Software-based Microarchitectural Attacks

Daniel Gruss
IAIK, Graz University of Technology

June 14, 2017 — PhD Defense
Thesis in numbers
Thesis in numbers

- 32 months
Thesis in numbers

- **32** months

- **10** invited talks and presentations at international venues
Thesis in numbers

- **32** months
- **10** invited talks and presentations at international venues
- **13** publications co-authored (**7** times tier 1)
Thesis in numbers

- **32** months

- **10** invited talks and presentations at international venues

- **13** publications co-authored (**7** times tier 1)

- **6** included in thesis (**3** times tier 1)
Software-based Side-Channel Attacks

- security and privacy rely on secrets (unknown to attackers)
- secrets can leak through side channels
Software-based Side-Channel Attacks

- security and privacy rely on secrets (unknown to attackers)
- secrets can leak through side channels
- software-based $\rightarrow$ no physical access
Plan (from March 2015)

Outlook

- Page Dedup.
- Page Dedup. in JS
- P+P
- P+P in JS
- F+R
- F+R on Memory
- F+R in JS
- CTA
- F+R on ARM
Plan (how it worked out)

- Page Dedup.
- P+P
- P+P in JS
- F+R
- F+R in JS
- F+R on Memory
- F+R on ARM
- CTA
- Page Dedup. in JS

Daniel Gruss, IAIK
June 14, 2017 — PhD Defense
Plan (how it worked out)

- Page Dedup.
- P+P
- P+P in JS
- F+R
- F+R in JS
- F+R on Memory
- F+R on ARM
- CTA

Daniel Gruss, IAIK
June 14, 2017 — PhD Defense
Plan (how it worked out)

- Page Dedup.
- P+P
- P+P in JS
- F+R
- F+R in JS
- F+R on Memory
- F+R on ARM
- CTA
Plan (how it worked out)

- Page Dedup.
- P+P
- P+P in JS
- F+R
- F+R on Memory
- Rowhammer.js
- CTA
- F+R on ARM
Plan (how it worked out)

Page Dedup. → Page Dedup. in JS

P+P → P+P in JS

F+R → F+R on Memory

CTA

ARMageddon

Rowhammer.js
Plan (how it worked out)

- Page Dedup.
- P+P
- F+R
- Page Dedup. in JS
- P+P in JS
- DRAMA
- Rowhammer.js
- CTA
- ARMageddon
Relation of the papers

minimization of requirements

novel side channels automation of attacks
Relation of the papers

minimization of requirements

novel side channels

automation of attacks

CTA
Relation of the papers

minimization of requirements

novel side channels

Dedup.js

CTA

automation of attacks
Relation of the papers

minimization of requirements

Dedup.js

RH.js

CTA

novel side channels

automation of attacks
Relation of the papers

minimization of requirements

Dedup.js
RH.js

novel side channels
automation of attacks

F+F
CTA
The diagram illustrates the relation of the papers, focusing on three main areas:

1. **Minimization of Requirements**
   - Dedup.js
   - RH.js
   - ARMageddon

2. **Novel Side Channels**
   - F+F

3. **Automation of Attacks**
   - CTA

The diagram shows the connections between these areas and the tools and concepts discussed in the papers.
Relation of the papers

minimization of requirements

Dedup.js

RH.js

ARMageddon

F+F

Prefetch

novel side channels

automation of attacks

Daniel Gruss, IAIK
June 14, 2017 — PhD Defense
1. Introduction

2. Background

3. Contributions

4. Conclusion
CPU Caches

- buffer frequently used slow memory for the fast CPU
- every memory reference goes through the cache
- transparent to OS and programs
Memory Access Latency

Number of Accesses

Latency in Cycles

Cached
Not Cached
Memory Access Latency

![Graph showing memory access latency with bars for 'Cached' and 'Not Cached' categories.]
A simple cache
A simple cache

Memory Address

offset

Cache
A simple cache

Memory Address

| Index | Offset |

Cache

$2^n$ cache sets

Daniel Gruss, IAIK
June 14, 2017 — PhD Defense
A simple cache

Memory Address

- Tag
- Index
- Offset

Cache

- Way 1 Tag
- Way 2 Tag
- Way 1 Data
- Way 2 Data

$2^n$ cache sets
Date and Instruction Caches

last-level cache:
- shared
- inclusive
Date and Instruction Caches

last-level cache:
- shared
- inclusive

→ shared memory shared is in cache, across cores!
Date and Instruction Caches

last-level cache:
- shared
- inclusive
→ shared memory shared is in cache, across cores!

function maps addresses to slices (Maurice, Le Scouarnec, et al. 2015)
Flush+Reload

Attacker address space

Cache

Victim address space
Flush+Reload

Attacker address space

Cache

cached

cached

Victim address space
Flush+Reload

Attacker address space

Cache

Victim address space

flushes
Flush+Reload

Attacker address space

Cache

loads data

Victim address space
Flush+Reload

Attacker address space

Cache

Victim address space

reloads data
3. Contributions
   – Cache Template Attacks
   – Page Deduplication Attacks in JavaScript
   – Rowhammer.js
   – Flush+Flush
   – ARMageddon
   – Prefetch Attacks
% sleep 2; ./spy 300 7f05140a4088-7f051417b000 r-xp 0x20000 08:02 268050
/usr/lib/x86_64-linux-gnu/gedit/libgedit.so

shark% ./spy
Cache Template

KEY

ADDRESS

0x7c680
0x7c6c0
0x7c800
0x7c840
0x7c880
0x7c900
0x7c940
0x7c980
0x7c9c0
0x7ca00
0x7cb80
0x7cc40
0x7cc80
0x7ccc0
0x7cd00
3. Contributions

- Cache Template Attacks
- Page Deduplication Attacks in JavaScript
- Rowhammer.js
- Flush+Flush
- ARMageDDon
- Prefetch Attacks
Page Deduplication Attack

Virtual Address Space

Physical Address Space
Page Deduplication Attack

JavaScript

Virtual Address Space

Physical Address Space
Page Deduplication Attack

JavaScript

Virtual Address Space

Physical Address Space
Page Deduplication Attack

JavaScript

Virtual Address Space

Victim

Physical Address Space
Page Deduplication Attack

![Diagram showing JavaScript, Virtual Address Space, and Physical Address Space, with overlapping memory regions marked in green, blue, and red.]

