocs/org/apache/http/conn/ssl/BrowserCompatHostnameVerifier.html r. 2014-03-11

55 http://hc.apache.org/httpcomponents-client-ga/httpclient/apidocs/org/apache/http/conn/ssl/StrictHostnameVerifier.html r. 2014-03-11

56 http://developer.android.com/reference/android/net/SSLCertificateSocketFactory.html r. 2014-03-11

57 https://developer.android.com/training/articles/security-ssl.html r. 2014-03-11

58 http://docs.oracle.com/javase/8/docs/api/javax/net/ssl/X509ExtendedTrustManager.html r. 2014-03-11

```
public final boolean verify(String host, SSLSession session) {
        try {
                Certificate[] certs = session.getPeerCertificates();
                X509Certificate x509 = (X509Certificate) certs[0];
                verify(host, x509);
                return true;
        } catch(SSLException e) { return false; }
}
```

Figure 1: Apache HttpClient X509HostnameVerifier Internal Code

> matches the targeted hostname. TLS does not require such checks, but some protocols over TLS (such as HTTPS) do. In earlier versions of the JDK, the certificate chain checks were done at the SSL/TLS layer, and the hostname verification checks were done at the layer over TLS. This class allows for the checking to be done during a single call to this class.

This class extends the `checkServerTrusted` and `checkClientTrusted` methods with an additional parameter for the socket reference, allowing the TrustManager to obtain the hostname that was used for the connection, thus making it possible to actually verify that hostname.

To retrofit this into the old `X509TrustManager` interface, all instances of `X509TrustManager` are internally wrapped into an `AbstractTrustManagerWrapper` that performs hostname verification according to the socket's `SSLParameters`[59]. All this happens transparently, all you need to do is to initialize your socket with the hostname and then set the right parameters:

```
SSLParameters p = sslSocket
.getSSLParameters();
p.setEndpointIdentificationAlgorithm
("HTTPS");
sslSocket.setSSLParameters(p);
```

If you do not set the endpoint identification algorithm, the socket will behave in the same way as in earlier versions of Java, accepting *any* valid certificate from *any* server name.

However, if you *do* run the above code, the certificate will be checked against the IP address or hostname that you are connecting to. If the service you are using employs DNS SRV[60], the hostname (the actual machine you are connecting to, e.g. `xmpp-042.example.com`) might differ from the service name (what the user entered, like `example.com`). However, the certificate will be issued for the *service* name, so the verification will fail. As such protocols are most often combined with `STARTTLS`, you will need to wrap your `SSLSocket` around your plain `Socket`, for which you can use the following code:

```
sslSocket=sslContext.getSocketFactory()
.createSocket(plainSocket, serviceName,
plainSocket.getPort(),
true);
```

1.2.5 API Confusion Conclusion

To summarize the different »platforms«:

- On Java 1.6 or earlier, no hostname verification mechanisms are available.
- On Android, use `SSLCertificateSocketFactory`[61] and be happy.
- If you have Apache HttpClient, add a `StrictHostnameVerifier.verify()`[62] call right after you connect your socket, **and check its return value!**
- On Java 1.7 or newer, do not forget to set the right `SSLParameters`[63], so the runtime takes care of hostname verification.

1.3 Java SSL In the Literature

There is a large amount of *good* and *bad* advice out there, you just need to be a security expert to separate the wheat from the chaff.

1.3.1 Negative Examples

The most expensive advice is free advice. And the Internet is full of it. First, there is code to let Java trust all certificates[64], because self-signed certificates are a subset of all certificates, obviously. Then, there is a software engineer deliberately disable certificate validation[65], because all these security exceptions only get into our way. Even after the Snowden revelations,

59 http://docs.oracle.com/javase/8/docs/api/javax/net/ssl/SSLParameters.html r. 2014-03-11

60 http://en.wikipedia.org/wiki/SRV_record r. 2014-03-11

61 http://developer.android.com/reference/android/net/SSLCertificateSocketFactory.html r. 2014-03-11

62 http://hc.apache.org/httpcomponents-client-ga/httpclient/apidocs/org/apache/http/conn/ssl/StrictHostnameVerifier.html r. 2014-03-11

63 http://docs.oracle.com/javase/8/docs/api/javax/net/ssl/SSLParameters.html r. 2014-03-11

64 http://runtime32.blogspot.de/2008/11/let-java-ssl-trust-all-certificates.html r. 2014-03-11

65 http://www.nakov.com/blog/2009/07/16/disable-certificate-validation-in-java-ssl-connections/ r. 2014-03-11

recipes for disabling SSL certificate validation[66] are still written. The suggestions are all very similar, and all pretty bad.

