Oh my Cache! 2
More fun with caches.

Daniel Gruss
Graz University of Technology

October 13, 2017 — QSP Lab
Whoami

- Daniel Gruss
- Post-Doc @ Graz University of Technology
- Twitter: @lavados
- Email: daniel.gruss@iaik.tugraz.at
Get your computer ready!

Within the first two hours we will:

- Checkout https://github.com/IAIK/cache_template_attacks
- Make a histogram
- Key stroke attack on an editor
- Try to establish a covert channel
1. Quick Start

2. Measuring and exploiting timing leakage

3. CPU caches

4. Cache attacks

5. Cache covert channels

6. Cache template attacks

7. Page Deduplication Attacks

8. Bitflips!

9. How to exploit bit flips?

10. How to mitigate Rowhammer?
Information leakage

Shared hardware

Memory
- Memory deduplication
- Memory bus

x86 CPU
- Branch prediction unit
- Arithmetic logic unit
- Data and instruction cache
Why targeting the cache?

- shared across cores
- fast
Why targeting the cache?

- shared across cores
- fast

→ fast cross-core attacks!
Timing differences

- caches improve performance
- SRAM is expensive $\rightarrow$ small caches
- different timings for memory accesses
  - data is *cached* $\rightarrow$ cache hit $\rightarrow$ fast
  - data is *not cached* $\rightarrow$ cache miss $\rightarrow$ slow
1. Quick Start

2. Measuring and exploiting timing leakage

3. CPU caches

4. Cache attacks

5. Cache covert channels

6. Cache template attacks

7. Page Deduplication Attacks

8. Bitflips!

9. How to exploit bit flips?

10. How to mitigate Rowhammer?
Mesuring timing leakage

How every timing attack works:

- learn timing of different corner cases
Mesuring timing leakage

How every timing attack works:

- learn timing of different corner cases
- later, we recognize these corner cases by timing only
Calibration

git clone https://github.com/IAIK/cache_template_attacks.git
cd calibration
make
./calibration
Steps

1. build two cases: cache hits and cache misses
2. time each case many times (get rid of noise)
Steps

1. build two cases: cache hits and cache misses
2. time each case many times (get rid of noise)
3. we have a histogram!
Steps

1. build two cases: cache hits and cache misses
2. time each case many times (get rid of noise)
3. we have a histogram!
4. find a threshold to distinguish the two cases
Step 1.1. Cache hits

Loop:

1. measure time
2. access variable (always cache hit)
3. measure time
4. update histogram with delta
Step 1.2. Cache misses

Loop:

1. measure time
2. access variable (always cache \textit{miss})
3. measure time
4. update histogram with delta
5. \textbf{flush} variable (\texttt{clflush} instruction)
Step 2. Accurate timings

- very short timings
- `rdtsc` instruction: cycle-accurate timestamps

  
  ```
  [...] 
  rdtsc 
  function() 
  rdtsc 
  [...] 
  ```
Step 2. Accurate timings

