rt-solutions.de

experts you can trust.

Java's SSLSocket: How Bad APIs Compromise Security

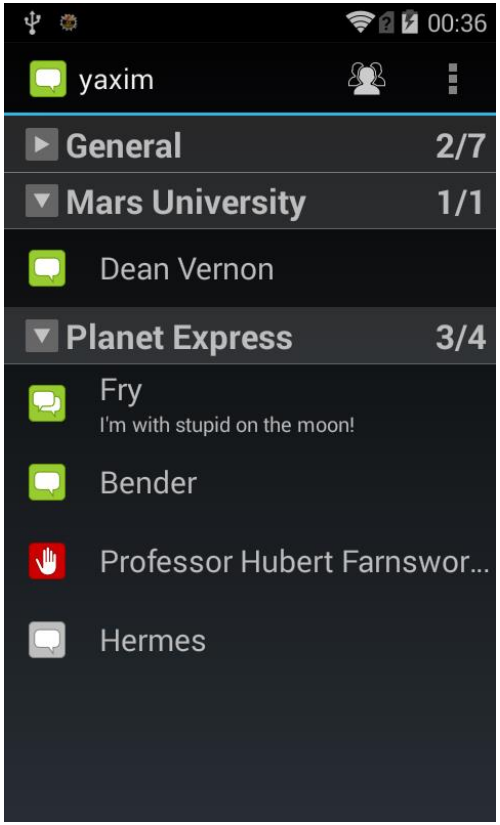
Tale of a Frustrated Android Developer

Dr. Georg Lukas <lukas@rt-solutions.de>

About the Speaker

rt-solutions.de

- IT Consultant at rt-solutions.de
  (ITSec, Smartphone Payment)
- Open Source developer
  (embedded Linux, Android)
- Maintainer of yaxim
  (yet another XMPP instant messenger)
- Operator of yax.im (Public XMPP service)
- Mobile / Wireless / Security geek

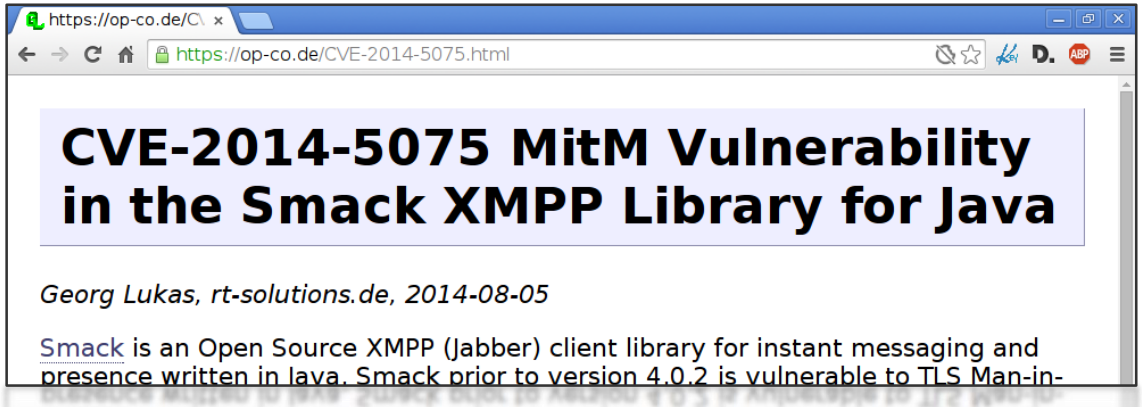


Motivation – Why Am I Here Today

rt-solutions.de

- Development of yaxim – Open-Source XMPP app
  - XMPP uses TLS for securing sessions (logins, chat content)
  - yaxim uses Smack for XMPP + MemorizingTrustManager for TLS
  - Added hostname checking to MTM

  - ➢ no place in Smack to add?!?



Agenda

- A brief history of SSL/TLS
- Java TLS APIs: All-or-nothing security
- Making your (Android) application more secure
- TLS in the Post-Snowden Era

A Brief History of SSL/TLS

rt-solutions.de

- Early 1990ies: Wild West Internet
  - Everybody uses telnet, ftp, nfs, …
- 1995: Netscape releases SSL 2.0 (Secure Sockets Layer)
- 1996: SSL 3.0 (redesign due to security flaws)
- 1999: TLS (Transport Layer Security) RFC based on SSLv3
- 1999, 2000: HTTP, IMAP, … over TLS, w/ hostname checks
- 2001: Sun creates JSSE library with JDK 1.4