- JavaScript
- Virtual Address Space
  - Victim
- Physical Address Space
Page Deduplication Attack

JavaScript

Virtual Address Space

Victim

Attacker generates a page suspected in victim process

Physical Address Space

Daniel Gruss, IAIK
June 14, 2017 — PhD Defense
Page Deduplication Attack

- JavaScript
- Virtual Address Space
- Victim
- Physical Address Space
Page Deduplication Attack

Attacker waits for deduplication

JavaScript

Virtual Address Space

Victim

Physical Address Space
Page Deduplication Attack

```javascript
t = time();
p[0] = p[0];
\Delta = time() - t;
```
Page Deduplication Attack

JavaScript

Virtual Address Space

Time

0

4

\( \Delta \) in \( \mu \)s

Victim

Physical Address Space

\( \downarrow \) measure \( \Delta \)
Page Deduplication Attack

![Diagram of page deduplication attack]

- **JavaScript**
- **Virtual Address Space**
- **Victim**
- **Physical Address Space**

\[ \Delta \text{ in } \mu s \]

- Time

Daniel Gruss, IAIK
June 14, 2017 — PhD Defense
Page Deduplication Attack

![Diagram showing JavaScript, Virtual Address Space, and Physical Address Space with time measurement and color coding.]
Page Deduplication Attack
Page Deduplication Attack

JavaScript

Virtual Address Space

Victim

△ in µs

Time

Physical Address Space

measure △
Page Deduplication Attack

JavaScript

Virtual Address Space

Victim

Time

\[ \Delta \text{ in } \mu s \]

\[ 0 \]

\[ 4 \]

measure \( \Delta \)

Physical Address Space
Page Deduplication Attack

Virtual Address Space

Victim

Physical Address Space

JavaScript

\[ \Delta \text{ in } \mu s \]

Time

measure \( \downarrow \)
Page Deduplication Attack

![Diagram showing the relationship between JavaScript, Virtual Address Space, Physical Address Space, and Victim.](image)

- Δ in µs
- Time
- Physical Address Space
- Virtual Address Space
- Victim

Daniel Gruss, IAIK
June 14, 2017 — PhD Defense
Page Deduplication Attack

Virtual Address Space

Physical Address Space

Victim

JavaScript

\[ \Delta \text{ in } \mu s \]

Time

\(0\) \(4\)

\(\downarrow\) measure

Daniel Gruss, IAIK
June 14, 2017 — PhD Defense
Page Deduplication Attack

JavaScript

Virtual Address Space

Victim

Physical Address Space

\[ \Delta \text{in } \mu s \]

Time

Daniel Gruss, IAIK
June 14, 2017 — PhD Defense
Page Deduplication Attack

JavaScript

Virtual Address Space

Victim

$\Delta$ in µs

Time

Physical Address Space

measure $\Delta$
Page Deduplication Attack

![Diagram showing JavaScript, Virtual Address Space, and Physical Address Space with measurements.]
Page Deduplication Attack

Virtual Address Space

Physical Address Space

\[ \Delta \text{ in } \mu s \]

0

4

Time

Victim

measure \( \Delta \)

JavaScript
Page Deduplication Attack

Virtual Address Space

Physical Address Space

Victim

JavaScript

Time

$\Delta$ in $\mu s$

Measure $\downarrow$
Page Deduplication Attack

JavaScript

Virtual Address Space

\[ \Delta \text{ in \mu s} \]

Victim

\[ \Delta \]

Time

0

4

measure

Physical Address Space

Daniel Gruss, IAIK
June 14, 2017 — PhD Defense
Page Deduplication Attack

Virtual Address Space

Physical Address Space

JavaScript

Victim

$\Delta$ in $\mu$s

$\Delta$

measure

Time

0

4

Daniel Gruss, IAIK
June 14, 2017 — PhD Defense
Page Deduplication Attack

Virtual Address Space

Time

µs

JavaScript

Physical Address Space

Victim

\[ \Delta \]
Page Deduplication Attack

JavaScript  Virtual Address Space  Victim

Physical Address Space

$\Delta$ in $\mu$s

Time

Daniel Gruss, IAIK
June 14, 2017 — PhD Defense
Page Deduplication Attack

![Diagram of page deduplication attack, showing JavaScript, Virtual Address Space, and Physical Address Space with measures in µs]
Page Deduplication Attack

![Diagram showing JavaScript, Virtual Address Space, Time, and Victim.]

\[ \Delta \text{ in } \mu s \]

\[ \neq \]
Page Deduplication Attack

JavaScript

Virtual Address Space

Physical Address Space

Victim

$\Delta$ in $\mu$s

Time

$\neq$

measure

Daniel Gruss, IAIK
June 14, 2017 — PhD Defense
Page Deduplication Attack

\[ \Delta \text{ in } \mu s \]

\[ 0 \]

\[ 4 \]

Virtual Address Space

JavaScript

Victim

Time

\[ \neq \]

Physical Address Space

\[ \text{measure} \]

Daniel Gruss, IAIK
June 14, 2017 — PhD Defense
Page Deduplication Attack

JavaScript

Virtual Address Space

Victim

\[ \Delta \text{ in } \mu \text{s} \]

Time

\[ \neq \]

