TLB Poisoning Attacks on AMD Secure Encrypted Virtualization

Mengyuan Li¹, Yinqian Zhang², Huibo Wang³, Kang Li³ and Yueqiang Cheng⁴

¹The Ohio State University, ²Southern University of Science and Technology, ³Baidu Security, ⁴NIO Security Research
“SEV technology is built around a threat model where an attacker is assumed to have access to not only execute user level privileged code on the target machine, but can potentially execute malware at the higher privileged hypervisor level as well.”
Hardware Memory Encryption

- **CPU**
  - AMD Secure Processor
    - Manages AES Keys
    - Handle SEV API
  - Memory Controller
    - Memory Encryption Engine (MEE)
    - AES encryption/decryption

- **Hypervisor**
  - Guest OS
  - VM
Hardware Memory Encryption

- Data are unencrypted in CPU.
Hardware Memory Encryption

- Data are encrypted in the memory.
New vulnerabilities?

Hardware Memory Encryption

Trusted Host

Traditional VMs
Virtualization

SEV-enabled VMs
Virtualization

New design & settings
New threat model

Untrusted Host

Root of Trust
AMD Secure Processor

6
New vulnerabilities?

Traditional VMs
Virtualization

+

Trusted Host

New design & settings
New threat model

SEV-enabled VMs
Virtualization

New vulnerabilities caused by the inconsistency?

Hardware Memory Encryption
New vulnerabilities?

- Traditional VMs
  - Virtualization
- Trusted Host
- SEV-enabled VMs
  - Virtualization
- Untrusted Host

New design & settings
New threat model

New vulnerabilities caused by the inconsistency?

Hardware Memory Encryption

TLB isolation mechanism
ASID-based TLB Isolation in VM’s Lifetime

Address Space Identifier (ASID)
ASID-based TLB Isolation in VM’s Lifetime

Context switch

TLB

vCPU0 ↔ vCPU1 ↔ vCPU2 ↔ vCPU3

CPU

gVA  sPA

gVA  sPA

gVA  sPA

gVA  sPA

Total TLB flush
ASID-based TLB Isolation in VM’s Lifetime

- vCPU0
- vCPU1
- Context switch
- vCPU2
- vCPU3

Address Space Identifier (ASID)

TLB

<table>
<thead>
<tr>
<th>ASID</th>
<th>gVA</th>
<th>sPA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

No TLB flush is enforced
ASID assignment in traditional VM

Guest VM’s world

VMEXIT

Save states into VM Control Block (VMCB)

Host’s world

Host handles VMEXIT

ASID

VMCB

ASID?

VMCB

Launch states from VMCB

VMRUN

Check TLB condition before VMRUN
ASID-based TLB Isolation in traditional VM

Guest VM’s world

VMEXIT

Save states into VM Control Block (VMCB)

ASID

VMCB

New ASID

VMCB

Host’s world

Launch states from VMCB

VMRUN

Host handles VMEXIT

Check TLB condition before VMRUN

a) move to a new CPU core => Assign a new ASID
b) Observed vCPU-switch => ASID++
ASID-based TLB Isolation in traditional VM

- Guest VM’s world
- Host’s world

**VMEXIT**
Save states into VM Control Block (VMCB)

- **Host handles VMEXIT**

**Check TLB condition before VMRUN**
- a) move to a new CPU core => Assign a new ASID
- b) Observed vCPU-switch => ASID++
- c) Otherwise => Unchanged ASID

- Launch states from VMCB

**VMRUN**
ASID’s new role in SEV

- Each VMs as well as hypervisor have their own and unique AES keys. Those VM Encryption Keys (VEKs) are stored in AMD-SP.

**Address Space Identifier (ASID)**
ASID’s new role in SEV

- Each VMs as well as hypervisor have their own and unique AES keys. Those VM Encryption Keys (VEKs) are stored in AMD-SP.