- do you measure what you *think* you measure?
- out-of-order execution → what is really executed

```
rdtsc
function()
[...]  # Example sequence of instructions
rdtsc
function()
```

```rtdtsc
rdtsc
[...]
rdtsc
function()
[...]```

Daniel Gruss, Graz University of Technology
October 13, 2017 — QSP Lab
Step 2. Accurate timings

- use pseudo-serializing instruction \texttt{rdtscp} (recent CPUs)
Step 2. Accurate timings

- use pseudo-serializing instruction `rdtscp` (recent CPUs)
- and/or use serializing instructions like `cpuid`
Step 2. Accurate timings

- use pseudo-serializing instruction `rdtscp` (recent CPUs)
- and/or use serializing instructions like `cpuid`
- and/or use fences like `mfence`
Step 2. Accurate timings

- use pseudo-serializing instruction \texttt{rdtscp} (recent CPUs)
- and/or use serializing instructions like \texttt{cpuid}
- and/or use fences like \texttt{mfence}

Step 3. Histogram

![Histogram Chart]

- **Latency in Cycles**
- **Number of Accesses**

Legend:
- **Cached**
- **Not Cached**
Step 3. Histogram

Latency in Cycles

Number of Accesses

Cached
Not Cached
Step 4. Find threshold

- as high as possible
- most cache hits are below
- no cache miss below
Side-channel attack on user input

- locate **key-dependent** memory accesses
- with cache template attacks
Profiling Phase: one event

Attacker address space

Victim address space

Cache is empty

Cache
Profiling Phase: one event

Attacker triggers an event
Profiling Phase: one event

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

Attacker address space

Cache

Victim address space

Update number of cache hits per event
Profiling Phase: one event

Attacker address space

| Shared 0x0 |

Cache

| Shared 0x0 |

Victim address space

| Shared 0x0 |

Attacker flushes shared memory
Profiling Phase: one event

Attacker address space

<table>
<thead>
<tr>
<th>Shared 0x0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Cache

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

Victim address space

<table>
<thead>
<tr>
<th>Shared 0x0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Repeat for higher accuracy
Profiling Phase: one event

Attacker address space

Victim address space

Shared \(0x40\)

Continue with next address
Profiling Phase: one event

Attacker address space

Victim address space

Cache

Continue with next address
What to profile?

# ps -A | grep gedit
# cat /proc/pid/maps
00400000-00489000 r-xp 00000000 08:11 396356 /usr/bin/gedit
7f5a96991000-7f5a96a51000 r-xp 00000000 08:11 399365 /usr/lib/x86_64-linux-gnu/libgdk-3.so.0.1400.14 ...

memory range, access rights, offset, –, –, file name
Profiling a single event

cd ..;/profiling/generic_low_frequency_example
# put the threshold into spy.c (MIN_CACHE_MISS_CYCLES)
make
./spy
# start the targeted program
sleep 2; ./spy 200 400000-489000 -- 20000
-- -- /usr/bin/gedit

... and hold down key in the targeted program
save addresses with peaks!
Exploitation phase

cd ../exploitation/generic
# put the threshold into spy.c (MIN_CACHE_MISS_CYCLES)
make
./spy file offset
1. Quick Start
2. Measuring and exploiting timing leakage
3. CPU caches
4. Cache attacks
5. Cache covert channels
6. Cache template attacks
7. Page Deduplication Attacks
8. Bitflips!
9. How to exploit bit flips?
10. How to mitigate Rowhammer?
Directly mapped cache

Memory Address
Directly mapped cache

Memory Address

Cache
Directly mapped cache

<table>
<thead>
<tr>
<th>Memory Address</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag</td>
</tr>
<tr>
<td>Data</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
Directly mapped cache

<table>
<thead>
<tr>
<th>Memory Address</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag</td>
</tr>
<tr>
<td>-----</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
Directly mapped cache

<table>
<thead>
<tr>
<th>Memory Address</th>
<th>Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>$b$ bits</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$2^b$ bytes
Directly mapped cache

Memory Address

\[ \begin{array}{cc}
\text{Tag} & \text{Data} \\
\end{array} \]

Cache

\[ 2^b \text{ bytes} \]
Directly mapped cache

Memory Address

\[ \begin{array}{c|c}
\text{Memory Address} & \text{Cache} \\
\hline
\text{Tag} & \text{Data} \\
2^n \text{ cache lines} & 2^b \text{ bytes} \\
\end{array} \]

Cache Index

\( n \) bits \hspace{1cm} \( b \) bits
Directly mapped cache

Memory Address

\[ f \]

\[ n \text{ bits} \quad b \text{ bits} \]

Cache

\[ 2^n \text{ cache lines} \]

Tag

Data

Hit/Miss

\[ 2^b \text{ bytes} \]
Directly mapped cache

Problem: working on congruent addresses
2-way set associativity

Memory Address

\[
f \rightarrow n \text{ bits} \quad b \text{ bits}
\]

Cache Index

\[2^n \text{ cache lines}\]

Cache

\[
\begin{array}{|c|c|}
\hline
\text{Tag} & \text{Data} \\
\hline
\end{array}
\]
2-way set associativity

Memory Address

\[ \begin{array}{c|c}
\text{Cache Index} & \text{Way 2 Tag} \\
\text{$2^n$ cache sets} & \text{Way 2 Data} \\
\end{array} \]

\[ \begin{array}{c|c|c}
\text{Way 1 Tag} & \text{Way 1 Data} \\
\text{Way 2 Tag} & \text{Way 2 Data} \\
\end{array} \]

\[ f \]

\[ \begin{array}{c|c|c}
\text{n bits} & \text{b bits} \\
\end{array} \]
2-way set associativity
2-way set associativity

Memory Address

\[ f \]

\[ \begin{array}{c|c|c}
\hline
	& n \text{ bits} & b \text{ bits} \\
\hline
\end{array} \]

Cache

\[ 2^n \text{ cache sets} \]

Cache Index

\[ =? \]

Tag

\[ =? \]

Data

Way 1 Tag

Way 2 Tag

Way 1 Data

Way 2 Data

Daniel Gruss, Graz University of Technology
October 13, 2017 — QSP Lab
2-way set associativity

Memory Address

\[ f \]

\[ \begin{array}{|c|c|}
\hline
\text{ } & \text{ } \\
\hline
n \text{ bits} & b \text{ bits} \\
\hline
\end{array} \]

Cache

\[ \begin{array}{|c|c|}
\hline
\text{Way 1 Tag} & \text{Way 1 Data} \\
\hline
\text{Way 2 Tag} & \text{Way 2 Data} \\
\hline
\end{array} \]

2\(^n\) cache sets

Cache Index

Tag

\[ \Rightarrow \] replacement policy

→ replacement policy
Caches today

- L1 and L2 are private
- last-level cache:
  - divided in slices
  - shared across cores
  - inclusive
Cache levels: Latency comparison

On current Intel CPUs:
Cache levels: Latency comparison

On current Intel CPUs:

- L1 cache: 4 cycles
Cache levels: Latency comparison

On current Intel CPUs:

- L1 cache: 4 cycles
- L2 cache: 12 cycles
Cache levels: Latency comparison

On current Intel CPUs:

- L1 cache: 4 cycles
- L2 cache: 12 cycles
- L3 cache: 26-31 cycles
Cache levels: Latency comparison

On current Intel CPUs:

- L1 cache: 4 cycles
- L2 cache: 12 cycles
- L3 cache: 26-31 cycles
- DRAM memory: >120 cycles
(Unprivileged) cache maintenance

User programs can optimize cache usage:

- **prefetch**: suggest CPU to load data into cache
- **clflush**: throw out data from all caches

... based on virtual addresses
1. Quick Start
2. Measuring and exploiting timing leakage
3. CPU caches
4. Cache attacks
5. Cache covert channels
6. Cache template attacks
7. Page Deduplication Attacks
8. Bitflips!
9. How to exploit bit flips?
10. How to mitigate Rowhammer?
CPU cache attacks

- cache-based keylogging
- crypto key recovery
  - various implementations (AES, RSA, ECC, ...)
  - up to 97% key bits recovered after 1 encryption
- cross-VM, cross-core, even cross-CPU
- any CPU vendor
Cross-core attacks?

- using the inclusive property
Cross-core attacks?

- using the **inclusive** property
- last-level cache is a superset of L1 and L2
Cross-core attacks?

- using the inclusive property
- last-level cache is a superset of L1 and L2
- data evicted from last-level cache $\rightarrow$ evicted from L1 and L2
Cross-core attacks?

- using the inclusive property
- last-level cache is a superset of L1 and L2
- data evicted from last-level cache $\rightarrow$ evicted from L1 and L2
- a core can evict lines in the private L1 of another core
Access-driven attacks

Attacker monitors its own activity to find sets accessed by victim.

Flush+Reload
- Gullasch et al. 2011
- Yarom and Falkner 2014
- Gruss, Spreitzer, et al. 2015

Prime+Probe
- Percival 2005
- Liu et al. 2015
- Maurice, Neumann, et al. 2015

Same techniques for covert and side channels
Flush+Reload: Building Blocks

- Shared Library / load binary twice / page deduplication
Flush+Reload: Building Blocks

- Shared Library / load binary twice / page deduplication
- `clflush` throws data out of cache

→ We can throw other shared code out of the cache
Flush+Reload: Building Blocks

- Shared Library / load binary twice / page deduplication
- `clflush` throws data out of cache
  → We can throw other shared code out of the cache
- `rdtsc` / `rdtscp` give accurate timing information
  → We can measure whether shared code is in the cache
Flush+Reload: First steps

- Measure timing of cached memory
- Measure timing of non-cached memory (flush before measuring)
- Draw a histogram
Flush+Reload

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

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

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

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

**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+Reload

**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 reloads data → fast access if the victim loaded the line
Flush+Reload

Pros: fine granularity (1 line)

Cons: restrictive

1. needs clflush instruction (not available e.g., in JS)
2. needs shared memory
Variants of Flush+Reload

Prime+Probe

step 0: attacker fills the cache (prime)
Prime+Probe

**step 0**: attacker fills the cache (prime)
Prime+Probe

step 0: attacker fills the cache (prime)
Prime+Probe

**Step 0**: Attacker fills the cache (prime)

**Step 1**: Victim evicts cache lines while performing encryption
Prime+Probe

**Step 0**: Attacker fills the cache (prime)

**Step 1**: Victim evicts cache lines while performing encryption
Prime+Probe

step 0: attacker fills the cache (prime)
step 1: victim evicts cache lines while performing encryption
**Prime + Probe**

**Step 0**: attacker fills the cache (prime)

**Step 1**: victim evicts cache lines while performing encryption
Prime+Probe

**step 0**: attacker fills the cache (prime)
**step 1**: victim evicts cache lines while performing encryption
Prime+Probe

**Step 0**: attacker fills the cache (prime)

**Step 1**: victim evicts cache lines while performing encryption

**Step 2**: attacker probes data to determine if the set was accessed
Prime+Probe

**step 0**: attacker fills the cache (prime)

**step 1**: victim evicts cache lines while performing encryption

**step 2**: attacker probes data to determine if the set was accessed
Prime+Probe

**step 0**: attacker fills the cache (prime)
**step 1**: victim evicts cache lines while performing encryption
**step 2**: attacker probes data to determine if the set was accessed
Prime+Probe

Pros: less restrictive

1. no need for `clflush` instruction (not available e.g., in JS)
2. no need for shared memory

Cons: coarser granularity (1 set)
Issues with Prime+Probe

We need to evict caches lines without `clflush` or shared memory:

1. which addresses do we access to have congruent cache lines?
2. without any privilege?
3. and in which order do we access them?
#1.1: Which physical addresses to access?

“LRU eviction”:

- assume that cache uses LRU replacement
- accessing \( n \) addresses from the same cache set to evict an \( n \)-way set
- eviction from last level → from whole hierarchy (it’s inclusive!)
#1.2: Which addresses map to the same set?

- function H that maps slices is undocumented
- reverse-engineered by Maurice, Le Scouarnec, et al. 2015; Inci et al. 2015; Yarom, Ge, et al. 2015
#1.2: Which addresses map to the same set?

- function H that maps slices is undocumented
- reverse-engineered by Maurice, Le Scouarnec, et al. 2015; Inci et al. 2015; Yarom, Ge, et al. 2015
- hash function basically an XOR of address bits
#1.2: Which addresses map to the same set?

3 functions, depending on the number of cores

<table>
<thead>
<tr>
<th>Address bit</th>
<th>3</th>
<th>6</th>
<th>5</th>
<th>3</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>2</th>
<th>8</th>
<th>7</th>
<th>2</th>
<th>6</th>
<th>5</th>
<th>2</th>
<th>2</th>
<th>2</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 cores</td>
<td>$o_0$</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
</tr>
<tr>
<td></td>
<td>$o_1$</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
</tr>
<tr>
<td>4 cores</td>
<td>$o_0$</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
</tr>
<tr>
<td></td>
<td>$o_1$</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
</tr>
<tr>
<td>8 cores</td>
<td>$o_0$</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
</tr>
<tr>
<td></td>
<td>$o_1$</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
</tr>
<tr>
<td></td>
<td>$o_2$</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
<td>⊕</td>
</tr>
</tbody>
</table>
#2: Obtain information without root privileges

- last-level cache is physically indexed
#2: Obtain information without root privileges

- last-level cache is physically indexed
- root privileges needed for physical addresses
#2: Obtain information without root privileges

- last-level cache is physically indexed
- root privileges needed for physical addresses
- use 2 MB pages → lowest 21 bits are the same as virtual address
#2: Obtain information without root privileges

- last-level cache is physically indexed
- root privileges needed for physical addresses
- use 2 MB pages → lowest 21 bits are the same as virtual address
  → enough to compute the cache set
#3.1: Replacement policy on older CPUs

“LRU eviction” memory accesses

![Cache set diagram]

- LRU replacement policy: oldest entry first
- Timestamps for every cache line
- Access updates timestamp

Daniel Gruss, Graz University of Technology
October 13, 2017 — QSP Lab
#3.1: Replacement policy on older CPUs

“LRU eviction” memory accesses

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

“LRU eviction” memory accesses

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

“LRU eviction” memory accesses

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

![Diagram of cache set and timestamps](image)
#3.1: Replacement policy on older CPUs

“LRU eviction” memory accesses

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

“LRU eviction” memory accesses

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

![Diagram of cache set with timestamps](image_url)
#3.1: Replacement policy on older CPUs

“LRU eviction” memory accesses

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

“LRU eviction” memory accesses

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

“LRU eviction” memory accesses

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

“LRU eviction” memory accesses

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

“LRU eviction” memory accesses

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

“LRU eviction” memory accesses

- no LRU replacement
#3.2: Replacement policy on recent CPUs

“LRU eviction” memory accesses

- no LRU replacement
#3.2: Replacement policy on recent CPUs

“LRU eviction” memory accesses

- no LRU replacement
#3.2: Replacement policy on recent CPUs

“LRU eviction” memory accesses

- no LRU replacement
#3.2: Replacement policy on recent CPUs

“LRU eviction” memory accesses

- no LRU replacement
#3.2: Replacement policy on recent CPUs

“LRU eviction” memory accesses

- no LRU replacement
#3.2: Replacement policy on recent CPUs

“LRU eviction” memory accesses

- no LRU replacement
#3.2: Replacement policy on recent CPUs

“LRU eviction” memory accesses

- no LRU replacement
#3.2: Replacement policy on recent CPUs

“LRU eviction” memory accesses

- no LRU replacement
#3.2: Replacement policy on recent CPUs

“LRU eviction” memory accesses

- no LRU replacement
- only 75% success rate on Haswell
#3.2: 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
#3.3: Cache eviction strategy

Figure: Fast and effective on Haswell. Eviction rate >99.97%.
1. Quick Start
2. Measuring and exploiting timing leakage
3. CPU caches
4. Cache attacks
5. Cache covert channels
6. Cache template attacks
7. Page Deduplication Attacks
8. Bitflips!
9. How to exploit bit flips?
10. How to mitigate Rowhammer?
Side channels vs covert channels

- side channel: attacker spies a victim process
- *covert channel*: communication between two processes
  - that are not supposed to communicate
  - that are collaborating
1-bit cache covert channels

ideas for 1-bit channels:
1-bit cache covert channels

ideas for 1-bit channels:

- Prime+Probe: use one cache set to transmit
  
  0: sender does not access the set $\rightarrow$ low access time in receiver
  1: sender does access the set $\rightarrow$ high access time in receiver
1-bit cache covert channels

ideas for 1-bit channels:

- Prime+Probe: use one cache set to transmit
  0: sender does not access the set → low access time in receiver
  1: sender does access the set → high access time in receiver

- Flush+Reload/Flush+Flush/Evict+Reload: use one address to transmit
  0: sender does not access the address → high access time in receiver
  1: sender does access the address → low access time in receiver
1-bit covert channels

- 1 bit data, 0 bit control?
1-bit covert channels

- 1 bit data, 0 bit control?
- idea: divide time into slices (e.g., 50μs frames)
- synchronize sender and receiver with a shared clock
Problems of 1-bit covert channels

- errors?
Problems of 1-bit covert channels

- errors? → error-correcting codes
- retransmission may be more efficient (less overhead)
- desynchronization
- optimal transmission duration may vary
Multi-bit covert channels

- combine multiple 1-bit channels
Multi-bit covert channels

- combine multiple 1-bit channels
- avoid interferences
  → higher performance
Multi-bit covert channels

- combine multiple 1-bit channels
- avoid interferences
→ higher performance
- use 1-bit for sending = true/false
Packets / frames

Organize data in packets / frames:

- some data bits
- check sum
- sequence number

→ keep sender and receiver synchronous
→ check whether retransmission is necessary
## State of the art

<table>
<thead>
<tr>
<th>method</th>
<th>raw capacity</th>
<th>err. rate</th>
<th>true capacity</th>
<th>env.</th>
</tr>
</thead>
<tbody>
<tr>
<td>F+F Gruss, Maurice, Wagner, et al. 2016</td>
<td>3968Kbps</td>
<td>0.840%</td>
<td>3690Kbps</td>
<td>native</td>
</tr>
<tr>
<td>F+R Gruss, Maurice, Wagner, et al. 2016</td>
<td>2384Kbps</td>
<td>0.005%</td>
<td>2382Kbps</td>
<td>native</td>
</tr>
<tr>
<td>E+R Lipp et al. 2016</td>
<td>1141Kbps</td>
<td>1.100%</td>
<td>1041Kbps</td>
<td>native</td>
</tr>
<tr>
<td>P+P Liu et al. 2015</td>
<td>600Kbps</td>
<td>1.000%</td>
<td>552Kbps</td>
<td>virt</td>
</tr>
</tbody>
</table>
1. Quick Start
2. Measuring and exploiting timing leakage
3. CPU caches
4. Cache attacks
5. Cache covert channels
6. Cache template attacks
7. Page Deduplication Attacks
8. Bitflips!
9. How to exploit bit flips?
10. How to mitigate Rowhammer?
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

Shared 0x0

Cache

Victim address space

Shared 0x0

Cache is empty
Profiling Phase

Attacker triggers an event
Profiling Phase

Attacker address space

Cache

Victim address space

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

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

Cache

Victim address space

Repeat for all events
Profiling Phase

Attacker address space

Cache

Victim address space

Repeat for all events

Shared 0x0

Shared 0x0
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

![Image of profiling output showing hexadecimal values and text in a terminal window.]
Profiling a Single Event
Profiling a Single Event

[Image of a terminal window with a file listing and a code editor window with some text highlighted.]
Profiling a Single Event
Profiling a Single Event

The image shows a terminal window displaying several lines of text, indicating file paths related to a tool named `gedit`. The file paths are sorted alphabetically, and each line includes a number at the beginning, possibly indicating a version or a process index. The terminal window also contains a text editor window with a few lines of text highlighted in blue, suggesting some form of annotation or selection.
Profiling a Single Event
Profiling a Single Event
Profiling a Single Event
Profiling a Single Event
Profiling a Single Event

```
/usr/bin/gedit, 0x1d9c0, 0
/usr/bin/gedit, 0x1da00, 0
/usr/bin/gedit, 0x1da40, 0
/usr/bin/gedit, 0x1da80, 0
/usr/bin/gedit, 0x1dab0, 0
/usr/bin/gedit, 0x1db90, 0
/usr/bin/gedit, 0x1db40, 0
/usr/bin/gedit, 0x1db80, 0
/usr/bin/gedit, 0x1dbc0, 0
/usr/bin/gedit, 0x1dc00, 0
/usr/bin/gedit, 0x1dc40, 0
/usr/bin/gedit, 0x1dc80, 0
/usr/bin/gedit, 0x1dcc0, 0
/usr/bin/gedit, 0x1dd00, 0
```

```
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

Plain Text  
Tab Width: 2  
Ln 1, Col 240  
INS
Profiling a Single Event
Profiling a Single Event
Profiling a Single Event

![Image of profiling output]

