Cache Template Attacks:
Automating Attacks on Inclusive Last-Level Caches

Daniel Gruss, Raphael Spreitzer, and Stefan Mangard,
IAIK, Graz University of Technology
August 14, 2015
Cache Template Attacks
Cache Template Attacks

- State of the art: cache attacks are powerful
- Problem: manual identification of attack targets
Cache Template Attacks

- State of the art: cache attacks are powerful
- Problem: manual identification of attack targets
- Solution: Cache Template Attacks
- Automatically find any secret-dependent cache access
- Can be used for attacks and to improve software
Cache Template Attacks

- State of the art: cache attacks are powerful
- Problem: manual identification of attack targets
- **Solution: Cache Template Attacks**
- Automatically find any secret-dependent cache access
- Can be used for attacks and to improve software
- Examples:
  - Cache-based keylogger
  - Automatic attacks on crypto algorithms
CPU Caches

- Reduce memory access latency
Memory Access Latency

![Histogram of Memory Access Latency](image-url)
Memory Access Latency
CPU Caches

- Reduce memory access latency
- Last-level cache on modern Intel CPUs:
  - Inclusive (to lower levels)
  - Not in Last-level cache ⇔ not cached
  - Shared (between cores)
Flush+Reload

- Powerful cache attack
- Works on shared binaries/libraries
- Application on crypto algorithms
Flush+Reload

Attacker address space

Cache

Victim address space

Cache is empty
Flush+Reload

Attacker address space

Cache

Victim address space

Victim maps shared library

Daniel Gruss, IAIK, Graz University of Technology
August 14, 2015
Flush+Reload

Attacker address space

Shared Library

Cache

Victim address space

Shared Library

Attacker maps shared library
Flush+Reload

Attacker accesses shared library
Flush+Reload

Attacker address space

Shared Library

Cache

load

Shared Library

Victim address space

Shared Library

Loading into cache...
Flush+Reload

Loading into cache...

Attacker address space

Victim address space

Cache

Shared Library

cached as well

Shared Library

Shared Library
Flush+Reload

Attacker address space

Cache

Victim address space

Shared Library

Attacker measures high latency

Daniel Gruss, IAIK, Graz University of Technology
August 14, 2015
Flush+Reload

Attacker address space

Cache

Victim address space

Attacker flushes shared library ("flush")
Flush+Reload

Attacker address space

Shared Library

Cache

Victim address space

Shared Library

Cache is empty again
Flush+Reload

Victim accesses shared library
Flush+Reload

Attacker address space

Cache

Victim address space

Attacker accesses shared library ("reload")
Flush+Reload

Attacker address space

Cache

already cached

Victim address space

Attacker measures low latency

Shared Library

Shared Library

Shared Library

Daniel Gruss, IAIK, Graz University of Technology
August 14, 2015
Flush+Reload

Attacker address space

Cache

Victim address space

Attacker flushes shared library (“flush”)
Flush+Reload

Attacker address space

Shared Library

Cache

Victim address space

Shared Library

Cache is empty again
Can We Build a Cache-Based Keylogger?
Can We Build a Cache-Based Keylogger?
Can We Build a Cache-Based Keylogger?

```
dgruss@t420dg: ~ cache/sc/generic (git)-[master] % /spy /usr/lib/x86_64-linux-gnu/libgdk-3.so 0x2d0f0
key down [after 3358683 cycles], t= 6684002717 ns
Key up [after 151936 cycles], t= 6994748676 ns
```

```
dgruss@t420dg: ~
```

Daniel Gruss, IAIK, Graz University of Technology
August 14, 2015
Can We Build a Cache-Based Keylogger?