Physical Address Space
Page Deduplication Attack

Virtual Address Space

Physical Address Space

Victim

JavaScript

Δ in µs

Time

0

4

measure

≠
Page Deduplication Attack

\[ \Delta \text{ in } \mu s \]

\[ \text{Time} \]

JavaScript

Virtual Address Space

Victim

Physical Address Space
Page Deduplication Attack

![Diagram showing JavaScript, Virtual Address Space, Physical Address Space, and Victim areas.](image-url)
Page Deduplication Attack

JavaScript

Virtual Address Space

Victim

Time

Physical Address Space

∆ in µs

0

4
Page Deduplication Attack

write and measure $\Delta$
Page Deduplication Attack

JavaScript

Virtual Address Space

Victim

Physical Address Space

write and measure $\Delta$

$\Delta$ in $\mu$s

Time

copy
Page Deduplication Attack

![Diagram showing JavaScript, Virtual Address Space, and Physical Address Space with write operation and time differences.](image-url)
Page Deduplication Attack

Attacker learns that another process had an identical page

Physical Address Space

Virtual Address Space

\[ \Delta \text{ in } \mu s \]

Time

0

4

Victim

JavaScript

Page Deduplication Attack

Attacker learns that another process had an identical page

Time
Δ in µs

0
4

Virtual Address Space

JavaScript

Victim

Physical Address Space
Page Deduplication Attack

Attacker learns that another process had an identical page
Page Deduplication Attack

Attacker learns that another process had an identical page.

Virtual Address Space

Physical Address Space

JavaScript

Victim
Page Deduplication Attack

Attacker learns that another process had an identical page.
Page Deduplication Attack

Attacker learns that another process had an identical page
Page Deduplication Attack

Attacker learns that another process had an identical page

JavaScript

Virtual Address Space

Victim

Physical Address Space

\[ \Delta \text{ in } \mu s \]
Page Deduplication Attack

Inspector learns that another process had an identical page.
Page Deduplication Attack

Attacker learns that another process had an identical page.
Page Deduplication Attack

Attacker learns that another process had an identical page

Daniel Gruss, IAIK
June 14, 2017 — PhD Defense
Page Deduplication Attack

Attacker learns that another process had an identical page

JavaScript

Virtual Address Space

Victim

Physical Address Space
Page Deduplication Attack

Attacker learns that another process had an identical page
Page Deduplication Attack

Attacker learns that another process had an identical page
Page Deduplication Attack

Attacker learns that another process had an identical page

Virtual Address Space

Physical Address Space

JavaScript

Victim

$\Delta \text{ in } \mu s$

Time

0

4

Daniel Gruss, IAIK
June 14, 2017 — PhD Defense
Page Deduplication Attack

Attacker learns that another process had an identical page
Page Deduplication Attack

Attacker learns that another process had an identical page

JavaScript

Virtual Address Space

Victim

Physical Address Space

Δ in µs

Time

0

4
Our Attack

First page deduplication attack which

- detects CSS files/images on websites,
- runs in JavaScript (no rdtsc, no addresses),
- runs on KVM, Windows 8.1 and Android.
Detect Image (JavaScript, Cross-VM, KVM)

Image not loaded

Image loaded

Nanoseconds

Page
3. Contributions

- Cache Template Attacks
- Page Deduplication Attacks in JavaScript
- Rowhammer.js
  - Flush+Flush
  - ARMageddon
  - Prefetch Attacks
Rowhammer

- Rowhammer: DRAM bug that causes bit flips (Kim et al. 2014)
- Bug used in security exploits (Seaborn 2015)
- Only non-cached accesses reach DRAM
- Very similar to Flush+Reload
Rowhammer (with clflush)
Rowhammer (with clflush)

cache set 1

clflush

DRAM bank

cache set 2

clflush
Rowhammer (with clflush)

- cache set 1
- cache set 2
- DRAM bank
Rowhammer (with clflush)

DRAM bank

cache set 1

cache set 2
Rowhammer (with clflush)

cache set 1

reload

DRAM bank

cache set 2
Rowhammer (with clflush)
Rowhammer (with clflush)

- cache set 1
- cache set 2
- DRAM bank

clflush
Rowhammer (with clflush)

DRAM bank

reload

cache set 1

reload

cache set 2
Rowhammer (with clflush)

cache set 1

clflush

DRAM bank

clflush

cache set 2
Rowhammer (with clflush)
Rowhammer (with clflush)
Rowhammer (with clflush)

- cache set 1
- cache set 2
- DRAM bank

Daniel Gruss, IAIK
June 14, 2017 — PhD Defense
Rowhammer (with clflush)

Daniel Gruss, IAIK
June 14, 2017 — PhD Defense
Rowhammer (with clflush)

cache set 1

cache set 2

DRAM bank

reload

bit flip!
Rowhammer without `clflush`

- Cache set 1
- Cache set 2

- DRAM bank
Rowhammer without clflush

cache set 1

load

load

cache set 2

DRAM bank
Rowhammer without \texttt{clflush}

- Cache set 1
- Cache set 2
- DRAM bank
Rowhammer without clflush

Cache set 1

Cache set 2

DRAM bank

load

load
Rowhammer without `clflush`

Cache set 1

Cache set 2

DRAM bank
Rowhammer without clflush

cache set 1

load

cache set 2

load
Rowhammer without clflush

cache set 1

cache set 2

DRAM bank

load

load
Rowhammer without clflush

- Cache set 1
- Cache set 2
- DRAM bank
Rowhammer without \texttt{clflush}

cache set 1

cache set 2

DRAM bank

Daniel Gruss, IAIK
June 14, 2017 — PhD Defense
Rowhammer without clflush

Cache set 1

Cache set 2

DRAM bank

Reload

Reload
Rowhammer without `clflush`

Repeat!
Rowhammer without clflush

Cache set 1

Cache set 2

DRAM bank

reload

wait for it...
Rowhammer without clflush

Drum bank

cache set 1

cache set 2

bit flip!
Rowhammer without clflush

Challenges:

1. How to get accurate timing (in JS)?
2. How to get physical addresses (in JS)?
3. Which physical addresses to access?
4. In which order to access them?
Rowhammer without clflush

Challenges:

1. How to get accurate timing (in JS)? → easy
2. How to get physical addresses (in JS)? → easy
3. Which physical addresses to access? → already solved
4. In which order to access them? → our contribution
Replacement policy on older CPUs

“LRU eviction” memory accesses

cache set

Daniel Gruss, IAIK
June 14, 2017 — PhD Defense
Replacement policy on older CPUs

“LRU eviction” memory accesses

- LRU replacement policy: oldest entry first
Replacement policy on older CPUs

“LRU eviction” memory accesses

- LRU replacement policy: oldest entry first
- Timestamps for every cache line
Replacement policy on older CPUs

“LRU eviction” memory accesses

- LRU replacement policy: oldest entry first
- timestamps for every cache line
- access updates timestamp
Replacement policy on older CPUs

“LRU eviction” memory accesses

- LRU replacement policy: oldest entry first
- timestamps for every cache line
- access updates timestamp
Replacement policy on older CPUs

“LRU eviction” memory accesses

- LRU replacement policy: oldest entry first
- timestamps for every cache line
- access updates timestamp
Replacement policy on older CPUs

“LRU eviction” memory accesses

- LRU replacement policy: oldest entry first
- timestamps for every cache line
- access updates timestamp
Replacement policy on older CPUs

“LRU eviction” memory accesses

- LRU replacement policy: oldest entry first
- timestamps for every cache line
- access updates timestamp
Replacement policy on older CPUs

“LRU eviction” memory accesses

- LRU replacement policy: oldest entry first
- timestamps for every cache line
- access updates timestamp
Replacement policy on older CPUs

“LRU eviction” memory accesses

- LRU replacement policy: oldest entry first
- timestamps for every cache line
- access updates timestamp
Replacement policy on older CPUs

“LRU eviction” memory accesses

- LRU replacement policy: oldest entry first
- timestamps for every cache line
- access updates timestamp
Replacement policy on recent CPUs

“LRU eviction” memory accesses

- no LRU replacement
Replacement policy on recent CPUs

“LRU eviction” memory accesses

- no LRU replacement
Replacement policy on recent CPUs

“LRU eviction” memory accesses

- no LRU replacement

load

10 5 8 9 7 6 3 4

cache set

no LRU replacement
Replacement policy on recent CPUs

“LRU eviction” memory accesses

- no LRU replacement
Replacement policy on recent CPUs

“LRU eviction” memory accesses

- no LRU replacement
Replacement policy on recent CPUs

“LRU eviction” memory accesses

- no LRU replacement
Replacement policy on recent CPUs

“LRU eviction” memory accesses

- no LRU replacement
Replacement policy on recent CPUs

“LRU eviction” memory accesses

- no LRU replacement
Replacement policy on recent CPUs

“LRU eviction” memory accesses

- no LRU replacement
Replacement policy on recent CPUs

“LRU eviction” memory accesses

- no LRU replacement
- only 75% success rate on Haswell
Replacement policy on recent CPUs

“LRU eviction” memory accesses

- no LRU replacement
- only 75% success rate on Haswell
- more accesses → higher success rate, but too slow
Cache eviction strategy: Notation (1)

Write eviction strategies as: $\mathcal{P}-C-D-L-S$

\[
\begin{align*}
&\text{for (s = 0; s <= } S - D \text{; s += L)} \\
&\quad \text{for (c = 0; c <= } C \text{; c += 1)} \nonumber \\
&\quad \quad \text{for (d = 0; d <= } D \text{; d += 1)} \nonumber \\
&\quad \quad \quad \quad \text{*a[s+d];} \nonumber
\end{align*}
\]
Cache eviction strategy: Notation (1)

Write eviction strategies as: $P-C-D-L-S$

$S$: total number of different addresses (= set size)

```
for (s = 0; s <= S - D; s += L)
  for (c = 0; c <= C; c += 1)
    for (d = 0; d <= D; d += 1)
      *a[s+d];
```
Cache eviction strategy: Notation (1)

Write eviction strategies as: $P$-$C$-$D$-$L$-$S$

$S$: total number of different addresses (= set size)

$D$: different addresses per inner access loop

$$
\begin{align*}
\text{for (s = 0; s }\leq\text{ S - D; s += L )} \\
\text{for (c = 0; c }\leq\text{ C; c += 1) } \\
\text{for (d = 0; d }\leq\text{ D; d += 1) } \\
*\text{a}[\text{s+d}] \\
\end{align*}
$$
Cache eviction strategy: Notation (1)

Write eviction strategies as: \( P-C-D-L-S \)

- \( S \): total number of different addresses (= set size)
- \( D \): different addresses per inner access loop
- \( L \): step size of the inner access loop

```
for (s = 0; s <= S - D; s += L)
    for (c = 0; c <= C; c += 1)
        for (d = 0; d <= D; d += 1)
            *a[s+d];
```
Cache eviction strategy: Notation (1)

Write eviction strategies as: $\mathcal{P}$-$C$-$D$-$L$-$S$

$S$: total number of different addresses (= set size)

$D$: different addresses per inner access loop

$L$: step size of the inner access loop

$C$: number of repetitions of the inner access loop

for (s = 0; s <= S - D; s += L)
  for (c = 0; c <= C; c += 1)
    for (d = 0; d <= D; d += 1)
      *a[s+d];
Cache eviction strategy: Notation (2)

for (s = 0; s <= \( S - D \); s += \( L \))
    for (c = 1; c <= \( C \); c += 1)
        for (d = 1; d <= \( D \); d += 1)
            \( a[s+d] \);
Cache eviction strategy: Notation (2)

```
for (s = 0; s <= S - D; s += L)
    for (c = 1; c <= C; c += 1)
        for (d = 1; d <= D; d += 1)
            *a[s+d];
```