File: /usr/bin/gedit, 0x1da80, 0
File: /usr/bin/gedit, 0x1dac0, 0
File: /usr/bin/gedit, 0x1db00, 0
File: /usr/bin/gedit, 0x1db40, 0
File: /usr/bin/gedit, 0x1db80, 0
File: /usr/bin/gedit, 0x1dc00, 0
File: /usr/bin/gedit, 0x1dc40, 0
File: /usr/bin/gedit, 0x1dc80, 0
File: /usr/bin/gedit, 0x1dcc0, 0
File: /usr/bin/gedit, 0x1dd00, 0
File: /usr/bin/gedit, 0x1dd40, 0
File: /usr/bin/gedit, 0x1dd80, 0
File: /usr/bin/gedit, 0x1ddc0, 0
Profiling a Single Event

![Image of profiling output with hexadecimal values and basic text editor interface]
Profiling a Single Event
Profiling a Single Event

![Image of code output]

**File**
/usr/bin/gedit, 0x1db80, 0
/usr/bin/gedit, 0x1dbc0, 0
/usr/bin/gedit, 0x1dc00, 0
/usr/bin/gedit, 0x1dc40, 0
/usr/bin/gedit, 0x1dc80, 0
/usr/bin/gedit, 0x1dcc0, 0
/usr/bin/gedit, 0x1dd00, 0
/usr/bin/gedit, 0x1dd40, 0
/usr/bin/gedit, 0x1dd80, 0
/usr/bin/gedit, 0x1de00, 0
/usr/bin/gedit, 0x1de40, 0
/usr/bin/gedit, 0x1de80, 0
/usr/bin/gedit, 0x1dec0, 0

**Code Output**
```
1 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
   aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
   aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
   aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
   aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
   aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
   aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
   aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
   aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
   aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```
Profiling a Single Event
Profiling a Single Event
Profiling a Single Event

![Image of Gedit text editor with hexadecimal values]
Profiling a Single Event
Profiling a Single Event

![Image showing profiling output]
Profiling a Single Event

<table>
<thead>
<tr>
<th>File Path</th>
<th>Address</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>/usr/bin/gedit, 0x1dd00</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>/usr/bin/gedit, 0x1dd40</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>/usr/bin/gedit, 0x1dd80</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>/usr/bin/gedit, 0x1ddc0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>/usr/bin/gedit, 0x1d000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>/usr/bin/gedit, 0x1d040</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>/usr/bin/gedit, 0x1d080</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>/usr/bin/gedit, 0x1dec0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>/usr/bin/gedit, 0x1df00</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>/usr/bin/gedit, 0x1df40</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>/usr/bin/gedit, 0x1df80</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>/usr/bin/gedit, 0x1e000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>/usr/bin/gedit, 0x1e040</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Profiling a Single Event
Profiling a Single Event
Profiling a Single Event
Profiling a Single Event
Profiling a Single Event

[Image of a terminal window showing memory addresses and characters]

File Edit View Search Terminal Help
/usr/bin/gedit, 0x1de80, 0
/usr/bin/gedit, 0x1dec0, 0
/usr/bin/gedit, 0x1df00, 0
/usr/bin/gedit, 0x1df40, 0
/usr/bin/gedit, 0x1df80, 0
/usr/bin/gedit, 0x1dfc0, 0
/usr/bin/gedit, 0x1e000, 0
/usr/bin/gedit, 0x1e040, 0
/usr/bin/gedit, 0x1e080, 0
/usr/bin/gedit, 0x1e0c0, 0
/usr/bin/gedit, 0x1e100, 0
/usr/bin/gedit, 0x1e140, 0
/usr/bin/gedit, 0x1e180, 0
/usr/bin/gedit, 0x1e1c0, 0
Profiling a Single Event
Profiling a Single Event
Profiling a Single Event

![Image showing a text editor interface with hexadecimal memory addresses and a code snippet highlighted in blue.]

1 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
   aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
   aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
   aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
   aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
   aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
   aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
   aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
   aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
   aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
   aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

Plain Text   Tab Width: 2   Ln 1, Col 262   INS
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 a graphical user interface showing file paths and text editing]

1 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
Profiling a Single Event

```
File Edit View Search Terminal Help
/usr/bin/gedit, 0x1e180, 0
/usr/bin/gedit, 0x1e1c0, 0
/usr/bin/gedit, 0x1e200, 0
/usr/bin/gedit, 0x1e240, 0
/usr/bin/gedit, 0x1e280, 0
/usr/bin/gedit, 0x1e2c0, 0
/usr/bin/gedit, 0x1e300, 0
/usr/bin/gedit, 0x1e340, 0
/usr/bin/gedit, 0x1e380, 0
/usr/bin/gedit, 0x1e3c0, 0
/usr/bin/gedit, 0x1e400, 0
/usr/bin/gedit, 0x1e440, 0
/usr/bin/gedit, 0x1e480, 0
/usr/bin/gedit, 0x1e4c0, 0
```

Plain Text  Tab Width: 2  Ln 1, Col 270  INS
Profiling a Single Event
Profiling a Single Event

![Profiling Example](image_url)

- Profiling output showing memory addresses and values.
- Text editor displaying a simple example of profiling results.
Profiling a Single Event
Profiling a Single Event

<table>
<thead>
<tr>
<th>File Path</th>
<th>Address</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>/usr/bin/gedit, 0x1e280</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>/usr/bin/gedit, 0x1e2c0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>/usr/bin/gedit, 0x1e300</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>/usr/bin/gedit, 0x1e340</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>/usr/bin/gedit, 0x1e380</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>/usr/bin/gedit, 0x1e3c0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>/usr/bin/gedit, 0x1e400</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>/usr/bin/gedit, 0x1e440</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>/usr/bin/gedit, 0x1e480</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>/usr/bin/gedit, 0x1e4c0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>/usr/bin/gedit, 0x1e500</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>/usr/bin/gedit, 0x1e540</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>/usr/bin/gedit, 0x1e580</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>/usr/bin/gedit, 0x1e5c0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
Profiling a Single Event

![Image of profiling output]

```
/usr/bin/gedit, 0x1e2c0, 0
/usr/bin/gedit, 0x1e300, 0
/usr/bin/gedit, 0x1e340, 0
/usr/bin/gedit, 0x1e380, 0
/usr/bin/gedit, 0x1e3c0, 0
/usr/bin/gedit, 0x1e400, 0
/usr/bin/gedit, 0x1e440, 0
/usr/bin/gedit, 0x1e480, 0
/usr/bin/gedit, 0x1e4c0, 0
/usr/bin/gedit, 0x1e500, 0
/usr/bin/gedit, 0x1e540, 0
/usr/bin/gedit, 0x1e580, 0
/usr/bin/gedit, 0x1e5c0, 0
/usr/bin/gedit, 0x1e600, 0
```
Profiling a Single Event

