TLSkex: Harnessing virtual machine introspection for decrypting TLS communication

Benjamin Taubmann, Dominik Dusold, Christoph Frädrich, Hans P. Reiser
Juniorprofessur für Sicherheit in Informationssystemen
Universität Passau

31.03.2016
Motivation

- Encrypted communication (especially TLS) became ubiquitous in the Internet (https, voip, mail, etc.)
- **Why do we need to decrypt encrypted communication?**
  - Adversaries use encryption to hide attacks
  - Network intrusion detection systems can not decrypt network traffic (especially Infrastructure-as-a-service based Clouds)
  - Distinction between legitimate and malicious traffic is hard
  - Malware analysis
Active Approaches

- **Man-in-the-middle attack (MITM)**
  - Force client not to encrypt: proxy replaces `https` with `http` in URLs of websites (sslstrip [4])
  - TLS Proxy that uses a fake certificate (sslsniff [3], sslsplit [1])

- **Disadvantages:**
  - can be detected
  - does not work with certificate pinning
  - (may) reduce the security level
Passive Approaches

- Decryption of the traffic with the private **RSA key** (ssldump [2])
  - Key acquisition from hard disk or main memory
  - Extraction of the RSA keys from main memory (simple pattern matching for the ASN.1 structure);

- **Disadvantages:**
  - Only applicable when the hard disk or main memory can be accessed
  - Not feasible for malware analysis
  - Fails when DH or ECDH session keys are used
TLSkex

Approach

- Monitor the network traffic of virtual machines (passively)
- Extract the *TLS session keys* from main memory of virtual machines
- Disadvantage: Works only when access to main memory is given (virtual machines, Firewire, ... )
TLSkex Architecture
Background: TLS Key Exchange

**TLSInspector actions**

- extract Client Random
- extract Server Random
- take snapshot
- extract first record

Client

- Hello
- Change Cipher Spec

Server

- Hello
- Change Cipher Spec
- first encrypted TLS record

[Diagram showing the TLS handshake process with labeled actions and messages between the client and server.]
**Timing**

- Snapshot must be taken when the key material is exchanged and before the connection is terminated.
- When a communication partner has computed the key material it sends a `ChangeCipherSpec` message.
- TLSInspector monitors the network traffic and triggers the snapshot when a `ChangeCipherSpec` message is sent.
TLSInspector Architecture

Network

Network logging

A

Network dump

Network dump

Key extraction

D

Viewer

Trigger mechanism

B

Memory acquisition

C

Virtual Machine

Process Memory

Main Memory

B

depicts the Trigger mechanism, which triggers a snapshot. This snapshot is then acquired by the Memory acquisition module, which can process both Main Memory and Process Memory. The acquired memory is then used by the Key extraction module to extract keys, which are then visualized by the Viewer.

The diagram also shows the network logging component, which captures network traffic, and the online/offline state of the system.
Snapshot

- The memory of a virtual machine is accessed via libVMI
- Only the memory of the process that handles the connection
- Only *write-able* and *anonymous* pages
- Extract information from kernel memory *(task_struct)*
Save only modified pages

- Set memory event on every page of a process when TCP connections is established
- Save dirty pages
- When hand shake is done, take snapshot of
  - dirty pages
  - newly allocated pages
Background: TLS Key Exchange

Client

- Hello

Server

- Hello

TLSInspector actions

- extract Client Random
- extract Server Random
- take snapshot
- extract first record
## Evaluation - Snapshot time

<table>
<thead>
<tr>
<th>Process</th>
<th>total</th>
<th>anon&amp;writeable</th>
<th>new</th>
<th>modified</th>
<th>dumped</th>
<th>t_{snap_start}</th>
<th>t_{snap_stop}</th>
<th>t_{search}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache2</td>
<td>72090</td>
<td>3715</td>
<td>0</td>
<td>26</td>
<td>26</td>
<td>4.3 ms</td>
<td>4.4 ms</td>
<td>30 ms</td>
</tr>
<tr>
<td>Curl</td>
<td>38264</td>
<td>3438</td>
<td>15</td>
<td>19</td>
<td>34</td>
<td>3.3 ms</td>
<td>4.0 ms</td>
<td>2 ms</td>
</tr>
<tr>
<td>Wget</td>
<td>22813</td>
<td>1378</td>
<td>16</td>
<td>13</td>
<td>29</td>
<td>4.0 ms</td>
<td>3.5 ms</td>
<td>2 ms</td>
</tr>
<tr>
<td>s_client</td>
<td>6114</td>
<td>152</td>
<td>9</td>
<td>22</td>
<td>31</td>
<td>0.4 ms</td>
<td>0.6 ms</td>
<td>8 ms</td>
</tr>
</tbody>
</table>

