black hat USA 2024

AUGUST 7-8, 2024

BRIEFINGS

All Your Secrets Belong to Us: Leveraging Firmware Bugs to Break TEEs

Tom Dohrmann

whoami

- 👋 Tom Dohrmann
- 🕵️ Low-level enthusiast
- ❤️ Coding
- ❤️ Hacking

Outline

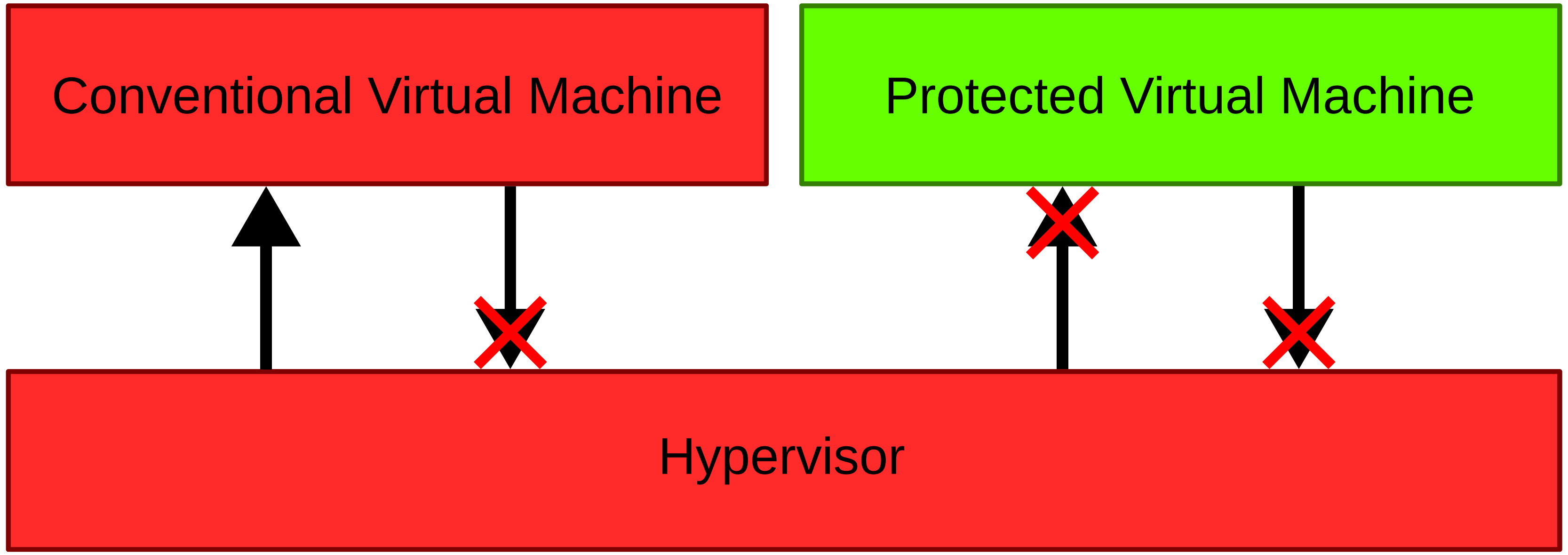
- Short Intro to TEEs and AMD SEV-SNP
- Prerequisites
  - Platform Security Processor & Firmware
  - Reverse Map Table
- Bug #1
  - Simple Exploit
  - Improved Exploit
- Bug #2
  - Exploit
- Wrap-up and take-aways

What‘s a TEE Anyway?

- TEE = Trusted Execution Environment
- A secure area of a main processor
- Workloads are protected from conventionally privileged parts of an OS e.g. the kernel
- For a lot of applications leakage of secrets is a bad as arbitrary code execution.
- Many implementations:
- AMD SEV(-ES/-SNP)
- Intel SGX, Intel TDX → “Compromising Confidential Compute, One Bug at a Time”
- Arm TrustZone, Arm CCA
- IBM SE
- RISC-V CoVE
- NVIDIA H100

Very Short Intro to AMD SEV-SNP

- AMD SEV-SNP implements a Trusted Execution Environment (TEE).
- It aims to shield protected virtual machines from untrusted and even malicious hypervisors.
- All data and code is encrypted and integrity protected.
- Upon creation of a VM, the initial memory contents are measured and can be verified through attestation reports.



Platform Security Processor (PSP)

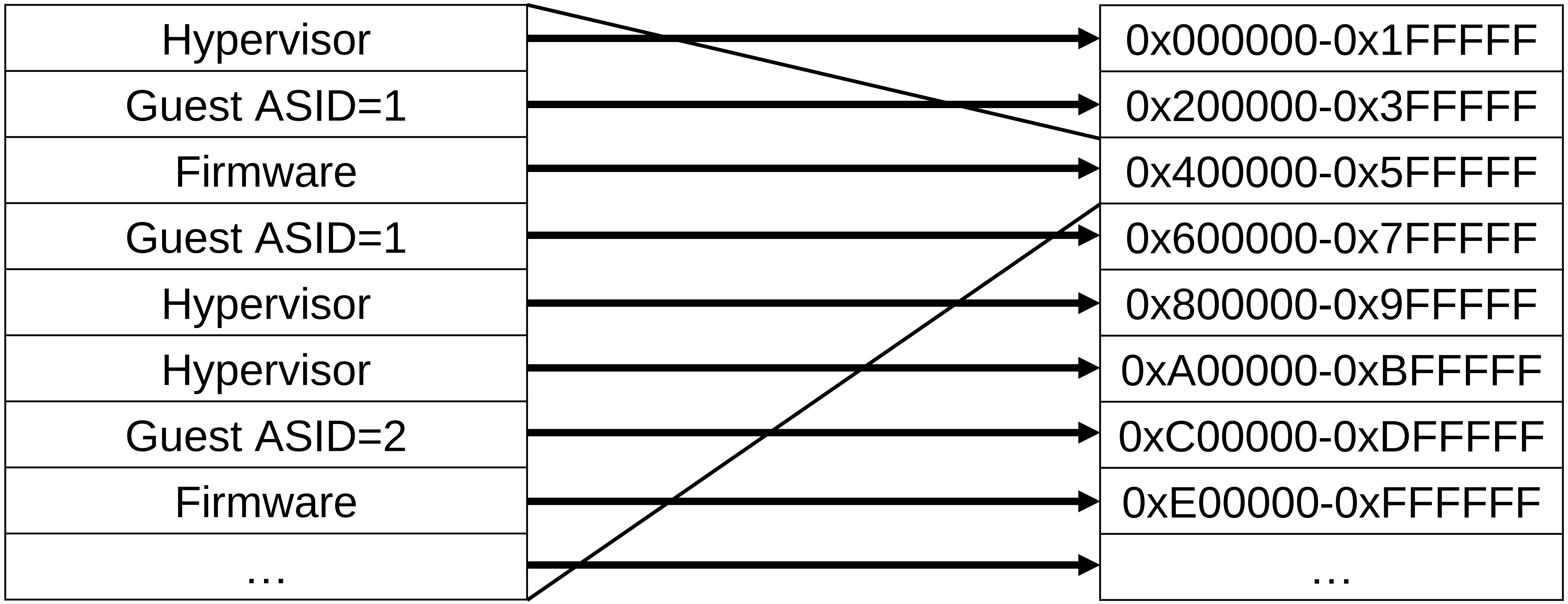
- The Platform Security Processor is a highly privileged components of AMD SoCs.
- In the context of SEV, the PSP implements the root of trust and is required to create, attest, migrate, delete SEV-SNP virtual machines.
- The SEV firmware is also used with the SEV-SNP's predecessors, SEV and SEV-ES.
- The firmware can be live-updated.
- Parts of the firmware were published in August 2023.

Reverse Map Table (RMP)

- The RMP is used to protect the integrity of memory.
- It contains an entry for every guest-assignable page of memory to track its state.
- Before every write access, the CPU checks the RMP to decide whether the access is allowed. These checks are done for all privilege levels including hypervisor and SMM accesses.
- The firmware is more privileged and can write to any memory → It needs to do these checks manually.
- The RMP is managed by the CPU through special instructions and by the SEV firmware.
- A lot of trust is put into the RMP permission and state checks being enforced correctly (foreshadowing!).