```
/usr/bin/gedit, 0x1e340, 0
/usr/bin/gedit, 0x1e380, 0
/usr/bin/gedit, 0x1e3c0, 0
/usr/bin/gedit, 0x1e400, 0
/usr/bin/gedit, 0x1e440, 0
/usr/bin/gedit, 0x1e480, 0
/usr/bin/gedit, 0x1e4c0, 0
/usr/bin/gedit, 0x1e500, 0
/usr/bin/gedit, 0x1e540, 0
/usr/bin/gedit, 0x1e580, 0
/usr/bin/gedit, 0x1e5c0, 0
/usr/bin/gedit, 0x1e600, 0
/usr/bin/gedit, 0x1e640, 0
/usr/bin/gedit, 0x1e680, 1
```
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

A screenshot showing a terminal window with file paths and a text editor window displaying some text. The terminal window shows file paths like `/usr/bin/gedit, 0x1e540, 0` and `/usr/bin/gedit, 0x1e580, 0`. The text editor window contains several lines of text with a highlighted section.
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

0x7c800

Key:

| A | B | C | D | E | F | 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

KEY

ADDRESS

Daniel Gruss, Graz University of Technology
October 13, 2017 — QSP Lab
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 uses T-Tables (precomputed from S-Boxes)

- 4 T-Tables

\[ T_0 \left[ k\{0,4,8,12\} \oplus p\{0,4,8,12\} \right] \]

\[ T_1 \left[ k\{1,5,9,13\} \oplus p\{1,5,9,13\} \right] \]

... 

- If we know which entry of \( T \) is accessed, we know the result of \( k_i \oplus p_i \).
- Known-plaintext attack (\( p_i \) is known) \( \rightarrow k_i \) can be determined
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

\[ k_0 = 0x00 \]

\[ k_0 = 0x55 \]

(transposed)
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
- Works on virtually all Intel CPUs
- Works even with unknown binaries
Conclusion

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

- Works on virtually all Intel CPUs
- Works even with unknown binaries
- Marks a change of perspective:
Conclusion

- Novel technique to find any cache side-channel leakage
  - Attacks
  - Detect vulnerabilities
- Works on virtually all Intel CPUs
- Works even with unknown binaries
- Marks a change of perspective:
  - Large scale analysis of binaries
  - Large scale automated attacks
1. Quick Start
2. Measuring and exploiting timing leakage
3. CPU caches
4. Cache attacks
5. Cache covert channels
6. Cache template attacks
7. Page Deduplication Attacks
8. Bitflips!
9. How to exploit bit flips?
10. How to mitigate Rowhammer?
Yet another cache: Linux page cache

- Files buffered page-wise in “page cache”
- Lower access time for frequently accessed data
Yet another cache: Linux page cache

- Files buffered page-wise in “page cache”
- Lower access time for frequently accessed data
- Use up all the memory
- Pages are freed on demand
- Deduplicate pages (copy-on-write)
Copy-on-Write

Virtual Address Space

Physical Address Space
Copy-on-Write

Virtual Address Space

Physical Address Space
Copy-on-Write

Process A

Virtual Address Space

Physical Address Space
Copy-on-Write

Virtual Address Space

Process A

Physical Address Space
Copy-on-Write
Copy-on-Write

Virtual Address Space

Process A

Physical Address Space
Copy-on-Write

Virtual Address Space

Physical Address Space

Process A

Process B

fork
Copy-on-Write

Virtual Address Space

Physical Address Space
Copy-on-Write

Virtual Address Space

Physical Address Space

Process A

Process B
Copy-on-Write
Copy-on-Write
Copy-on-Write

Process A

Virtual Address Space

Process B

Physical Address Space

Process B tries to write
Copy-on-Write

Process A

Virtual Address Space

Process B tries to write

copy

Physical Address Space

Process B

Daniel Gruss, Graz University of Technology
October 13, 2017 — QSP Lab
Copy-on-Write

Virtual Address Space

Physical Address Space

Process A

Process B

write
Write vs. Copy-on-Write

- Regular write access < 0.2μs
- Write access with copy-on-write pagefault > 3.0μs
- Clearly distinguishable
Page Deduplication

Process A

Virtual Address Space

Process B

Physical Address Space
Page Deduplication

Processes started independently

Virtual Address Space

Physical Address Space

Process A

Process B
Page Deduplication

Virtual Address Space

Process A

Physical Address Space

Process B
Page Deduplication

Virtual Address Space

Process A

Process B

Physical Address Space
Page Deduplication

Virtual Address Space

Process A

Process B

Physical Address Space
Page Deduplication

Virtual Address Space

Process A

Physical Address Space

Process B

Process A

Physical Address Space

Process B
Page Deduplication

Virtual Address Space

Process A

Deduplication Thread

Process B

Physical Address Space
Page Deduplication

Virtual Address Space

Deduplication Thread

Physical Address Space
Page Deduplication

Virtual Address Space

Deduplication Thread

Physical Address Space
Page Deduplication

Virtual Address Space

Process A

Process B

Deduplication Thread

Physical Address Space

\[\neq\]
Page Deduplication

Virtual Address Space

Deduplication Thread

Physical Address Space
Page Deduplication

Virtual Address Space

Process A

Process B

Deduplication Thread

Physical Address Space
Page Deduplication

Virtual Address Space

Process A

Deduplication Thread

≠

Process B

Physical Address Space
Page Deduplication

Virtual Address Space

Process A

Process B

Deduplication Thread

Physical Address Space
Page Deduplication

Virtual Address Space

Deduplication Thread

Physical Address Space
Page Deduplication

Virtual Address Space

Deduplication Thread

Physical Address Space
Page Deduplication

Virtual Address Space

Process A

Deduplication Thread

Done!

Process B

Physical Address Space
Page Deduplication

Virtual Address Space

Process A

Deduplication Thread

Process B

Physical Address Space
Page Deduplication (without fork)

- Deduplication between processes:
  1. in same OS instance (Android, Windows)
  2. in different VMs (KVM, VMWare, ...)

Time until deduplication 2-45 minutes depends on system configuration.
Page Deduplication (without fork)

- Deduplication between processes:
  1. in same OS instance (Android, Windows)
  2. in different VMs (KVM, VMWare, ...)
- Code pages, data pages - even kernel pages
- Time until deduplication 2-45 minutes
  - depends on system configuration
Page Deduplication Attack

Attacker Process A

Virtual Address Space

Benign Process B

Physical Address Space
Page Deduplication Attack

Attacker Process A

Virtual Address Space

Benign Process B

Attacker generates a page suspected in process B

Physical Address Space
Page Deduplication Attack

Attacker Process A

Virtual Address Space

Benign Process B

Physical Address Space
Page Deduplication Attack

Attacker Process A

Virtual Address Space

Attacker waits for deduplication

Benign Process B

Physical Address Space
Page Deduplication Attack

Attacker Process A

Virtual Address Space

Attacker waits for deduplication

Benign Process B

Physical Address Space

\[ t = \text{time}(); \]
\[ p[0] = p[0]; \]
\[ \Delta = \text{time}() - t; \]
Page Deduplication Attack

Virtual Address Space

Attacker Process A

Benign Process B

Physical Address Space

$\Delta$ in µs

Time

Daniel Gruss, Graz University of Technology
October 13, 2017 — QSP Lab
Page Deduplication Attack

Attacker Process A

Virtual Address Space

Benign Process B

Physical Address Space

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

Time

measure \( \Delta \)
Page Deduplication Attack

Attacker Process A

Virtual Address Space

Benign Process B

Physical Address Space

$\Delta$ in $\mu$s

Time

$\Delta$
Page Deduplication Attack

Attack Process A

Virtual Address Space

Benign Process B

Physical Address Space

Δ in µs

0

4

Time

measure ∆
Page Deduplication Attack

Attacker Process A

Virtual Address Space

Benign Process B

Physical Address Space

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

Time

\[ \Delta \]

measure \( \Delta \)
Page Deduplication Attack

Attacker Process A

Virtual Address Space

Benign Process B

Physical Address Space

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

Time

\[ \Delta \]

measure \( \Delta \)
Page Deduplication Attack

Attacker Process A

Virtual Address Space

Benign Process B

Physical Address Space

$\Delta$ in $\mu$s

Time

Daniel Gruss, Graz University of Technology
October 13, 2017 — QSP Lab
Page Deduplication Attack

Attacker Process A

Virtual Address Space

Benign Process B

Physical Address Space

$\Delta$ in $\mu$s

Time

$\Delta$ in $\mu$s

measure $\Delta$
Page Deduplication Attack

Attacker Process A

Virtual Address Space

Benign Process B

Physical Address Space

$\Delta$ in µs

Time

$\Delta$

measure $\Delta$
Page Deduplication Attack

Attacker Process A

Virtual Address Space

Benign Process B

Physical Address Space

$\Delta$ in µs

Time

$\Delta$

measure $\Delta$
Page Deduplication Attack

Attacker Process A

Virtual Address Space

Benign Process B

Physical Address Space

$\Delta$ in µs

Time

measure $\Delta$
Page Deduplication Attack

Attacker Process A

Benign Process B

Virtual Address Space

Physical Address Space

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

Time

Daniel Gruss, Graz University of Technology
October 13, 2017 — QSP Lab
Page Deduplication Attack

Virtual Address Space

Physical Address Space

Attacker Process A

Benign Process B

$\Delta$ in $\mu$s

Time

measure $\Delta$
Page Deduplication Attack

Attacker Process A

Virtual Address Space

Benign Process B

Physical Address Space

Δ in μs

Time

measure Δ
Page Deduplication Attack

Attacker Process A

Virtual Address Space

Benign Process B

Physical Address Space

$\Delta$ in $\mu$s

Time

$\Delta$
Page Deduplication Attack

Attacker Process A

Virtual Address Space

Benign Process B

Physical Address Space

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

Time

$\Delta$

measure $\Delta$
Page Deduplication Attack

Virtual Address Space

Physical Address Space

Attacker Process A

Benign Process B

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

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

Time

measure \( \Delta \)
Page Deduplication Attack

Attacker Process A

Virtual Address Space

Benign Process B

Physical Address Space

$\Delta$ in $\mu$s

Time

measure $\Delta$
Page Deduplication Attack

Attacker Process A

Virtual Address Space

Benign Process B

Physical Address Space

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

Time

\[ \Delta \neq \]

measure \( \Delta \)

Daniel Gruss, Graz University of Technology
October 13, 2017 — QSP Lab
Page Deduplication Attack

Attacker Process A

Virtual Address Space

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

Time

\[ \neq \]

Benign Process B

Physical Address Space

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

Daniel Gruss, Graz University of Technology
October 13, 2017 — QSP Lab
Page Deduplication Attack

Attacker Process A

Virtual Address Space

Benign Process B

Physical Address Space

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

Time

\[ \neq \]

measure \( \Delta \)
Page Deduplication Attack

Attacker Process A

Virtual Address Space

Benign Process B

Physical Address Space

Δ in µs

Time

Measure Δ

Daniel Gruss, Graz University of Technology
October 13, 2017 — QSP Lab
Page Deduplication Attack

Attacker Process A

Virtual Address Space

Benign Process B

Physical Address Space

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

\[ \Delta \neq \]

Time

Daniel Gruss, Graz University of Technology
October 13, 2017 — QSP Lab
Page Deduplication Attack

Attacker Process A

Virtual Address Space

Benign Process B

Physical Address Space

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

Time

\[ \neq \]

measure \( \Delta \)
Page Deduplication Attack

Attacker Process A

Virtual Address Space

Benign Process B

Δ in μs

Time

Physical Address Space
Page Deduplication Attack

Attacker Process A

Virtual Address Space

Benign Process B

Physical Address Space

$\Delta$ in $\mu$s

Time

Daniel Gruss, Graz University of Technology
October 13, 2017 — QSP Lab
Page Deduplication Attack

Attacker Process A

Virtual Address Space

Benign Process B

Δ in µs

Time

Physical Address Space

Daniel Gruss, Graz University of Technology
October 13, 2017 — QSP Lab
Page Deduplication Attack

Attacker Process A

Virtual Address Space

Benign Process B

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

\[ \text{Time} \]

write and measure \( \Delta \)

Physical Address Space
Page Deduplication Attack

Attacker Process A

Virtual Address Space

Benign Process B

Δ in µs

Copy

Physical Address Space

Write and measure Δ

Time

Daniel Gruss, Graz University of Technology
October 13, 2017 — QSP Lab
Page Deduplication Attack

Attacker Process A

Virtual Address Space

Benign Process B

Physical Address Space

Δ in µs

Time

write

Daniel Gruss, Graz University of Technology
October 13, 2017 — QSP Lab
Page Deduplication Attack

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

Attacker learns that another process had an identical page

Attacker Process A

Virtual Address Space

Benign Process B

Physical Address Space

Daniel Gruss, Graz University of Technology
October 13, 2017 — QSP Lab
Page Deduplication Attack

Attacker Process A

Virtual Address Space

Benign Process B

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

Time

Attacker learns that another process had an identical page

Physical Address Space
Page Deduplication Attack

Attacker Process A

Virtual Address Space

Benign Process B

Attacker learns that another process had an identical page

Physical Address Space

Δ in µs

Time

Daniel Gruss, Graz University of Technology
October 13, 2017 — QSP Lab
Page Deduplication Attack

Attacker Process A

Virtual Address Space

Benign Process B

Physical Address Space

Attacker learns that another process had an identical page

$\Delta$ in $\mu$s

Time

Daniel Gruss, Graz University of Technology
October 13, 2017 — QSP Lab
Page Deduplication Attack

Attacker Process A

Virtual Address Space

Benign Process B

Attacker learns that another process had an identical page

Physical Address Space

Time

$\Delta$ in µs

0

4

Daniel Gruss, Graz University of Technology
October 13, 2017 — QSP Lab
What can be attacked?

- Detect binary versions in co-located VMs
- Detect downloaded image in Firefox under certain conditions
  → Attacks on hypervisors
- Native code only

What can be attacked?

- Detect CSS files and images of opened websites
  - Chrome, Firefox and Internet Explorer
- Perform the attack in JavaScript
  → Attacks on KVM, Windows 8.1 and Android
Attacking Browsers

- Images and CSS files are page-aligned in memory
- Load them into memory for all websites of interest
- Detect deduplication
  → Malicious ad networks: alternative to tracking pixels?
Detect Image (Native, Cross-VM, KVM)

- Image not loaded
- Image loaded

Cycles

Page
Challenges of JavaScript-based attacks

- No cycle counting (`rdtsc`)
- No access to virtual addresses
Page Deduplication Attacks in JavaScript

- Only require microsecond accuracy
  - `performance.now()` is accurate enough
  - Can even work with millisecond accuracy
    - Accumulate time difference
    - Only possible with enough image/CSS data

- Large typed arrays are allocated page-aligned
Detect Image (JavaScript, Cross-VM, KVM)

Image not loaded  Image loaded

Nanoseconds

Page

10^5
10^4
10^3
10^2
500 1,000 1,500 2,000 2,500 3,000 3,500

Daniel Gruss, Graz University of Technology
October 13, 2017 — QSP Lab
Detection of Open Websites

- Attacker chosen set of websites
- Load website images and CSS files into arrays
- Reuse HTTP headers of system under attack
Countermeasures

JavaScript:
- Reduce timer accuracy?
- Prevent page-aligned arrays?
- Website diversification?
- Prevent control over full pages
  - Every $n$-th byte not part of JavaScript array
Countermeasures

JavaScript:

- Reduce timer accuracy?
- Prevent page-aligned arrays?
- Website diversification?
- Prevent control over full pages
  - Every $n$-th byte not part of JavaScript array

Generic:

- Disable page deduplication (for writable pages)
1. Quick Start
2. Measuring and exploiting timing leakage
3. CPU caches
4. Cache attacks
5. Cache covert channels
6. Cache template attacks
7. Page Deduplication Attacks
8. Bitflips!
9. How to exploit bit flips?
10. How to mitigate Rowhammer?
DRAM organization
DRAM organization

channel 0

channel 1
DRAM organization

channel 0

back of DIMM: rank 1

channel 1

front of DIMM: rank 0
DRAM organization

channel 0

back of DIMM: rank 1

channel 1

front of DIMM: rank 0

chip
DRAM organization

- bits in cells in rows
- access: activate row, copy to row buffer

bank 0

row 0
row 1
row 2
...
row 32767

row buffer
How reading from DRAM works

CPU wants to access row 1

row buffer

DRAM bank
How reading from DRAM works

CPU wants to access row 1 → row 1 activated
How reading from DRAM works

CPU wants to access row 1
→ row 1 activated
→ row 1 copied to row buffer
How reading from DRAM works

CPU wants to access row 1
→ row 1 activated
→ row 1 copied to row buffer
How reading from DRAM works

DRAM bank

CPU wants to access row 2

row buffer
How reading from DRAM works

DRAM bank

CPU wants to access row 2
→ row 2 activated

...
How reading from DRAM works

CPU wants to access row 2
→ row 2 activated
→ row 2 copied to row buffer
How reading from DRAM works

CPU wants to access row 2
→ row 2 activated
→ row 2 copied to row buffer
How reading from DRAM works

CPU wants to access row 2
→ row 2 activated
→ row 2 copied to row buffer
→ slow (row conflict)
How reading from DRAM works

DRAM bank

CPU wants to access row 2—again

row buffer
How reading from DRAM works

DRAM bank

CPU wants to access row 2—again
→ row 2 already in row buffer
How reading from DRAM works

CPU wants to access row 2—again
⇒ row 2 already in row buffer
How reading from DRAM works

CPU wants to access row 2—again
→ row 2 already in row buffer
→ fast (row hit)
How reading from DRAM works

DRAM bank

row buffer = cache

row buffer
DRAM refresh

- cells leak $\rightarrow$ repetitive refresh necessary
- refresh $\approx$ reading (destructive) + writing same data again
- maximum interval between refreshes to guarantee data integrity
DRAM refresh

- cells leak $\rightarrow$ repetitive refresh necessary
- refresh $\approx$ reading (destructive) + writing same data again
- maximum interval between refreshes to guarantee data integrity
- cells leak faster upon proximate accesses $\rightarrow$ Rowhammer
Rowhammer

“It’s like breaking into an apartment by repeatedly slamming a neighbor’s door until the vibrations open the door you were after” – Motherboard Vice