Address Space Identifier (ASID)
- SEV VM’s vCPUs need to have the same ASID
ASID-based TLB Isolation in SEV VM

Guest VM’s world

VMEXIT

Save states into VM Control Block (VMCB)

Host’s world

Launch states from VMCB

ASID

VMCB

ASID

VMCB

Host handles VMEXIT

Check TLB condition before VMRUN

a) move to a new CPU core => TLB flush

b) Observed vCPU-switch (the same VM) => TLB flush

c) Otherwise => Unchanged ASID
ASID-based TLB Isolation in SEV VM

Guest VM’s world

VMEXIT

Save states into VM Control Block (VMCB)

ASID

VMCB

Host’s world

Launch states from VMCB

ASID

VMCB

Hypervisor controlled

Host handles VMEXIT

Check TLB condition before VMRUN

a) move to a new CPU core => TLB flush

b) Observed vCPU-switch (the same VM) => TLB flush

c) Otherwise => Unchanged ASID
TLB POISONING ATTACKS - OUTLINE

• Attack Primitives
  - TLB Misuse across vCPUs
  - TLB Misuse within the Same vCPU

• TLB Poisoning Attacks
  - TLB poisoning with assisting process
  - TLB poisoning without assisting process

• Discussion

• Conclusion
TLB POISONING ATTACKS - OUTLINE

• Attack Primitives
  - TLB Misuse across vCPUs
  - TLB Misuse within the Same vCPU

• TLB Poisoning Attacks
  - TLB poisoning with assisting process
  - TLB poisoning without assisting process

• Discussion

• Conclusion
TLB Misuse across vCPUs

Context switch

Skip TLB flush
TLB Misuse across vCPUs

Program 0

CPU

ASID  gVA   sPA
ASID  gVA   sPA
ASID  gVA   sPA
ASID  gVA   sPA

vCPU0

Context switch

vCPU1

Program 1

CPU

ASID  gVA   sPA
ASID  gVA   sPA
ASID  gVA   sPA
ASID  gVA   sPA

TLB

Program 1 can
- Execute P0’s instruction
- Read P0’s data

Skip TLB flush
TLB POISONING ATTACKS - OUTLINE

• Attack Primitives
  ● TLB Misuse across vCPUs
  ● TLB Misuse within the Same vCPU

• TLB Poisoning Attacks
  ● TLB poisoning with assisting process
  ● TLB poisoning without assisting process

• Discussion

• Conclusion
TLB Misuse within the Same vCPU

Context switch

Hardware-enforced TLB flush

Same vCPU

Program0

Program1

Empty

CPU

CPU

TLB

TLB

ASID | gVA | sPA
--|---|---
ASID | gVA | sPA
ASID | gVA | sPA
ASID | gVA | sPA
TLB Misuse within the Same vCPU

Program0

CPU0

ASID | gVA | sPA
-----|-----|-----
  ASID | gVA | sPA
  ASID | gVA | sPA
  ASID | gVA | sPA
  ASID | gVA | sPA

Fill TLB entries

Program0

VMCB switching

CPU1

ASID | gVA | sPA
-----|-----|-----
  ASID | gVA | sPA
  ASID | gVA | sPA
  ASID | gVA | sPA
  ASID | gVA | sPA

Context switch

Hardware-enforced TLB flush

CPU1

ASID | gVA | sPA
-----|-----|-----
  ASID | gVA | sPA
  ASID | gVA | sPA
  ASID | gVA | sPA
  ASID | gVA | sPA

Program 1 can
- Execute P0’s instruction
- Read P0’s data
TLB POISONING ATTACKS - OUTLINE

• Attack Primitives
  ● TLB Misuse across vCPUs
  ● TLB Misuse within the Same vCPU

• TLB Poisoning Attacks
  ● TLB poisoning with assisting process
  ● TLB poisoning without assisting process

• Discussion

• Conclusion
Assume:
• Attacker process inside the VM and controlled by the attacker
• Attacker process is unprivileged
• Attacker process and victim process are on different vCPUs
• Attacker process know victim process’s address space (e.g., Crossline attack)

Goal:
• Control privileged process’s execution
TLB poisoning with assisting process (Openssh)

- vCPU0 (Victim Process (OpenSSH Daemon))
- vCPU1 (Attacker Process)
- Context switch
TLB poisoning with assisting process (Openssh)

Victim Process (OpenSSH Daemon)

Attacker Process

mmap gVA0 to gPA1

gVA0: Virtual Address of *pam_authenticate*
gPA0: Physical Address of *pam_authenticate*
gPA1: Physical Address of malicious code

