Secure Cryptographic Protocol
Execution based on
Runtime Verification

Secure Communication in the Quantum Era (SPS G5448)
February 5th, 2020

Christian Colombo
Mark Vella
Cryptographic Protocols

Design

Proofs to validate design against threat models

Implementation

Difficult to make it fully secure…
So many things can go wrong!
Levels of abstraction of security threats

(High level) Wrong protocol implementation
The protocol implementation might deviate from the verified (theoretical) design

Medium level threats
Malware, Data leaks, etc

Low level threats
Arithmetic overflows, undefined downcasts, and invalid pointer references

Hardware
Can hardware be trusted? Side Channel attacks?
It is difficult to make implementation fully secure…

but we can raise the bar as much as possible.
Our strategy

Isolate!
Our strategy

Isolate!

Monitor!
Monitor code while executing (High level)

Commodity h/w and stock OS
- User-mode
- AGKE code
- Token calls
- Kernel trap
- USB/network drivers

Crypto h/w token
- Crypto OS
- Generate/Retrieve
- APDU
- Keys
- Ciphers
- MCU

AGKE network exchanges

Monitor for data leaks (medium level)

Crypto OS calls
- Plain/ciphertext data exchanges
- NO key transfers

Full isolation

NO external code provisioning
Preliminary case study

Web server

Firefox implementation (C code)

Elliptic Curve Diffie-Hellman Exchange (ECDHE)
Preliminary implementation

Setup using Binary-level instrumentation

- Web server
- Binary instrumentation
- Firefox implementation (C code)
Preliminary implementation

Setup using Binary-level instrumentation

Through which monitors can gain visibility
Properties verified (High level) on ECDHE

Digital certificate verification is done (in order to authenticate public keys sent by peers)
Properties verified (High level) on ECDHE

Validation of remote peer's **public key** on each exchange is done (unless the session is aborted)
Properties verified (High level) on ECDHE

Once master secret is established, private keys should be **scrubbed from memory** (to limit the impact of memory leak attacks such as Heartbleed, irrespective of whether the session is aborted)
Feasibility study of approach

Is the approach possible for a realistic code base?

Is the approach feasible in terms of overheads?

Used the Firefox case study on top 100 Alexa sites
### Feasibility study

<table>
<thead>
<tr>
<th>TID 0x1003</th>
<th>PR_Close()</th>
</tr>
</thead>
<tbody>
<tr>
<td>200 ms</td>
<td></td>
</tr>
<tr>
<td>TID 0xffb</td>
<td></td>
</tr>
<tr>
<td>312 ms</td>
<td>SSL_ImportFD()</td>
</tr>
<tr>
<td>312 ms</td>
<td>ret:0x7faa43591940</td>
</tr>
<tr>
<td>312 ms</td>
<td>SSL_AuthCertificateHook()</td>
</tr>
<tr>
<td>312 ms</td>
<td>fd:0x7faa43591940</td>
</tr>
<tr>
<td>312 ms</td>
<td>ret:0x0</td>
</tr>
<tr>
<td>312 ms</td>
<td>PR_Connect()</td>
</tr>
<tr>
<td>312 ms</td>
<td>fd:0x7faa43591940</td>
</tr>
<tr>
<td>531 ms</td>
<td>SECKEY_CreateECPrivateKey()</td>
</tr>
<tr>
<td>531 ms</td>
<td>cx:0x7faa3deda988</td>
</tr>
</tbody>
</table>

```c
| 532 ms     | EC_ValidatePublicKey() |
| 532 ms     | ret:0x0 |
| 532 ms     | ret:0x7faa3dd66020::7faa3dd66020 c0 fe d9 |
| 3d         | ...=   |
| 533 ms     | SECKEY_CreateECPrivateKey() |
| 533 ms     | cx:0x7faa3deda988 |
| 534 ms     | EC_ValidatePublicKey() |
| 539 ms     | ret:0x0 |
| 539 ms     | ret:0x7faa3dd68020::7faa3dd68020 00 f1 10 |
```

- **Web server**
- **Binary instrumentation**
- **Firefox implementation (C code)**
- **Runtime Verification**
Overheads measurement

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Pages</th>
<th>Page load time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>mean</td>
</tr>
<tr>
<td>No RV</td>
<td>1,000</td>
<td>6,918.37</td>
</tr>
<tr>
<td>With RV</td>
<td>1,000</td>
<td>7,282.35</td>
</tr>
<tr>
<td>Mean overhead</td>
<td></td>
<td>0.05</td>
</tr>
<tr>
<td>Wilcoxon signed-rank test</td>
<td></td>
<td>p=0.281</td>
</tr>
</tbody>
</table>
## Overheads measurement

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Pages</th>
<th>Page load, mean [ms]</th>
<th>Page load, std. dev. [ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td>No RV</td>
<td>1,000</td>
<td>6,918.37</td>
<td>24,870.86</td>
</tr>
<tr>
<td>With RV</td>
<td>1,000</td>
<td>7,282.35</td>
<td>27,328.9</td>
</tr>
<tr>
<td>Mean overhead</td>
<td></td>
<td>0.05</td>
<td></td>
</tr>
<tr>
<td>Wilcoxon signed-rank test</td>
<td></td>
<td>p=0.281</td>
<td></td>
</tr>
</tbody>
</table>
Lessons learnt

Good start with promising results - approach seems feasible

Beware:

Program comprehension is required, both for setting up function hooks as well as to enable individual TLS session monitoring

Real-world code tends to be written in a manner to favor efficient execution rather than monitorability (eg, was difficult to keep track of particular sessions on the server)
Secure Communication in the Quantum Era

NATO Science for Peace and Security Programme, Project no. G5448

Partners:

- Slovakia - Slovak University of Technology
- Malta - University of Malta
- Spain - Universidad Rey Juan Carlos
- US - Florida Atlantic University

http://re-search.info/