```
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
...```

```
1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

row buffer
Rowhammer

“It’s like breaking into an apartment by repeatedly slamming a neighbor’s door until the vibrations open the door you were after” – Motherboard Vice

Diagram of Rowhammer process:
- **DRAM bank** with rows of bits:
  - Row buffer
  - Activate:
  - Copy:

Diagram showing memory addresses and bit manipulation.
Rowhammer

“It’s like breaking into an apartment by repeatedly slamming a neighbor’s door until the vibrations open the door you were after” – Motherboard Vice

DRAM bank

activate

...  

row buffer

copy
Rowhammer

“It’s like breaking into an apartment by repeatedly slamming a neighbor’s door until the vibrations open the door you were after” – Motherboard Vice
Rowhammer

“It’s like breaking into an apartment by repeatedly slamming a neighbor’s door until the vibrations open the door you were after” – Motherboard Vice

DRAM bank

activate

row buffer

copy
Rowhammer

“It’s like breaking into an apartment by repeatedly slamming a neighbor’s door until the vibrations open the door you were after” – Motherboard Vice
Requirements

Memory accesses must be

- **uncached**: reach DRAM
- **fast**: race against the next row refresh
- **targeted**: reach specific row
How do we get enough uncached accesses?
Impact of the CPU cache

- only non-cached accesses reach DRAM
Impact of the CPU cache

- only non-cached accesses reach DRAM
- either remove data from cache
Impact of the CPU cache

- only non-cached accesses reach DRAM
- either remove data from cache
- or don’t put it there in the first place
Impact of the CPU cache

- only non-cached accesses reach DRAM
- either remove data from cache
- or don’t put it there in the first place
  → next access will be served from DRAM
Access techniques

1. `clflush` instruction → original paper (Kim et al. 2014)
2. cache eviction (Gruss, Maurice, and Mangard 2016; Aweke et al. 2016)
3. non-temporal accesses (Qiao and Seaborn 2016)
4. uncached memory (Veen et al. 2016)
#1 Hammering with clflush

cache set 1

cache set 2

DRAM bank
#1 Hammering with clflush

- cache set 1
- cache set 2
- DRAM bank

Daniel Gruss, Graz University of Technology
October 13, 2017 — QSP Lab
#1 Hammering with clflush

cache set 1

(cache set 2)

DRAM bank
#1 Hammering with clflush

cache set 1

cache set 2

DRAM bank
#1 Hammering with `clflush`

cache set 1

cache set 2

DRAM bank
#1 Hammering with \texttt{clflush}

![Diagram showing cache sets and DRAM bank with reload arrows]

- cache set 1
- cache set 2
- DRAM bank

**reload**

Daniel Gruss, Graz University of Technology
October 13, 2017 — QSP Lab
#1 Hammering with clflush

![Diagram showing cache sets and clflush operations]
#1 Hammering with clflush

```
cache set 1

---

cache set 2

---

DRAM bank
```

reload

reload
#1 Hammering with clflush

![Diagram showing cache sets and DRAM bank with clflush operations]

- cache set 1
- cache set 2
- DRAM bank

Daniel Gruss, Graz University of Technology
October 13, 2017 — QSP Lab
#1 Hammering with clflush

cache set 1

DRAM bank

cache set 2

reload

reload
#1 Hammering with clflush

cache set 1

cache set 2

DRAM bank

cflush
cflush
cflush
#1 Hammering with clflush

```
cache set 1

cache set 2

DRAM bank
```

reload

reload
#1 Hammering with clflush

- cache set 1
- cache set 2
- DRAM bank
  - clflush
  - wait for it...

Daniel Gruss, Graz University of Technology
October 13, 2017 — QSP Lab
#1 Hammering with clflush

cache set 1

cache set 2

DRAM bank

reload

bit flip!
How widespread is the issue?

**DDR3:**
- Kim et al.: 110/129 modules from 3 vendors, all but 3 since mid-2011
- Seaborn and Dullien: 15/29 laptops

**DDR4 believed to be safe:**
- we showed bit flips (Pessl et al. 2016)

Figure: *
Flush, reload, flush, reload. . .

- the core of Rowhammer is essentially a Flush+Reload loop
- as much an attack on DRAM as on cache
#2 Hammering with cache eviction

- idea: avoid `clflush` to be independent of specific instructions
  → no `clflush` in JavaScript
#2 Hammering with cache eviction

- idea: avoid `clflush` to be independent of specific instructions
  → no `clflush` in JavaScript
- our approach: use regular memory accesses for eviction
  → techniques from cache attacks!
#2 Hammering with cache eviction

- idea: avoid `clflush` to be independent of specific instructions
  - → no `clflush` in JavaScript

- our approach: use regular memory accesses for eviction
  - → techniques from cache attacks!
  - → Rowhammer, Prime+Probe style!
#2 Hammering with cache eviction

DRAM bank

- cache set 1
- cache set 2
#2 Hammering with cache eviction

DRAM bank

cache set 1

load

cache set 2

load
#2 Hammering with cache eviction

- cache set 1
- cache set 2
- DRAM bank

Daniel Gruss, Graz University of Technology
October 13, 2017 — QSP Lab
#2 Hammering with cache eviction

cache set 1

cache set 2

DRAM bank
#2 Hammering with cache eviction

Cache set 1

Cache set 2

DRAM bank

load

load
#2 Hammering with cache eviction

Cache set 1

Cache set 2

DRAM bank
#2 Hammering with cache eviction

cache set 1

DRAM bank

cache set 2

load

load
#2 Hammering with cache eviction

cache set 1

DRAM bank

cache set 2

load

load
#2 Hammering with cache eviction

![Diagram showing cache sets and load operations](image-url)
#2 Hammering with cache eviction

cache set 1

cache set 2

DRAM bank

reload

reload
#2 Hammering with cache eviction

cache set 1

cache set 2

repeat!

DRAM bank
#2 Hammering with cache eviction

![Diagram showing cache eviction]

- **cache set 1**
- **cache set 2**
- **DRAM bank**

wait for it...
#2 Hammering with cache eviction

- cache set 1
- cache set 2

DRAM bank

bit flip!
Cache eviction strategies

Not as simple as that → replacement policy is not LRU
Cache eviction strategies

Not as simple as that → replacement policy is not LRU

→ fast and effective on Haswell: eviction rate >99.97%
Cache eviction strategies

Not as simple as that → replacement policy is not LRU

→ fast and effective on Haswell: eviction rate >99.97%
→ we evaluated 10 000+ strategies to find the best one
Hammering with cache eviction on Haswell

![Graph showing the number of bit flips within 15 minutes against refresh interval in μs (BIOS configuration).]
#3 Hammering with non-temporal accesses

- non-temporal accesses: data accessed just once, not in the future
- NTA instructions → **bypass cache** to minimize cache pollution
#3 Hammering with non-temporal accesses

- non-temporal accesses: data accessed just once, not in the future
- NTA instructions $\rightarrow$ **bypass cache** to minimize cache pollution
- NT stores to 1 address are combined at WC buffer
- only last write goes to DRAM $\rightarrow$ rate not sufficient
#3 Hammering with non-temporal accesses

- non-temporal accesses: data accessed just once, not in the future
- NTA instructions $\rightarrow$ bypass cache to minimize cache pollution
- NT stores to 1 address are combined at WC buffer
- only last write goes to DRAM $\rightarrow$ rate not sufficient
- following cached access to same address
#3 Hammering with non-temporal accesses

begin:
  movnti %eax, (X)
  movnti %eax, (Y)
  mov %eax, (X)
  mov %eax, (Y)
  jmp begin
#4 Hammering with uncached memory

Sometimes, everything fails,
#4 Hammering with uncached memory

Sometimes, everything fails, e.g., on mobile devices.
#4 Hammering with uncached memory

Sometimes, everything fails, e.g., on mobile devices

- ARMv7 flush instruction is privileged
#4 Hammering with uncached memory

Sometimes, everything fails, e.g., on mobile devices

- ARMv7 flush instruction is privileged
- cache eviction seems to be too slow
#4 Hammering with uncached memory

Sometimes, everything fails, e.g., on mobile devices

- ARMv7 flush instruction is privileged
- cache eviction seems to be too slow
- ARMv8 non-temporal stores are still cached in practice
#4 Hammering with uncached memory

- ION: memory management since Android 4.0
- apps can use `/dev/ion` for **uncached**, physically contiguous memory
- no privilege and no permission needed
How do we target accesses?
Physical addresses and DRAM

- fixed map: physical addresses $\rightarrow$ DRAM cells
- undocumented for Intel
- reverse-engineering for Sandy Bridge (Seaborn 2015)
- and by us for Sandy, Ivy, Haswell, Skylake, ... (Pessl et al. 2016)
- using the timing difference between row hits and row conflicts
Rowhammer preparations

For starting it’s easier with an empty file cache

```bash
sync && echo 3 | sudo tee /proc/sys/vm/drop_caches
```
and swap disabled
```bash
sudo swapoff -a
```
and with full CPU speed
```bash
sudo cpupower -c all set -b 0
```
How do I reverse my own DRAM?

https://github.com/IAIK/DRAMA

taskset 0x4 sudo ./measure -p 0.5 -s 16
# taskset core for stability
# sudo for pagemap access
# -p 0.5 allocate 50% of memory, the more the better
# -s I expect at least 16 sets (I have 32)
How do I flip bits?

https://github.com/IAIK/rowhammerjs

Copy functions from measure result

make ivy # or your microarchitecture
sudo ./rowhammer-ivy -d 2
# sudo for pagemap
# -d 2, for 2 DIMMs
sudo ./rowhammer-ivy -d 2 -f 0
# -f 0, only test offset 0 of every row
Rowhammer without `clflush`?

- idea: avoid `clflush` to be independent of specific instructions
  → no `clflush` in JavaScript
Rowhammer without `clflush`?

- idea: avoid `clflush` to be independent of specific instructions
  → no `clflush` in JavaScript

- our approach: use regular memory accesses for eviction
  → techniques from cache attacks!

Daniel Gruss, Graz University of Technology
October 13, 2017 — QSP Lab
Rowhammer without `clflush`?

- idea: avoid `clflush` to be independent of specific instructions
  - no `clflush` in JavaScript
- our approach: use regular memory accesses for eviction
  - techniques from cache attacks!
  - Rowhammer, Prime+Probe style!
Rowhammer without clflush

- Cache set 1
- Cache set 2

DRAM bank
Rowhammer without `clflush`

```
cache set 1

load

load

DRAM bank

load

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
Rowhammer without `clflush`

Cache set 1

Cache set 2

DRAM bank
Rowhammer without *clflush*

```
cache set 1

load

DRAM bank

cache set 2

load
```
Rowhammer without clflush

cache set 1

cache set 2

DRAM bank

load

load
Rowhammer without \texttt{clflush}
Rowhammer without clflush

cache set 1

load

DRAM bank

load

cache set 2
Rowhammer without clflush
Rowhammer without `clflush`

Cache set 1

cache set 2

DRAM bank

repeat!
Rowhammer without clflush

cache set 1

cache set 2

wait for it...
Rowhammer without clflush

cache set 1

cache set 2

DRAM bank

bit flip!
Requirements for Rowhammer

1. **uncached** memory accesses: need to reach DRAM
2. **fast** memory accesses: race against the next row refresh
Requirements for Rowhammer

1. **uncached** memory accesses: need to reach DRAM
2. **fast** memory accesses: race against the next row refresh

→ optimize the eviction rate and the timing
Rowhammer.js: the 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.js: the challenges

1. how to get accurate timing in JS? → easy
2. how to get physical addresses in JS?
3. which physical addresses to access?
4. in which order to access them?
Rowhammer.js: the 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?
4. in which order to access them?
Rowhammer.js: the 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?
Rowhammer.js: the 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? → already earlier today
Challenge #1: accurate timing in JavaScript?