Reverse Map Table (RMP)

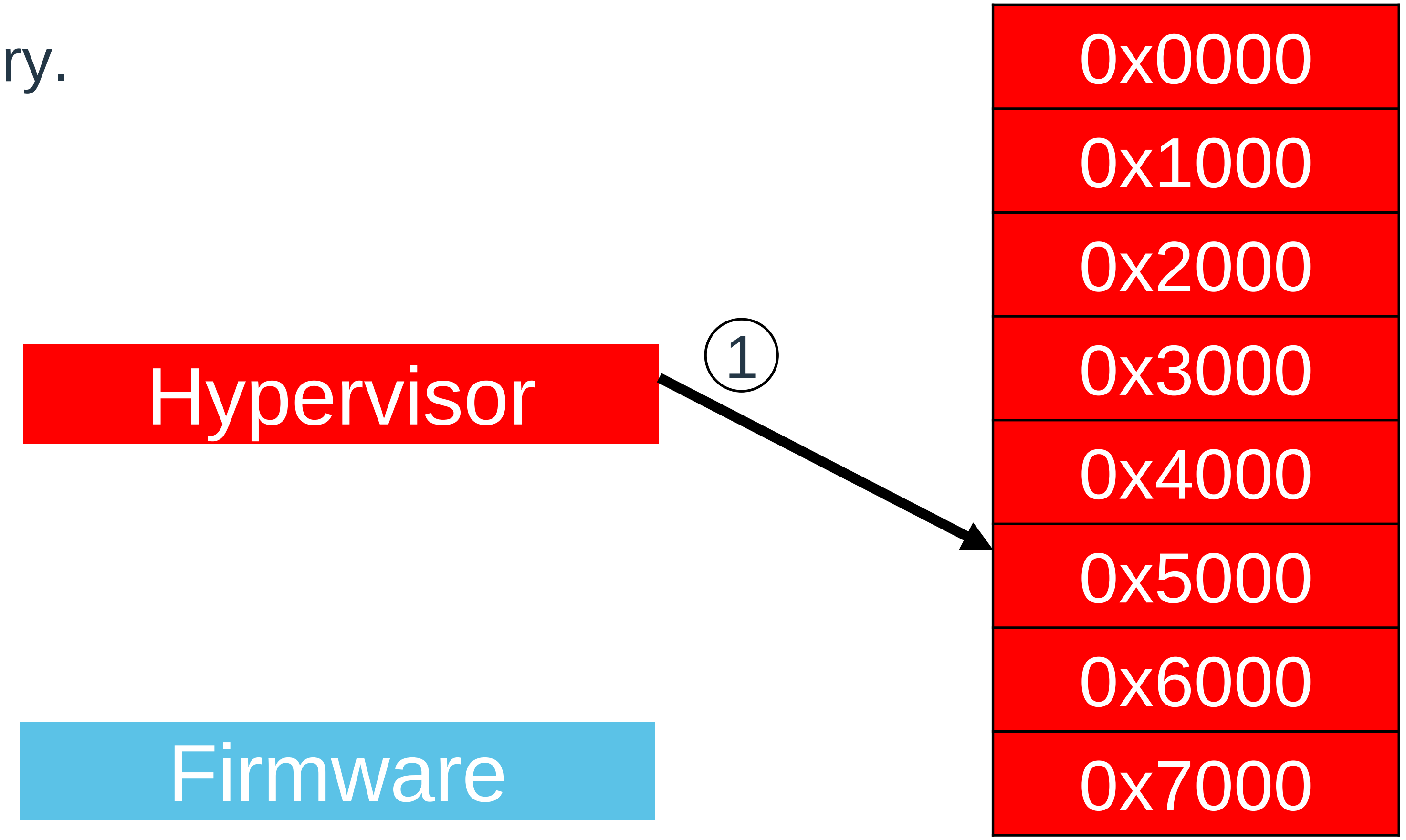
- Each page can be owned by the hypervisor, a virtual machine, or the SEV firmware.