  …
- 2006: TLS 1.1 fixes padding and CBC attack (BEAST, 2011)
- 2008: TLS 1.2 fixes timing oracle (Lucky13, 2013)
- 2011: Deprecation of SSL… version 2

A Brief History of SSL/TLS (2)

rt-solutions.de

- 2011: Hostname checking unified in RFC6125, named…

"Representation and Verification
of
Domain-Based Application Service Identity
Within
Internet Public Key Infrastructure
Using
X.509 (PKIX) Certificates
in the Context of
Transport Layer Security (TLS)"

A Brief History of SSL/TLS (3)

rt-solutions.de

- 2012, 2013: CRIME and BREACH attacks on compression
- 2014: POODLE attack deprecates SSLv3

Challenges for Developers

rt-solutions.de

How hard can secure communication with TLS be?

- Certificate Verification
  - Is the presented certificate valid (in terms of time)?
  - Is it signed by a “trusted” Certificate Authority?
- Hostname Verification
  - Does the certificate match the server we want to talk to?
- Development/Production
  - TLS stands in the way during application development
  - Got a cert for „www-dev.intranet“?
- Users want Self-Signed / Expired / Wrong-hostname Certs
  - Typically in „private cloud“ installations

How to use TLS in Java?

Theory:

```
public abstract class SSLSocket extends Socket
```

This class extends `Sockets` and provides secure socket using protocols such as the "Secure Sockets Layer" (SSL) or IETF "Transport Layer Security" (TLS) protocols.

Such sockets are normal stream sockets, but they **add a layer of security protections** over the underlying network transport protocol, such as TCP. Those protections include:

- *Integrity Protection*. SSL protects against modification of messages by an active wiretapper.
- *Authentication*. In most modes, SSL provides peer authentication. Servers are usually authenticated, and clients may be authenticated as requested by servers.
- *Confidentiality (Privacy Protection)*. In most modes, SSL encrypts data being sent between client and server. This protects the confidentiality of data, so that passive wiretappers won't see sensitive data such as financial information or personal information of many kinds.

How to use TLS in Java?

Theory:

`public class HttpsURLConnection extends HttpURLConnection`

HttpsURLConnection extends HttpURLConnection with support for **https-specific features**.

See http://www.w3.org/pub/WWW/Protocols/ and RFC 2818 for more details on the https specification.

This class uses HostnameVerifier and SSLSocketFactory. There are default implementations defined for both classes. However, the implementations can be replaced on a per-class (static) or per-instance basis. All new HttpsURLConnections instances will be assigned the "default" static values at instance creation, but they can be overriden by calling the appropriate per-instance set method(s) before connecting.

How to use TLS in Java?

rt-solutions.de

Practice:

```
java.security.cert.CertPathValidatorException: Trust anchor for certification path not found.
        at org.apache.harmony.xnet.provider.jsse.TrustManagerImpl.checkTrusted(TrustManagerImpl.java:282)
        at org.apache.harmony.xnet.provider.jss                    mpl.checkServerTrusted(TrustManagerImpl.jav
        at de.duenndns.ssl.MemorizingTrustManag                    usted(MemorizingTrustManager.java:392)
        at de.duenndns.ssl.MemorizingTrustManager.checkServerTrusted(MemorizingTrustManager.java:430)
        at org.apache.harmony.xnet.provider.jsse.OpenSSLSocketImpl                 ificateChain(OpenSSLSocketIm
        at org.apache.harmony.xnet.provider.jsse.NativeCrypto.SSL_do_handshake(Native Method)
        at org.apache.harmony.xnet.provider.jsse.OpenSSLSocketImpl.startHandshake(OpenSSLSocketImpl.java:
        at libcore.net.http.HttpConnection.setupSecureSocket(HttpConnection.java:209)
        at libcore.net.http.HttpsURLConnectionImpl$HttpsEngine.makeSslConnection(HttpsURLConnectionImpl.j
        at libcore.net.http.HttpsURLConnectionImpl$HttpsEngine.connect(HttpsURLConnectionImpl.java:433)
        at libcore.net.http.HttpEngine.sendSocketRequest(HttpEngine.java:290)
        at libcore.net.http.HttpEngine.sendRequest(HttpEngine.java:240)
        at libcore.net.http.HttpURLConnectionImpl.connect(HttpURLConnectionImpl.java:81)
        at libcore.net.http.HttpsURLConnectionImpl.connect(HttpsURLConnectionImpl.java:165)
        at de.duenndns.mtmexample.MTMExample$2.run(MTMExample.java:101)
Caused by: java.security.cert.CertPathValidatorException: Trust anchor for certification path not found.
        ... 15 more
```

anchor?!?

path!?!