![Image of terminal output showing key press times]

- Key down (after 3358683 cycles), t = 6684002717 ns
- Key up (after 151936 cycles), t = 6994749676 ns
Can We Build a Cache-Based Keylogger?
Can We Build a Cache-Based Keylogger?
Can We Build a Cache-Based Keylogger?

```
File Edit View Search Terminal Help

dgruss@t420dg:~/cache/sc/generic (git)-[master] % /spy/usr/lib/x86_64-linux-gnu/libgdk-3.so 0x2d0f0
key down [after 3358683 cycles], t= 6684602717 ns
key up [after 151936 cycles], t= 6394748676 ns
key down [after 428989 cycles], t= 7893566978 ns
key up [after 132857 cycles], t= 8652147911 ns
key down [after 346702 cycles], t= 8716639893 ns
```

```
dgruss@t420dg:~ % sc
```
Can We Build a Cache-Based Keylogger?
Can We Build a Cache-Based Keylogger?
Can We Build a Cache-Based Keylogger?
Can We Build a Cache-Based Keylogger?
Can We Build a Cache-Based Keylogger?

```
key down (after 3358683 cycles), t= 668402717 ns
key up (after 161936 cycles), t= 6994748676 ns
key down (after 426989 cycles), t= 7893566978 ns
key up (after 132057 cycles), t= 8652147911 ns
key down (after 346702 cycles), t= 8716839893 ns
key up (after 117388 cycles), t= 9861549977 ns
key down (after 311478 cycles), t= 9565911057 ns
key up (after 133242 cycles), t= 9814428341 ns
```

```
dgruss@t420dg ~ % sudo
```
Can We Build a Cache-Based Keylogger?

![Image of terminal output showing key presses and timings]

- key down (after 338683 cycles), t= 6684002717 ns
- key up (after 161936 cycles), t= 6994748676 ns
- key down (after 428989 cycles), t= 7893566978 ns
- key up (after 132057 cycles), t= 9652147911 ns
- key down (after 346702 cycles), t= 8716639893 ns
- key up (after 117388 cycles), t= 8985159977 ns
- key down (after 311478 cycles), t= 9565911057 ns
- key up (after 133242 cycles), t= 9814428341 ns
- key down (after 322659 cycles), t= 10415918868 ns

- `dgruss@t420dg: ~ % sudo`
Can We Build a Cache-Based Keylogger?

![Image of terminal output showing key press timings and sudo command execution]
Can We Build a Cache-Based Keylogger?
Can We Build a Cache-Based Keylogger?
Can We Build a Cache-Based Keylogger?
Can We Build a Cache-Based Keylogger?
Can We Build a Cache-Based Keylogger?
Can We Build a Cache-Based Keylogger?
Can We Build a Cache-Based Keylogger?
Can We Build a Cache-Based Keylogger?
Can We Build a Cache-Based Keylogger?
Can We Build a Cache-Based Keylogger?

![Image of system command output]

```
dgruss@t420dg:~/spy/lib/x86_64-linux-gnu/flb/gdk-3.so 0x20f0
```

```
dgruss@t420dg: sudo zsh
[dudo] password for dgruss: 
```
Can We Build a Cache-Based Keylogger?
Can We Build a Cache-Based Keylogger?
Can We Build a Cache-Based Keylogger?
Can We Build a Cache-Based Keylogger?

```
key up (after 132054 cycles), t= 11473800443 ns
key down (after 251339 cycles), t= 11970896613 ns
key up (after 131647 cycles), t= 12219441040 ns
Key down (after 366253 cycles), t= 12993786546 ns
key up (after 142718 cycles), t= 13173011741 ns
key down (after 1310221 cycles), t= 15681968205 ns
key up (after 104063 cycles), t= 15888962301 ns
key down (after 2668233 cycles), t= 19849303484 ns
key up (after 132367 cycles), t= 29697814864 ns
```

```
dgruss@t420dg:~ % sudo zsh
[sudo] password for dgruss: 
```
Can We Build a Cache-Based Keylogger?
Can We Build a Cache-Based Keylogger?
Can We Build a Cache-Based Keylogger?
Can We Build a Cache-Based Keylogger?
Can We Build a Cache-Based Keylogger?

```
dgruss@t420dg:~/spy/usr/lib/x86_64-linux-gnu/flbgdk-3.so 0x2d0f0
```

```
dgruss@t420dg: sudo zsh
[dgruss@t420dg ~ %] sudo zsh
[sudo] password for dgruss:
```
Can We Build a Cache-Based Keylogger?
Can We Build a Cache-Based Keylogger?