- native code: rdtsc
- JavaScript: `window.performance.now()`
Challenge #1: accurate timing in JavaScript?

- native code: `rdtsc`
- JavaScript: `window.performance.now()`
- recent patch: time rounded to 5 microseconds
- still works: we measure millions of accesses
Challenge #2: physical addresses and JavaScript

- OS optimization: use 2MB pages
- last 21 bits (2MB) of physical address
- = last 21 bits (2MB) of virtual address
Challenge #2: physical addresses and JavaScript

- OS optimization: use 2MB pages
  - last 21 bits (2MB) of physical address
  - = last 21 bits (2MB) of virtual address
  - = last 21 bits (2MB) of JS array indices Gruss, Bidner, et al. 2015
Challenge #2: physical addresses and JavaScript

- OS optimization: use 2MB pages
  - last 21 bits (2MB) of **physical address**
  - = last 21 bits (2MB) of **virtual address**
  - = last 21 bits (2MB) of **JS array indices** Gruss, Bidner, et al. 2015

- several DRAM rows per 2MB page
- several congruent addresses per 2MB page
Challenge #3: physical addresses and DRAM

- fixed map: physical addresses $\rightarrow$ DRAM cells
- undocumented for Intel CPUs
- reverse-engineered for Sandy Bridge Seaborn 2015
- and by us for Sandy, Ivy, Haswell, Skylake, ... Pessl et al. 2016
Challenge #3: physical addresses and cache sets

- fixed map: physical addresses $\rightarrow$ cache sets
- undocumented for Intel CPUs but reverse-engineered Maurice, Le Scouarnec, et al. 2015
Challenge #4: replacement policy

→ fast and effective on Haswell: eviction rate >99.97%
Cache eviction strategy: New representation

- represent accesses as a sequence of numbers:
  1, 2, 1, 2, 2, 3, 2, 3, 3, 4, 3, 4, ...
- can be a long sequence
- all congruent addresses are indistinguishable w.r.t eviction strategy
Cache eviction strategy: New representation

- represent accesses as a sequence of numbers: 
  1, 2, 1, 2, 2, 3, 2, 3, 3, 4, 3, 4, ...
- can be a long sequence
- all congruent addresses are indistinguishable w.r.t. eviction strategy
  → adding more unique addresses can increase eviction rate
  → multiple accesses to one address can increase the eviction rate
  → indistinguishable → balanced number of accesses
Cache eviction strategy: Notation (1)

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

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

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

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

```plaintext
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

```
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: \( P-C-D-L-S \)

- \( S \): 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

```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 (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];
```

- \( P-2-2-1-4 \rightarrow 1, 2, 1, 2, 2, 3, 2, 3, 3, 4, 3, 4 \)
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];
```

- $P-2-2-1-4 \rightarrow 1, 2, 1, 2, 2, 3, 2, 3, 3, 4, 3, 4$  
  $S = 4$
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];
```

- \( P - 2 - 2 - 1 - 4 \) → 1, 2, 1, 2, 2, 3, 2, 3, 3, 4, 3, 4
  
  \( S = 4 \)
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];
```

$P - 2 - 2 - 1 - 4 \rightarrow 1, 2, 1, 2, 2, 3, 2, 3, 3, 4, 3, 4$

$S = 4$

$D = 2$
Cache eviction strategy: Notation (2)

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

- \(P-2-2-1-4 \rightarrow 1, 2, 1, 2, 2, 3, 2, 3, 3, 4, 3, 4\)
- \(S = 4\)
- \(D = 2\)
- \(C = 2\)
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];

\[ \text{P-2-2-1-4} \rightarrow 1, 2, 1, 2, 2, 3, 2, 3, 3, 4, 3, 4 \]

\[ L = 1, D = 2, C = 2, S = 4 \]
Cache eviction strategy: Notation (2)

\[
\text{for } (s = 0; s \leq S - D; s += L) \\
\quad \text{for } (c = 1; c \leq C; c += 1) \\
\quad \text{for } (d = 1; d \leq D; d += 1) \\
\quad \ast a[s+d];
\]

- \(P-2-2-1-4 \rightarrow 1, 2, 1, 2, 2, 3, 2, 3, 3, 4, 3, 4\) 
  \(S = 4\) 
  \(L = 1\) 
  \(D = 2\) 
  \(C = 2\)

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

We evaluated more than 10000 strategies...¹

<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>

¹ Daniel Gruss, Graz University of Technology
October 13, 2017 — QSP Lab
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>

Daniel Gruss, Graz University of Technology
October 13, 2017 — QSP Lab
Cache eviction strategies: Evaluation

We evaluated more than 10000 strategies...¹

<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>

¹ Daniel Gruss, Graz University of Technology
October 13, 2017 — QSP Lab
Cache eviction strategies: Evaluation

We evaluated more than 10000 strategies...\(^1\)

<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></td>
<td></td>
</tr>
</tbody>
</table>

Daniel Gruss, Graz University of Technology
October 13, 2017 — QSP Lab
Cache eviction strategies: Evaluation

We evaluated more than 10000 strategies...\(^1\)

<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></td>
</tr>
</tbody>
</table>
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>
Cache eviction strategies: Evaluation

We evaluated more than 10000 strategies...\(^1\)

<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></td>
<td></td>
</tr>
</tbody>
</table>
Cache eviction strategies: Evaluation

We evaluated more than 10000 strategies... \(^1\)

<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>
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>$P\text{-}1\text{-}1\text{-}1\text{-}1\text{-}17$</td>
<td>17</td>
<td>74.46%</td>
<td>307 ns</td>
</tr>
<tr>
<td>$P\text{-}1\text{-}1\text{-}1\text{-}20$</td>
<td>20</td>
<td>99.82%</td>
<td>934 ns</td>
</tr>
<tr>
<td>$P\text{-}2\text{-}1\text{-}1\text{-}17$</td>
<td>34</td>
<td>99.86%</td>
<td>191 ns</td>
</tr>
<tr>
<td>$P\text{-}2\text{-}2\text{-}1\text{-}17$</td>
<td>64</td>
<td>99.98%</td>
<td>180 ns</td>
</tr>
</tbody>
</table>
Cache eviction strategies: Evaluation

We evaluated more than 10000 strategies...¹

<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?
Cache eviction strategies: Illustration

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

\[ P-2-1-1-17 \text{ (34 accesses, 191ns)} \]
Cache eviction strategies: Illustration

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

$P-2-1-1-17$ (34 accesses, 191ns)
Cache eviction strategies: Illustration

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

\[ P-2-1-1-17 \ (34 \text{ accesses, } 191\text{ns}) \]
Cache eviction strategies: Illustration

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

\[ P-2-1-1-17 \text{ (34 accesses, 191ns)} \]
Cache eviction strategies: Illustration

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

\[ P-2-1-1-17 \text{ (34 accesses, 191ns)} \]
Cache eviction strategies: Illustration

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

$P-2-1-1-17$ (34 accesses, 191ns)
Cache eviction strategies: Illustration

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

\[ P-2-1-1-17 \ (34 \text{ accesses}, \ 191\text{ns}) \]

Time in ns
Cache eviction strategies: Illustration

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

\[ P-2-1-1-17 \text{ (34 accesses, 191ns)} \]
Cache eviction strategies: Illustration

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

\[ P-2-1-1-17 \text{ (34 accesses, 191ns)} \]
Cache eviction strategies: Illustration

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

\[ P-2-1-1-17 \] (34 accesses, 191\,\text{ns})

Time in \text{ns}
Cache eviction strategies: Illustration

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

$P-2-1-1-17$ (34 accesses, 191ns)
Cache eviction strategies: Illustration

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

\[ P-2-1-1-17 \] (34 accesses, 191ns)

Time in ns
Cache eviction strategies: Illustration

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

$P-2-1-1-17$ (34 accesses, 191ns)

Time in ns
Cache eviction strategies: Illustration

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

$P-2-1-1-17$ (34 accesses, 191ns)
Cache eviction strategies: Illustration

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

\[ P-2-1-1-17 \text{ (34 accesses, 191ns)} \]

Time in ns
Cache eviction strategies: Illustration

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

\[ P-2-1-1-17 \text{ (34 accesses, 191ns)} \]
Cache eviction strategies: Illustration

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

**P-2-1-1-17** (34 accesses, 191ns)
Cache eviction strategies: Illustration

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

$P-2-1-1-17$ (34 accesses, 191ns)
Cache eviction strategies: Illustration

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

$P-2-1-1-17$ (34 accesses, 191ns)
Cache eviction strategies: Illustration

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

\( P-2-1-1-17 \) (34 accesses, 191ns)

Time in ns
Cache eviction strategies: Illustration

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

\[ P-2-1-1-17 \text{ (34 accesses, 191ns)} \]
Cache eviction strategies: Illustration

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

```
Miss (intended) | Miss (intended) | Miss | Miss | Miss | Hit | Miss
```

**P-2-1-1-17 (34 accesses, 191ns)**

```
Miss (intended) | Miss (intended) | Miss | Miss | Miss | Miss | Miss
```

Time in ns
Cache eviction strategies: Illustration

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

\[ P-2-1-1-17 \text{ (34 accesses, 191ns)} \]
Cache eviction strategies: Illustration

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

$P-2-1-1-17$ (34 accesses, 191ns)
Cache eviction strategies: Illustration

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

$P-2-1-1-17$ (34 accesses, 191ns)

Time in ns
Cache eviction strategies: Illustration

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

\[ P-2-1-1-17 \] (34 accesses, 191ns)

Time in ns
Cache eviction strategies: Illustration

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

\[ P-2-1-1-17 \text{ (34 accesses, 191ns)} \]

Time in ns
Cache eviction strategies: Illustration

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

```
Miss (intended) Miss (intended) Miss Miss Hit Miss
```

**P-2-1-1-17** (34 accesses, 191ns)

