Symbolic Execution of Security Protocol Impl.: Handling Cryptographic Primitives

Mathy Vanhoef — @vanhofem

USENIX WOOT, Baltimore, US, 14 August 2018
Overview

Symbolic Execution

4-way handshake

Handling Crypto

Results
Overview

Symbolic Execution

Handling Crypto

4-way handshake

Results
void recv(data, len) {
    if (data[0] != 1)
        return
    if (data[1] != len)
        return
    int num = len/data[2]
    ...
}
Symbolic Execution

\[
data[0] \neq 1
\]

```c
void recv(data, len) {
    if (data[0] != 1)
        return;
    if (data[1] != len)
        return;
    int num = len/data[2];
    ...
}
```

\[
data[0] == 1
\]

```c
void recv(data, len) {
    if (data[0] != 1)
        return;
    if (data[1] != len)
        return;
    int num = len/data[2];
    ...
}
```
Symbolic Execution

```
data[0] != 1
```n

```
data[0] == 1
```n

Continue execution:
```
if (data[1] != len)
```n

PC = Path Constraint
Symbolic Execution

- data[0] != 1
- data[0] == 1 && data[1] != len
- data[0] == 1 && data[1] == len

Continue execution
Symbolic Execution

```c
void recv(data, len) {
    if (data[0] != 1)
        return
    if (data[1] != len)
        return
    int num = len/data[2]
    ...
}
```

Can data[2] equal zero under the current PC?
Symbolic Execution

\[
data[0] == 1 &&
data[1] == len
\]

```c
void recv(data, len) {
    if (data[0] != 1)
        return
    if (data[1] != len)
        return
    int num = len/data[2]
    ...
}
```

Yes! Bug detected!

Can data[2] equal zero under the current PC?
Implementations

We build upon KLEE
› Works on LLVM bytecode
› Actively maintained

Practical limitations:
› $|\textit{paths}| = 2^{|\textit{if-statement}|}$
› Infinite-length paths
› SMT query complexity
Overview

Symbolic Execution

Handling Crypto

4-way handshake

Results
void recv(data, len) {
    plain = decrypt(data, len)
    if (plain == NULL) return

    if (plain[0] == COMMAND)
        process_command(plain)
    else
        ...
}

Mark data as symbolic
Motivating Example

```c
void recv(data, len) {
    plain = decrypt(data, len)
    if (plain == NULL) return

    if (plain[0] == COMMAND)
        process_command(plain)
    else
        ...
}
```

Mark data as symbolic

Summarize crypto algo. (time consuming)

Analyze crypto algo. (time consuming)

Won’t reach this code!
Efficiently handling decryption?

Decrypted output = fresh symbolic variable
Example

```c
void recv(data, len) {
    plain = decrypt(data, len)
    if (plain == NULL) return

    if (plain[0] == COMMAND) {  // Normal analysis
        process_command(plain)
    } else {
        ...
    }
}
```

Mark data as symbolic

Create fresh symbolic variable

Can now analyze code that parses decrypted data
Other Applications

Handling hash functions
› Output = fresh symbolic variable
› Also works for HMACs (Message Authentication Codes)

Tracking use of crypto primitives?
› Recording relationship between input & output
› Treating fresh variable as information flow taint
Detecting Crypto Misuse

Timing side-channels

- $\forall\ (paths)$: all bytes of MAC in path constraint?
- If not: comparison exits on first difference

Decryption oracles

- Behavior depends on unauth. decrypted data
- Decrypt data is in path constraint, but not in MAC
Overview

Symbolic Execution

Handling Crypto

4-way handshake

Results
The 4-way handshake

Used to connect to any protected Wi-Fi network

Mutual authentication

Negotiates fresh PTK: pairwise transient key
4-way handshake (simplified)

optional 802.1x authentication
4-way handshake (simplified)

optional 802.1x authentication

Msg1(r, ANonce)

Derive PTK

Msg2(r, SNonce)

Derive PTK
4-way handshake (simplified)

PTK = Combine(shared secret, ANonce, SNonce)
4-way handshake (simplified)

optional 802.1x authentication

Msg1(r, ANonce)

Derive PTK

Msg2(r, SNonce)

Msg3(r+1; GTK)