<table>
<thead>
<tr>
<th>Command</th>
<th>Time (cycles)</th>
<th>Time (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>key down</td>
<td>2868233</td>
<td>1964303494</td>
</tr>
<tr>
<td>key up</td>
<td>132367</td>
<td>2007814864</td>
</tr>
<tr>
<td>key down</td>
<td>189008</td>
<td>2945683595</td>
</tr>
<tr>
<td>key up</td>
<td>143324</td>
<td>2971968013</td>
</tr>
<tr>
<td>key down</td>
<td>221377</td>
<td>2113466814</td>
</tr>
<tr>
<td>key up</td>
<td>132327</td>
<td>2138321106</td>
</tr>
<tr>
<td>key down</td>
<td>200026</td>
<td>2175663048</td>
</tr>
<tr>
<td>key up</td>
<td>133371</td>
<td>2295967593</td>
</tr>
<tr>
<td>key down</td>
<td>496999</td>
<td>2362133492</td>
</tr>
</tbody>
</table>

Daniel Gruss, IAIK, Graz University of Technology
August 14, 2015
Can We Build a Cache-Based Keylogger?
Can We Build a Cache-Based Keylogger?
Can We Build a Cache-Based Keylogger?
Can We Build a Cache-Based Keylogger?

![Image of a terminal window showing keydown events and sudo command]

- key down (after 21377 cycles), t= 21134668145 ns
- key up (after 13327 cycles), t= 21383211068 ns
- key down (after 20026 cycles), t= 21756630481 ns
- key up (after 133371 cycles), t= 22696575935 ns
- key down (after 496999 cycles), t= 23621334925 ns
- key up (after 144034 cycles), t= 23290571171 ns
- key down (after 323057 cycles), t= 23892694006 ns
- key up (after 126538 cycles), t= 24140599794 ns
- key down (after 356326 cycles), t= 24894458397 ns

*dgruss@d420dg:~* $ sudo zsh
[sudo] password for *dgruss*: *
Can We Build a Cache-Based Keylogger?

![Image of terminal output showing key values and times]

- **Key Up**: Various values for cycles and time (in nanoseconds)
- **Key Down**: Similar to Key Up, showing different cycle times

Terminal output:
- `dgruss@t420dg:~$ sudo zsh
[sudo] password for dgruss:`
Can We Build a Cache-Based Keylogger?
Can We Build a Cache-Based Keylogger?
Can We Build a Cache-Based Keylogger?
Can We Build a Cache-Based Keylogger?

![Graphical representation of cache-based keylogger output]

- Key up: 133371 cycles, time = 22695675935 ns
- Key down: 496999 cycles, time = 23621334925 ns
- Key up: 144034 cycles, time = 23290571171 ns
- Key down: 323957 cycles, time = 23892894006 ns
- Key up: 128538 cycles, time = 24146599794 ns
- Key down: 356326 cycles, time = 24884458937 ns
- Key up: 117490 cycles, time = 25673573813 ns
- Key down: 429069 cycles, time = 25882450809 ns
- Key up: 144070 cycles, time = 26151774831 ns

- `dgruss@d420dg:~% sudo zsh`
  - [sudo] password for dgruss: [redacted]
Can We Build a Cache-Based Keylogger?
Can We Build a Cache-Based Keylogger?

![Cache-Based Keylogger Example]

```
dgruss@d420dg:~$ sudo zsh
[dgruss] password for dgruss:
```
Can We Build a Cache-Based Keylogger?
Can We Build a Cache-Based Keylogger?
Can We Build a Cache-Based Keylogger?
Can We Build a Cache-Based Keylogger?
Can We Build a Cache-Based Keylogger?

![Image of terminal output showing keypress timing and sudo command execution]

- `key up [after] 128538 cycles, t= 24146596794 ns`
- `key down [after] 356326 cycles, t= 24884458397 ns`
- `key up [after] 117498 cycles, t= 25673573813 ns`
- `key down [after] 429059 cycles, t= 25882450809 ns`
- `key up [after] 144070 cycles, t= 26151774831 ns`
- `key down [after] 888504 cycles, t= 27852155406 ns`
- `key up [after] 131787 cycles, t= 28190614206 ns`
- `key down [after] 488957 cycles, t= 29613173961 ns`
- `key up [after] 136781 cycles, t= 29282495205 ns`

```
dgruss@t420dg:~ % sudo zsh
[sudo] password for dgruss:
root@t420dg:~ #
```

