Systematic Fuzzing and Testing of TLS Libraries

Juraj Somorovsky
Transport Layer Security

- The most important crypto protocol
- HTTP, SMTP, IMAP ...
TLS History

Secure Sockets Layer (SSL), SSLv2

SSLv3

Transport Layer Security

1995

Wagner, Schneier: Analysis of SSLv3

Bleichenbacher’s attack

Padding oracle attack

2000

2005

TLS 1.1

TLS 1.2

2010

BEAST, CRIME, BREACH, Lucky 13

2015

TLS 1.3

Juraj Somorovsky. Systematic Fuzzing and Testing of TLS Libraries
Questions

• How can we test these attacks?
• Can we find such attacks automatically?
Approach [SP2-17]

1. Collect TLS libraries
2. 
3. Profit
Approach [SP2-17]

1. Collect TLS libraries
2. 
3. Profit
Contributions

- Flexible TLS framework
- Fuzzing, testing, writing attacks ...
- High impact vulnerability in OpenSSL
- Additional vulnerabilities in Botan, MatrixSSL...
- [https://github.com/RUB-NDS/TLS-Attacker](https://github.com/RUB-NDS/TLS-Attacker)
Overview

1. TLS Protocol
2. Attacks
3. Framework Prerequisites
4. TLS-Attacker Design
5. Fuzzing
6. Results
7. Conclusions
TLS RSA Handshake

ClientHello → ServerHello

ClientKeyExchange → ServerHelloDone

ChangeCipherSpec → Certificate

(Client-) Finished

ChangeCipherSpec → (Server-) Finished

Application → Application
TLS is complex ...

• Different versions
• Crypto primitives: RSA, EC, AES, 3DES, RC4, Chacha, Poly1305, New Hope
• Extensions
• Protocol flows
TLS is complex ...

ClientHello → ServerHello → Certificate → ServerKeyExchange → ServerHelloDone → ClientKeyExchange → CertificateVerify → ChangeCipherSpec → (Client-) Finished → ChangeCipherSpec → (Server-) Finished → Heartbeat → Heartbeat
Overview

1. TLS Protocol
2. Attacks
3. Framework Prerequisites
4. TLS-Attacker Design
5. Fuzzing
6. Results
7. Conclusions
Secure Sockets Layer (SSL), SSLv2

SSLv3

Transport Layer Security

TLS 1.1

TLS 1.2

TLS 1.3

1995

Wagner, Schneier: Analysis of SSLv3

Bleichenbacher’s attack

2000

Padding oracle attack

2005

BEAST, CRIME, BREACH, Lucky 13

2010

2015

Juraj Somorovsky. Systematic Fuzzing and Testing of TLS Libraries
Early CCS

Server computes the master key based on a zero value
Early CCS

• Man-in-the-Middle attacks

• Further state machine attacks in 2015:
  – Beurdouche et al.: FREAK
  – de Ruiter and Poll
Heartbleed

[Heartbleed]

[Heartbeat]
00 07 DeepSec

[Heartbeat]
00 07 DeepSec

[Server]
Padding oracle attacks

- Adaptive chosen-ciphertext attacks

AES-CBC: Vaudenay’s attack
RSA-PKCS#1: Bleichenbacher’s attack
Overview

1. TLS Protocol
2. Attacks
3. Framework Prerequisites
4. TLS-Attacker Design
5. Fuzzing
6. Results
7. Conclusions
Recent Attacks on TLS

• Not only crypto attacks ...

• Attacks on TLS state machines
  – FREAK
  – Early CCS

• Buffer overflows / overreads
  – Heartbleed
  – CVE-2016-6307 (High) -> CVE-2016-6309 (Critical)

• Tool for flexible protocol executions needed
Framework Prerequisites

- Flexible protocol flow definition
- Message modifications
- Invalid behavior detection
- Protocol flow reproduction
Overview

1. TLS Protocol
2. Attacks
3. Framework Prerequisites
4. TLS-Attacker Design
5. Fuzzing
6. Results
7. Conclusions
Modifiable variables

• Define basic data types (integer, byte, arrays) with modifications

• Example:

```java
ModifiableInteger i = new ModifiableInteger();
i.setValue(30);
i.setModification(new AddModification(20));
System.out.println(i.getValue());  // 50
```

• Further modifications: xor, shuffle, delete, ...
Protocol messages

• ClientHello

<table>
<thead>
<tr>
<th>ClientHelloMessage</th>
</tr>
</thead>
<tbody>
<tr>
<td>cipherSuites: ModifiableByteArray</td>
</tr>
<tr>
<td>cipherSuiteLength: ModifiableInteger</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>getCipherSuites()</td>
</tr>
<tr>
<td>getCipherSuiteLength()</td>
</tr>
</tbody>
</table>