**Step1:** Create a mapping in attacker process’s address space
TLB poisoning with assisting process (Openssh)

vCPU0
Victim Process
(OpenSSH Daemon)

Call pam_authenticate()

vCPU1
Attacker Process

Unset all P bit

Intercept NPF of gPA0

Attacker

mmap gVA0 to gPA1

CPU Core

gVA0: Virtual Address of pam_authenticate

gPA0: Physical Address of pam_authenticate

gPA1: Physical Address of malicious code

Step 2:
Unset all P bit, and Intercept target function (pam_authenticate)
TLB poisoning with assisting process (Openssh)

**Steps:**
1. **Victim Process** (OpenSSH Daemon)
   - Call `pam_authenticate()`

2. **Attacker Process**
   - Unset all P bit
   - Intercept NPF of gPA
   - Trap vCPU0, Run vCPU1

3. **Attacker**
   - Access gVA
   - TLB <gVA0, gPA1>

**Variables:**
- **gVA0**: Virtual Address of `pam_authenticate`
- **gPA0**: Physical Address of `pam_authenticate`
- **gPA1**: Physical Address of malicious code

**Poisoning:**
- **Step 3**: Poison TLB entries by accessing gVA0
TLB poisoning with assisting process (Opensh)

vCPU0

Victim Process
(OpenSSH Daemon)

Call pam_authenticate() → Intercept NPF of gPA0

vCPU1

Attacker Process

mmap gVA0 to gPA1

Trap vCPU0, Run vCPU1

Trap vCPU1, Run vCPU0, Skip TLB flush

Access gVA0

Time

Step 4:
Skip TLB flush caused by vCPU switching.

CPU Core

Attacker

Unset all P bit

TLB <gVA0, gPA1>

gVA0: Virtual Address of pam_authenticate

gPA0: Physical Address of pam_authenticate

gPA1: Physical Address of malicious code

32
TLB poisoning with assisting process (Openssh)

- Victim Process (OpenSSH Daemon)
  - Call pam_authenticate()
  - TLB <gVA0, gPA1>
  - Execute code in gPA1

- Attacker Process
  - Unset all P bit
  - Intercept NPF of gPA0
  - Trap vCPU0, Run vCPU1
  - Trap vCPU1, Run vCPU0
  - Skip TLB flush
  - mmap gVA0 to gPA1
  - Access gVA0

- Attacker

- CPU Core

Result:
Bypass authentication (execute arbitrary code)

- gVA0: Virtual Address of pam_authenticate
- gPA0: Physical Address of pam_authenticate
- gPA1: Physical Address of malicious code
TLB POISONING ATTACKS - OUTLINE

• Attack Primitives
  - TLB Misuse across vCPUs
  - TLB Misuse within the Same vCPU

• TLB Poisoning Attacks
  - TLB poisoning with assisting process
  - TLB poisoning without assisting process

• Discussion

• Conclusion
Network-interface applications:
- Use fork() to serve different requests
- Children processes have similar VMA

Target:
Dropbear SSH: lightweight open-source SSH server
Network-interface applications:
• Use fork() to serve different requests
• Children processes have similar VMA

Target:
Dropbear SSH: lightweight open-source SSH server

Goal:
• Bypass password authentication without assisting process when ASLR is enabled
TLB poisoning without assisting process

- vCPU0
  - Victim Process
  - (SSH connection from VM owner)

- vCPU1
  - Attacker Process
  - (SSH connection from attacker)

CPU Core
TLB poisoning without assisting process

- **vCPU0**: Victim Process (SSH connection from VM owner)
- **vCPU1**: Attacker Process (SSH connection from attacker)
- **Attacker**
- **CPU Core**

Unset all P bit

Login Req with wrong PWD

gPA0: Physical Address of `constant_time_strcmp`
gPA1: Physical Address of `svr_auth_pwd`

**Step 1:**
Attacker login with a wrong PWD
TLB poisoning without assisting process

vCPU0
Victim Process (SSH connection from VM owner)

vCPU1
Attacker Process (SSH connection from attacker)