- \( P - 2 - 2 - 1 - 4 \) → 1, 2, 1, 2, 2, 3, 2, 3, 3, 4, 3, 4
Cache eviction strategy: Notation (2)

for (s = 0; s <= S - D; s += L)
   for (c = 1; c <= C; c += 1)
      for (d = 1; d <= D; d += 1)
         *a[s+d];

- \( P-2-2-1-4 \rightarrow 1, 2, 1, 2, 2, 3, 2, 3, 3, 4, 3, 4 \)

- \( P-1-1-1-4 \rightarrow 1, 2, 3, 4 \rightarrow \text{LRU eviction with set size 4} \)
Cache eviction strategies: Evaluation

We evaluated more than 10000 strategies...

<table>
<thead>
<tr>
<th>strategy</th>
<th># accesses</th>
<th>eviction rate</th>
<th>loop time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P-1-1-1-17</td>
<td>17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P-1-1-1-20</td>
<td>20</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Executed in a loop, on a Haswell with a 16-way last-level cache
Cache eviction strategies: Evaluation

We evaluated more than 10000 strategies...

<table>
<thead>
<tr>
<th>strategy</th>
<th># accesses</th>
<th>eviction rate</th>
<th>loop time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P-1-1-1-17</td>
<td>17</td>
<td>74.46%</td>
<td>✗</td>
</tr>
<tr>
<td>P-1-1-1-20</td>
<td>20</td>
<td>99.82%</td>
<td>✓</td>
</tr>
</tbody>
</table>

Executed in a loop, on a Haswell with a 16-way last-level cache
Cache eviction strategies: Evaluation

We evaluated more than 10000 strategies...

<table>
<thead>
<tr>
<th>strategy</th>
<th># accesses</th>
<th>eviction rate</th>
<th>loop time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_{-1-1-1-17}$</td>
<td>17</td>
<td>74.46%</td>
<td>307 ns</td>
</tr>
<tr>
<td>$P_{-1-1-1-20}$</td>
<td>20</td>
<td>99.82%</td>
<td>934 ns</td>
</tr>
</tbody>
</table>

Executed in a loop, on a Haswell with a 16-way last-level cache
Cache eviction strategies: Evaluation

We evaluated more than 10000 strategies...

<table>
<thead>
<tr>
<th>strategy</th>
<th># accesses</th>
<th>eviction rate</th>
<th>loop time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{P}$-1-1-1-17</td>
<td>17</td>
<td>74.46% $\times$</td>
<td>307 ns $\checkmark$</td>
</tr>
<tr>
<td>$\mathcal{P}$-1-1-1-20</td>
<td>20</td>
<td>99.82% $\checkmark$</td>
<td>934 ns $\times$</td>
</tr>
<tr>
<td>$\mathcal{P}$-2-1-1-17</td>
<td>34</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Executed in a loop, on a Haswell with a 16-way last-level cache
Cache eviction strategies: Evaluation

We evaluated more than 10000 strategies...

<table>
<thead>
<tr>
<th>strategy</th>
<th># accesses</th>
<th>eviction rate</th>
<th>loop time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{P}$-1-1-1-17</td>
<td>17</td>
<td>74.46% $\times$</td>
<td>307 ns $\checkmark$</td>
</tr>
<tr>
<td>$\mathcal{P}$-1-1-1-20</td>
<td>20</td>
<td>99.82% $\checkmark$</td>
<td>934 ns $\times$</td>
</tr>
<tr>
<td>$\mathcal{P}$-2-1-1-17</td>
<td>34</td>
<td>99.86% $\checkmark$</td>
<td></td>
</tr>
</tbody>
</table>

Executed in a loop, on a Haswell with a 16-way last-level cache
Cache eviction strategies: Evaluation

We evaluated more than 10000 strategies...

<table>
<thead>
<tr>
<th>strategy</th>
<th># accesses</th>
<th>eviction rate</th>
<th>loop time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{P}$-1-1-1-17</td>
<td>17</td>
<td>74.46% X</td>
<td>307 ns ✓</td>
</tr>
<tr>
<td>$\mathcal{P}$-1-1-1-20</td>
<td>20</td>
<td>99.82% ✓</td>
<td>934 ns X</td>
</tr>
<tr>
<td>$\mathcal{P}$-2-1-1-17</td>
<td>34</td>
<td>99.86% ✓</td>
<td>191 ns ✓</td>
</tr>
</tbody>
</table>

Executed in a loop, on a Haswell with a 16-way last-level cache
Cache eviction strategies: Evaluation

We evaluated more than 10000 strategies...

<table>
<thead>
<tr>
<th>strategy</th>
<th># accesses</th>
<th>eviction rate</th>
<th>loop time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{P}-1-1-1-17$</td>
<td>17</td>
<td>74.46% ✓</td>
<td>307 ns ✓</td>
</tr>
<tr>
<td>$\mathcal{P}-1-1-1-20$</td>
<td>20</td>
<td>99.82% ✓</td>
<td>934 ns x</td>
</tr>
<tr>
<td>$\mathcal{P}-2-1-1-17$</td>
<td>34</td>
<td>99.86% ✓</td>
<td>191 ns ✓</td>
</tr>
<tr>
<td>$\mathcal{P}-2-2-1-17$</td>
<td>64</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Executed in a loop, on a Haswell with a 16-way last-level cache
Cache eviction strategies: Evaluation

We evaluated more than 10000 strategies...