Admittedly, an encrypted but unvalidated connection is still a little bit better than a plaintext connection. However, with the advent of free WiFi networks and SSL MitM software, everybody with a little energy can invade your »secure« connections, which you use to transmit really sensitive information. The effect of this can reach from funny over embarassing and up to life-threatening, if you are a journalist in a crisis zone.

The personal favorite of the author is this StackOverflow question[67] about avoiding the certificate warning message in yaxim[68], which is caused by MemorizingTrustManager[69].

Fortunately, the situation on StackOverflow has been improving over the years. Some time ago, readers were overwhelmed with `DO_NOT_VERIFY`[70] `HostnameVerifier`s and all-accepting `DefaultTrustManager`s[71], where the authors conveniently forgot to mention that their code turns the big red »security« switch to OFF.

The better answers on StackOverflow at least come with a warning[72] or even suggest certificate pinning[73].

1.3.2 Positive Examples

In 2012, Kevin Locke has created[74] a proper HostnameVerifier using the internal sun.security.util.HostnameChecker[75] class which seems to exist in Java SE 6 and 7. This `HostnameVerifier` is used with `AsyncHttpClient`, but is suitable for other use-cases as well.

Fahl et al.[76] have analyzed[77] the sad state of SSL in Android apps in 2012. Their focus was on HTTPS, where they did find a massive amount of applications deliberately misconfigured to accept invalid or mismatching certificates (probably added during app development). In a 2013 followup[78], they have developed a mechanism to enable certificate checking and pinning according to special flags in the application manifest.

Will Sargent from Terse Systems has an[79] excellent[80] series[81] of[82] articles[83] on everything TLS, with videos, examples and plentiful background information, which is strongly recommended to watch.

There is even an excellent StackOverflow answer by Bruno[84], outlining the proper hostname validation options with Java 7, Android and »other« Java platforms, in a very concise way.

1.4 Mitigation Possibilities

So you are an app developer, and you get this pesky `CertificateException` you could not care less about. What can you do to get rid of it, in a secure way? That depends on your situation.

1.4.1 Cloud-Connected App: Certificate Pinning

If your app is always connecting to known-in-advance servers under you control (like only your company's »cloud«), employ Certificate Pinning[85].

If you want a cheap and secure solution, create your own Certificate Authority (CA)[86] (**and guard its keys!**), deploy its certificate as the only trusted CA in the app, and sign[87] all your server keys with it. This approach provides you with the ultimate control over the whole security infrastructure, you do not need to pay certificate extortion fees to greedy CAs, and a compromised CA can not issue certificates that would allow to MitM your app. The only drawback is that you might not be as good as a commercial CA at guarding your CA keys, and these are the keys to

66 http://mariuszprzydatek.com/2013/07/19/disabling-ssl-certificate-validation/ r. 2014-03-11

67 http://stackoverflow.com/questions/20544193/avoid-accept-unknown-certificate-warning-in-android-while-using-xmpp r. 2014-03-11

68 https://yaxim.org r. 2014-03-11

69 https://github.com/ge0rg/MemorizingTrustManager/ r. 2014-03-11

70 http://stackoverflow.com/questions/995514/https-connection-android/1000205#1000205 r. 2014-03-11

71 http://stackoverflow.com/questions/1828775/how-to-handle-invalid-ssl-certificates-with-apache-httpclient/1828840#1828840 r. 2014-03-11

72 http://stackoverflow.com/questions/2642777/trusting-all-certificates-using-httpclient-over-https/4837230#4837230 r. 2014-03-11

73 http://stackoverflow.com/questions/2893819/telling-java-to-accept-self-signed-ssl-certificate/2893932#2893932 r. 2014-03-11

74 http://kevinlocke.name/bits/2012/10/03/ssl-certificate-verification-in-dispatch-and-asynchttpclient/ r. 2014-03-11

75 http://www.docjar.com/docs/api/sun/security/util/HostnameChecker.html r. 2014-03-11

76 http://android-ssl.org/ r. 2014-03-11

77 http://android-ssl.org/files/p50-fahl.pdf r. 2014-03-11

78 http://android-ssl.org/files/p49.pdf r. 2014-03-11

79 http://tersesystems.com/2014/01/13/fixing-the-most-dangerous-code-in-the-world/ r. 2014-03-11

80 http://tersesystems.com/2014/03/20/fixing-x509-certificates/ r. 2014-03-11

81 http://tersesystems.com/2014/03/22/fixing-certificate-revocation/ r. 2014-03-11

82 http://tersesystems.com/2014/03/23/fixing-hostname-verification/ r. 2014-03-11

83 http://tersesystems.com/2014/03/31/testing-hostname-verification/ r. 2014-03-11

84 http://stackoverflow.com/a/18174689/539443 r. 2014-03-11

85 https://www.owasp.org/index.php/Certificate_and_Public_Key_Pinning r. 2014-03-11

86 https://jamielinux.com/articles/2013/08/act-as-your-own-certificate-authority/ r. 2014-03-11

87 https://jamielinux.com/articles/2013/08/create-and-sign-ssl-certificates-certificate-authority/ r. 2014-03-11

your kingdom.