CVE-2024-21980

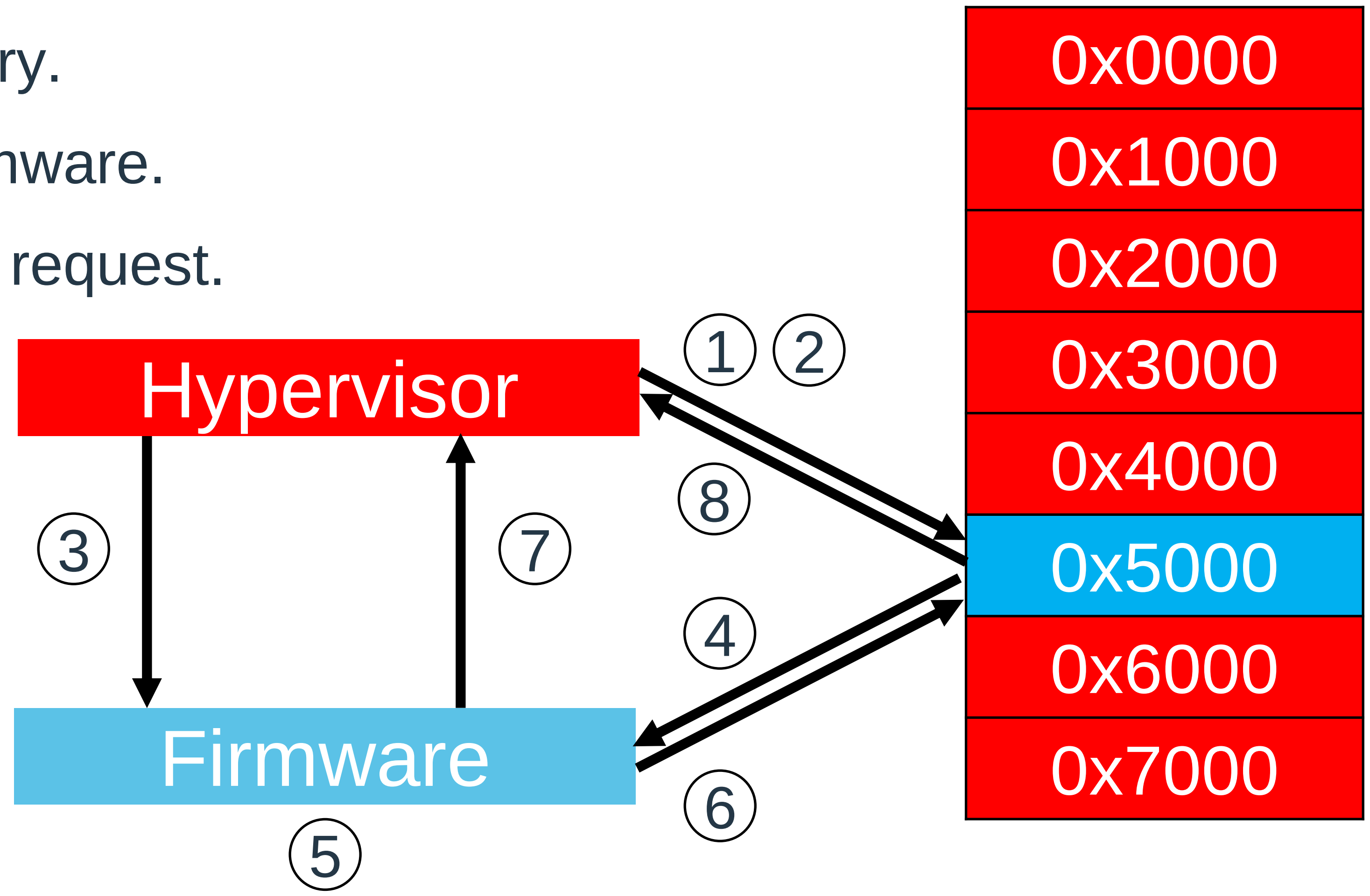
Command Dispatch

1. The hypervisor writes the request to memory.



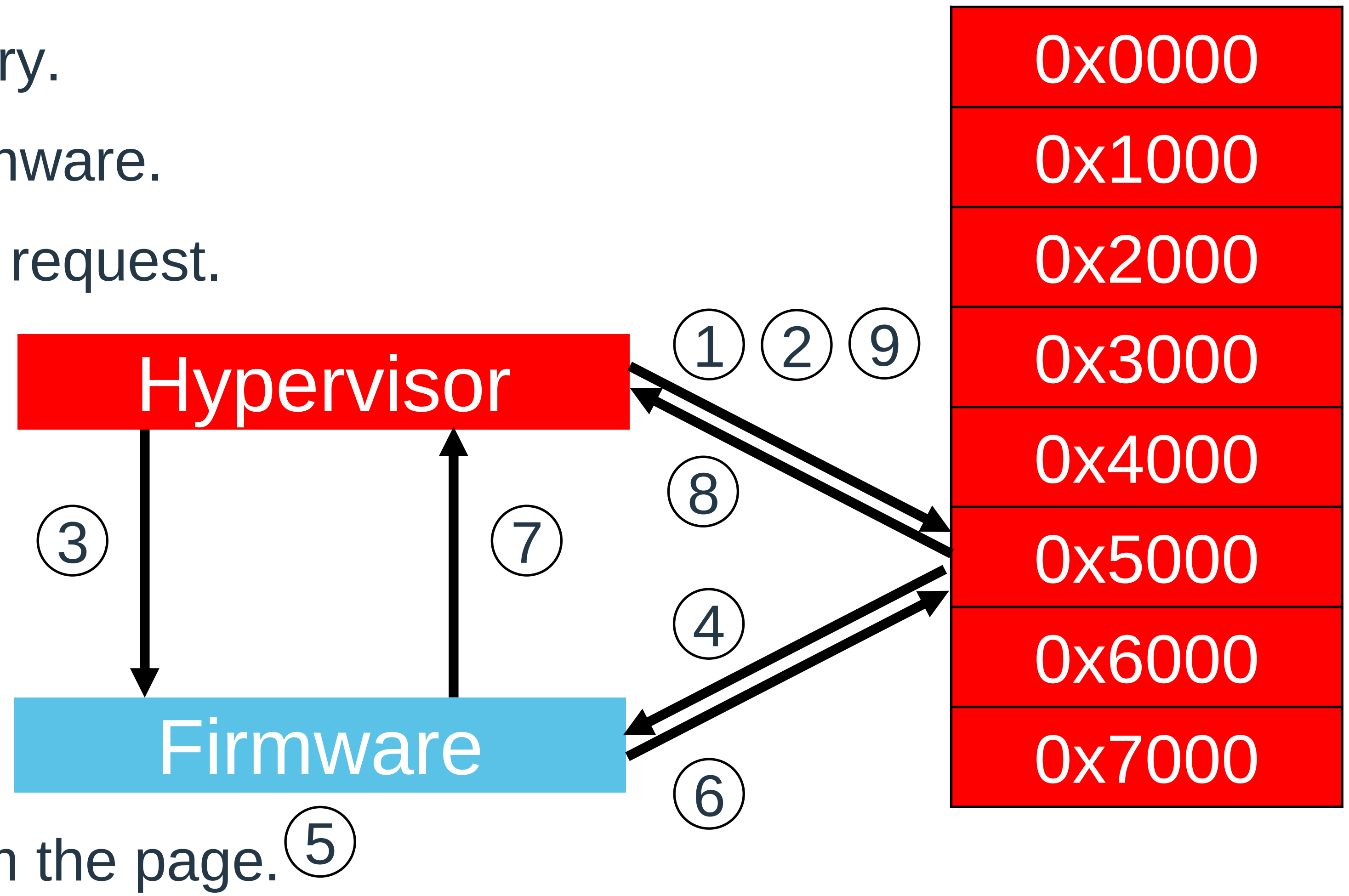
Command Dispatch

1. The hypervisor writes the request to memory.
2. The hypervisor donates the page to the firmware.
3. The hypervisor tells the firmware about the request.
4. The firmware reads the request.
5. The firmware processes the request.
6. The firmware writes the response back.
7. The firmware tells the hypervisor it's done.
8. The hypervisor reads the response.



Command Dispatch

1. The hypervisor writes the request to memory.
2. The hypervisor donates the page to the firmware.
3. The hypervisor tells the firmware about the request.
4. The firmware reads the request.
5. The firmware processes the request.
6. The firmware writes the response back.
7. The firmware tells the hypervisor it's done.
8. The hypervisor reads the response.
9. The hypervisor asks the firmware to reclaim the page.



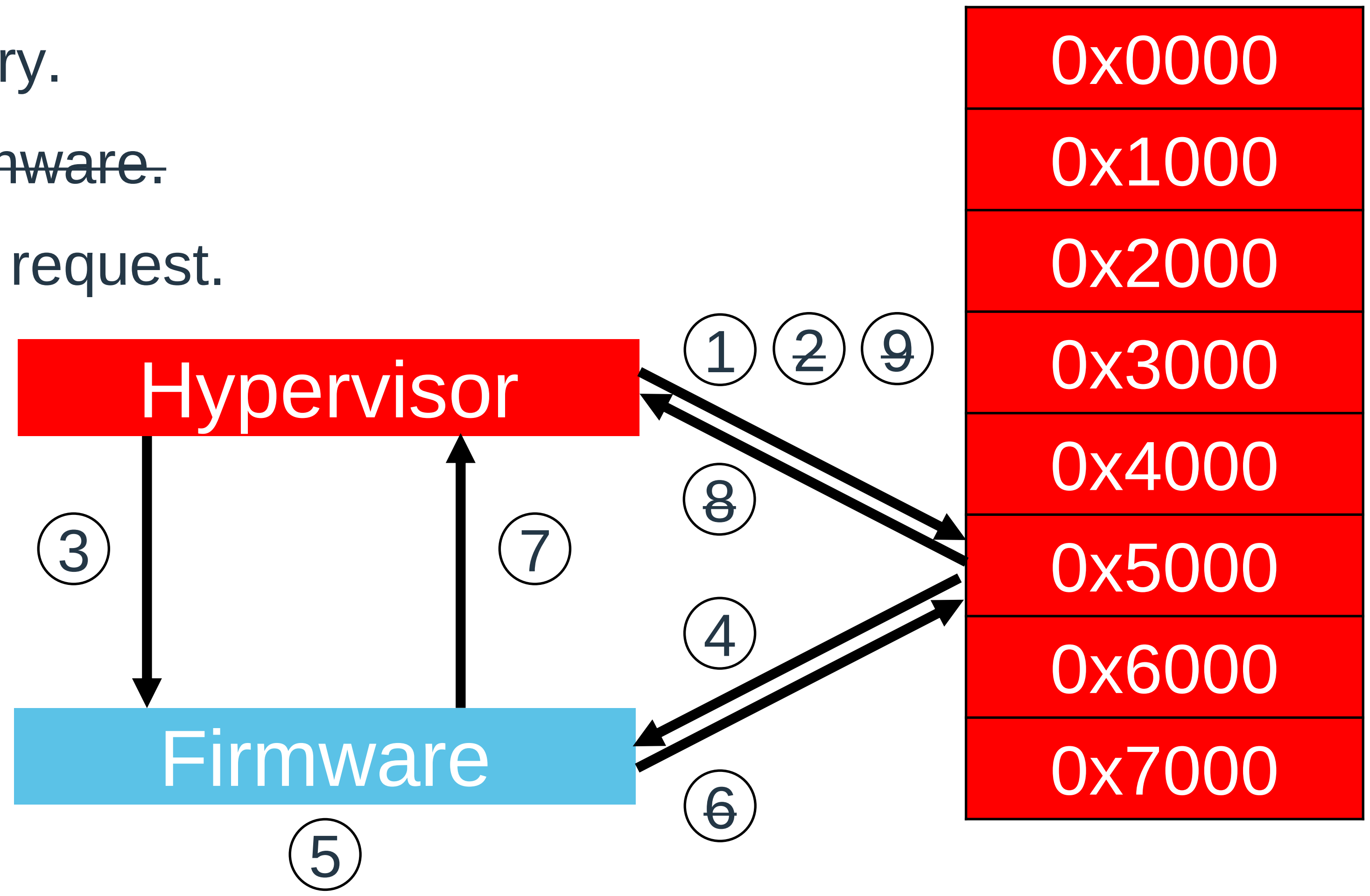
TL;DR: Command requests and responses are written to regular memory.

→ During step 6, the firmware needs to check whether it's allowed to write to memory.

Command Dispatch (w/o Response)

1. The hypervisor writes the request to memory.
2. ~~The hypervisor donates the page to the firmware.~~
3. The hypervisor tells the firmware about the request.
4. The firmware reads the request.
5. The firmware processes the request.
6. ~~The firmware writes the response back.~~
7. The firmware tells the hypervisor it's done.
8. ~~The hypervisor reads the response.~~
9. ~~The hypervisor reclaims the page.~~

→ The firmware only has to check the RMP if it writes back a response.



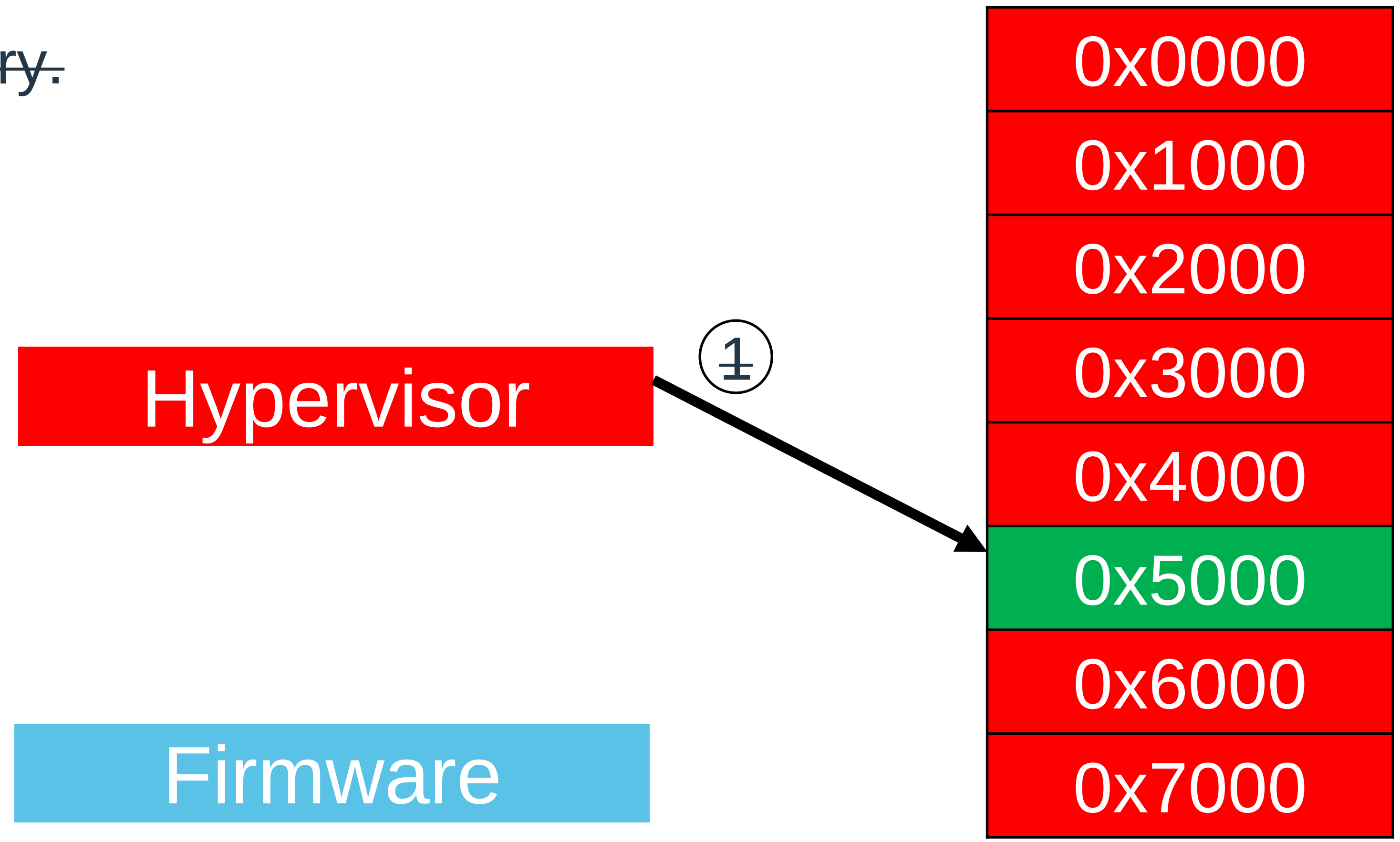
black hat USA 2024

Bug #1
One Of These It Not Like The Others...