Derive PTK

Msg4(r+1)
4-way handshake (simplified)

optional 802.1x authentication

Msg1(r, ANonce)

Derive PTK

Encrypted with PTK

Msg3(r+1; GTK)

Derive PTK

Msg4(r+1)
4-way handshake (simplified)

optional 802.1x authentication

Msg1(r, ANonce)

Derive PTK

Msg2(r, SNonce)

Msg3(r+1; GTK)

Derive PTK

Msg4(r+1)

Install PTK & GTK

Install PTK
4-way handshake (simplified)

optional 802.1x authentication

Msg1(r, ANonce)

Derive PTK

Msg2(r, SNonce)

Msg3(r+1; GTK)

Derive PTK

Msg4(r+1)

Install PTK & GTK

Install PTK

encrypted data frames can now be exchanged
4-way handshake (simplified)

optional 802.1x authentication

derive PTK

Msg1(r, ANonce)

Authenticated with a MAC

Msg2(r, SNonce)

Msg3(r+1; GTK)

Msg4(r+1)

Install PTK & GTK

Install PTK

encrypted data frames can now be exchanged
We focus on the client

Symbolic execution of

Intel’s iwd daemon  wpa_supplicant  kernel driver

How to get these working under KLEE?
Intel’s iwd

Avoid running full program under KLEE
› Would need to model Wi-Fi stack symbolically

Our approach
› iwd contains unit test for the 4-way handshake
› Reuse initialization code of unit test!
› Symbolically execute only receive function
wpa_supplicant

Unit test uses virtual hardware and runs full AP
› Still need to simulate Wi-Fi stack…

Alternative approach:
› Write unit test that isolates 4-way handshake like iwd
› Then symbolically execute receive function!
› Need to modify code of wpa_supplicant (non-trivial)
Mediatek’s Driver

No unit tests & it’s a Linux driver
› Symbolically executing the Linux kernel?!

Inspired by previous cases
› Write unit test & simulate used kernel functions in userspace
› Verify extracted code is correctly simulated in userspace!
Not all our unit tests are created equally

https://github.com/vanhoefm/woot2018
Overview

Symbolic Execution

Handling Crypto

4-way handshake

Results
Discovered Bugs I

Timing side-channels
› Authentication tag not checked in constant time
› MediaTek and iwd are vulnerable

Denial-of-service in iwd
› Caused by integer underflow
› Leads to huge malloc that fails
Discovered Bugs II

Buffer overflow in MediaTek kernel driver
› Occurs when copying the group key
› May lead to remote code execution

Flawed AES unwrap crypto primitive
› Also in MediaTek’s kernel driver
› Manually discovered
Decryption oracle in wpa_supplicant

Decryption oracle:
› Doesn’t check authenticity of malformed handshake message
› But does decrypt and process data

→ Decrypt group key (GTK) in Message 3 (Msg3)
Decryption oracle in wpa_supplicant II

Msg3': decrypted using RC4, but not authenticated

<table>
<thead>
<tr>
<th>header</th>
<th>221</th>
<th>38</th>
<th>$x_0 \ldots x_{37}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td></td>
<td></td>
<td>GTK</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>header</th>
<th>221</th>
<th>36</th>
<th>$x_0 \ldots x_{35}$</th>
<th>$x_{36}$</th>
<th>$x_{37}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td></td>
<td></td>
<td>GTK'</td>
<td>Type'</td>
<td>Length'</td>
</tr>
</tbody>
</table>

→ Parsing only succeeds if $x_{37}$ is zero
Future work

Short-term
› Efficiently simulate reception of multiple packets
› If 1\textsuperscript{st} packet doesn’t affect state, stop exploring this path

Long-term
› Extract packet formats and state machine
› Verify basic properties of protocol
Conclusion

› Symbolic execution of protocols
› Simple simulation of crypto
› Interesting future work
As a final note...

I wrote a vulnerability scanner that abstracts all the predicates in a binary, traverses the callgraph and generates phormulae to run then with a SMT solver. I found 1 vuln in 3 days with this tool.

He wrote a dumb ass fuzzer and found 5 vulns in 1 day.

Good thing I'm not a n00b like that guy.
Thank you!

Questions?