To implement the client side, you need to store the CA cert in a key file, which you can use to create an `X509TrustManager` that will only accept server certificates signed by your CA (Fig. 2).

If you rather prefer to trust the establishment (or if your servers are to be used by web browsers as well), you need to get all your server keys signed by an »official« Root CA. However, you can still store that single CA into your key file and use the above code. You just won't be able to switch to a different CA later on if they try to extort more money from you.

1.4.2 User-configurable Servers (a.k.a. »Private Cloud«): TOFU/POP

In the context of TLS, TOFU/POP is neither vegetarian music nor frozen food, but stands for »*Trust on First Use / Persistence of Pseudonymity*«.

The idea behind TOFU/POP is that when you connect to a server for the first time, your client stores its certificate, and checks it on each subsequent connection. This is the same mechanism as used in SSH. If you had no evildoers between you and the server the first time, later MitM attempts will be discovered. OpenSSH displays Fig. 3 on a key change.

In case you fell victim to a MitM attack the first time you connected, you will see the nasty warning as soon as the attacker goes away, and can start investigating. Your information will be compromised, but at least you will know it.

The problem with the TOFU approach is that it does not mix well with the PKI[88] infrastructure model used in the TLS world: with TOFU, you create *one* key when the server is configured for the first time, and that key remains bound to the server *forever* (there is no concept of key revocation).

With PKI, you create a key and request a certificate, which is typically valid for one or two years. Before that certificate expires, you *must* request a new certificate (optionally using a new private key), and replace the expiring certificate on the server with the new one.

If you let an application »pin« the TLS certificate on first use, you are in for a surprise within the next year or two. If you »pin« the server public key, you must be aware that you will have to stick to that key (and renew certificates for it) forever. Of course you can create your own, self-signed, certificate with a ridiculously long expiration time, but this practice is frowned upon (for self-signing *and* long expiration times).

Currently, some ideas[89] exist about how to combine PKI with TOFU, but the only sensible thing that an app can do is to give a shrug and ask the user.

Because asking the user is non-trivial from a background networking thread, the author has developed MemorizingTrustManager[90] (MTM) for Android. MTM is a library that can be plugged into your apps' TLS connections, that leverages the system's ability for certificate and hostname verification, and asks the user if the system does not consider a given certificate/hostname combination as legitimate. Internally, MTM is using a key store where it collects all the certificates that the user has permanently accepted.

1.4.3 Browser

If you are developing a browser that is meant to support HTTPS, please stop here, get a security expert into your team, and only go on with her. This article has shown that using T LS is horribly hard even if you can leverage existing components to perform the actual verification of certificates and hostnames. Writing such checks in a browser-compliant way is far beyond the scope of this piece.

1.5 Outlook

1.5.1 DNS + TLS = DANE

Besides of TOFU/POP, which is not yet ready for TLS primetime, there is an alternative approach to link the server name (in DNS) with the server identity (as represented by its TLS certificate): DNS-based Authentication of Named Entities (DANE)[91].

With this approach, information about the server's TLS certificate can be added to the DNS database, in the form of different certificate constraint records:

1. a *CA constraint* can require that the presented server certificate MUST be signed by the referenced CA public key, and that this CA must be a known Root CA.
2. a *service certificate constraint* can define that the server MUST present the referenced certificate, and that certificate must be signed by a known Root CA.
3. a *trust anchor assertion* is like a CA constraint, except it does not need to be a Root CA known to the client. This allows a server administrator to run their own CA.
4. a *domain issued certificate* is analogous to a service certificate constraint, but like in (2), there is no need to involve a Root CA.