• Stored in a message list

• Serializable in XML
Defining a protocol flow

```xml
<protocolMessages>
  <ClientHello>
    <supportedCipherSuites>
      <CipherSuite>TLS_RSA_WITH_AES_128_CBC_SHA</CipherSuite>
    </supportedCipherSuites>
  </ClientHello>
  <ServerHello/>
  <Certificate/>
  <ServerHelloDone/>
  <RSAClientKeyExchange/>
  <ChangeCipherSpec/>
  <Finished/>
  <ChangeCipherSpec/>
  <Finished/>
  <Application/>
</protocolMessages>
```
Defining a protocol flow

```xml
<protocolMessages>
  <ClientHello>
    <supportedCipherSuites>
      <CipherSuite>tls_rsa_with_aes_128_cbc_sha</CipherSuite>
    </supportedCipherSuites>
  </ClientHello>
  <ServerHello/>
  <Certificate/>
  <ServerHelloDone/>
  <RSAClientKeyExchange/>
  <ChangeCipherSpec/>
  <Finished/>
  <ChangeCipherSpec/>
  <Finished/>
  <Heartbeat/>
</protocolMessages>

<Heartbeat>
  <payloadLength>
    <integerAddModification>20000</integerAddModification>
  </payloadLength>
</Heartbeat>
```
Overview

1. TLS Protocol
2. Attacks
3. Framework Prerequisites
4. TLS-Attacker Design
5. Fuzzing
6. Results
7. Conclusions
Vulnerability detection

• How do we detect invalid server behavior?

1. Different TLS alerts
   – Useful by padding oracle attacks

2. Address Sanitizer (ASan)
   – Detects memory errors at runtime
   – Available in recent compilers, e.g. GCC

• Vulnerability found -> protocol stored in XML
Two-stage concept

• Currently only server evaluation

1. Crypto
   – Padding oracles, Bleichenbacher attack, invalid curve attacks, POODLE ...

2. Fuzzing for boundary violations
   – 3 phases
Fuzzing for boundary violations

1. Variable filtering
   - Not all variables suitable

2. Fuzzing with filtered variables
   - Random modifications (add, delete, xor)
   - Boundary values (-128, -1, 0, 32768, ...)

3. Fuzzing with modified protocol flows

ClientHelloMessage
- cipherSuites
- cipherSuiteLength
- clientRandom
- extensions
- extensionLength
- ....

Demo
Overview

1. TLS Protocol
2. Attacks
3. Framework Prerequisites
4. TLS-Attacker Design
5. Fuzzing
6. Results
7. Conclusions
Results

• Padding oracle attack
  – OpenSSL (CVE-2016-2107)
  – Botan 1.11.21 (CVE-2015-7824)
  – MatrixSSL 3.8.2

• Bleichenbacher attack
  – MatrixSSL 3.8.2

• Missing length checks
  – GnuTLS 3.4.9
  – OpenSSL 1.0.1

• Out-of-bound reads / writes
  – OpenSSL-1.1.0-pre1 (stack overflow)
  – Botan 1.11.28 (Out-of-bound read)
Padding oracle attack

- Applicable to AES-CBC
- Challenge: not to reveal padding validity
  1. **Same** error message
  2. **Constant time** padding and HMAC validation
AES-CBC in TLS

• MAC-Pad-Encrypt

• Example:
  – Two blocks
  – Message: Hello
  – MAC size: 20 bytes (SHA-1)
  – Padding size: $32 - 5 - 20 = 7$

Hello

\[
\begin{array}{cccccccc}
06 & 06 & 06 & 06 & 06 & 06 & 06 & 06
\end{array}
\]
AES-CBC in TLS

• Challenge: not to reveal padding validity
• Always:
  – Padding validation
  – MAC validation
• Same error message and timing

What can go wrong?

Hello

06 06 06 06 06 06 06 06
Constant Time Validation

Decrypted data:

<table>
<thead>
<tr>
<th>H</th>
<th>e</th>
<th>l</th>
<th>l</th>
<th>o</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Valid = true

Mask data:

<table>
<thead>
<tr>
<th>H</th>
<th>e</th>
<th>l</th>
<th>l</th>
<th>o</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>06</th>
<th>06</th>
<th>06</th>
<th>06</th>
<th>06</th>
<th>06</th>
<th>06</th>
</tr>
</thead>
</table>
Constant Time Validation

**Decrypted data**

*Valid = false*

**Mask data**
OpenSSL Vulnerability

Decrypted data

Valid = true

Mask data
OpenSSL Vulnerability (CVE-2016-2107)

- Introduced by patching Lucky 13
- Only when using AES-NI
- Leads to a different server response

http://web-in-security.blogspot.co.at/2016/05/curious-padding-oracle-in-openssl-cve.html

Can this be even worse?
Yes

- MatrixSSL 3.8.2
- Timing attack -> buffer overflow
Overview

1. TLS Protocol
2. Attacks
3. Framework Prerequisites
4. TLS-Attacker Design
5. Fuzzing
6. Results
7. Conclusions
Conclusions and future work

• Maintaining a crypto library is hard
• New code / patches can introduce new flaws
• Systematic fuzzing and evaluation needed
• TLS-Attacker
  – For researchers, pentesters
  – For developers
• Development / fuzzing improvements needed
  – TLS client-side tests
  – Better fuzzing strategies