SSL implementieren – aber sicher!

Karlsruher Entwicklertag 2014
21.05.2014

Dr. Yun Ding
SSL in the news

2011
BEAST
Compromised CAs

2012
CRIME

2013
Lucky 13
RC4 biases
BREACH
DRBG Backdoor

2014
Apple goto fail
GnuTLS goto
OpenSSL HeartBleed
SSL in scientific publications

2012

“The Most Dangerous Code in the World: Validating SSL Certificates …”


2013

“Rethinking SSL Development in an Applied World”


Apps vulnerable to MITM

- **Android Apps**: 8% of 1,074 out of 13,500 Apps
- **iOS Apps**: 14% of 98 out of 697 Apps
# Layers of SSL-based applications

<table>
<thead>
<tr>
<th>Human</th>
<th>Application</th>
<th>Banking</th>
<th>Shopping</th>
<th>Messaging</th>
<th>Browser</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Middleware/ Wrappers</td>
<td>Apache HttpClient</td>
<td>cURL</td>
<td>PhoneGap</td>
<td>MKNetworkKit</td>
</tr>
<tr>
<td>SSL Libraries</td>
<td>GnuTLS</td>
<td>Apple Secure Transport</td>
<td>OpenSSL</td>
<td>JSSE</td>
<td></td>
</tr>
<tr>
<td>SSL Protocols</td>
<td>Secure Protocols</td>
<td>Cipher Suites</td>
<td>Renegotiation</td>
<td>Compression</td>
<td></td>
</tr>
<tr>
<td>Cryptographic Primitives</td>
<td>Random Number Generators</td>
<td>Hash</td>
<td>Encryption</td>
<td>Authentication</td>
<td></td>
</tr>
</tbody>
</table>
How does SSL work?

Handshake: Key Exchange

Encrypted Data Communication

How does SSL break?

<table>
<thead>
<tr>
<th>Trick user <em>not</em> to encrypt</th>
<th>SSL stripping</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predict the key</td>
<td>DRBG backdoor</td>
</tr>
<tr>
<td>Trick user to use attacker’s key</td>
<td>Apple goto fail, GnuTLS goto, MITM</td>
</tr>
<tr>
<td>Trick server to expose keys</td>
<td>OpenSSL Heartbleed</td>
</tr>
<tr>
<td>Perform cryptographic analysis to decrypt</td>
<td>RC4 biases, Lucky13, CRIME, BEAST, Breach</td>
</tr>
</tbody>
</table>

Copyright: © 2012 dny3d, bigstock.com, Scanrail, bigstock.com
SSL relies on Trust in Certificates ...
SSL relies on Valid Certificates

1. Make sure certificate validation is not turned off!
2. Verify the certificate is valid: not expired, not revoked
3. Validate “Chain of Trust”
4. Don’t accept self-signed certificates
5. Make sure hostname validation is set
What went wrong

- Insecure coding
  - Skipped or broken certificate validation
- Badly designed APIs
  - Expose low-level SSL protocol details, complex options
  - Complex relationship between return values and error status
  - Unsafe defaults (+missing warning in API Doc)
- Delegate responsibility to application developers
## Default behavior in SSL lib. & wrappers

<table>
<thead>
<tr>
<th>Libraries/Wrappers</th>
<th>Chain of Trust</th>
<th>Hostname Verification</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenSSL</td>
<td>✔️</td>
<td>✗</td>
</tr>
<tr>
<td>GnuTLS</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>CyaSSL</td>
<td>✔️</td>
<td>✗</td>
</tr>
<tr>
<td>JSSE SSLSocketFactory</td>
<td>✔️</td>
<td>✗</td>
</tr>
<tr>
<td>SSLSocketFactory</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>HttpsURLConnection</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Apache HttpClient 3.*</td>
<td>✔️</td>
<td>✗</td>
</tr>
<tr>
<td>HttpClient 4.*</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Python ssl module</td>
<td>✔️</td>
<td>✗</td>
</tr>
</tbody>
</table>
What went wrong

- Lack of understanding of how SSL works and breaks
- Misinterpretation of manifold SSL parameters & options
- Delegate responsibility to end users with warnings
- “Security gets in the way”
When Security gets in the way ...