Command	Buffer Type	RMP Write Checks
INIT	Input Only	No
SHUTDOWN	Ignore	No
PLATFORM_RESET	Ignore	No
PLATFORM_STATUS	Output Only	Yes
…	…	…
ATTESTATION	Input & Output & Error	No
SEND_START	Input & Output & Error	Yes
SEND_UPDATE_DATA	Input & Output & Error	Yes
SEND_UPDATE_VMSA	Input & Output & Error	Yes

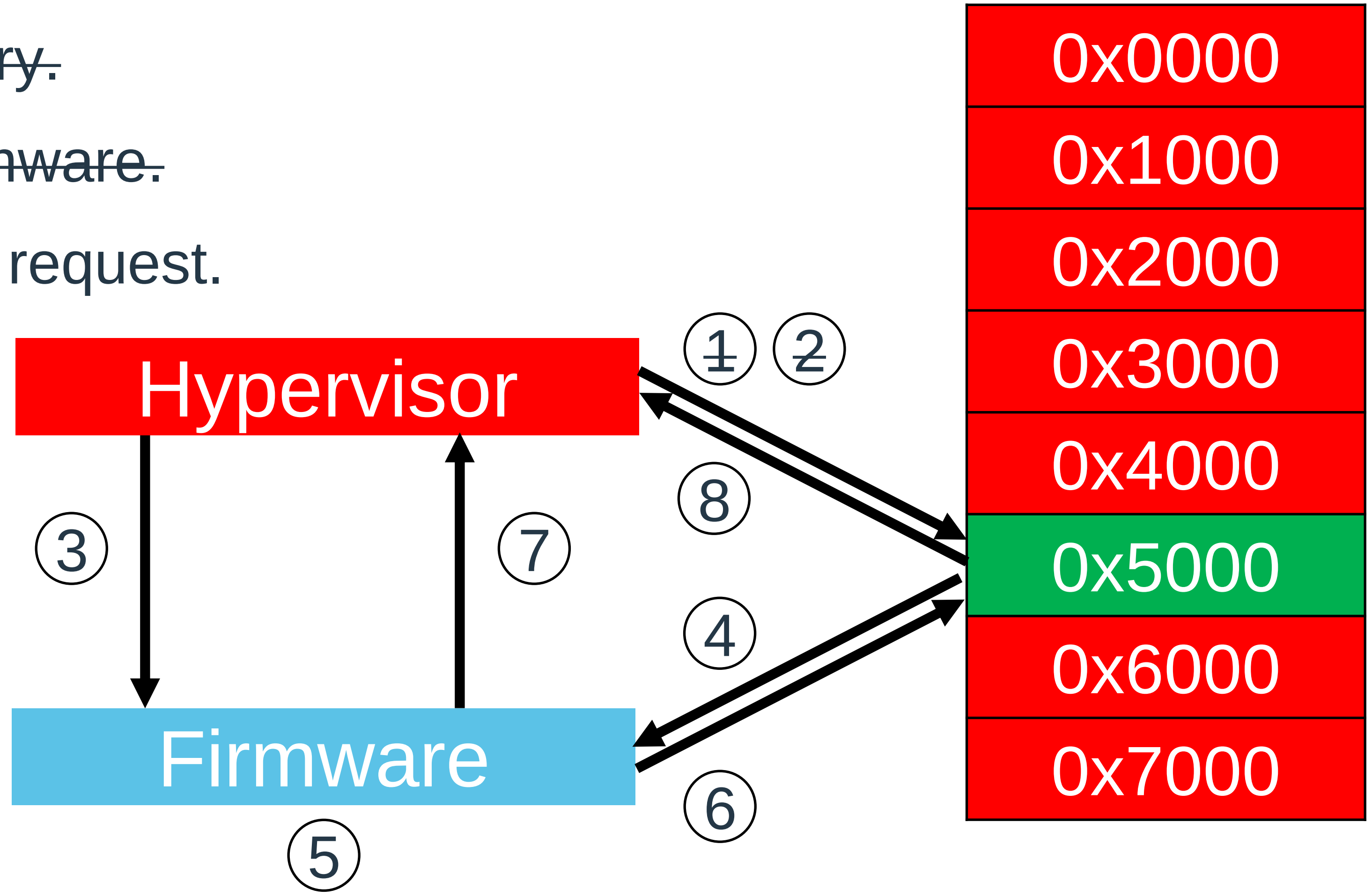
Command Dispatch

1. ~~The hypervisor writes the request to memory.~~



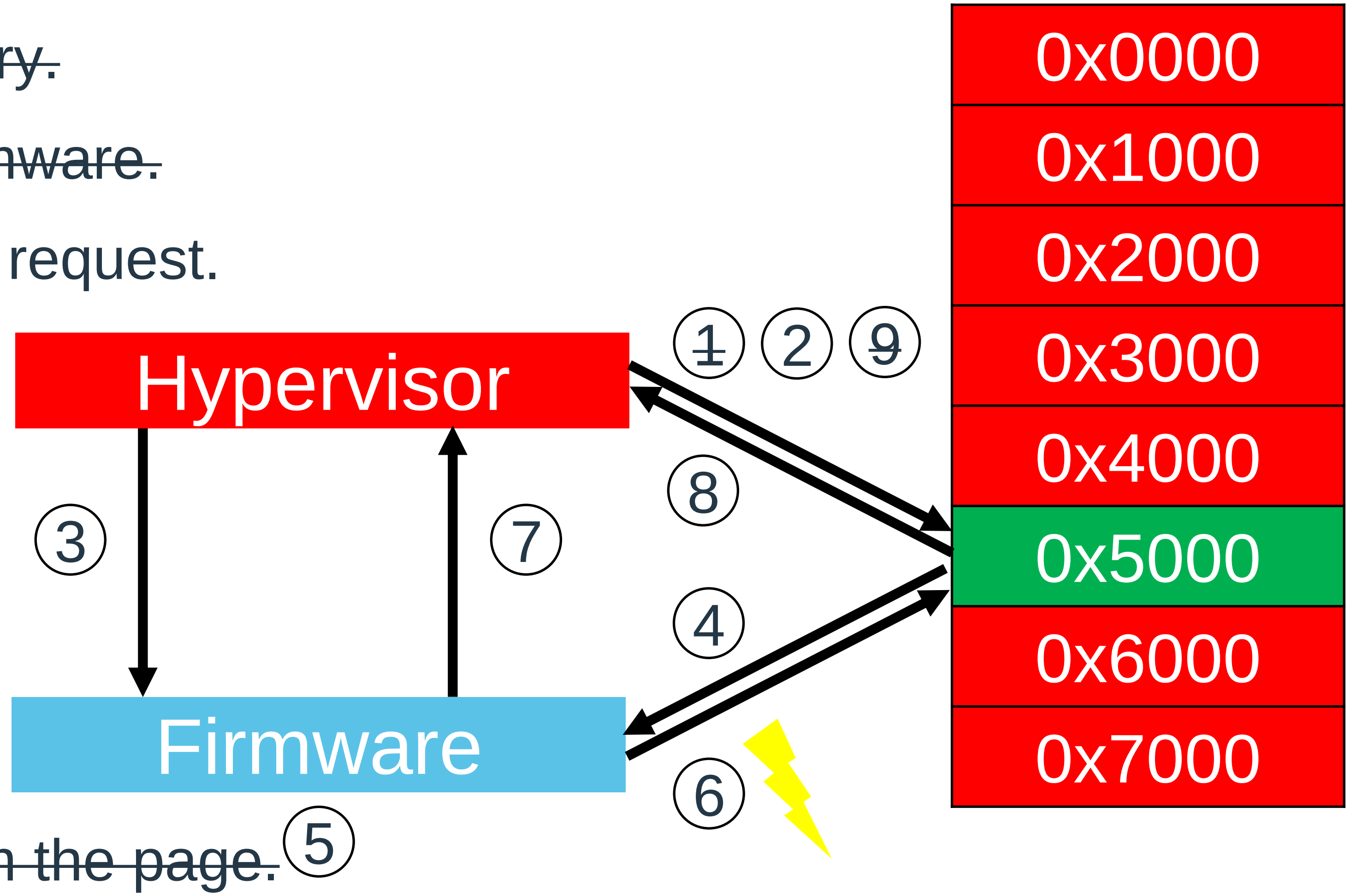
Command Dispatch

1. ~~The hypervisor writes the request to memory.~~
2. ~~The hypervisor donates the page to the firmware.~~
3. The hypervisor tells the firmware about the request.
4. The firmware reads the request.
5. The firmware processes the request.
6. The firmware writes the response back.
7. The firmware tells the hypervisor it's done.
8. The hypervisor reads the response.