- Certificate Verification is too secure!

Certificate Verification in Java 1.4-1.6

rt-solutions.de

- SSLSocket / SSLEngine (basic building blocks)
  - somehow uses X509TrustManager to check certificates

```
interface X509TrustManager extends TrustManager {
    void checkClientTrusted(X509Certificate[], String) throws CertificateException;
    void checkServerTrusted(X509Certificate[], String) throws CertificateException;
    X509Certificate[] getAcceptedIssuers();
}
```

- SSLSocket created by SSLSocketFactory
- SSLSocketFactory obtained from SSLContext
- SSLContext initialized with TrustManager(s)!

```
void checkServerTrusted(X509Certificate[], String) {
    return; // accepts all certificates, buried deep in your production code
}
```

- ✔ Certificate Verification: All-or-Nothing solution

Hostname Verification in Java 1.4-1.6

rt-solutions.de

- SSLSocket documentation: Yes, sir! We are Secure!
- SSLSocket reality: this is an application-layer problem!
- Application layer code in Java JRE: HttpsUrlConnection

```
HttpsUrlConnection.setHostnameVerifier(HostnameVerifier v):

public interface HostnameVerifier {
    boolean verify(String hostname, SSLSession session);
}
```

- To be called right after the TLS handshake
- Attention: returns boolean instead of exception!

Hostname Verification in Java 1.4-1.6

rt-solutions.de

Hostname verification in your own (non-HTTPS) code:

- Call `hostnameVerifier.verify(hostname, session)` right after completing the TLS handshake…
  - **…and check the return value!**
- `HostnameVerifier hostnameVerifier = ???`
- Reference implementation in Java?
  - None available ☹
- `HttpsUrlConnection.getDefaultHostnameVerifier()`?
  - It always returns false ☹
    „[Only] if [HttpsUrlConnection's] standard hostname verification logic fails, the implementation will call the `verify` method"

Hostname Verification in Java 1.4-1.6

rt-solutions.de

- Use Java's Secure Socket Extension Reference Guide:

"For example:

```
public class MyHostnameVerifier implements HostnameVerifier {
    public boolean verify(String hostname, SSLSession session) {
        // pop up an interactive dialog box
        // or insert additional matching logic
        if (good_address) {
            return true;
        } else {
            return false;
        }
    }
}
```

"

Hostname Verification in Java 1.4-1.6

- Sounds easy! Lets write our own HostnameVerifier!
  - CommonName vs. SubjectAltName(s)
  - International Domain Name (IDN) encoding
  - WildCard certificates (think „*.co.uk“)
  - IP addresses
  - IPv6 addresses
  - Embedded NUL bytes
  - …
  - RFC6125 is 57 pages

Hostname Verification: Apache

rt-solutions.de

Maybe somebody else wrote one?

- Apache HttpClient has a working verifier (also in Android)
  `interface X509HostnameVerifier extends HostnameVerifier`
- Watch out for the API!
  - Apache: void, throws SSLException
  - Java: returns boolean

- `StrictHostnameVerifier`
- `BrowserCompatHostnameVerifier` (less strict with wildcards)
- `AllowAllHostnameVerifier` (not strict at all)

- ✔ Once again: All-or-Nothing

Hostname Verification: Java 7

What about Java 7+?

```
public class X509ExtendedTrustManager implements X509TrustManager
```

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 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.

```
public abstract class X509ExtendedTrustManager implements X509TrustManager {
    void checkServerTrusted(X509Certificate[], String, Socket)
            throws CertificateException;
    void checkServerTrusted(X509Certificate[], String, SSLEngine)
            throws CertificateException;

    ….
}
```

Hostname Verification: Java 7

rt-solutions.de

- Java Runtime checks if passed TrustManager is an X509ExtendedTrustManager, calls the right methods ☺
- Must be enabled manually before connecting:

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

- To wrap around regular sockets („STARTTLS“):

```
sslSocket = sslContext.getSocketFactory().createSocket(
      plainSocket,
      hostName, /* ← use actual service/domain name */
      plainSocket.getPort(), true);
// set hostname checking parameter, per above
```

- ✓ Secure Hostname Verification beyond HTTPS
- ✓ Still All-or-Nothing

What about Android?