Override (secure) standard certificate validation

- disable or break certificate validation
- disabled in development & forget to remove in production
Customized Trust Manager in Java

```java
import javax.net.ssl.*;
import java.security.cert.*;

public class DisableValidationTrustManager implements X509TrustManager {
    public DisableValidationTrustManager() {
    }
    public void checkServerTrusted(java.security.cert.X509Certificate[] pl, String p2) {
        System.out.println("I don't validate any certs!");
        return;
    }
}

TrustManager tm[] = {new DisableValidationTrustManager()};
SSLContext context;
try {
    context = SSLContext.getInstance("TLS");
    context.init(null, tm, null);
} catch (NoSuchAlgorithmException e) {
    e.printStackTrace();
} catch (KeyManagementException e) {
    e.printStackTrace();
}
```
Hostname Verification in HttpClient (4.3)

- Skip hostname verification: communicate with another host
- Customization to skip hostname verification:

  ```java
  new SSLConnectionSocketFactory(sslContext, new AllowAllHostnameVerifier())
  ```
Decouple test and production code

- Don’t hardcode insecure certificate validation (and forget)
- Use best practices in software architecture for decoupling
  - Abstract Factory Design Pattern
  - Dependency Injection, configuration instead of programming
Customization for more Security!

- SSL Certificate or Public Key Pinning
  - Whitelist expected Certificates or Public Keys
  - Pre-existing binding between the server and its certificate/public key

Sample code available on OWASP
Secure SSL configuration

- Use secure protocols: TLS v1.2, TLS v1.1, TLS v1.0
- Use secure cipher suites
  - Support authentication & encryption ≥ 128 bit  Avoid
  - Use ECDHE for forward secrecy
  - Avoid anonymous DH, null cipher, RC4, 3DES
- RSA and DSA key must be ≥ 2048 bits
- Disable client-initiated Renegotiation
- Disable TLS compression
Secure SSL configuration

- Avoid mixed TLS and non-TLS content
- Secure cookies
- Deploy HTTP Strict Transport Security (HSTS)
- Prevent caching of sensitive content

“SSL/TLS Deployment Best Practices” of Qualys SSL Labs

OWASP “Transport Layer Protection Cheat Sheet”
https://owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet
Test SSL

- Perform adversarial testing: abnormal certificates, MITM attacking tools (sslsniff, mitmproxy)
- Testing for SSL/TLS ciphers, protocols, keys and know vulnerabilities (e.g., BEAST, CRIME, Heartbleed)

<table>
<thead>
<tr>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Protocols</strong></td>
</tr>
<tr>
<td>TLS 1.2</td>
</tr>
<tr>
<td>TLS 1.1</td>
</tr>
<tr>
<td>TLS 1.0</td>
</tr>
<tr>
<td>SSL 3</td>
</tr>
<tr>
<td>SSL 2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cipher Suites (SSL 3+ suites in server-preferred order, then SSL 2 suites where used)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLS_RSA_WITH_RC4_128_SHA (0x1)</td>
</tr>
<tr>
<td>TLS_RSA_WITH_RC4_128_MD5 (0x4)</td>
</tr>
<tr>
<td>TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)</td>
</tr>
<tr>
<td>ECDH 256 bits (96, 3072 bits RSA)</td>
</tr>
</tbody>
</table>

- [http://thoughtcrime.org/software/sslsniff/](http://thoughtcrime.org/software/sslsniff/)
- [http://mitmproxy.org/](http://mitmproxy.org/)
Tools: Creating Keys and Certs

- Java Keytool
- OpenSSL: powerful, but complex
  - Based on OpenSSL
  - Provides a Graphical User Interface (GUI)
  - Based on GnuTLS
  - Provides GUI and command line support
Tools: Creating Keys and Certs with xca
Securely implement SSL!

- Understand how SSL works and breaks
- Use SSL libraries and middleware securely
  - Don’t rely on default settings of SSL libraries and middleware/wrappers
  - Look out for badly designed SSL API (return value, error status)
- Perform certificate validation properly
  - Verify the certificate is valid: not expired, not revoked
  - Validate “Chain of Trust”
  - Don’t accept self-signed certificates
  - Make sure hostname validation is set
- Decouple insecure customized certificate handling from production code
- Test for insecure SSL configurations
Engineering SSL is System Security Engineering

Interdisciplinary

developers

hackers

secure coding

end users

assumptions

weakest link

dependencies

boundaries

interactions

security community

psychology

open source

policies

usability

System Security Engineering
References

