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 → 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
- F+R in JS
- F+R on ARM
- CTA

Daniel Gruss, IAIK
March 27, 2015
Plan (how it worked out)

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

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

Page Dedup. \(\rightarrow\) Page Dedup. in JS

P+P \(\rightarrow\) P+P in JS

F+R \(\rightarrow\) F+R on Memory

CTA \(\rightarrow\) F+R in JS

F+R \(\rightarrow\) F+R on ARM

P+P in JS

F+R on Memory

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
- Rowhammer.js
- CTA
- F+R on ARM
- F+R on ARM

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 on Memory
- Rowhammer.js
- CTA
- ARMageddon
Plan (how it worked out)

- Page Dedup.
- Page Dedup. in JS
- P+P
- P+P in JS
- F+R
- 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
- automation of attacks
- novel side channels

CTA
Relation of the papers

- minimization of requirements
- Dedup.js
- novel side channels
- automation of attacks
- CTA
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

F+F

CTA

novel side channels

automation of attacks
Relation of the papers

minimization of requirements

Dedup.js

RH.js

ARMageddon

F+F

CTA

novel side channels

automation of attacks
Relation of the papers

- minimization of requirements
- automation of attacks
- novel side channels

CTA

Dedup.js

RH.js

ARMageddon

F+F

Prefetch

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

![Graph showing memory access latency with latency in cycles on the x-axis and number of accesses on the y-axis. The graph is divided into two categories: Cached and Not Cached.]
Memory Access Latency

Number of Accesses vs. Latency in Cycles

- Cached
- Not Cached
A simple cache
A simple cache

Memory Address

Offset

Cache
A simple cache

Memory Address

| Index | Offset |

Cache

$2^n$ cache sets
A simple cache

Memory Address

| Tag | Index | Offset |

Cache

<table>
<thead>
<tr>
<th>Way 1 Tag</th>
<th>Way 1 Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Way 2 Tag</td>
<td>Way 2 Data</td>
</tr>
</tbody>
</table>

$2^n$ cache sets
Date and Instruction Caches

- core 0
  - L1
  - L2
- core 1
  - L1
  - L2
- core 2
  - L1
  - L2
- core 3
  - L1
  - L2

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

Daniel Gruss, IAIK
June 14, 2017 — PhD Defense
Flush+Reload

Attacker address space  Cache  Victim address space

flushes
Flush+Reload

Attacker address space

Cache

Victim address space

loads data
Flush+Reload

Attacker address space

Cache

reloads data

Victim address space
3. Contributions
   – Cache Template Attacks
   – Page Deduplication Attacks in JavaScript
   – Rowhammer.js
   – Flush+Flush
   – ARMageddon
   – Prefetch Attacks
% sleep 2; ./spy 300 7f05140a4080-7f051417b000 r-xp 0x20000 08:02 26 8050
/usr/lib/x86_64-linux-gnu/gedit/libgedit.so

shark% ./spy
Cache Template

KEY

ADDRESS

0x7c680
0x7c6c0
0x7c700
0x7c740
0x7c780
0x7c7c0
0x7c800
0x7c840
0x7c880
0x7c8c0
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

JavaScript

Virtual Address Space

Victim

Physical Address Space
Page Deduplication Attack

Attacker generates a page suspected in victim process
Page Deduplication Attack
Page Deduplication Attack

Virtual Address Space

Attacker waits for deduplication

Physical Address Space

Victim

JavaScript

Attacker waits for deduplication
Page Deduplication Attack

```javascript
let t = time();
p[0] = p[0];
let Δ = time() - t;
```
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