rt-solutions.de

- Android features `SSLCertificateSocketFactory` API
  - Available since API level 1 (on **all** devices!)
  - Well-documented security properties (warnings all around)
  - Development-mode support for „disabling security“
    - `.getInsecure()` socket factory
    - "setprop socket.relaxsslcheck yes" to disable (on your phone)
  - Warnings in LogCat when used insecurely

- ✓ Certificate **and** Hostname Verification
- ✓ Support for development mode
- ✓ Production: All-or-Nothing with easy toggle

- ✓ Used by: <5% of TLS-using Android apps

„Hardcoded“ Backend Servers

rt-solutions.de

- Using your corporate backend („api.mycompany.com“)
  - Well-known server(s)
  - Well-known certificates
- Certificate Pinning: „pin“ the server identity to your app
  - Robustness against Root-/Corporate-CA MitM
  - Works with self-signed / private CA
- Pinning of server/CA *public key*
  - Replace (expired) certificate without invalidating pin
  - Explicit pin expiration mechanism needed
  - AndroidPinning (moxie); java-pinning (Flowdalic)
- Pinning of server/CA *certificate*
  - Must update app before changing server cert
  - 6 lines of Java code

„Hardcoded“ Backend Servers (2)

rt-solutions.de

- Pin(s) bundled in a keystore file

```
keytool -import -trustcacerts -alias ca -file ca.crt -keystore app_pins.jks
Enter keystore password: password
Re-enter new password: password
```

- Keystore file used as trust root

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

- Use separate keystores for dev and production
- ✔ Secure use of self-signed/official certificates

User-Configured Servers

rt-solutions.de

- „Private Cloud“ applications (XMPP, contact sync, …)
  - Users are „creative“ with their TLS certificates
  - Self-signed/expired/wrong hostname (or all of the above)
- Trust On First Use/Persistence of Pseudonymity (TOFU/POP)
  - Store certificate|public-key on first connect
  - Check with local copy on subsequent connects
  - Connection error if server credentials change
- TOFU key lifetime management mismatches PKI
  - Distinguish MitM attack from server key update
  - SSH-like interactive approach required
  - Challenge: network thread vs. UI thread

User-Configured Servers (2)

rt-solutions.de

- Android: MemorizingTrustManager
  - Plug-in TrustManager
  - Interactive dialog for untrusted certificate / server names
  - Android library project
  - Open Source (MIT):

https://github.com/ge0rg/MemorizingTrustManager/

- Fine-grained security model

- Problem: users do not care!
  - Always click „Always“

TLS Beyond Root CAs

rt-solutions.de

- Problem 1: Users do not (want to) care
- Problem 2: Root Certificate Authority system flawed
  - 650+ CAs trusted by Windows, Mozilla, …
  - Any of them can sign any domain name!
  - Comodo/UserTrust, DigiNotar, TurkTrust
- Certificate Pinning: partial/intermediate solution
  - Hardcoded pins are inflexible
  - Does not scale well (TLS terminators, load-balancers, …)
  - Easy to screw up, hard to recover
- Solution: make pinning more flexible

TLS Beyond Root CAs

rt-solutions.de

- Trust Assertions for Certificate Keys (TACK)
  - Independent signing key (TSK) for server certificates
  - Short-lived pinning of TSK/hostname (max. 30 days)
  - In-band transmission (hard, but possible to MitM)
  - TLS Extension (hard to roll out)
  - Draft RFC (expired in 2013 ☹)
  - No Java implementation available ☹
- DNS-based Authentication of Named Entities (DANE)
  - Server/CA identity stored in DNS
  - Support for Root-CA signed and self-signed certificates
  - TLS association (TLSA) record
  - Depends on (hierarchical) DNSSEC infrastructure
  - dnsjava / DNSSEC4j / dnssecjava – significant work needed!

Summary (for Developers)

rt-solutions.de

	Protection against MitM	Usable for Private Cloud	Dev/Production Switch	Code available
Just SSLSocket	☹	☹	☹	☺
Apache HttpClient	☺	☹	☹	☺
SSLCertificate-SocketFactory	☺	☹	☺	☺
KeyStore file / java-pinning	☺☺☺	😐	☺	☺
AndroidPinning	☺☺☺	☹	☹	☺
MemorizingTrustM.	😐	☺☺☺	☺	☺
TACK	☺☺	☹	☹	☹
DANE	☺☺☺	☺	☺	☹

MitM protection against…
- ☺ normal attackers
- ☺☺☺ government attackers

Private cloud: supports…
- ☺ self-signed certs
- ☺☺☺ expired certs / invalid hostnames