Daniel Gruss, IAIK, Graz University of Technology
August 14, 2015
Can We Build a Cache-Based Keylogger?
Challenges

- How to locate key-dependent memory accesses?
Challenges

- How to locate key-dependent memory accesses?
- It’s complicated:
  - Large binaries and libraries (third-party code)
  - Many libraries (gedit: 60MB)
  - Closed-source / unknown binaries
  - Self-compiled binaries
Challenges

- How to locate key-dependent memory accesses?
- It’s complicated:
  - Large binaries and libraries (third-party code)
  - Many libraries (gedit: 60MB)
  - Closed-source / unknown binaries
  - Self-compiled binaries

- Difficult to find all exploitable addresses
Cache Template Attacks
Cache Template Attacks

Profiling Phase

- Preprocessing step to find exploitable addresses automatically
  - w.r.t. “events” (keystrokes, encryptions, ...)
  - called “Cache Template”
Cache Template Attacks

Profiling Phase

- Preprocessing step to find exploitable addresses automatically
  - w.r.t. “events” (keystrokes, encryptions, ...)
  - called “Cache Template”

Exploitation Phase

- Monitor exploitable addresses
Profiling Phase

Attacker address space

Cache

Victim address space

Cache is empty

Shared 0x0

Shared 0x0
Profiling Phase

Attacker triggers an event
Profiling Phase

Attacker checks one address for cache hits ("Reload")
Profiling Phase

Attacker address space

Cache

Victim address space

Shared 0x0

Shared 0x0

Shared 0x0

Update cache hit ratio (per event and address)
Profiling Phase

Attacker address space

Cache

Victim address space

Attacker flushes shared memory
Profiling Phase

Attacker address space

Victim address space

Cache

Repeat for higher accuracy
Profiling Phase

Attacker address space

Victim address space

Shared 0x0

Cache

Repeat for all events
Profiling Phase

Attacker address space

Victim address space

Cache

Repeat for all events

Shared 0x0

Shared 0x0

C

Daniel Gruss, IAIK, Graz University of Technology
August 14, 2015
Profiling Phase

Attacker address space

Cache

Victim address space

Shared 0x40

Continue with next address
### Profiling Phase

- **Attacker address space**
- **Cache**
- **Victim address space**

Continue with next address
Profiling a Single Event
Profiling a Single Event
Profiling a Single Event
Profiling a Single Event
Profiling a Single Event
Profiling a Single Event
Profiling a Single Event
Profiling a Single Event
Profiling a Single Event
Profiling a Single Event
Profiling a Single Event
Profiling a Single Event
Profiling a Single Event
Profiling a Single Event
Profiling a Single Event
Profiling a Single Event
Profiling a Single Event
Profiling a Single Event
Profiling a Single Event
Profiling a Single Event
Profiling a Single Event
Profiling a Single Event
Profiling a Single Event
Profiling a Single Event
Profiling a Single Event
Profiling a Single Event
Profiling a Single Event
Profiling a Single Event
Profiling a Single Event
Profiling a Single Event
Profiling a Single Event
Profiling a Single Event
Profiling a Single Event

![Image of terminal output]

Daniel Gruss, IAIK, Graz University of Technology
August 14, 2015
Profiling a Single Event
Profiling a Single Event
Profiling a Single Event
Profiling a Single Event
Profiling a Single Event
Profiling a Single Event
Profiling a Single Event
Profiling a Single Event
Profiling a Single Event
Profiling a Single Event
Profiling a Single Event
Profiling a Single Event
Profiling a Single Event
Profiling a Single Event
Profiling a Single Event
Profiling a Single Event
Profiling a Single Event
Proﬁling Phase: 1 Event, 1 Address

Address: 0x7c800

Key n
Profiling Phase: 1 Event, 1 Address

Example: Cache Hit Ratio for \((0x7c800, n)\): 200 / 200
Profiling Phase: All Events, 1 Address

Address: 0x7c800

Key:

- g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z
Profiling Phase: All Events, 1 Address

Example: Cache Hit Ratio for \((0x7c800, u)\): 13 / 200
Profiling Phase: All Events, 1 Address

