Demystifying Pointer Authentication on Apple M1

Zechao Cai¹,², Jiaxun Zhu¹,², Wenbo Shen¹,², Yutian Yang¹,², Rui Chang¹,², Yu Wang³, Jinku Li⁴, and Kui Ren¹,²

¹Zhejiang University, Hangzhou, China
²ZJU-Hangzhou Global Scientific and Technological Innovation Center, Hangzhou, China
³Hangzhou Cyberserval Co., Ltd., Hangzhou, China
⁴Xidian University, Xi’an, China
Outline

- Pointer Authentication
  - What is Pointer Authentication (PAC)
  - Current Research State of Apple PAC
- Our Motivation
  - Lack of systematic analysis of Apple PAC Protection
- Our Contribution
  - m1n1-based reverse engineering framework
  - Disclosure of Apple’s hardware implementation
  - Comprehensive analysis of PA-based XNU kernel protection
- Our Findings
  - Apple’s PA Hardware
  - PA-based XNU Kernel Protection
  - Security analysis
Pointer Authentication
ARMv8.3 Specification

1 control register
- SCTLR_EL1.EnIA/IB/DA/DB

5 key registers
- APIA/APIB/APDA/APDB/APGA

2 kinds of instructions
- pac*/aut*
Pointer Authentication

Used for:
1. Function Call
2. Memory Access

Diagram:
- CPU
- Register
  - Pointer A
- Memory
  - Pointer A
Pointer Authentication

Used for:
1. Function Call
2. Memory Access

Attacker with memory write primitive
Pointer Authentication

- **CPU**
  - Register: Pointer A
  - Register: Pointer B

- **Memory**
  - Pointer B

- **Attacker with memory write primitive**

1. Function Call
2. Memory Access

Used for
Pointer Authentication
**Pointer Authentication**

An attacker with a memory write primitive can modify a pointer, leading to an error code being set. This diagram illustrates the process of authentication and potential security vulnerabilities in pointer management.
Pointer Authentication

Attacker with memory write primitive

Modify Pointer and PAC

Error Code

Pointer A

Pointer B

PAC

Register

Memory

CPU

pacia x1, x2

autia x1, x2

pacia x1, x2

auth x1, x2

Authentication
Apple’s PA Protection

- **Apple is the first one** to implement and deploy PA hardware
  - State-of-the-art mitigation against pointer corruption attack
  - Since its debut, the number of public kernel exploits has decreased

Our Motivation

- Although there are a lot of research works on Apple’s PA
  - Most of them focus on PA-based software protection
  - How does Apple implements Apple PA hardware remain unknown
Our Motivation

- Although there are a lot of research works on Apple’s PA
  - Most of them focus on PA-based **software** protection
  - How does Apple implement Apple PA **hardware remain unknown**

Brandon Azad:

> It seemed that the A12 manages to break either cross-EL PAC symmetry or cross-key PAC symmetry.

> pointer read from kernel memory. These two values differed despite the fact that the PAC keys should be the same in userspace and the kernel, so once again it appeared that the A12 was conjuring some sort of dark magic.
Our Motivation

• The **hardware** implementation remains **unknown**

• As a result, the PA **software** can not be analyzed systematically

• There is still **no systematic analysis** of Apple PA hardware and software
Our Contribution

- We build a reverse engineering framework based on m1n1 (an open-sourced hypervisor) to analyze PA hardware and software on Apple M1.

- We reveal how Apple customizes the PA hardware to introduce undisclosed security properties — Cross-domain (Cross-EL/Key/VM/Boot) attack mitigation.

- We analyze the implementation of PA-based XNU kernel protection and identify four attack surfaces. Apple acknowledged our findings publicly, fixed these issues in a security update and assigned us a CVE.
m1n1-based RE Framework

• Required Capabilities (RC)

**RC1: Identify** undisclosed PA-related Apple-specific system registers

**RC2: Read/Write** actual PA key values

**RC3: Profile** the undisclosed PA instruction behavior

**RC4: Debug** the XNU kernel dynamically
m1n1-based RE Framework

RC1: Identify undisclosed PA-related Apple-specific system registers

Challenge: Apple introduced a lot of undisclosed system registers

Our solution: We identify registers based on

- Binary information
- System Register Redirection Hardware Feature

Binary

Initial Register Set
(Most are _EL1 encoding)

Alias Registers
(_EL12/EL2 encoding)

Registers Set

Final Registers Set

String data,
Test functions...

Dynamic analysis

Binary analysis

Validate the results

Bootstrap Process
m1n1-based RE Framework

- **RC2: Read/Write** the actual key value

**Challenge:** Apple implemented a **hardware-based PAC key protection**

**Our techniques:**

- For **EL1 Key**, Read/Write the key **from EL2** exception level
- For **EL2 Key**, Read/Write the key **before Apple PA is enabled**

![Diagram showing the m1n1-based RE Framework with EL1 and EL2 levels, and the interaction between APIAKeyLo_EL1 and APIAKeyLo_EL2 keys.]
m1n1-based RE Framework

- **RC3: Profile** the undisclosed PA instruction behavior

**Challenge:** We need to analyze the complex interplays between registers and instructions

**Our solution:**

![Diagram showing the m1n1-based RE Framework]

- **Step 1:** Control Registers
- **Step 2:** Read/Write actual key
- **Step 3:** Key R/W Ability