```
Miss (intended) Miss (intended) Miss Miss Miss Hit Hit Miss
```
Cache eviction strategies: Illustration

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

$P$-2-1-1-17 (34 accesses, 191ns)

Time in ns
Cache eviction strategies: Illustration

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

\[ P-2-1-1-17 \text{ (34 accesses, 191ns)} \]
Cache eviction strategies: Illustration

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

P-2-1-1-17 (34 accesses, 191ns)

Time in ns
Cache eviction strategies: Illustration

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

**P-2-1-1-17** (34 accesses, 191ns)

Time in ns
Cache eviction strategies: Illustration

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

\[ P-2-1-1-17 \text{ (34 accesses, 191ns)} \]
Cache eviction strategies: Illustration

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

\( P-2-1-1-17 \) (34 accesses, 191ns)

Time in ns
Cache eviction strategies: Illustration

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

\[ P-2-1-1-17 \text{ (34 accesses, 191ns)} \]

Time in ns
Cache eviction strategies: Illustration

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

$P-2-1-1-17 \ (34 \text{ accesses, } 191\text{ns})$

Time in ns
Cache eviction strategies: Illustration

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

\[ P-2-1-1-17 \] (34 accesses, 191\text{ns})
Cache eviction strategies: Illustration

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

**P-2-1-1-17** (34 accesses, 191ns)
Cache eviction strategies: Illustration

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

\textit{P-2-1-1-17} (34 accesses, 191\,ns)

Time in ns
Cache eviction strategies: Illustration

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

$P-2-1-1-17$ (34 accesses, 191ns)

Time in ns
Cache eviction strategies: Illustration

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

\[ P-2-1-1-17 \text{ (34 accesses, 191ns)} \]
Cache eviction strategies: Illustration

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

\(P-2-1-1-17\) (34 accesses, 191ns)
Cache eviction strategies: Illustration

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

\[
\begin{array}{cccccccccccccccc}
\text{Miss (intended)} & \text{Miss (intended)} & \text{Miss} & \text{Miss} & \text{Miss} & \text{Miss} & \text{Miss} & \text{Miss} & \text{Miss} & \text{Miss} & \text{Miss} & \text{Miss} & \text{Miss} & \text{Miss} & \text{Miss} & \text{Miss} \\
\end{array}
\]

\( P-2-1-1-17 \) (34 accesses, 191ns)

\[
\begin{array}{cccccccccccccccc}
\text{Miss (intended)} & \text{Miss (intended)} & \text{Miss} & \text{Miss} & \text{Miss} & \text{Miss} & \text{Miss} & \text{Miss} & \text{Miss} & \text{Miss} & \text{Miss} & \text{Miss} \\
\end{array}
\]

Time in ns
Execution time vs. bit flips

→ low execution time is better.
Eviction rate vs. bit flips

→ high eviction rate is better. Average: 73.96%.
## Eviction strategies on Haswell

Table: The fastest 5 eviction strategies with an eviction rate above 99.75% compared to \textit{clflush} and LRU eviction on Haswell.

<table>
<thead>
<tr>
<th>C</th>
<th>D</th>
<th>L</th>
<th>S</th>
<th>Accesses</th>
<th>Hits</th>
<th>Misses</th>
<th>Time (ns)</th>
<th>Eviction</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>2</td>
<td>2</td>
<td>60</td>
<td>99.9999%</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>2</td>
<td>18</td>
<td>90</td>
<td>34</td>
<td>4</td>
<td>179</td>
<td>99.9624%</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>17</td>
<td>64</td>
<td>35</td>
<td>5</td>
<td>180</td>
<td>99.9820%</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>17</td>
<td>34</td>
<td>47</td>
<td>5</td>
<td>191</td>
<td>99.8595%</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>2</td>
<td>18</td>
<td>108</td>
<td>34</td>
<td>5</td>
<td>216</td>
<td>99.9365%</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>17</td>
<td>17</td>
<td>96</td>
<td>13</td>
<td>307</td>
<td>74.4593%</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>2</td>
<td>20</td>
<td>80</td>
<td>41</td>
<td>23</td>
<td>329</td>
<td>99.7800%</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>20</td>
<td>20</td>
<td>187</td>
<td>78</td>
<td>934</td>
<td>99.8200%</td>
</tr>
</tbody>
</table>
Evaluation on Haswell

Figure: Number of bit flips within 15 minutes.
1. Quick Start
2. Measuring and exploiting timing leakage
3. CPU caches
4. Cache attacks
5. Cache covert channels
6. Cache template attacks
7. Page Deduplication Attacks
8. Bitflips!
9. How to exploit bit flips?
10. How to mitigate Rowhammer?
How to exploit random bit flips?

- They are not random → highly reproducible flip pattern!

1. choose a data structure that you can place at arbitrary memory locations
2. scan for “good” flips
3. place data structure there
4. trigger bit flip again
Strategy: Modify instructions

- idea from Seaborn and Dullien 2015
- x86 op codes are variable length
  - unsafe op codes (syscall) ∈ safe but long multi-byte op codes
  - only a problem with jumps to arbitrary addresses
- flip a bit in a validated NaCl instruction sequence
  - safe + validated jump → arbitrary jump
## Page Table Entries

<table>
<thead>
<tr>
<th>P</th>
<th>RW</th>
<th>US</th>
<th>WT</th>
<th>UC</th>
<th>R</th>
<th>D</th>
<th>S</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

Daniel Gruss, Graz University of Technology
October 13, 2017 — QSP Lab
Page Table Entries

<table>
<thead>
<tr>
<th>P</th>
<th>RW</th>
<th>US</th>
<th>WT</th>
<th>UC</th>
<th>R</th>
<th>D</th>
<th>S</th>
<th>G</th>
<th>Ignored</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Ignored

X
### Page Table Entries

<table>
<thead>
<tr>
<th>P</th>
<th>RW</th>
<th>US</th>
<th>WT</th>
<th>UC</th>
<th>R</th>
<th>D</th>
<th>S</th>
<th>G</th>
<th>Ignored</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Physical Page Number**

<table>
<thead>
<tr>
<th>Ignored</th>
<th>X</th>
</tr>
</thead>
</table>
Page Table Entries

Each 4 KB page table consists of 512 such entries
Page Table Manipulation

0x7000 – 0x7FFF
0x6000 – 0x6FFF
0x5000 – 0x5FFF
0x4000 – 0x4FFF
0x3000 – 0x3FFF
0x2000 – 0x2FFF
0x1000 – 0x1FFF
0x0 – 0xFFF

PTE 0
PTE 1
PTE 2
PTE 3
PTE 4
PTE 5
PTE 6
PTE 7

User Page
User Page
User Page
User Page
User Page
User Page
User Page
Kernel Page
Page Table
User Page
User Page
Page Table Manipulation

0x7000 – 0x7FFF
0x6000 – 0x6FFF
0x5000 – 0x5FFF
0x4000 – 0x4FFF
0x3000 – 0x3FFF
0x2000 – 0x2FFF
0x1000 – 0x1FFF
0x0 – 0xFFF

PTE 7
PTE 6
PTE 5
PTE 4
PTE 3
PTE 2
PTE 1
PTE 0

User Page
User Page
User Page
User Page
User Page
User Page
User Page
Kernel Page
Page Table
User Page
User Page
User Page
Page Table Manipulation

- **0x7000 – 0x7FFF** (User Page)
- **0x6000 – 0x6FFF** (User Page)
- **0x5000 – 0x5FFF** (User Page)
- **0x4000 – 0x4FFF** (User Page)
- **0x3000 – 0x3FFF** (User Page)
- **0x2000 – 0x2FFF** (Kernel Page)
- **0x1000 – 0x1FFF** (User Page)
- **0x0 – 0xFFF** (User Page)

**Page Table**

PTE 0

PTE 1

PTE 2

PTE 3

PTE 4

PTE 5

PTE 6

PTE 7

0x0 - 0xFFF
Page Table Manipulation