<table>
<thead>
<tr>
<th>strategy</th>
<th># accesses</th>
<th>eviction rate</th>
<th>loop time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{P}-1-1-1-17$</td>
<td>17</td>
<td>74.46% $\times$</td>
<td>307 ns $\checkmark$</td>
</tr>
<tr>
<td>$\mathcal{P}-1-1-1-20$</td>
<td>20</td>
<td>99.82% $\checkmark$</td>
<td>934 ns $\times$</td>
</tr>
<tr>
<td>$\mathcal{P}-2-1-1-17$</td>
<td>34</td>
<td>99.86% $\checkmark$</td>
<td>191 ns $\checkmark$</td>
</tr>
<tr>
<td>$\mathcal{P}-2-2-1-17$</td>
<td>64</td>
<td>99.98% $\checkmark$</td>
<td></td>
</tr>
</tbody>
</table>

Executed in a loop, on a Haswell with a 16-way last-level cache
Cache eviction strategies: Evaluation

We evaluated more than 10000 strategies...

<table>
<thead>
<tr>
<th>strategy</th>
<th># accesses</th>
<th>eviction rate</th>
<th>loop time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P-1-1-1-17</td>
<td>17</td>
<td>74.46%</td>
<td>307 ns</td>
</tr>
<tr>
<td>P-1-1-1-20</td>
<td>20</td>
<td>99.82%</td>
<td>934 ns</td>
</tr>
<tr>
<td>P-2-1-1-17</td>
<td>34</td>
<td>99.86%</td>
<td>191 ns</td>
</tr>
<tr>
<td>P-2-2-1-17</td>
<td>64</td>
<td>99.98%</td>
<td>180 ns</td>
</tr>
</tbody>
</table>

Executed in a loop, on a Haswell with a 16-way last-level cache
Cache eviction strategies: Evaluation

We evaluated more than 10000 strategies...

<table>
<thead>
<tr>
<th>strategy</th>
<th># accesses</th>
<th>eviction rate</th>
<th>loop time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P-1-1-1-17</td>
<td>17</td>
<td>74.46%</td>
<td>307 ns</td>
</tr>
<tr>
<td>P-1-1-1-20</td>
<td>20</td>
<td>99.82%</td>
<td>934 ns</td>
</tr>
<tr>
<td>P-2-1-1-17</td>
<td>34</td>
<td>99.86%</td>
<td>191 ns</td>
</tr>
<tr>
<td>P-2-2-1-17</td>
<td>64</td>
<td>99.98%</td>
<td>180 ns</td>
</tr>
</tbody>
</table>

→ more accesses, smaller execution time? Executed in a loop, on a Haswell with a 16-way last-level cache
Cache eviction strategies (illustration)

$P-1-1-1-17$ (17 accesses, 307ns)

$P-2-1-1-34$ (34 accesses, 191ns)
Cache eviction strategies (illustration)

$P-1-1-1-17$ (17 accesses, 307ns)

$P-2-1-1-34$ (34 accesses, 191ns)

Time in ns
Cache eviction strategies (illustration)

$\mathcal{P}$-1-1-1-17 (17 accesses, 307ns)

$\mathcal{P}$-2-1-1-34 (34 accesses, 191ns)

Time in ns
Cache eviction strategies (illustration)

\( \mathcal{P}-1-1-1-17 \) (17 accesses, 307ns)

\( \mathcal{P}-2-1-1-34 \) (34 accesses, 191ns)

Time in ns
Cache eviction strategies (illustration)

$P_{-1-1-1-17}$ (17 accesses, 307ns)

$P_{-2-1-1-34}$ (34 accesses, 191ns)

Time in ns
Cache eviction strategies (illustration)

\( \mathcal{P}-1-1-1-17 \) (17 accesses, 307ns)

\( \mathcal{P}-2-1-1-34 \) (34 accesses, 191ns)

Time in ns
Cache eviction strategies (illustration)

\[ \mathcal{P}-1-1-1-17 \] (17 accesses, 307ns)

\[ \mathcal{P}-2-1-1-34 \] (34 accesses, 191ns)

Time in ns
Cache eviction strategies (illustration)

$P-1-1-1-17$ (17 accesses, 307ns)

$P-2-1-1-34$ (34 accesses, 191ns)

Time in ns
Cache eviction strategies (illustration)

$P-1-1-1-17$ (17 accesses, 307ns)

$P-2-1-1-34$ (34 accesses, 191ns)

Time in ns
Cache eviction strategies (illustration)

\[ P-1-1-1-17 \] (17 accesses, 307ns)

\[ P-2-1-1-34 \] (34 accesses, 191ns)

Time in ns
Cache eviction strategies (illustration)

$P_{-1-1-1-17}$ (17 accesses, 307ns)

$P_{-2-1-1-34}$ (34 accesses, 191ns)

Time in ns
Cache eviction strategies (illustration)

\( P-1-1-1-17 \) (17 accesses, 307ns)

\( P-2-1-1-34 \) (34 accesses, 191ns)

Time in ns
Cache eviction strategies (illustration)

$\mathcal{P}-1-1-1-17$ (17 accesses, 307ns)

$\mathcal{P}-2-1-1-34$ (34 accesses, 191ns)

Time in ns
Cache eviction strategies (illustration)

$P-1-1-1-17$ (17 accesses, 307ns)

$P-2-1-1-34$ (34 accesses, 191ns)

Time in ns
Cache eviction strategies (illustration)

\(P-1-1-1-17\) (17 accesses, 307ns)

\(P-2-1-1-34\) (34 accesses, 191ns)

Time in ns
Cache eviction strategies (illustration)

$\mathcal{P}-1-1-1-17$ (17 accesses, 307ns)

$\mathcal{P}-2-1-1-34$ (34 accesses, 191ns)