Distinguish \( n \) from other keys by monitoring \( 0x7c800 \)
Profiling Phase: All Events, All Addresses
Exploitation Phase

- Monitor addresses from Cache Template
Exploitation Phase

- Monitor addresses from Cache Template
- Report to log file / attacker
Exploitation Phase

- Monitor addresses from Cache Template
- Report to log file / attacker
- Manual analysis of log file
  - Find password in keypress log, etc.
Example Attacks
Attack 1: Keystroke Timings

- Spy on keystroke timings on Linux, Windows and OS X
Attack 1: Keystroke Timings

- Spy on keystroke timings on Linux, Windows and OS X
- Sub-microsecond accuracy
Attack 1: Keystroke Timings

- Spy on keystroke timings on Linux, Windows and OS X
- Sub-microsecond accuracy
- Derive text input from timings
Attack 2: Keylogging

- Linux with GTK: monitor keystrokes of specific keys
- Detect groups of keys
- Some keys distinct
Attack 3: Locate AES T-Tables

AES T-Table implementation from OpenSSL 1.0.2
Attack 3: Locate AES T-Tables

AES T-Table implementation from OpenSSL 1.0.2

- Most addresses in two groups:
  - Cache hit ratio 100% (always cache hits)
  - Cache hit ratio 0% (no cache hits)
Attack 3: Locate AES T-Tables

AES T-Table implementation from OpenSSL 1.0.2

- Most addresses in two groups:
  - Cache hit ratio 100% (always cache hits)
  - Cache hit ratio 0% (no cache hits)

- One 4096 byte memory block:
  - Cache hit ratio of 92%
  - Cache hits depend on key value and plaintext value
  - The T-Tables
Attack 4: AES T-Table Template Attack

AES T-Table implementation from OpenSSL 1.0.2

- Known-plaintext attack
- Events: encryption with only one fixed key byte
Attack 4: AES T-Table Template Attack

AES T-Table implementation from OpenSSL 1.0.2

- Known-plaintext attack
- Events: encryption with only one fixed key byte
- Profile each event
Attack 4: AES T-Table Template Attack

AES T-Table implementation from OpenSSL 1.0.2

- Known-plaintext attack
- Events: encryption with only one fixed key byte
- Profile each event
- Exploitation phase:
  - Eliminate key candidates
Attack 4: AES T-Table Template Attack

AES T-Table implementation from OpenSSL 1.0.2

- Known-plaintext attack
- Events: encryption with only one fixed key byte
- Profile each event
- Exploitation phase:
  - Eliminate key candidates
  - Reduction of key space in first-round attack:
    - 64 bits after 16–160 encryptions
Attack 4: AES T-Table Template Attack

AES T-Table implementation from OpenSSL 1.0.2

- Known-plaintext attack
- Events: encryption with only one fixed key byte
- Profile each event
- Exploitation phase:
  - Eliminate key candidates
  - Reduction of key space in first-round attack:
    - 64 bits after 16–160 encryptions
  - State of the art: full key recovery after 30000 encryptions
Attack 4: AES T-Table Template

\[
\begin{align*}
\text{VALUE OF } p_0 & \quad \text{ADDRESS} \\
0 & \quad \begin{array}{c}
\begin{array}{c}
\text{\(k_0 = 0x00\)}
\end{array}
\end{array} \\
255 & \quad \begin{array}{c}
\begin{array}{c}
\text{\(k_0 = 0x55\)}
\end{array}
\end{array}
\end{align*}
\]
Conclusion

- Novel technique to find any cache side-channel leakage
  - Attacks
  - Detect vulnerabilities
Conclusion

- Novel technique to find any cache side-channel leakage
  - Attacks
  - Detect vulnerabilities
- Marks a change of perspective:
Conclusion

- Novel technique to find any cache side-channel leakage
  - Attacks
  - Detect vulnerabilities
- Marks a change of perspective:
  - Large scale analysis of (unknown) binaries
  - Large scale automated attacks
Cache Template Attacks:

Automating Attacks on Inclusive Last-Level Caches

Daniel Gruss, Raphael Spreitzer, and Stefan Mangard,
IAIK, Graz University of Technology

August 14, 2015