Protecting Page Tables from RowHammer Attacks using Monotonic Pointers in DRAM True-Cells

Xin-Chuan Wu (University of Chicago)
Timothy Sherwood (University of California, Santa Barbara)
Frederic T. Chong (University of Chicago)
Yanjing Li (University of Chicago)

ASPLOS 2019 April 16 - Secure Memory Session
Outline

• Background

• Our technique: Cell-Type Aware (CTA) memory allocation

• Implementation

• Evaluation and Results
RowHammer - Prevalent in Modern DRAM Systems

- Prevalent in Modern DRAM Systems
- Frequently open/close
- Accelerated charge leaking
- Bitflip Errors

[Kim 14]
RowHammer: Critical Security Challenge

• PTE (Page Table Entry) based privilege escalation attacks:
  • Allow attacker to access entire physical memory
  • Many real-world demonstrations

<table>
<thead>
<tr>
<th>Technique</th>
<th>[Seaborn 15]</th>
<th>[Xiao 16]</th>
<th>[Gruss 16]</th>
<th>[Veen 16]</th>
<th>[Cheng 18]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Victim Platform</td>
<td>x86</td>
<td>VM</td>
<td>x86</td>
<td>ARM</td>
<td>x86</td>
</tr>
</tbody>
</table>

• Attack time: less than 20 seconds.
RowHammer Countermeasures

- Countermeasures
  - Prevent bit-flips
    - HW changes
    - Power consumption
    - Performance overhead
  - Control bit-flips only in user data
    - Performance overhead
    - High SW complexity
  - Our work:
    - Allow bit-flips in the system
      - Low SW complexity
      - (18 lines of code change)
      - Provable security
      - No performance impact
Virtual Address Translation

- Key data structure: page table

- Page tables also specify access permissions

```
Virtual address

Physical memory

Object in Virtual space

Page Table Entry

Data Object

Read/write permissions are granted to the user process
```
RowHammer Attacks - Inducing Bit-Flips in PTE

- **PTE Self Reference**
  - PTE points to page tables
  - Instead of regular data

- **Necessary condition**
  - For all PTE-based privilege escalation attacks

---

- Gained access to the entire physical memory
- Read/write permissions are granted to the user process
Outline

• Background

• Our technique: Cell-Type Aware (CTA) memory allocation

• Implementation

• Evaluation and Results
Key Observation - Asymmetry in DRAM Cells

• Two types of memory cells in modern DRAM chips

• True-cells
  • Charged: logic 1
  • Discharged: logic 0
  • Charge leaking: ‘1’→‘0’ errors

• Anti-cells
  • Charged: logic 0
  • Discharged: logic 1
  • Charge leaking induces ‘0’→‘1’ errors
Large Blocks of True-Cells

- Cell type identifiable through system tests
- A row consists of the same memory cell type
- True-cell and anti-cell rows are interleaved
  - every $N$ physical DRAM rows
  - $N = 512$ being a common number

Physical memory:

```
<table>
<thead>
<tr>
<th>True-cell Rows</th>
<th>Anti-cell Rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>True-cell Rows</td>
<td></td>
</tr>
<tr>
<td></td>
<td>:</td>
</tr>
<tr>
<td>True-cell Rows</td>
<td>Anti-cell Rows</td>
</tr>
<tr>
<td>True-cell Rows</td>
<td></td>
</tr>
</tbody>
</table>
```
Cell-Type-Aware (CTA) Memory Allocation

• Key idea: leverage DRAM cell types to destroy the “self-reference” condition
  • All page tables must be allocated above a low water mark
  • All regular data objects must be allocated below the low water mark
  • Only true-cells are used above the low water mark
  • Monotonicity in page table pointers
    • Pointers only decrease in address value in the presence of RowHammer-induced bit-flips
CTA Destroys Self-Reference: The Proof

**Theorem (No Self-Reference Theorem).** Given that page tables are stored above a low water mark $P$, containing pointers to pages all below $P$, and that all pointers in page table entries are stored in true-cells, then no pointer $p$ can point back to any page table entry $e$ after a RowHammer attack.