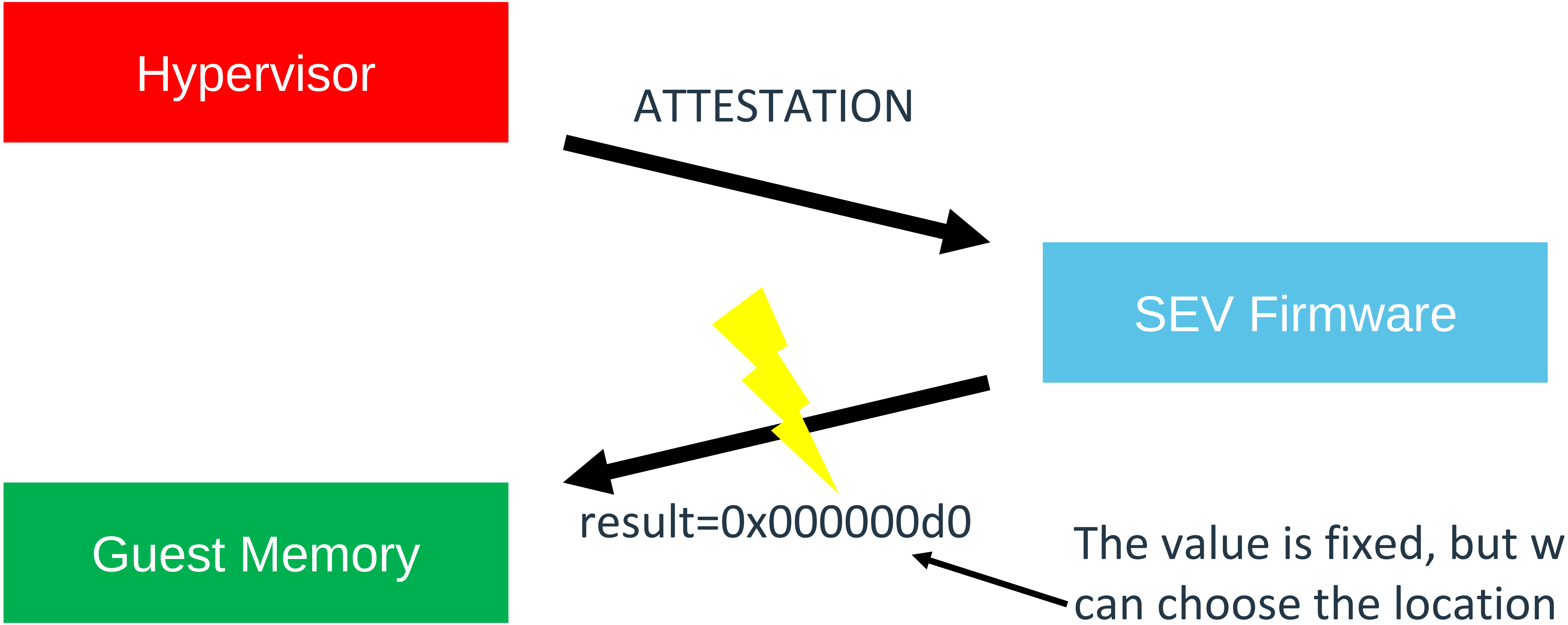
Command Dispatch

1. ~~The hypervisor writes the request to memory.~~
2. ~~The hypervisor donates the page to the firmware.~~
3. The hypervisor tells the firmware about the request.
4. The firmware reads the request.
5. The firmware processes the request.
6. The firmware writes the response back.
7. The firmware tells the hypervisor it's done.
8. The hypervisor reads the response.
9. ~~The hypervisor asks the firmware to reclaim the page.~~



→ The firmware just corrupted the memory of a protected guest.

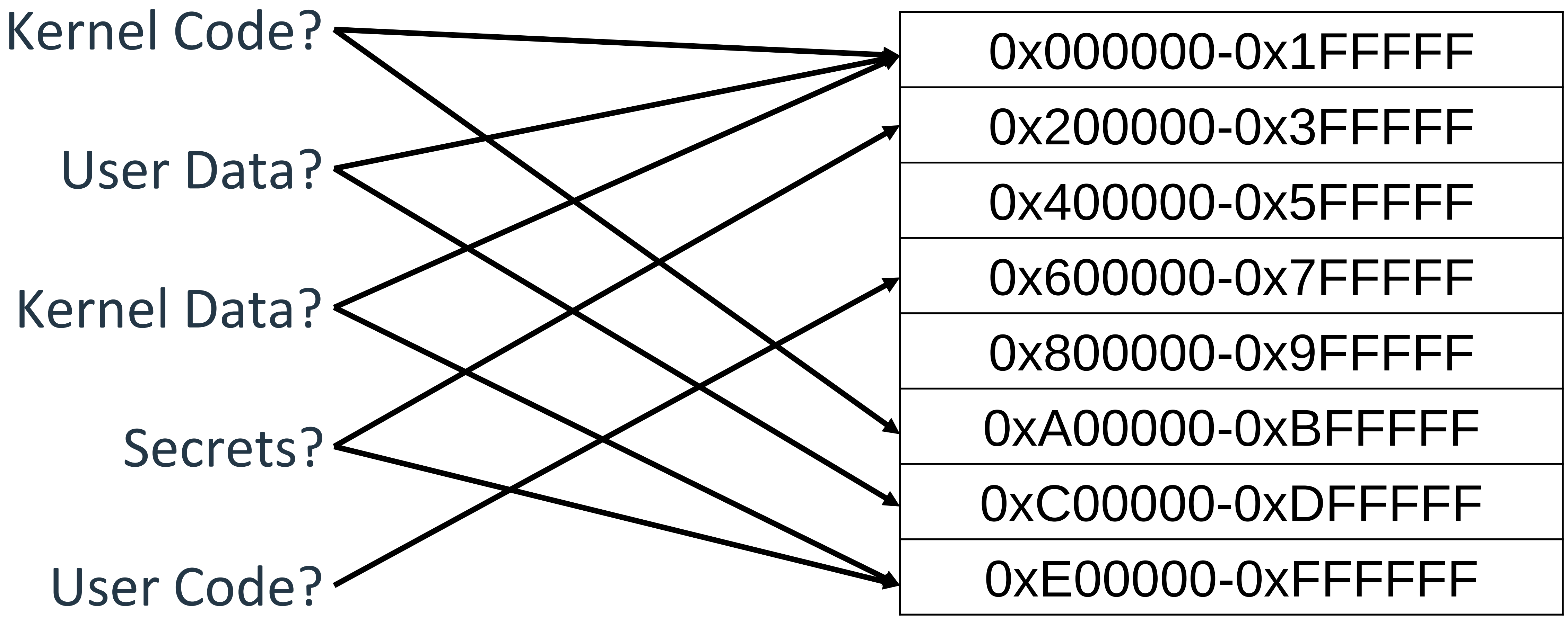
Primitive Exploit



The value is fixed, but we can choose the location (with some limitations).

Choosing a Target

- It’s not always easy to know what each guest memory region contains.



Choosing a Target

- The attacker has very little control over the plaintext values for the corrupted ciphertext.

Ciphertext

6D	4A	0D	CE	F9	82	C2	53	...

Plaintext

55	48	89	E5	48	83	EC	10	...

Ciphertext

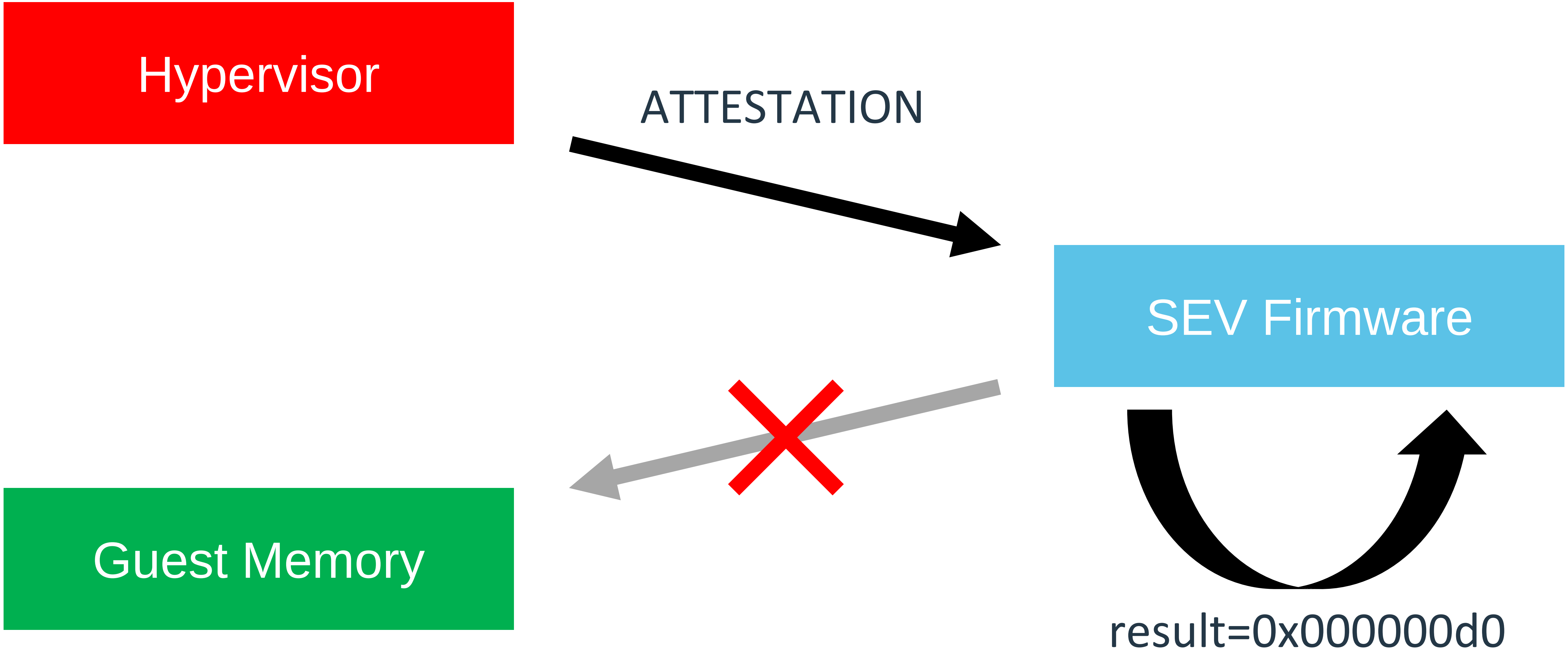
D0	00	00	00	C3	2F	1E	81	...