Multiple constraints can be specified to tighten the checks, encoded in TLSA records (for TLS association). TLSA records are always specific to a given server name and port. For example, to make a secure XMPP connection with `zombofant.net`, first

88 http://en.wikipedia.org/wiki/Public_key_infrastructure r. 2014-03-11

89 https://dev.guardianproject.info/projects/bazaar/wiki/Chained_TLS_Cert_Verification r. 2014-03-11

90 https://github.com/ge0rg/MemorizingTrustManager/ r. 2014-03-11

91 http://en.wikipedia.org/wiki/DNS-based_Authentication_of_Named_Entities r. 2014-03-11

```
KeyStore ks = KeyStore.getInstance(KeyStore.getDefaultType());
ks.load(new FileInputStream(keyStoreFile), "keyStorePassword".toCharArray());
TrustManagerFactory tmf = TrustManagerFactory.getInstance("X509");
tmf.init(ks);
SSLContext sc = SSLContext.getInstance("TLS");
sc.init(null, tmf.getTrustManagers(), new java.security.SecureRandom());
// use 'sc' for your HttpsURLConnection / SSLSocketFactory / ...
```

Figure 2: Example for An Explicit List of Root CAs

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
```

Figure 3: OpenSSH warning message

the XMPP SRV record (`_xmpp-client._tcp`) needs to be obtained:

```
$ host -t SRV \
  _xmpp-client._tcp.zombofant.net
_xmpp-client._tcp.zombofant.net has  \
 SRV record 0 0 5222 xmpp.zombofant.net.
```

Then, the TLSA record(s) for `xmpp.zombofant.net:5222` must be obtained:

```
$ host -t TLSA \
 _5222._tcp.xmpp.zombofant.net
_5222._tcp.xmpp.zombofant.net \
has TLSA  record 3 0 1  \
75E6A12CFE74A2230F3578D5E98C6F251AE2 \
043EDEBA09F9D952A4C1 C317D81D
```

This record reads as: the server is using a domain issued certificate (3) with the full certificate (0) represented via its SHA-256 hash (1):
`75:E6:A1:2C:FE:74:A2:23:`
`0F:35:78:D5:E9:8C:6F:25:`
`1A:E2:04:3E:DE:BA:09:F9:`
`D9:52:A4:C1:C3:17:D8:1D`.

And indeed, if we check the server certificate using `openssl s_client`, the SHA-256 hash does match:

```
Subject: CN=zombofant.net
Issuer: O=Root CA,\
 OU=http://www.cacert.org, \
 CN=CA Cert Signing Authority/ \
emailAddress=support@cacert.org
Validity
 Not Before: Apr  8 07:25:35 2014 GMT
 Not After : Oct  5 07:25:35 2014 GMT
SHA256 Fingerprint=75:E6:A1:2C:FE:74:\
 A2:23:0F:35:78:D5:E9:8C:6F:25:1A:E2:\
 04:3E:DE:BA:09:F9:D9:52:A4:C1:C3:17:\
 D8:1D
```

Of course, this information can only be relied upon if the DNS records are secured by DNSSEC[92]. And DNSSEC can be abused by the same entities that already can manipulate Root CAs and perform large-scale Man-in-the-Middle attacks. However, this kind of attack is made significantly harder: while a typical Root CA list contains hundreds of entries, with an unknown number of intermediate CAs each, and it is sufficient to compromise any one of them to screw you, with DNSSEC, the attacker needs to obtain the keys to your domain (`zombofant.net`), to your top-level domain (`.net`) or the master root keys (`.`). In addition to that improvement, another benefit of DANE is that server operators can replace (paid) Root CA services with (cheaper/free) DNS records.

However, there is a long way until DANE can be used in Java. Java's own DNS code is very limited (no SRV support, TLSA - what are you dreaming of?) The dnsjava[93] library claims to provide *partial* DNSSEC verification, there is the unmaintained DNSSEC4j[94] and the GSoC work-in-progress dnssecjava[95]. All that remains is for somebody to step up and implement a DANETrustManager[96] based on one of these components.

1.6 Conclusion

Internet security is hard. Let's go bake some cookies!

2 About the Author

Georg Lukas completed a Ph.D. degree in Computer Science in 2012, focusing the research on wireless communication and security. Currently, he is working as an IT Security consultant at rt-solutions.de GmbH in Cologne. He is developing smart-phone applications and working on mobile payment solutions.

You can contact the author at lukas@rt-solutions.de.

92 http://en.wikipedia.org/wiki/Domain_Name_System_Security_Extensions r. 2014-03-11

93 http://www.dnsjava.org/ r. 2014-03-11

94 https://github.com/adamfisk/DNSSEC4J r. 2014-03-11

95 https://github.com/jitsi/dnssecjava r. 2014-03-11

96 http://stackoverflow.com/questions/23683398/how-to-use-dane-with-java r. 2014-03-11