**Run pac Instructions**

- **EL0:** PAC
- **EL1:** PAC
- **EL2:** PAC

Different Key in different ELs, Reboot
m1n1-based RE Framework

- **RC4: Debug** the XNU kernel dynamically

  **Challenge:** LLDB (provided by Apple) **does not support active kernel debugging** on Apple M1

  **Our solution:** We implement active kernel debugging based m1n1 hypervisor
Our Findings
Apple’s PA Hardware

• Finding Overview

① Controllability, PAC algorithm

② - ⑤ Cross-domain Attack Mitigation
Apple’s PA Hardware

• Controllability & PAC Algorithm

APCTL_EL1

- bit[0]: Enable Apple PA

- bit[2], bit[3]: Enable PA on user (bit[2]) or kernel space (bit[3])

- bit[1], bit[4]: Enable EXTRAKEY on user (bit[4]) or kernel space (bit[1])

PAC Algorithm is not QARMA

- (Modifier XOR Key Value) is one of the inputs
Apple’s PA Hardware

• Cross-domain Attack

  • *Pointer substitution attack across different domains*

Cross-VM: From VM to Host and Other VMs

Cross-Key: E.g., From APIA-signed to APIB-signed

Cross-Boot: From Boot Round 1 to Round 2

Cross-EL: From User space to Kernel space

Formalization of Cross-domain Attack in the paper 😊
Apple’s PA Hardware

• Cross-VM attack mitigation

Setting the **higher 64-bit PAC Key** will trigger a **Key Transformation**

② **VMDIV_EL2** is used for differentiate the Key Transformation between VM and Host
Apple’s PA Hardware

• Cross-Key attack mitigation

Key Transformation introduces **per-key-type salts** to differentiate the results for different key types.

③ **VMDIV_EL2 ⊕ per-key-type salts** is one of the inputs for Key Transformation.
Apple’s PA Hardware

• Cross-Boot attack mitigation

EL2 Key Transformation introduces **per-boot diversifier** to differentiate the results for different CPU Boots
Apple’s PA Hardware

• Cross-EL attack mitigation

Apple introduces an **EXTRAKEY** to differentiate the PAC computation between user and kernel space

5 **EXTRAKEY ⊕ APKeys (APIA/IB/DA/DB/GA)** is the actual key value for PAC computation

Controlled by APCTL_EL1 (bit[1]: Kernel, bit[4]: User (XNU Kernel only enable bit[4]))
Sign/Auth Interfaces Analysis

- We analyze all pac instructions in XNU kernel
- The result shows that XNU kernel uses
  - 9 types of signing modifiers
    - (5 types in official documentation)
  - 6 policies for generating modifier constant
    - (2 policies in official documentation)
PA-based Kernel Protection

• Key Management
  • XNU kernel configures the keys
    • Global: APIA/DA/GA
    • Per-Process: APIB/DB, EXTRAKEY

<table>
<thead>
<tr>
<th>Key</th>
<th>APIA</th>
<th>APDA</th>
<th>APGA</th>
<th>APIB</th>
<th>APDB</th>
<th>EXTRAKEY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scope</td>
<td>Global</td>
<td>Global</td>
<td>Global</td>
<td>Per-Process</td>
<td>Per-Process</td>
<td>Per-Process</td>
</tr>
</tbody>
</table>
PA-based Kernel Protection

• Key Management

• PAC instruction scope
  
  • pacia/da/ga: global in kernel space, per-process in user space
  
  • pacib/db: per-process
  
  • For non-arm64e process, the XNU kernel disable the user space PAC

<table>
<thead>
<tr>
<th>PAC instructions</th>
<th>pacia</th>
<th>pacda</th>
<th>pacga</th>
<th>pacib</th>
<th>pacdb</th>
</tr>
</thead>
<tbody>
<tr>
<td>User (arm64e)</td>
<td>Per-Process</td>
<td>Per-Process</td>
<td>Per-Process</td>
<td>Per-Process</td>
<td>Per-Process</td>
</tr>
<tr>
<td>User (Non-arm64e)</td>
<td>-</td>
<td>-</td>
<td>Per-Process</td>
<td>Per-Process</td>
<td>-</td>
</tr>
<tr>
<td>Kernel</td>
<td>Global</td>
<td>Global</td>
<td>Global</td>
<td>Per-Process</td>
<td>Per-Process</td>
</tr>
</tbody>
</table>
Security Analysis

- We validate 4 attack surfaces (88 cases) and report them to Apple

1. Incomplete Sensitive Data Identification
   - Potential Enhancements

2. Incomplete Interrupt Context Protection
   - Fixed in a security update and acknowledged publicly

3. Signing Gadget
   - Fixed and assigned a CVE (CVE-2023-32424)

4. Key Leakage
   - Potential Enhancements

More detail about identification and validation in the paper 😊

<table>
<thead>
<tr>
<th>AS</th>
<th>Identified</th>
<th>Validated</th>
<th>Fixed Cases</th>
</tr>
</thead>
<tbody>
<tr>
<td>①</td>
<td>153</td>
<td>83</td>
<td>6</td>
</tr>
<tr>
<td>②</td>
<td>17+18*</td>
<td>2</td>
<td>2+18*</td>
</tr>
<tr>
<td>③</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>④</td>
<td>2</td>
<td>2</td>
<td>-</td>
</tr>
</tbody>
</table>

* 18 cases are identified in XNU-7195, and all of them are fixed in XNU-8019.
Thank you
Q&A