Time in ns
Cache eviction strategies (illustration)

\[ P-1-1-1-17 \] (17 accesses, 307ns)

\[ P-2-1-1-34 \] (34 accesses, 191ns)

Time in ns
Cache eviction strategies (illustration)

$\mathcal{P}-1-1-1-17$ (17 accesses, 307ns)

$\mathcal{P}-2-1-1-34$ (34 accesses, 191ns)

Time in ns
Cache eviction strategies (illustration)

$P-1-1-1-17$ (17 accesses, 307ns)

$P-2-1-1-34$ (34 accesses, 191ns)

Time in ns
Cache eviction strategies (illustration)

\( P-1-1-1-17 \) (17 accesses, 307ns)

\( P-2-1-1-34 \) (34 accesses, 191ns)
Cache eviction strategies (illustration)

$P-1-1-1-17$ (17 accesses, 307ns)

$P-2-1-1-34$ (34 accesses, 191ns)

Time in ns
Cache eviction strategies (illustration)

\[ \mathcal{P}-1-1-1-17 \] (17 accesses, 307ns)

\[ \mathcal{P}-2-1-1-34 \] (34 accesses, 191ns)
Cache eviction strategies (illustration)

\( P-1-1-1-17 \) (17 accesses, 307ns)

\( P-2-1-1-34 \) (34 accesses, 191ns)

Time in ns
Cache eviction strategies (illustration)

\( \mathcal{P}-1-1-1-17 \) (17 accesses, 307ns)

\( \mathcal{P}-2-1-1-34 \) (34 accesses, 191ns)

Time in ns
Cache eviction strategies (illustration)

\( \mathcal{P}-1-1-1-17 \) (17 accesses, 307ns)

\( \mathcal{P}-2-1-1-34 \) (34 accesses, 191ns)

Time in ns
Cache eviction strategies (illustration)

\( P-1-1-1-17 \) (17 accesses, 307ns)

\( P-2-1-1-34 \) (34 accesses, 191ns)
Cache eviction strategies (illustration)

\( P-1-1-1-17 \) (17 accesses, 307ns)

\( P-2-1-1-34 \) (34 accesses, 191ns)

Time in ns
Cache eviction strategies (illustration)

\( P-1-1-1-17 \) (17 accesses, 307ns)

\( P-2-1-1-34 \) (34 accesses, 191ns)

Time in ns
Cache eviction strategies (illustration)

\[ P-1-1-1-17 \ (17 \text{ accesses, } 307\text{ns}) \]

\[ P-2-1-1-34 \ (34 \text{ accesses, } 191\text{ns}) \]

Time in ns
Cache eviction strategies (illustration)

\( P-1-1-1-17 \) (17 accesses, 307ns)

\( P-2-1-1-34 \) (34 accesses, 191ns)

Time in ns
Cache eviction strategies (illustration)

\[ P-1-1-1-17 \] (17 accesses, 307ns)

\[ P-2-1-1-34 \] (34 accesses, 191ns)

Time in ns
Cache eviction strategies (illustration)

$\mathcal{P}-1-1-1-17$ (17 accesses, 307ns)

$\mathcal{P}-2-1-1-34$ (34 accesses, 191ns)
Cache eviction strategies (illustration)

\( p-1-1-1-17 \) (17 accesses, 307ns)

\( p-2-1-1-34 \) (34 accesses, 191ns)

Time in ns
Cache eviction strategies (illustration)

\( P-1-1-1-17 \) (17 accesses, 307ns)

\( P-2-1-1-34 \) (34 accesses, 191ns)
Cache eviction strategies (illustration)

\( P-1-1-1-17 \) (17 accesses, 307\,ns)

\( P-2-1-1-34 \) (34 accesses, 191\,ns)

Time in ns
Cache eviction strategies (illustration)

$P-1-1-1-17$ (17 accesses, 307ns)

$P-2-1-1-34$ (34 accesses, 191ns)

Time in ns
Cache eviction strategies (illustration)

$\mathcal{P}-1-1-1-17$ (17 accesses, 307ns)

$\mathcal{P}-2-1-1-34$ (34 accesses, 191ns)
Cache eviction strategies (illustration)

$P-1-1-1-17$ (17 accesses, 307ns)

$P-2-1-1-34$ (34 accesses, 191ns)
Cache eviction strategies (illustration)

$\mathcal{P}-1-1-1-17$ (17 accesses, 307ns)

$\mathcal{P}-2-1-1-34$ (34 accesses, 191ns)

Time in ns
Cache eviction strategies (illustration)

\[ P-1-1-1-17 \] (17 accesses, 307ns)

\[ P-2-1-1-34 \] (34 accesses, 191ns)

Time in ns
Cache eviction strategies (illustration)

$\mathcal{P}-1-1-1-17$ (17 accesses, 307ns)

$\mathcal{P}-2-1-1-34$ (34 accesses, 191ns)
Cache eviction strategies (illustration)

\( P-1-1-1-17 \) (17 accesses, 307ns)

\( P-2-1-1-34 \) (34 accesses, 191ns)

Time in ns
Cache eviction strategies (illustration)

\[ \mathcal{P}-1-1-1-17 \ (17 \text{ accesses, } 307\text{ns}) \]

\[ \mathcal{P}-2-1-1-34 \ (34 \text{ accesses, } 191\text{ns}) \]

Time in ns
Evaluation on Haswell

Figure: Number of bit flips within 15 minutes.
3. Contributions

- Cache Template Attacks
- Page Deduplication Attacks in JavaScript
- Rowhammer.js
- Flush+Flush
- ARMageddon
- Prefetch Attacks
Flush+Flush: Motivation

- cache attacks $\rightarrow$ many cache misses
- detect via performance counters
  $\rightarrow$ good idea, but not good enough
Flush+Reload

