ATTACKING ARM POINTER AUTHENTICATION WITH SPECULATIVE EXECUTION

Joseph Ravichandran*, Weon Taek Na*, Jay Lang, Mengjia Yan

*Both authors contributed equally to this work.
Joseph Ravichandran
1st Year PhD Student, MIT

Weon Taek Na
1st Year PhD Student, MIT
Memory Corruption Attacks

Microarchitectural Attacks
Contributions

1. New way of thinking about compounding threat models.
2. Hardware bypass for ARM Pointer Authentication.
The idea in 60 seconds.
Memory Corruption

Read/ Write Memory → Change Function Pointer → Arbitrary Code Execution
Memory Corruption

Read/Write Memory → Change Function Execution → Arbitrary Code Execution

Pointer Authentication blocks changing pointers.
Memory Corruption

1. Read/Write Memory
2. Write function pointer with forged hash
3. Arbitrary Code Execution
Just bruteforce it, right?
Key Insight: Avoid crashes using speculative execution!
Agenda

1. Background
2. High Level View
3. Data Attack
4. Instruction Attack
5. Analysis
Buffer Overflow

<table>
<thead>
<tr>
<th>Buffer[0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffer[1]</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>Function Pointer</td>
</tr>
</tbody>
</table>
Buffer Overflow

<table>
<thead>
<tr>
<th>Buffer[0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffer[1]</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td><strong>Function Pointer</strong></td>
</tr>
</tbody>
</table>

Buffer Overflow overwrites the function pointer!
Let's fix this bug with **Pointer Authentication**.
ARM Pointer Authentication

\[ \text{PAC} = \text{crypto\_fn}(\text{pointer}, \text{salt}, \text{key}) \]
Buffer Overflow

<table>
<thead>
<tr>
<th>Buffer[0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffer[1]</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>PAC</td>
</tr>
<tr>
<td>Function Pointer</td>
</tr>
</tbody>
</table>
Buffer Overflow

Buffer[0]
Buffer[1]
...
PAC
Function Pointer

Buffer Overflow corrupts the PAC

Invalid PAC means we crash!
The goal is to reveal the PAC for an arbitrary pointer without crashing.
Break PAC with Hardware Attacks

- Guess a PAC *speculatively* to prevent crashes
- Leak verification results via side channel
Speculative Execution

if (true)
A
else
B

In Order

Branch
A

Speculative

Branch
Speculate B
Undo B
A

Microarchitectural side effects NOT undone
We use side channels to transmit the verification results of a pointer.
Threat Model

- Read/ write memory corruption bug
- Local code execution
- Can trigger PACMAN Gadget
Bird's Eye View

Write PAC guess into memory with existing software bug

Run PACMAN Gadget with guess

Correct
Observes Load!

Incorrect
No Load
Data Gadget

```python
if (condition):
    verified_ptr = check_pac(guess_ptr)
    load(verified_ptr)
```
if (condition):
    verified_ptr = check_pac(guess_ptr)
    load(verified_ptr)
if (condition):
    verified_ptr = check_pac(guess_ptr)
load(verified_ptr)
Data Attack

\[
\text{if (condition):} \\
\text{verified_ptr = check_pac(guess_ptr)} \\
\text{load(verified_ptr)}
\]
if (condition):
    verified_ptr = check_pac(guess_ptr)
load(verified_ptr)
```python
if (condition):
    verified_ptr = check_pac(guess_ptr)
    load(verified_ptr)
```
Data Attack

```python
if (condition):
    verified_ptr = check_pac(guess_ptr)
    load(verified_ptr)
```

Correct PAC

Mispredict Branch \(\rightarrow\) PAC Check Succeeds \(\rightarrow\) Speculative Load!

Incorrect PAC

Mispredict Branch \(\rightarrow\) PAC Check Fails \(\rightarrow\) Speculative Exception
if (condition): #BR1
    verified_ptr = check_pac(guess_ptr)
    call(verified_ptr) #BR2
The world's first desktop CPU that supports Pointer Authentication.
Challenges of Real World HW

- No documentation of microarchitectural details.
- No high resolution timer.
- macOS is a difficult system to integrate attacks on.

Essentially, we had to reinvent the wheel.
Conjectured TLB Hierarchy

- **L1 User iTLB**
  - 32 sets, 4 ways

- **L1 Kernel iTLB**
  - 32 sets, 4 ways

- **L1 dTLB**
  - 256 sets, 12 ways

- **L2 TLB**
  - 2048 sets, 23 ways
Conjectured TLB Hierarchy

- L1 User iTLB: 32 sets, 4 ways
- L1 Kernel iTLB: 32 sets, 4 ways
- L1 dTLB: 256 sets, 12 ways
- L2 TLB: 2048 sets, 23 ways
Conjectured TLB Hierarchy

L1 User iTLB
32 sets, 4 ways

L1 Kernel iTLB
32 sets, 4 ways

L1 dTLB
256 sets, 12 ways

L2 TLB
2048 sets, 23 ways
Experiment Testbed:

We insert a vulnerable kernel extension.
PAC Oracle Accuracy

Data

Instructions
Under the PACMAN kext, we find each run takes 2.69ms. This will likely be longer for real kernel code.

We can bruteforce an entire 16-bit PAC (from 0x0000 to 0xFFFF) in under 3 minutes.
<table>
<thead>
<tr>
<th>Data Gadgets</th>
<th>Instruction Gadgets</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>13,867</td>
<td>41,292</td>
<td>55,159</td>
</tr>
</tbody>
</table>

PACMAN Gadgets are readily available in large codebases.

This list is not exhaustive, and no exploitability analysis was performed.
PacmanOS
A Rust-based bare metal environment for performing experiments.
Top news

The M1 has a big security loophole, and Apple can't patch it
4 hours ago

MIT researchers uncover ‘unpatchable’ flaw in Apple M1 chips
51 minutes ago

MIT Finds New Arm Vulnerability Present in Apple M1, Demos PACMAN Attack
4 hours ago

PACMAN M1 chip attack defeats ‘the last line of security’ – but requires physical access
2 hours ago

All coverage

Phoronix
Apple M1 Affected By "PACMAN" Hardware Vulnerability In Arm Pointer Authentication
4 hours ago

VentureBeat
MIT researchers discover Apple M1 chip vulnerability
3 hours ago

Apple M1 chip contains hardware vulnerability that bypasses memory defense
4 hours ago

DARK Reading
Design Weakness Discovered in Apple M1 Kernel Protections
3 hours ago

Experts warn of 'PACMAN' flaw in M1 chip that can't be patched
1 hour ago
PACMAN: Attacking ARM Pointer Authentication with Speculative Execution
Follow us on Twitter!

@0xjprx
@weon_taek_na

PACMANATTACK.COM