Plaintext

CC	54	C3	D4	B5	D2	29	06	...

Attacking the guest directly is possible, but …
… It's far from trivial and …
… Exploits will likely have to be tailored to specific workloads.

Attacking the Firmware



Guest Context Pages

- Guest context pages contain metadata about a guest.
- Marked as owned by the SEV firmware in the RMP using a special *CONTEXT* state.
- Guest context pages are encrypted.

Guest Context Pages

UMC Key Seed															
51	7E	7B	D1	B1	66	DA	FE	05	D3	E8	A3	F7	AE	E5	CA
Offline Encryption Key															
16	04	B4	B1	51	3C	05	21	76	EA	A4	9F	28	20	CD	54
…															
Launch Digest															
81	B6	EC	B6	BD	D9	93	20	C0	D1	C6	57	54	3D	C1	23
…															
Offline Encryption IV								Handle				Policy			
00	00	00	00	00	00	00	00	00	00	00	00	SMT			
State				ASID				CCXs				Guest Flags			
RUNNING				D6	01	00	00	FF	00	00	00	SEV-ES			
…															

Guest Context Pages

UMC Key Seed															
51	7E	7B	D1	B1	66	DA	FE	05	D3	E8	A3	F7	AE	E5	CA
Offline Encryption Key															
16	04	B4	B1	51	3C	05	21	76	EA	A4	9F	28	20	CD	54
…															
Launch Digest															
81	B6	EC	B6	BD	D9	93	20	C0	D1	C6	57	54	3D	C1	23
…															
Offline Encryption IV								Handle				Policy			
00	00	00	00	00	00	00	00	00	00	00	00	SMT			
State				ASID				CCXs				Guest Flags			
RUNNING				D6	01	00	00	FF	00	00	00	SEV-ES			
…															

Guest Context Pages

UMC Key Seed															
51	7E	7B	D1	B1	66	DA	FE	05	D3	E8	A3	F7	AE	E5	CA
Offline Encryption Key															
16	04	B4	B1	51	3C	05	21	76	EA	A4	9F	28	20	CD	54

- When the guest is created, the firmware uses a secure RNG to generate the UMC key seed.
- Before the guest is first used, the firmware programs the UMC key seed into all the Unified Memory Controllers (UMC) on the platform.
- The UMCs use this key seed to derive the guest's encryption key.

Guest Context Pages

<table>
<tr><td colspan="16">UMC Key Seed</td></tr>
<tr><td>51</td><td>7E</td><td>7B</td><td>D1</td><td>B1</td><td>66</td><td>DA</td><td>FE</td><td>05</td><td>D3</td><td>E8</td><td>A3</td><td>F7</td><td>AE</td><td>E5</td><td>CA</td></tr>
<tr><td colspan="16">Offline Encryption Key</td></tr>
<tr><td>16</td><td>04</td><td>B4</td><td>B1</td><td>51</td><td>3C</td><td>05</td><td>21</td><td>76</td><td>EA</td><td>A4</td><td>9F</td><td>28</td><td>20</td><td>CD</td><td>54</td></tr>
<tr><td colspan="16">…</td></tr>
<tr><td colspan="16">Launch Digest</td></tr>
<tr><td>81</td><td>B6</td><td>EC</td><td>B6</td><td>BD</td><td>D9</td><td>93</td><td>20</td><td>C0</td><td>D1</td><td>C6</td><td>57</td><td>54</td><td>3D</td><td>C1</td><td>23</td></tr>
<tr><td colspan="16">…</td></tr>
<tr><td colspan="8">Offline Encryption IV</td><td colspan="4">Handle</td><td colspan="4">Policy</td></tr>
<tr><td>00</td><td>00</td><td>00</td><td>00</td><td>00</td><td>00</td><td>00</td><td>00</td><td>00</td><td>00</td><td>00</td><td>00</td><td colspan="4">SMT</td></tr>
<tr><td colspan="4">State</td><td colspan="4">ASID</td><td colspan="4">CCXs</td><td colspan="4">Guest Flags</td></tr>
<tr><td colspan="4">RUNNING</td><td>D6</td><td>01</td><td>00</td><td>00</td><td>FF</td><td>00</td><td>00</td><td>00</td><td colspan="4">SEV-ES</td></tr>
<tr><td colspan="16">…</td></tr>
</table>

Guest Context Pages

<table>
<tr><td colspan="16">UMC Key Seed</td></tr>
<tr><td>D0</td><td>00</td><td>00</td><td>00</td><td>B1</td><td>66</td><td>DA</td><td>FE</td><td>05</td><td>D3</td><td>E8</td><td>A3</td><td>F7</td><td>AE</td><td>E5</td><td>CA</td></tr>
<tr><td colspan="16">Offline Encryption Key</td></tr>
<tr><td>16</td><td>04</td><td>B4</td><td>B1</td><td>51</td><td>3C</td><td>05</td><td>21</td><td>76</td><td>EA</td><td>A4</td><td>9F</td><td>28</td><td>20</td><td>CD</td><td>54</td></tr>
<tr><td colspan="16">…</td></tr>
<tr><td colspan="16">Launch Digest</td></tr>
<tr><td>81</td><td>B6</td><td>EC</td><td>B6</td><td>BD</td><td>D9</td><td>93</td><td>20</td><td>C0</td><td>D1</td><td>C6</td><td>57</td><td>54</td><td>3D</td><td>C1</td><td>23</td></tr>
<tr><td colspan="16">…</td></tr>
<tr><td colspan="8">Offline Encryption IV</td><td colspan="4">Handle</td><td colspan="4">Policy</td></tr>
<tr><td>00</td><td>00</td><td>00</td><td>00</td><td>00</td><td>00</td><td>00</td><td>00</td><td>00</td><td>00</td><td>00</td><td>00</td><td colspan="4">SMT</td></tr>
<tr><td colspan="4">State</td><td colspan="4">ASID</td><td colspan="4">CCXs</td><td colspan="4">Guest Flags</td></tr>
<tr><td colspan="4">RUNNING</td><td>D6</td><td>01</td><td>00</td><td>00</td><td>FF</td><td>00</td><td>00</td><td>00</td><td colspan="4">SEV-ES</td></tr>
<tr><td colspan="16">…</td></tr>
</table>

Guest Context Pages

UMC Key Seed															
D0	D0	00	00	00	66	DA	FE	05	D3	E8	A3	F7	AE	E5	CA
Offline Encryption Key															
16	04	B4	B1	51	3C	05	21	76	EA	A4	9F	28	20	CD	54
…															
Launch Digest															
81	B6	EC	B6	BD	D9	93	20	C0	D1	C6	57	54	3D	C1	23
…															
Offline Encryption IV								Handle				Policy			
00	00	00	00	00	00	00	00	00	00	00	00	SMT			
State				ASID				CCXs				Guest Flags			
RUNNING				D6	01	00	00	FF	00	00	00	SEV-ES			
…															