CPU Core

Step 2:
Pause attacker process before PWD authentication

gPA0: Physical Address of constant_time_strcmp

Unset all P bit

Login Req with wrong PWD

Intercept NPF of gPA0

Call constant_time_strcmp

gPA1: Physical Address of svr_auth_pwd

Time
TLB poisoning without assisting process

Victim Process (SSH connection from VM owner) vCPU0

Attacker

Unset all P bit

Trap vCPU1, Run vCPU0

TLB Fill PWD Auth

Call constant_time_strcmp

vCPU0

Login Req with PWD

Call constant_time_strcmp

Intercept NPF of gPA0

Call svr_auth_pwd

CPU Core

Attacker Process (SSH connection from attacker) vCPU1

Login Req with wrong PWD

Call constant_time_strcmp

gPA0: Physical Address of constant_time_strcmp

gPA1: Physical Address of svr_auth_pwd

Step 3:
Pause Victim process after Victim process’s PWD auth
TLB poisoning without assisting process

Step 3:
Pause Victim process after Victim process’s PWD auth

TLB Fill → PWD Auth

Call constant_time_strcmp → Intercept NPF of gPA₀

Login Req with PWD → Trap vCPU1, Run vCPU0

Call constant_time_strcmp

Call svr_auth_pwd → Intercept NPF of gPA₁

CPU Core

gPA₀: Physical Address of constant_time_strcmp

Time

gPA₁: Physical Address of svr_auth_pwd
TLB poisoning without assisting process

Call `constant_time_strcmp`

Intercept NPF of gPA$\text{0}$

Login Req with PWD

Call `constant_time_strcmp`

TLB Fill

PWD Auth

Call `svr_auth_pwd`

Intercept NPF of gPA$\text{1}$

Trap vCPU1, Run vCPU0

Intercept NPF of gPA$\text{1}$

Trap vCPU0, Run vCPU1

Skip TLB flush

Exe `constant_time_strcmp`

Step 4:
Skip TLB flush caused by vCPU switching and resume Attacker process

gPA$\text{0}$: Physical Address of `constant_time_strcmp`

gPA$\text{1}$: Physical Address of `svr_auth_pwd`
TLB poisoning without assisting process

Step 5:
Attacker process pass PWD auth by using victim process’s PWD Buffer
TLB poisoning without assisting process

- **gPA₀**: Physical Address of `constant_time_strcmp`
- **gPA₁**: Physical Address of `svr_auth_pwd`

**Step6:**
Flush TLB
TLB poisoning without assisting process

CPU Core

- Call `constant_time_strcmp`
- Intercept NPF of gPA0
- Trap vCPU1, Run vCPU0
- Call `svr_auth_pwd`
- TLB Fill
- PWD Auth

- Call `svr_auth_pwd`
- Intercept NPF of gPA1
- Trap vCPU0, Run vCPU1
- Skip TLB flush
- Exc `constant_time_strcmp`
- TLB Misuse
- PWD Auth
- Call `svr_auth_pwd`
- Flush TLB

Result:
- 17 out of 20 connections
- Bypass the PWD auth

---

gPA0: Physical Address of `constant_time_strcmp`

gPA1: Physical Address of `svr_auth_pwd`
TLB POISONING ATTACKS - OUTLINE

• Attack Primitives
  ● TLB Misuse across vCPUs
  ● TLB Misuse within the Same vCPU

• TLB Poisoning Attacks
  ● TLB poisoning with assisting process
  ● TLB poisoning without assisting process

• Discussion

• Conclusion
Discussion

- **TLB Poisoning attacks on SEV-SNP**
  - SEV-SNP add additional TLB identifier fields in protected VMSA
  - TLB-flush mechanism is now controlled by hardware
Discussion

- **TLB Poisoning attacks on SEV-SNP**
  - SEV-SNP add additional TLB identifier fields in protected VMSA
  - TLB-flush mechanism is now controlled by hardware

- **Countermeasure on SEV/SEV-ES**
  - Network-related application should use exec() to ensure a completely new VMA for different connections (like OpenSSH)
Summary

• This work Demystifies AMD SEV’s mechanism for TLB management

• This work proposes the TLB Poisoning attacks

• This work discusses potential countermeasures
Q & A

Mengyuan Li: li.7533@osu.edu