**Table:** Amount of mapped and changed memory pages (4096 bytes) of different processes during the key negotiation procedure and the time to prepare \( t\_{\text{snap\_start}} \) and take \( t\_{\text{snap\_stop}} \) a differential snapshot; \( t\_{\text{search}} \) denotes the time to extract a key from a snapshot.
Problem
- There is no standardized way to store TLS session keys in memory

Brute force approach
- Test every byte sequence as a key for the TLS connection
- Use message authentication code in TLS record to verify if key is corrected and data has been decrypted correctly

**Advantage:** Implementation independent

**Problem:** Attacker can fool monitoring tool, e.g., split key
(D) Key Extraction Improvements

- Test only 4 byte aligned sequences
- Skip null byte areas
- Heuristics - Pretest keys
  - check if it is ASCII string
  - compare amount of zeroes and ones, should be uniformly distributed

\[ \sum_{\mu - k}^{\mu + k} \binom{n}{k} p^k (1 - p)^{n-k} \geq 0.89 \]

\[ k = 16, \mu = 192, p = 0.5 \]
## Evaluation - Heuristics

<table>
<thead>
<tr>
<th>Process</th>
<th>k=1</th>
<th>k=2</th>
<th>k=4</th>
<th>k=8</th>
<th>k=16</th>
<th>k=32</th>
<th>no string</th>
<th>not all 0 / 1</th>
<th>combined (k=16)</th>
</tr>
</thead>
<tbody>
<tr>
<td>key included</td>
<td>8.12</td>
<td>16.2</td>
<td>31.6</td>
<td>58.5</td>
<td>89.7</td>
<td>99.9</td>
<td>1 − 10⁻¹⁵</td>
<td>1 − 10⁻¹⁹</td>
<td>87.7</td>
</tr>
<tr>
<td>Apache2</td>
<td>0.10</td>
<td>0.28</td>
<td>0.64</td>
<td>1.27</td>
<td>2.33</td>
<td>4.26</td>
<td>85.49</td>
<td>43.54</td>
<td>1.69</td>
</tr>
<tr>
<td>Curl</td>
<td>0.15</td>
<td>0.45</td>
<td>1.04</td>
<td>2.11</td>
<td>3.50</td>
<td>4.75</td>
<td>77.53</td>
<td>10.55</td>
<td>3.32</td>
</tr>
<tr>
<td>Wget</td>
<td>0.15</td>
<td>0.46</td>
<td>1.06</td>
<td>2.15</td>
<td>3.60</td>
<td>4.91</td>
<td>78.10</td>
<td>10.68</td>
<td>3.38</td>
</tr>
<tr>
<td>s_client</td>
<td>0.054</td>
<td>0.18</td>
<td>0.49</td>
<td>0.96</td>
<td>1.89</td>
<td>3.40</td>
<td>56.52</td>
<td>37.35</td>
<td>1.63</td>
</tr>
</tbody>
</table>

**Table:** First row: probability that a key is not eliminated by the heuristic. Other rows: percentage of a memory snapshot that contains a 48 byte long and four byte aligned sequence with: a) $192 \pm k$ one bits, b) the byte sequence is not an ASCII string c) no 8 byte sequence with only zero or only one bits d) a to c combined
Discussion

- Easy to circumvent:
  - Start dedicated crypto process
  - Store key not in a 48 byte sequence
  - Mix byte order
  - Use different protocol or change TLS slightly

- Are kernel structures still trustworthy?

- Potential DoS vector?

- Ethical considerations
Conclusion

**Non Intrusive**
- No active manipulation of communication
- No modification of application

**Universal:**
- Independence of specific key exchange
- Independence of encryption algorithm
- Independence of client/server role
- Independence of the implementation
Questions
References

P. C. Heckel.
Use SSLsplit to transparently sniff TLS/SSL connections – including non-HTTP(S) protocols.

S. Iveson.
Using ssldump to decode/decrypt SSL/TLS packets.

M. Marlinspike.
sslsniff.

M. Marlinspike.
sslstrip.