Thank you! Questions?

rt-solutions.de

- Why Eve and Mallory Love Android: An Analysis of SSL (In)Security on Android, S. Fahl et al.;
  2012 ACM Conference on Computer and Communications Security
- The Most Dangerous Code in the World: Validating SSL Certificates in Non-Browser Software, M. Georgiev et al.;
  2012 ACM Conference on Computer and Communications Security
- Fixing the Most Dangerous Code in the World and follow-up articles,
  Will Sargent; 2014 Terse Systems
- Trust Assertions for Certificate Keys, M. Marlinspike and T. Perrin;
  2013 TLS-WG (draft-perrin-tls-tack-02.txt)
- Java/Android SSLSocket Vulnerable to MitM Attacks, G. Lukas;
  2014 op-co.de

Georg Lukas <lukas@rt-solutions.de>
rt-solutions.de GmbH, Köln

Reference Material

Enforcing Protocols And Ciphers

rt-solutions.de

- TLS 1.0 (not recommended since 2006)
  - Default in Java JRE<8
  - Default on Android <5.0 Lollipop
- TLS 1.2 must be enabled explicitly before connecting:

```
sslSocket.setEnabledProtocols(new String[] { "TLSv1.2" })
```

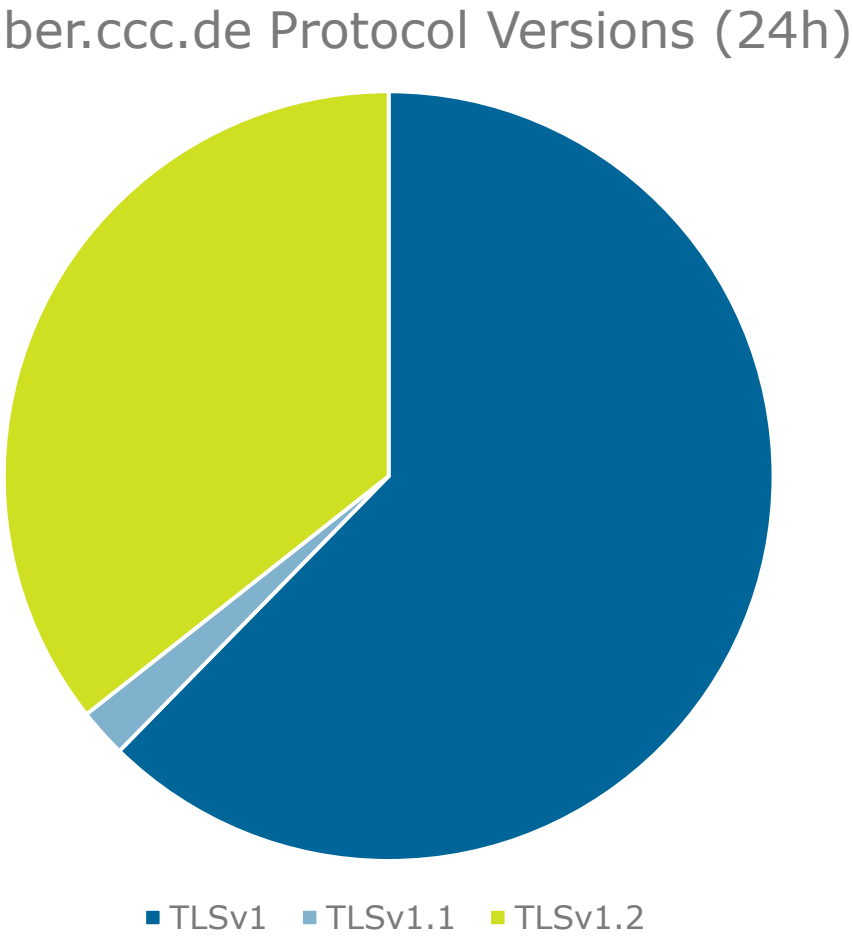
  - Java: requires JRE7
  - Android: requires 4.1 Jelly Bean (5.0 for async SSLEngine)
  - Better: filter results from `sslSocket.getSupportedProtocols()`
- Cipher suite: RC4 (proven insecure in 2013) is everywhere
  - Default in JRE6
  - Default on Android 2.3 – 4.4 (2010-2014)

```
sslSocket.setEnabledCipherSuites(new String[] { "????" })
```

  - Suggesting a sane default list almost impossible ☹

Some Numbers

rt-solutions.de



Some More Numbers

rt-solutions.de