Guest Context Pages

<table>
<tr><td colspan="16">UMC Key Seed</td></tr>
<tr><td>D0</td><td>D0</td><td>D0</td><td>00</td><td>00</td><td>00</td><td>DA</td><td>FE</td><td>05</td><td>D3</td><td>E8</td><td>A3</td><td>F7</td><td>AE</td><td>E5</td><td>CA</td></tr>
<tr><td colspan="16">Offline Encryption Key</td></tr>
<tr><td>16</td><td>04</td><td>B4</td><td>B1</td><td>51</td><td>3C</td><td>05</td><td>21</td><td>76</td><td>EA</td><td>A4</td><td>9F</td><td>28</td><td>20</td><td>CD</td><td>54</td></tr>
<tr><td colspan="16">…</td></tr>
<tr><td colspan="16">Launch Digest</td></tr>
<tr><td>81</td><td>B6</td><td>EC</td><td>B6</td><td>BD</td><td>D9</td><td>93</td><td>20</td><td>C0</td><td>D1</td><td>C6</td><td>57</td><td>54</td><td>3D</td><td>C1</td><td>23</td></tr>
<tr><td colspan="16">…</td></tr>
<tr><td colspan="8">Offline Encryption IV</td><td colspan="4">Handle</td><td colspan="4">Policy</td></tr>
<tr><td>00</td><td>00</td><td>00</td><td>00</td><td>00</td><td>00</td><td>00</td><td>00</td><td>00</td><td>00</td><td>00</td><td>00</td><td colspan="4">SMT</td></tr>
<tr><td colspan="4">State</td><td colspan="4">ASID</td><td colspan="4">CCXs</td><td colspan="4">Guest Flags</td></tr>
<tr><td colspan="4">RUNNING</td><td>D6</td><td>01</td><td>00</td><td>00</td><td>FF</td><td>00</td><td>00</td><td>00</td><td colspan="4">SEV-ES</td></tr>
<tr><td colspan="16">…</td></tr>
</table>

Guest Context Pages

UMC Key Seed															
D0	D0	D0	D0	00	00	00	FE	05	D3	E8	A3	F7	AE	E5	CA
Offline Encryption Key															
16	04	B4	B1	51	3C	05	21	76	EA	A4	9F	28	20	CD	54
…															
Launch Digest															
81	B6	EC	B6	BD	D9	93	20	C0	D1	C6	57	54	3D	C1	23
…															
Offline Encryption IV								Handle				Policy			
00	00	00	00	00	00	00	00	00	00	00	00	SMT			
State				ASID				CCXs				Guest Flags			
RUNNING				D6	01	00	00	FF	00	00	00	SEV-ES			
…															

Guest Context Pages

UMC Key Seed															
D0	D0	D0	D0	D0	D0	D0	D0	D0	D0	D0	D0	D0	D0	D0	D0
Offline Encryption Key															
00	00	00	B1	51	3C	05	21	76	EA	A4	9F	28	20	CD	54
…															
Launch Digest															
81	B6	EC	B6	BD	D9	93	20	C0	D1	C6	57	54	3D	C1	23
…															
Offline Encryption IV								Handle				Policy			
00	00	00	00	00	00	00	00	00	00	00	00	SMT			
State				ASID				CCXs				Guest Flags			
RUNNING				D6	01	00	00	FF	00	00	00	SEV-ES			
…															

Guest Context Pages

Identical Key Seeds = Identical Encryption Keys

UMC Key Seed															
D0	D0	D0	D0	D0	D0	D0	D0	D0	D0	D0	D0	D0	D0	D0	D0
Offline Encryption Key															
00	00	00	B1	51	3C	05	21	76	EA	A4	9F	28	20	CD	54
…															
Launch Digest															
81	B6	EC	B6	BD	D9	93	20	C0	D1	C6	57	54	3D	C1	23
…															
Offline Encryption IV								Handle				Policy			
00	00	00	00	00	00	00	00	00	00	00	00	SMT			
State				ASID				CCXs				Guest Flags			
RUNNING				D6	01	00	00	FF	00	00	00	SEV-ES			
…															

Victim guest

UMC Key Seed															
D0	D0	D0	D0	D0	D0	D0	D0	D0	D0	D0	D0	D0	D0	D0	D0
Offline Encryption Key															
00	00	00	B1	51	3C	05	21	76	EA	A4	9F	28	20	CD	54
…															
Launch Digest															
81	B6	EC	B6	BD	D9	93	20	C0	D1	C6	57	54	3D	C1	23
…															
Offline Encryption IV								Handle				Policy			
00	00	00	00	00	00	00	00	00	00	00	00	SMT | DEBUG			
State				ASID				CCXs				Guest Flags			
RUNNING				D6	01	00	00	FF	00	00	00	SEV-ES			
…															

Attacker guest with debugging enabled

Guest Context Pages

- Guest context pages contain metadata about a guest.
- Marked as owned by the SEV firmware in the RMP using a special *CONTEXT* state.
- **Guest context pages are encrypted.**

Ciphertext

UMC Key Seed								
D0	D0	D0	D0	D0	D0	D0	D0	...

↔

Plaintext

UMC Key Seed								
3A	CB	3E	3D	F6	8F	1F	BC	...

Location-Dependent Encryption

- All guest context pages are encrypted using the same key, but use a physical-address-dependent IV.

Ciphertext at 0x2000 — IV=f(0x2000) — Plaintext at 0x2000

UMC Key Seed								
D0	D0	D0	D0	D0	D0	D0	D0	...

⟷

UMC Key Seed								
0F	50	3A	A0	2A	0A	01	15	...

Ciphertext at 0x5000 — IV=f(0x5000) — Plaintext at 0x5000

UMC Key Seed								
D0	D0	D0	D0	D0	D0	D0	D0	...

⟷

UMC Key Seed								
3A	CB	3E	3D	F6	8F	1F	BC	...

→ We have to use the same physical address for the guest and attacker context pages.

→ We have to shut the victim guest down before starting the attack guest.

Improved Exploit

1. Launch victim guest.
2. Corrupt *UMC key seed* with fixed values.
3. Run victim guest and records its encrypted memory.
4. Decommission victim guest.
5. Launch attacker guest at the same location with debug options enabled.
6. Corrupt *UMC key seed* with the same fixed values.
7. Use debug commands with the attacker guest to decrypt the memory of the victim guest.

```
root@server:~/firmware-vuln-poc# cargo run -- --pfn 0x171a24
    Finished dev [unoptimized + debuginfo] target(s) in 0.03s
Creating VM with identical UMC key seed
Raw page:
000: 0300000000000000110fa00000000000000000000000000000000000a98d95ff
020: 29d761f0c5bdc1b4b4a69fd2f37c829ca6d30d439252f7daefd72fcd45262053
040: 3c10be491d825e35ea4166261486e417187c679efcc2d2be8553f32c2c62bbf3
060: 27ac6d99214a2ce1fc37d35a94475ff377e67caacc43add86e908a9369207343
080: 14c197ffbcfade378bd1b33819051d5d3628b5eb71a0b84daefed27671e8e202
0a0: 0000000000000000000000000000000000000000000000000000000000000000
0c0: 0000000000000000000000000000000000000000000000000000000000000000
0e0: 0000000000000000000000000000000000000000000000000000000000000000
100: 0000000000008000880000000eeff0000f0ffffffffffffffff3f000000000000
120: 0000000000000000000000000000000000000000000000000000000000000000
140: 0000000000000000000000000000000000000000000000000000000000000000
160: c800000000000000000000000000000000000000000000000000000000000000
```

```
Secrets page:
imi_en: false
FMS: 00a00f11
gosvw: 000000000000000000000000a98d95ff
vmpck0: 29d761f0c5bdc1b4b4a69fd2f37c829ca6d30d439252f7daefd72fcd45262053
vmpck1: 3c10be491d825e35ea4166261486e417187c679efcc2d2be8553f32c2c62bbf3
vmpck2: 27ac6d99214a2ce1fc37d35a94475ff377e67caacc43add86e908a9369207343
vmpck3: 14c197ffbcfade378bd1b33819051d5d3628b5eb71a0b84daefed27671e8e202
VMSA tweak bitmap: 000000000080008800000000eeff0000f0ffffffffffffffff3f00
000000000000000000000000000000
tsc_factor: 200
```

CVE-2024-21978

Bug #2

- The firmware stores some certificates in non-volatile storage.
- The *INIT_EX* command can be used to ask the firmware to use regular memory instead of on-chip SPI flash for non-volatile storage.
- The hypervisor has to donate memory to the firmware by converting some memory into the *FIRMWARE* state.
- The firmware only checks that the memory is in the *FIRMWARE* state when *INIT_EX* is executed. All following accesses skip the access checks.
- The hypervisor can use the *PAGE_RECLAIM* command to ask the firmware to convert unused *FIRMWARE* memory back into hypervisor state.

→ *PAGE_RECLAIM* doesn’t whether the address is being used for non-volatile storage.

Rough Plan of Attack

1. Convert some memory into the *FIRMWARE* state.
2. Use that memory with *INIT_EX* as non-volatile storage.
3. Reclaim the memory using *PAGE_RECLAIM*.
4. Assign the memory to a guest.
5. Trigger a command that causes the firmware to non-volatile storage.

Can We Better Than Exploit #1?

- Last time we were limited by the fixed value of the memory corruption.
- The *PDH_GEN* command regenerates some certificates and writes ~3 pages of random data to the memory backing used for non-volatile storage.