Attacker address space

Cache

Victim address space
Flush+Reload

Attacker address space

| cached |

Cache

| cached |

Victim address space

| cached |
Flush+Reload

step 1: attacker flushes the shared line
Flush+Reload

**step 1**: attacker flushes the shared line

**step 2**: victim loads data while performing encryption
Flush+Reload

**step 1**: attacker flushes the shared line
**step 2**: victim loads data while performing encryption
**step 3**: attacker reloads data → fast access if the victim loaded the line
Flush+Flush

step 0: attacker maps shared library $\rightarrow$ shared memory, shared in cache
Flush+Flush

**step 0:** attacker maps shared library → shared memory, shared in cache
Flush+Flush

**step 0:** attacker maps shared library → shared memory, shared in cache

**step 1:** attacker flushes the shared line
Flush+Flush

**step 0**: attacker maps shared library $\rightarrow$ shared memory, shared in cache

**step 1**: attacker flushes the shared line

**step 2**: victim loads data while performing encryption
Flush+Flush

**Step 0**: Attacker maps shared library $\rightarrow$ shared memory, shared in cache

**Step 1**: Attacker flushes the shared line

**Step 2**: Victim loads data while performing encryption

**Step 3**: Attacker flushes data $\rightarrow$ high execution time if the victim loaded the line
Flush+Flush: Conclusion

- 496 KB/s covert channel
- same side channel targets as Flush+Reload
- attacker causes no cache misses
  → fast
  → stealthy
3. Contributions

– Cache Template Attacks
– Page Deduplication Attacks in JavaScript
– Rowhammer.js
– Flush+Flush

– ARMageddon
– Prefetch Attacks
Cache Attacks on mobile devices?

- powerful cache attacks on Intel x86 in the last 10 years
- nothing like Flush+Reload or Prime+Probe on mobile devices

→ why?
ARMageddon in a nutshell

1. no flush instruction
2. pseudo-random replacement
3. cycle counters require root
4. last-level caches not inclusive
5. multiple CPUs
ARMageddon in a nutshell

1. no flush instruction $\rightarrow$ Evict+Reload
2. pseudo-random replacement
3. cycle counters require root
4. last-level caches not inclusive
5. multiple CPUs
ARMageddon in a nutshell

1. no flush instruction → Evict+Reload
2. pseudo-random replacement → eviction strategies from Rowhammer.js
3. cycle counters require root
4. last-level caches not inclusive
5. multiple CPUs
ARMageddon in a nutshell

1. no flush instruction $\rightarrow$ Evict+Reload
2. pseudo-random replacement $\rightarrow$ eviction strategies from Rowhammer.js
3. cycle counters require root $\rightarrow$ new timing methods
4. last-level caches not inclusive
5. multiple CPUs
ARMageddon in a nutshell

1. no flush instruction $\rightarrow$ Evict+Reload
2. pseudo-random replacement $\rightarrow$ eviction strategies from Rowhammer.js
3. cycle counters require root $\rightarrow$ new timing methods
4. last-level caches not inclusive $\rightarrow$ let L1 spill to L2
5. multiple CPUs
ARMageddon in a nutshell

1. no flush instruction → Evict+Reload
2. pseudo-random replacement → eviction strategies from Rowhammer.js
3. cycle counters require root → new timing methods
4. last-level caches not inclusive → let L1 spill to L2
5. multiple CPUs → remote fetches + flushes
3. Contributions

- Cache Template Attacks
- Page Deduplication Attacks in JavaScript
- Rowhammer.js
- Flush+Flush
- ARMageddon
- Prefetch Attacks
Prefetch: Motivation

Idea: Would this also work on inaccessible kernel memory?

<table>
<thead>
<tr>
<th>Mapping level</th>
<th>Execution time</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDPT</td>
<td>230</td>
</tr>
<tr>
<td>PD</td>
<td>246</td>
</tr>
<tr>
<td>PT</td>
<td>222</td>
</tr>
<tr>
<td>uncached P.</td>
<td>383</td>
</tr>
<tr>
<td>cached P.</td>
<td>181</td>
</tr>
</tbody>
</table>
Prefetch: Kernel Memory Layout

Physical memory

Virtual address space

User

Kernel

max. phys.

Direct map
Prefetching Kernel Addresses

The graph shows the minimum access latency (in a certain unit, likely time) as a function of the page offset in the kernel direct map. The x-axis represents the page offset, ranging from 0 to 240, while the y-axis shows the latency ranging from 100 to 250. The data points are visualized with a blue line, indicating a relatively stable latency with minor fluctuations.
Prefetch: Locate Kernel Driver (defeat KASLR)

Avg. execution time

Page offset in kernel driver region

Daniel Gruss, IAIK
June 14, 2017 — PhD Defense
Conclusions

1. microarchitectural attacks can be widely automated
2. unknown and novel side channels are likely to exist
3. minimal requirements enable attacks through websites
4. constructing countermeasures is difficult and requires solid understanding of attacks
Conclusions

1. microarchitectural attacks can be **widely automated**
Conclusions

1. microarchitectural attacks can be **widely automated**
2. unknown and **novel side channels** are likely to exist
Conclusions

1. microarchitectural attacks can be **widely automated**
2. unknown and **novel side channels** are likely to exist
3. **minimal requirements** enable attacks through websites
Conclusions

1. microarchitectural attacks can be *widely automated*
2. unknown and **novel side channels** are likely to exist
3. **minimal requirements** enable attacks through websites
4. constructing countermeasures is difficult and requires solid understanding of attacks
Author’s Publications in this Thesis I


Author’s Publications in this Thesis II


Further Contributions I


Further Contributions II


Software-based Microarchitectural Attacks

Daniel Gruss
IAIK, Graz University of Technology

June 14, 2017 — PhD Defense
Bibliography I


Bibliography II


Bibliography III