Virtual Address Space

Physical Address Space

Victim

JavaScript

\[ \Delta \text{ in } \mu s \] 0 4

Time

measure \( \downarrow \)
Page Deduplication Attack

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

Time

Physical Address Space

Virtual Address Space

Victim

JavaScript
Page Deduplication Attack

JavaScript  \hspace{1cm} Virtual Address Space  \hspace{1cm} Victim

\[ \Delta \text{ in µs} \]

\begin{align*}
\text{Time} & \\
0 & \quad 4
\end{align*}

Physical Address Space
Page Deduplication Attack

![Diagram showing JavaScript, Virtual Address Space, and Victim. The diagram illustrates the comparison of time measures in microseconds (µs) between the JavaScript and Victim processes, highlighting the physical address space.]
Page Deduplication Attack

- **JavaScript**
- **Virtual Address Space**
  - \( \Delta \text{in } \mu s \)
  - Time
- **Victim**
- **Physical Address Space**

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

JavaScript

Virtual Address Space

Time

Victim

Physical Address Space

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

\( \Delta \) measures the difference in time between the JavaScript and the Victim. The diagram shows the physical address space divided into regions, with different colors indicating the allocation of memory.
Page Deduplication Attack

![Diagram showing JavaScript, Virtual Address Space, and Physical Address Space with measures in µs over time](https://example.com/diagram.png)
Page Deduplication Attack

Virtual Address Space

Physical Address Space

JavaScript

Victim
Page Deduplication Attack

![Diagram showing 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
- Physical Address Space
- Victim

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Δ in μs</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Diagram showing the relationship between JavaScript, Virtual Address Space, and Physical Address Space, with a measure in μs over time.
Page Deduplication Attack

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

Time

Physical Address Space

Virtual Address Space

Victim

JavaScript
Page Deduplication Attack

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

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

Time

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

![Diagram showing JavaScript, Virtual Address Space, and Physical Address Space with a measure of time in microseconds (µs)].
Page Deduplication Attack

![Diagram showing virtual and physical address spaces with measures and their impacts on time.](diagram.png)
Page Deduplication Attack

Diagram showing JavaScript, Virtual Address Space, and Victim. The diagram illustrates the measure in \( \mu s \) over time."
Page Deduplication Attack

![Diagram of Page Deduplication Attack]

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

This diagram illustrates the concept of a Page Deduplication Attack, showing the interaction between the JavaScript environment, the virtual address space, and the physical address space of the victim. The attack measures the time difference (Δ in µs) over time (Time) to detect vulnerabilities. The diagram visualizes how data is accessed and compared in both the virtual and physical address spaces.
Page Deduplication Attack

Virtual Address Space

Physical Address Space

Victim

JavaScript

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

Time

\[ \neq \]

measure

\( \downarrow \)
Page Deduplication Attack

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

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

Time

\[ \neq \]

\( \bigtriangleup \)
Page Deduplication Attack

JavaScript  Virtual Address Space  Victim

Physical Address Space

$\Delta$ in $\mu$s

Time

$\neq$

measure

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

Virtual Address Space

Physical Address Space

Victim

JavaScript

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

Time

\[ \neq \]

Measure

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

Virtual Address Space

Physical Address Space

Victim

JavaScript

Time

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

\[ \begin{array}{c}
0 \\
4 \\
\end{array} \]

\[ \neq \]

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

Virtual Address Space

Time

0

4

Δ in µs

Victim

Physical Address Space

JavaScript

=}

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

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

![Diagram]

- **Physical Address Space**

- Δ in μs
- Time

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

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

**Virtual Address Space**

- **JavaScript**
- **Victim**

**Physical Address Space**

- **Time**
- **Write and measure** $\Delta$

$\Delta$ in $\mu$s

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

write and measure $\Delta$

copy

JavaScript

Virtual Address Space

Victim

Time

$\Delta$ in $\mu$s

Physical Address Space

$0$

$4$

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

\( \text{write} \)
Page Deduplication Attack

Attacker learns that another process had an identical page

<table>
<thead>
<tr>
<th>JavaScript</th>
<th>Virtual Address Space</th>
<th>Victim</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Δ in μs</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Time</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Physical Address Space
Page Deduplication Attack