```
<table>
<thead>
<tr>
<th>Address Range</th>
<th>PTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x7000 – 0x7FFF</td>
<td>PTE 7</td>
</tr>
<tr>
<td>0x6000 – 0x6FFF</td>
<td>PTE 6</td>
</tr>
<tr>
<td>0x5000 – 0x5FFF</td>
<td>PTE 5</td>
</tr>
<tr>
<td>0x4000 – 0x4FFF</td>
<td>PTE 4</td>
</tr>
<tr>
<td>0x3000 – 0x3FFF</td>
<td>PTE 3</td>
</tr>
<tr>
<td>0x2000 – 0x2FFF</td>
<td>PTE 2</td>
</tr>
<tr>
<td>0x1000 – 0x1FFF</td>
<td>PTE 1</td>
</tr>
<tr>
<td>0x0 – 0xFFF</td>
<td>PTE 0</td>
</tr>
</tbody>
</table>
```

- User Page
- User Page
- User Page
- User Page
- User Page
- User Page
- Page Table
- User Page
- User Page
- Kernel Page
- User Page
- User Page

Daniel Gruss, Graz University of Technology
October 13, 2017 — QSP Lab
Page Table Manipulation

- 0x7000 – 0x7FFF
- 0x6000 – 0x6FFF
- 0x5000 – 0x5FFF
- 0x4000 – 0x4FFF
- 0x3000 – 0x3FFF
- 0x2000 – 0x2FFF
- 0x1000 – 0x1FFF
- 0x0 – 0xFFF

User Page

Kernel Page

Page Table

User Page

User Page

User Page

User Page

User Page

User Page

User Page

Daniel Gruss, Graz University of Technology
October 13, 2017 — QSP Lab
Page Table Manipulation
Search for page with flip

Hammering memory locations in different rows
Search for page with flip

Hammering memory locations in different rows
Search for page with flip

Hammering memory locations in different rows
Search for page with flip

Hammering memory locations in different rows
Search for page with flip

Hammering memory locations in different rows
Search for page with flip

Hammering memory locations in different rows
Release page with flip
Release page with flip
Fill all remaining memory with page tables
Fill all remaining memory with page tables
Page Table Manipulation

- PTE 0
- PTE 1
- PTE 2
- PTE 3
- PTE 4
- PTE 5
- PTE 6
- PTE 7

- User Page
- Kernel Page
- Page Table
- Page Table
- Page Table
- Page Table
- Page Table
- Page Table
- Page Table
- Page Table
- Page Table
- Page Table
- Page Table

Daniel Gruss, Graz University of Technology
October 13, 2017 — QSP Lab
Page Table Manipulation

0x7000 – 0x7FFF
0x6000 – 0x6FFF
0x5000 – 0x5FFF
0x4000 – 0x4FFF
0x3000 – 0x3FFF
0x2000 – 0x2FFF
0x1000 – 0x1FFF
0x0 – 0xFFF

PTE 7
PTE 6
PTE 5
PTE 4
PTE 3
PTE 2
PTE 1
PTE 0

Page Table
Page Table
User Page
Page Table
Page Table
Kernel Page
Page Table
Page Table
Page Table
Page Table
Page Table
Page Table
Page Table
Page Table
Page Table
Page Table Manipulation

- PTE 7
- PTE 6
- PTE 5
- PTE 4
- PTE 3
- PTE 2
- PTE 1
- PTE 0

Page Table
User Page
Page Table
Page Table
Kernel Page
Page Table
Page Table
Page Table
Strategy: Flipping Page Table PPN bits

1. scan for flips
2. exhaust or massage memory to place a page table at target location
3. gain access to your own page table → kernel privileges
Flipping Page Table PPN bits

- idea from Seaborn and Dullien 2015
- same idea applied in several other works:
  - Rowhammer.js (Gruss, Maurice, and Mangard 2016)
  - One bit flips, one cloud flops (Y. Xiao et al. 2016)
  - Drammer (Veen et al. 2016)
Post-Rowhammer Exploitation

- scan entire physical memory (very fast) and:
  - modify binary pages executed in root privileges (Y. Xiao et al. 2016)
  - modify credential structs (Veen et al. 2016)
  - read keys (Y. Xiao et al. 2016)
  - corrupt RSA signatures (Bhattacharya and Mukhopadhyay 2016)
  - modify certificates
  - configurations
  - etc.

- pages are pretty unique: 32768 bits per page
Bit Flips + Page Deduplication
Bit Flips + Page Deduplication

Page with bit flip is filled with target content
Bit Flips + Page Deduplication

OS or hypervisor searches for duplicate pages
Bit Flips + Page Deduplication

OS or hypervisor searches for duplicate pages
Bit Flips + Page Deduplication

OS or hypervisor searches for duplicate pages
Bit Flips + Page Deduplication

OS or hypervisor searches for duplicate pages
Bit Flips + Page Deduplication

OS or hypervisor searches for duplicate pages
Bit Flips + Page Deduplication

OS or hypervisor searches for duplicate pages
Bit Flips + Page Deduplication

OS or hypervisor searches for duplicate pages
Bit Flips + Page Deduplication

OS or hypervisor searches for duplicate pages
Bit Flips + Page Deduplication

Hammer again + flip again
Bit Flips + Page Deduplication
Strategy: Flipping in Deduplicated Pages

1. scan for flips
2. place content for deduplication so that flip can be exploited
3. perform the bit change through Rowhammer
Flipping in Deduplicated Pages

- idea from Bosman et al. 2016
  - change data type (double → pointer)
  - change pointer to good object → counterfeit object
- and from Razavi et al. 2016
  - corrupt authorized SSH keys
  - corrupt Debian update URLs + RSA public key file
1. Quick Start
2. Measuring and exploiting timing leakage
3. CPU caches
4. Cache attacks
5. Cache covert channels
6. Cache template attacks
7. Page Deduplication Attacks
8. Bitflips!
9. How to exploit bit flips?

10. How to mitigate Rowhammer?
Mitigations

Different mitigations have been proposed:

- detection vs prevention
- software vs hardware
- short-term vs long-term
Quick fixes

- no `clflush` instruction
Quick fixes

- no `clflush` instruction → Rowhammer.js
Quick fixes

- no `clflush` instruction → Rowhammer.js
- increase the refresh rate
Quick fixes

- no `clflush` instruction → Rowhammer.js
- increase the refresh rate
  → would need to be increased by $7 \times$ to eliminate all bit flips

Figure: 
Errors depending on refresh interval (Kim et al. 2014)
Quick fixes

- no `clflush` instruction → Rowhammer.js
- increase the refresh rate
  → would need to be increased by $7 \times$ to eliminate all bit flips
  → implementation: increased by $2 \times$ by BIOS vendors

Figure: *

Errors depending on refresh interval (Kim et al. 2014)
What about ECC?

- ECC protection: server can handle or correct single bit errors
What about ECC?

- ECC protection: server can handle or correct single bit errors
- no standard for event reporting
What about ECC?

- ECC protection: server can handle or correct single bit errors
- no standard for event reporting
- in practice (Lanteigne 2016)
  - common: server counts ECC errors and report only if they reach a threshold (e.g., > 100 bit flips / hour)
What about ECC?

- ECC protection: server can handle or correct single bit errors
- no standard for event reporting
- in practice (Lanteigne 2016)
  - common: server counts ECC errors and report only if they reach a threshold (e.g., > 100 bit flips / hour)
  - some server vendors never report errors to the OS
What about ECC?

- ECC protection: server can handle or correct single bit errors
- **no standard** for event reporting
- in practice (Lanteigne 2016)
  - common: server counts ECC errors and report only if they reach a threshold (e.g., > 100 bit flips / hour)
  - some server vendors **never report errors** to the OS
  - one server **did not even halt** when bit flips were non-correctable
Detecting Rowhammer attacks

- Rowhammer: lots of cache misses that can be monitored with hardware performance counters (Herath and Fogh 2015; Gruss, Maurice, Wagner, et al. 2016; Chiappetta et al. 2015; Payer 2016)
Preventing Rowhammer attacks in hardware (1/3)

Original ideas from Kim et al. 2014

- making better DRAM chips that are not vulnerable,
- using error correcting codes (ECC)
- increasing the refresh rate
- remapping/retiring faulty cells after manufacturing
- identifying hammered rows at runtime and refreshing neighbors
Preventing Rowhammer attacks in hardware (1/3)

Original ideas from Kim et al. 2014

- making better DRAM chips that are not vulnerable,
- using error correcting codes (ECC)
- increasing the refresh rate
- remapping/retiring faulty cells after manufacturing
- identifying hammered rows at runtime and refreshing neighbors

→ expensive, performance overhead, or increased power consumption
Preventing Rowhammer attacks in hardware (2/3)

PARA - Probabilistic Adjacent Row Activation (Kim et al. 2014)

- one row closed $\rightarrow$ one adjacent row opened with low probability $p$

Rowhammer: one row opened and closed a high number of times statistically, neighbor rows are refreshed $\rightarrow$ no bit flip

Implementation at the memory controller level

Advantage: stateless $\rightarrow$ not expensive for $p = 0.001$ and $N = 100K$, experiencing one error in one year has a probability $9.4 \times 10^{-14}$
Preventing Rowhammer attacks in hardware (2/3)

PARA - Probabilistic Adjacent Row Activation (Kim et al. 2014)

- one row closed $\rightarrow$ one adjacent row opened with low probability $p$
- Rowhammer: one row opened and closed a high number of times $N_{th}$
Preventing Rowhammer attacks in hardware (2/3)

PARA - Probabilistic Adjacent Row Activation (Kim et al. 2014)

- one row closed $\rightarrow$ one adjacent row opened with low probability $p$
- Rowhammer: one row opened and closed a high number of times $N_{th}$
- statistically, neighbor rows are refreshed $\rightarrow$ no bit flip
Preventing Rowhammer attacks in hardware (2/3)

PARA - Probabilistic Adjacent Row Activation (Kim et al. 2014)

- one row closed → one adjacent row opened with low probability $p$
- Rowhammer: one row opened and closed a high number of times $N_{th}$
- statistically, neighbor rows are refreshed → no bit flip
- implementation at the memory controller level
Preventing Rowhammer attacks in hardware (2/3)

PARA - Probabilistic Adjacent Row Activation (Kim et al. 2014)

- one row closed $\rightarrow$ one adjacent row opened with low probability $p$
- Rowhammer: one row opened and closed a high number of times $N_{th}$
- statistically, neighbor rows are refreshed $\rightarrow$ no bit flip
- implementation at the memory controller level
- advantage: stateless $\rightarrow$ not expensive
Preventing Rowhammer attacks in hardware (2/3)

PARA - Probabilistic Adjacent Row Activation (Kim et al. 2014)

- one row closed $\rightarrow$ one adjacent row opened with low probability $p$
- Rowhammer: one row opened and closed a high number of times $N_{th}$
- statistically, neighbor rows are refreshed $\rightarrow$ no bit flip
- implementation at the memory controller level
- advantage: stateless $\rightarrow$ not expensive
- for $p = 0.001$ and $N_{th} = 100K$, experiencing one error in one year has a probability $9.4 \times 10^{-14}$
Preventing Rowhammer attacks in hardware (3/3)

Target Row Refresh (TRR)

- counter per row
- increment neighbor rows
- refresh when counter reaches a threshold
## Preventing Rowhammer attacks in hardware (3/3)

### Target Row Refresh (TRR)

- counter per row
- increment neighbor rows
- refresh when counter reaches a threshold

![Diagram of Target Row Refresh (TRR)]
Preventing Rowhammer attacks in hardware (3/3)

Target Row Refresh (TRR)

- counter per row
- increment neighbor rows
- refresh when counter reaches a threshold
Preventing Rowhammer attacks in hardware (3/3)

Target Row Refresh (TRR)

- counter per row
- increment neighbor rows
- refresh when counter reaches a threshold
Preventing Rowhammer attacks in hardware (3/3)

Target Row Refresh (TRR)

- counter per row
- increment neighbor rows
- refresh when counter reaches a threshold

Diagram showing the concept of Target Row Refresh (TRR) with counters and refresh actions.
Preventing Rowhammer attacks in hardware (3/3)

Target Row Refresh (TRR)

- counter per row
- increment neighbor rows
- refresh when counter reaches a threshold
Preventing Rowhammer attacks in hardware (3/3)

Target Row Refresh (TRR)

- counter per row
- increment neighbor rows
- refresh when counter reaches a threshold
Preventing Rowhammer attacks in hardware (3/3)

Target Row Refresh (TRR)

- counter per row
- increment neighbor rows
- refresh when counter reaches a threshold
Preventing Rowhammer attacks in hardware (3/3)

Target Row Refresh (TRR)

- counter per row
- increment neighbor rows
- refresh when counter reaches a threshold
Target Row Refresh (TRR)

- counter per row
- increment neighbor rows
- refresh when counter reaches a threshold
Preventing Rowhammer attacks in hardware (3/3)

Target Row Refresh (TRR)

- counter per row
- increment neighbor rows
- refresh when counter reaches a threshold
Preventing Rowhammer attacks in hardware (3/3)

Target Row Refresh (TRR)

- counter per row
- increment neighbor rows
- refresh when counter reaches a threshold
Preventing Rowhammer attacks in software

“nohammer” kernel module Corbet 2016

- refresh rate of 8 ms would prevent Rowhammer on most systems
- use PMC to measure cache misses per 64 ms interval
- limit cache miss rate to 1/8 of maximum
Preventing Rowhammer attacks in software

“nohammer” kernel module Corbet 2016

- refresh rate of 8 ms would prevent Rowhammer on most systems
- use PMC to measure cache misses per 64 ms interval
- limit cache miss rate to 1/8 of maximum

Wait for refresh
Preventing Rowhammer attacks in software

“nohammer” kernel module Corbet 2016

- refresh rate of 8 ms would prevent Rowhammer on most systems
- use PMC to measure cache misses per 64 ms interval
- limit cache miss rate to 1/8 of maximum
Preventing Rowhammer attacks in software

“nohammer” kernel module Corbet 2016

- refresh rate of 8 ms would prevent Rowhammer on most systems
- use PMC to measure cache misses per 64 ms interval
- limit cache miss rate to 1/8 of maximum
Preventing Rowhammer attacks in software

“nohammer” kernel module Corbet 2016

- refresh rate of 8 ms would prevent Rowhammer on most systems
- use PMC to measure cache misses per 64 ms interval
- limit cache miss rate to 1/8 of maximum

Wait for refresh
Wait for refresh
Preventing Rowhammer attacks in software

“nohammer” kernel module Corbet 2016

- refresh rate of 8 ms would prevent Rowhammer on most systems
- use PMC to measure cache misses per 64 ms interval
- limit cache miss rate to 1/8 of maximum

Wait for refresh
Preventing Rowhammer attacks in software

“nohammer” kernel module Corbet 2016

- refresh rate of 8 ms would prevent Rowhammer on most systems
- use PMC to measure cache misses per 64 ms interval
- limit cache miss rate to 1/8 of maximum
Preventing Rowhammer attacks in software

“nohammer” kernel module Corbet 2016

- refresh rate of 8 ms would prevent Rowhammer on most systems
- use PMC to measure cache misses per 64 ms interval
- limit cache miss rate to 1/8 of maximum

Wait for refresh
Preventing Rowhammer attacks in software

“nohammer” kernel module Corbet 2016

- refresh rate of 8 ms would prevent Rowhammer on most systems
- use PMC to measure cache misses per 64 ms interval
- limit cache miss rate to 1/8 of maximum
Preventing Rowhammer attacks in software

MASCAT - Stopping Microarchitectural Attacks Before Execution (Irazoqui et al. 2016)

- static analysis of the binary
- detect suspicious instruction sequences (clflush, rdtsc, fences, ...)
- open problem: false positives
Preventing Rowhammer attacks in software

ANVIL (Aweke et al. 2016)

- uses performance counters to detect rowhammer
- activate rows neighbor rows to prevent flips
- similar as PARA, but in software
Preventing Rowhammer attacks in software

ANVIL (Aweke et al. 2016)

- uses performance counters to detect rowhammer
- activate rows neighbor rows to prevent flips
- similar as PARA, but in software
Preventing Rowhammer attacks in software

- B-CATT: disable vulnerable physical memory (Brasser et al. 2017)
- G-CATT: isolate security domains in physical memory based on potential vulnerability (Brasser et al. 2017)
Preventing Rowhammer attacks in software

- B-CATT: disable vulnerable physical memory (Brasser et al. 2017)
- G-CATT: isolate security domains in physical memory based on potential vulnerability (Brasser et al. 2017)
Conclusion

- Rowhammer attacks are easy to mount
- works on most systems (if you know the DRAM mapping)
- most countermeasures are too expensive or ineffective
Oh my Cache! 2
More fun with caches.

Daniel Gruss
Graz University of Technology

October 13, 2017 — QSP Lab
Bibliography I


Bibliography II

Brasser, Ferdinand, Lucas Davi, David Gens, Christopher Liebchen, and Ahmad-Reza Sadeghi (2017). “CAn’t Touch This: Software-only Mitigation against Rowhammer Attacks targeting Kernel Memory”. In: USENIX Security Symposium.


Corbet, Jonathan (2016). Defending against Rowhammer in the kernel. URL: https://lwn.net/Articles/704920/.

Bibliography III


Bibliography IV


Bibliography V


Maurice, Clémentine, Nicolas Le Scouarnec, Christoph Neumann, Olivier Heen, and Aurélien Francillon (2015). “Reverse Engineering Intel Complex Addressing Using Performance Counters”. In: RAID.

Maurice, Clémentine, Christoph Neumann, Olivier Heen, and Aurélien Francillon (2015). “C5: Cross-Cores Cache Covert Channel”. In: DIMVA’15.


Bibliography VIII