**Proof.** $\forall p \in$ page table, where $p$ are the pointers to the memory addresses of allocated pages, $p < P$.

$\forall e$, where $e$ are memory addresses of entries in the page table, $e > P$.

Let $\gamma(p)$ be the value of pointer $p$ after a RowHammer bit-flip.

For $p$ stored in true-cells, ‘0’ bits in $p$ cannot change to ‘1’ bits.

Therefore, $\gamma(p) \leq p$

Since $\forall p < P$, then $\forall \gamma(p) < P$

Since $\forall e > P$, then $\forall \gamma(p) < \forall e$

Therefore, no $\gamma(p)$ can point to any page table entry $e$. □
CTA Destroys Self-Reference: Example
Outline

• Background

• Our technique: Cell-Type Aware (CTA) memory allocation

• Implementation

• Evaluation and Results
CTA Implementation in Linux

• Actual x86/Linux system prototype
• ZONE_PTP for page table page allocation

```
18 lines of code change
```
Memory Zone Map with CTA

• 32MB for ZONE_PTP is sufficient

• Typical true-cell blocks: 64MB
  • Enough to cover the entire ZONE_PTP
• Background

• Our technique: Cell-Type Aware (CTA) memory allocation

• Implementation

• Evaluation and Results
Security Evaluation

• Non-ideal reality: small probability of 0 → 1 errors in true-cells
  • 0.2 % (actual DRAM measurement data [Kim 14])

• CTA renders RowHammer PTE privilege escalation attacks impractical
  • Only 1 out of 204,000 systems is vulnerable to the attack
  • It takes many days to successfully attack a vulnerable system.

<table>
<thead>
<tr>
<th>Physical memory</th>
<th># of Exploitable PTEs</th>
<th>32MB PTP</th>
</tr>
</thead>
<tbody>
<tr>
<td>8GB</td>
<td>4.69×10^{-6}</td>
<td>230.7</td>
</tr>
<tr>
<td></td>
<td>Attack Time (Days)</td>
<td></td>
</tr>
<tr>
<td>16GB</td>
<td>6.03×10^{-6}</td>
<td>462.3</td>
</tr>
<tr>
<td></td>
<td>Attack Time (Days)</td>
<td></td>
</tr>
<tr>
<td>32GB</td>
<td>7.54×10^{-6}</td>
<td>925.5</td>
</tr>
<tr>
<td></td>
<td>Attack Time (Days)</td>
<td></td>
</tr>
</tbody>
</table>
Performance Evaluation

- Run-time measurement on System prototype
  - System 1
    - Ubuntu 14.04.5 (Linux kernel 4.4.0-124-generic)
    - Intel i7-6700 quad-core CPU at 3.4 GHz
    - 8GB physical memory
  - System 2
    - Ubuntu 16.04 (Linux kernel 4.4-0-141-generic)
    - Intel Xeon Silver 4110 32-core CPU at 2.1 GHz
    - 128GB physical memory

- Average performance impact of CTA vs. original system

<table>
<thead>
<tr>
<th>Workload</th>
<th>8GB System</th>
<th>128GB System</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPEC 2006</td>
<td>-0.07%</td>
<td>0.04%</td>
</tr>
<tr>
<td>Phoronix</td>
<td>-0.08%</td>
<td>0.25%</td>
</tr>
</tbody>
</table>
Future Work:
Broader Applicability of Data Monotonicity

• Effective solutions for various security problems
  • Permission vector protection
  • Countermeasure to coldboot attacks

• Efficient error detection
  • Place data on true-cells
  • Count hamming weight of data
    • Using only one instruction (popcnt, x86 or vcnt, ARM)
Conclusion

• Asymmetry in DRAM: enables data monotonicity

• CTA memory allocation
  • Countermeasure to RowHammer PTE privilege escalation attacks
    • Monotonicity in PTE pointers → Destroys PTE self-reference
  • Effective: provable secure
  • Extremely low-cost: 18 lines of code change, no HW/performance cost

• Broader applicability of data monotonicity