Attacker learns that another process had an identical page

Physical Address Space

Virtual Address Space

Victim

JavaScript

\[ \text{Time} \]

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

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
Page Deduplication Attack

Attacker learns that another process had an identical page

Time

$\Delta$ in $\mu$s

Physical Address Space

Virtual Address Space

Victim

JavaScript
Page Deduplication Attack

JavaScript

Virtual Address Space

Victim

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

Time

Attacker learns that another process had an identical page

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

JavaScript

Virtual Address Space

Victim

Physical Address Space

Δ in μs

Time

0 4

Daniel Gruss, IAIK
June 14, 2017 — PhD Defense

www.iaik.tugraz.at
Page Deduplication Attack

JavaScript

Virtual Address Space

Victim

Attacker learns that another process had an identical page

Physical Address Space
Page Deduplication Attack

Attacker learns that another process had an identical page

JavaScript

Virtual Address Space

Victim

Physical Address Space

Δ in μs

0

4

Time
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

\( \Delta \text{ in } \mu s \)

Time
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

JavaScript

Virtual Address Space

Time

0

4

Δ in µs

Victim

Physical Address Space
Our Attack

First page deduplication attack which

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

![Graph showing the time taken to load images](Image)

- Image not loaded
- Image loaded

Nanoseconds

Page

Daniel Gruss, IAIK
June 14, 2017 — PhD Defense
3. Contributions
   – Cache Template Attacks
   – Page Deduplication Attacks in JavaScript
   – Rowhammer.js
   – Flush+Flush
   – ARMedgeddon
   – 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

cache set 2

DRAM bank
Rowhammer (with clflush)
Rowhammer (with clflush)
Rowhammer (with clflush)

cache set 1

cache set 2

DRAM bank

reload
Rowhammer (with clflush)

cache set 1

cache set 2

DRAM bank

reload

reload
Rowhammer (with clflush)

cache set 1

clflush

cache set 2

clflush

DRAM bank
Rowhammer (with clflush)

- Cache set 1
- Cache set 2
- DRAM bank

reload
Rowhammer (with clflush)
Rowhammer (with clflush)

cache set 1

DRAM bank

cache set 2

reload

reload
Rowhammer (with clflush)

cache set 1

cache set 2

DRAM bank

clflush

clflush
Rowhammer (with `clflush`)
Rowhammer (with clflush)

cache set 1

clflush

cache set 2

clflush

DRAM bank

wait for it. . .
Rowhammer (with clflush)

cache set 1

cache set 2

DRAM bank

bit flip!

reload

reload
Rowhammer without clflush

cache set 1

cache set 2

DRAM bank
Rowhammer without \texttt{clflush}

cache set 1

load

load

DRAM bank

cache set 2
Rowhammer without `clflush`

Cache set 1

Cache set 2

DRAM bank
Rowhammer without clflush

Cache set 1

Cache set 2

DRAM bank

Load

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

cache set 1

load

DRAM bank

load

cache set 2
Rowhammer without `clflush`

Cache set 1

Cache set 2

DRAM bank

Load
Rowhammer without `clflush`

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

```

<table>
<thead>
<tr>
<th>DRAM bank</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

cache set 1

load

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

cache set 2

load

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

cache set 1

cache set 2

DRAM bank
Rowhammer without clflush

Cache set 1

Cache set 2

DRAM bank

reload

reload
Rowhammer without clflush

repeat!
Rowhammer without \texttt{clflush}

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

Cache set 1: 

Cache set 2: 

DRAM bank: 

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
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

```
cache set  2  5  8  9  7  6  3  4
```

load
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

Daniel Gruss, IAIK
June 14, 2017 — PhD Defense
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

![Diagram of cache set with-memory accesses](image)
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
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: \( P \cdot C \cdot D \cdot L \cdot S \)

```c
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)

```c
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

```c
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
- \( L \): step size of the inner access loop

```c
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
Cache eviction strategy: Notation (2)

\[
\begin{align*}
&\text{for } (s = 0; s \leq S - D; s += L) \\
&\quad \text{for } (c = 1; c \leq C; c += 1) \\
&\quad \text{for } (d = 1; d \leq D; d += 1) \\
&\quad \quad *a[s+d];
\end{align*}
\]
Cache eviction strategy: Notation (2)

```c
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];
```

- $\mathcal{P} - 2 - 2 - 1 - 4 \rightarrow 1, 2, 1, 2, 2, 3, 2, 3, 3, 4, 3, 4$
Cache eviction strategy: Notation (2)

```c
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];
```

- \( \mathcal{P}-2-2-1-4 \rightarrow 1, 2, 1, 2, 2, 3, 2, 3, 3, 4, 3, 4 \)
- \( \mathcal{P}-1-1-1-4 \rightarrow 1, 2, 3, 4 \rightarrow \) 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>1-1-1-17</td>
<td>17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>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% ( \times )</td>
<td>307 ns ( \checkmark )</td>
</tr>
<tr>
<td>( P-1-1-1-20 )</td>
<td>20</td>
<td>99.82% ( \checkmark )</td>
<td>934 ns ( \times )</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% ![X]</td>
<td>307 ns ![✓]</td>
</tr>
<tr>
<td>$P_{-1-1-1-20}$</td>
<td>20</td>
<td>99.82% ![✓]</td>
<td>934 ns ![X]</td>
</tr>
<tr>
<td>$P_{-2-1-1-17}$</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>$P$-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 X</td>
</tr>
<tr>
<td>$P$-2-1-1-17</td>
<td>34</td>
<td>99.86% ✓</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>
</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 ❌</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>$P_{1-1-1-17}$</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></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>$\mathcal{P}$-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>99.98% ✓</td>
<td>180 ns ✓</td>
</tr>
</tbody>
</table>

$\Rightarrow$ more accesses, smaller execution time? Executed in a loop, on a Haswell with a 16-way last-level cache
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)
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)

\[ \mathcal{P}-1-1-1-17 \] (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)

\[ \mathcal{P}-1-1-1-17 \] (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)

\( \mathcal{P}-1-1-1-17 \) (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)

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

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

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

\[ Miss \ (intended) \ Miss \ (intended) \ Miss \ Miss \ Miss \ Miss \]

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

\[ Miss \ (intended) \ Miss \ (intended) \ Miss \ Miss \ Miss \]

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)

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, 307\,ns)

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

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

<table>
<thead>
<tr>
<th>Miss (intended)</th>
<th>Miss (intended)</th>
<th>Miss</th>
<th>Miss</th>
<th>Miss</th>
<th>Miss</th>
<th>Miss</th>
<th>Miss</th>
<th>Miss</th>
</tr>
</thead>
</table>

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

<table>
<thead>
<tr>
<th>Miss (intended)</th>
<th>Miss (intended)</th>
<th>Miss</th>
<th>Miss</th>
<th>Miss</th>
<th>Miss</th>
<th>Miss</th>
<th>Miss</th>
<th>Miss</th>
</tr>
</thead>
</table>

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)
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)
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 → many cache misses
- detect via performance counters
  → good idea, but not good enough
Flush+Reload

Attacker address space

Cache

Victim address space
Flush+Reload

Attacker address space

Cache

Victim address space

cached

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 → shared memory, shared in cache
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 → 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 → 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 → 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 → 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 $\rightarrow$ Evict+Reload
2. pseudo-random replacement $\rightarrow$ 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 → 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
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 $\rightarrow$ 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 $\rightarrow$ remote fetches + flushes
shell@zeroflte:/data/local/tmp $ ./keyboard_spy -c 0
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?
Prefetch: Kernel Memory Layout
Prefetching Kernel Addresses

Min. access latency

Page offset in kernel direct map
Prefetch: Locate Kernel Driver (defeat KASLR)
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