Guest Context Pages

<table>
<tr><td colspan="16">UMC Key Seed</td></tr>
<tr><td>51</td><td>7E</td><td>7B</td><td>D1</td><td>B1</td><td>66</td><td>DA</td><td>FE</td><td>05</td><td>D3</td><td>E8</td><td>A3</td><td>F7</td><td>AE</td><td>E5</td><td>CA</td></tr>
<tr><td colspan="16">Offline Encryption Key</td></tr>
<tr><td>16</td><td>04</td><td>B4</td><td>B1</td><td>51</td><td>3C</td><td>05</td><td>21</td><td>76</td><td>EA</td><td>A4</td><td>9F</td><td>28</td><td>20</td><td>CD</td><td>54</td></tr>
<tr><td colspan="16">…</td></tr>
<tr><td colspan="16">Launch Digest</td></tr>
<tr><td>81</td><td>B6</td><td>EC</td><td>B6</td><td>BD</td><td>D9</td><td>93</td><td>20</td><td>C0</td><td>D1</td><td>C6</td><td>57</td><td>54</td><td>3D</td><td>C1</td><td>23</td></tr>
<tr><td colspan="16">…</td></tr>
<tr><td colspan="8">Offline Encryption IV</td><td colspan="4">Handle</td><td colspan="4">Policy</td></tr>
<tr><td>00</td><td>00</td><td>00</td><td>00</td><td>00</td><td>00</td><td>00</td><td>00</td><td>00</td><td>00</td><td>00</td><td>00</td><td colspan="4">SMT</td></tr>
<tr><td colspan="4">State</td><td colspan="4">ASID</td><td colspan="4">CCXs</td><td colspan="4">Guest Flags</td></tr>
<tr><td colspan="4">RUNNING</td><td>D6</td><td>01</td><td>00</td><td>00</td><td>FF</td><td>00</td><td>00</td><td>00</td><td colspan="4">SEV-ES</td></tr>
<tr><td colspan="16">…</td></tr>
</table>

Guest Context Pages

UMC Key Seed															
51	7E	7B	D1	B1	66	DA	FE	05	D3	E8	A3	F7	AE	E5	CA
Offline Encryption Key															
16	04	B4	B1	51	3C	05	21	76	EA	A4	9F	28	20	CD	54
…															

- Corrupting the *UMC key seed* isn't very useful because we have no control of the value.

Guest Context Pages

UMC Key Seed															
51	7E	7B	D1	B1	66	DA	FE	05	D3	E8	A3	F7	AE	E5	CA
Offline Encryption Key															
16	04	B4	B1	51	3C	05	21	76	EA	A4	9F	28	20	CD	54
…															
Launch Digest															
81	B6	EC	B6	BD	D9	93	20	C0	D1	C6	57	54	3D	C1	23
…															
Offline Encryption IV								Handle				Policy			
00	00	00	00	00	00	00	00	00	00	00	00	SMT			
State				ASID				CCXs				Guest Flags			
RUNNING				D6	01	00	00	FF	00	00	00	SEV-ES			
…															

Guest Context Pages

- After the UMC key seed has been programmed into the UMC, the encryption unit in the memory controller uses the address space identifier (*ASID*) to look up the encryption key for a guest.

Offline Encryption IV								Handle				Policy
00	00	00	00	00	00	00	00	00	00	00	00	SMT
State				ASID				CCXs				Guest Flags
RUNNING				D6	01	00	00	FF	00	00	00	SEV-ES
…												

Guest Context Pages

- After the UMC key seed has been programmed into the UMC, the encryption unit in the memory controller uses the address space identifier (*ASID*) to look up the encryption key for a guest.
- If we corrupt the *ASID* we can trick the firmware into using another guest's encryption keys.

Offline Encryption IV								Handle				Policy
F7	CC	FD	61	9E	D9	3D	FF	4B	97	D8	AF	VMSA_REG_PROT
State				ASID				CCXs				Guest Flags
LAUNCH				45	22	BB	A9	17	63	4B	23	SEV-ES
…												

Guest Context Pages

- After the UMC key seed has been programmed into the UMC, the encryption unit in the memory controller uses the address space identifier (*ASID*) to look up the encryption key for a guest.
- If we corrupt the *ASID* we can trick the firmware into using another guest's encryption keys.
- If we also corrupt the *Policy* we can issue debug commands for that other guest.

<table>
<tr><td colspan="8">Offline Encryption IV</td><td colspan="4">Handle</td><td>Policy</td></tr>
<tr><td>00</td><td>59</td><td>EF</td><td>80</td><td>2C</td><td>78</td><td>1D</td><td>CE</td><td>4D</td><td>99</td><td>67</td><td>51</td><td>DEBUG</td></tr>
<tr><td colspan="4">State</td><td colspan="4">ASID</td><td colspan="4">CCXs</td><td>Guest Flags</td></tr>
<tr><td colspan="4">INIT</td><td>E8</td><td>5F</td><td>73</td><td>60</td><td>24</td><td>87</td><td>5B</td><td>EA</td><td>(empty)</td></tr>
<tr><td colspan="13">…</td></tr>
</table>

Guest Context Pages

- After the UMC key seed has been programmed into the UMC, the encryption unit in the memory controller uses the address space identifier (*ASID*) to look up the encryption key for a guest.
- If we corrupt the *ASID* we can trick the firmware into using another guest's encryption keys.
- If we also corrupt the *Policy* we can issue debug commands for that other guest.
- There are only relatively few valid *ASIDs* (<509 or <1006 depending on the CPU).
- We can query both the *ASID* and the *policy* using the *GUEST_STATUS* command.

Offline Encryption IV								Handle				Policy
00	59	EF	80	2C	78	1D	CE	4D	99	67	51	DEBUG
State				ASID				CCXs				Guest Flags
INIT				E8	5F	73	60	24	87	5B	EA	(empty)
…												

Guest Context Pages

- After the UMC key seed has been programmed into the UMC, the encryption unit in the memory controller uses the address space identifier (*ASID*) to look up the encryption key for a guest.
- If we corrupt the *ASID* we can trick the firmware into using another guest's encryption keys.
- If we also corrupt the *Policy* we can issue debug commands for that other guest.
- There are only relatively few valid *ASIDs* (<509 or <1006 depending on the CPU).
- We can query both the *ASID* and the *policy* using the *GUEST_STATUS* command.

Offline Encryption IV								Handle				Policy
A9	AD	C0	7D	C3	40	CB	45	7E	BC	36	4E	SMT | DEBUG
State				ASID				CCXs				Guest Flags
INIT				F7	3E	19	2E	90	B3	52	C4	SEV-ES
…												

Guest Context Pages

- After the UMC key seed has been programmed into the UMC, the encryption unit in the memory controller uses the address space identifier (*ASID*) to look up the encryption key for a guest.
- If we corrupt the *ASID* we can trick the firmware into using another guest's encryption keys.
- If we also corrupt the *Policy* we can issue debug commands for that other guest.
- There are only relatively few valid *ASIDs* (<509 or <1006 depending on the CPU).
- We can query both the *ASID* and the *policy* using the *GUEST_STATUS* command.

Offline Encryption IV								Handle				Policy
A1	12	2B	90	00	7E	AC	F9	9E	FA	CA	73	SMT
State				ASID				CCXs				Guest Flags
RUNNING				FB	46	05	00	23	73	7A	01	SEV-ES
…												

Guest Context Pages

- After the UMC key seed has been programmed into the UMC, the encryption unit in the memory controller uses the address space identifier (*ASID*) to look up the encryption key for a guest.
- If we corrupt the *ASID* we can trick the firmware into using another guest's encryption keys.
- If we also corrupt the *Policy* we can issue debug commands for that other guest.
- There are only relatively few valid *ASIDs* (<509 or <1006 depending on the CPU).
- We can query both the *ASID* and the *policy* using the *GUEST_STATUS* command.

<table>
<tr><td colspan="8">Offline Encryption IV</td><td colspan="4">Handle</td><td colspan="4">Policy</td></tr>
<tr><td>DE</td><td>FD</td><td>97</td><td>D5</td><td>8F</td><td>B9</td><td>1C</td><td>F3</td><td>D1</td><td>7E</td><td>91</td><td>6E</td><td colspan="4">SMT | DEBUG</td></tr>
<tr><td colspan="4">State</td><td colspan="4">ASID</td><td colspan="4">CCXs</td><td colspan="4">Guest Flags</td></tr>
<tr><td colspan="4">INIT</td><td>89</td><td>1E</td><td>00</td><td>00</td><td>88</td><td>3F</td><td>71</td><td>91</td><td colspan="4">(empty)</td></tr>
<tr><td colspan="16">…</td></tr>
</table>

Guest Context Pages

- After the UMC key seed has been programmed into the UMC, the encryption unit in the memory controller uses the address space identifier (*ASID*) to look up the encryption key for a guest.
- If we corrupt the *ASID* we can trick the firmware into using another guest's encryption keys.
- If we also corrupt the *Policy* we can issue debug commands for that other guest.
- There are only relatively few valid *ASIDs* (<509 or <1006 depending on the CPU).
- We can query both the *ASID* and the *policy* using the *GUEST_STATUS* command.

Offline Encryption IV								Handle				Policy
E8	D5	E6	AA	43	CA	81	7E	5D	85	15	06	DEBUG
State				ASID				CCXs				Guest Flags
LAUNCH				A4	01	00	00	61	3F	08	CF	SEV-ES
…												

Guest Context Pages

- We repeatedly corrupt guest context pages until hit an *ASID* < 509/1006 and *Policy* allows debugging.
- The chances of getting everything right are about 1 in 20,000,000.
- We can corrupt 300 guest context pages per second.
- We expect to get a hit about once a day.
- This can be in advance before launching the victim guest.

Offline Encryption IV								Handle				Policy
E8	D5	E6	AA	43	CA	81	7E	5D	85	15	06	DEBUG
State				ASID				CCXs				Guest Flags
LAUNCH				A4	01	00	00	61	3F	08	CF	SEV-ES
…												

```
freax13@server:~/code/cve-2024-21978-poc$ bash exploit.sh
Corrupt guest context page so that ASID is in range 1..510
Smallest ASID: 0x0000001f iterations: 14052175 zeros: 10539628 unique asids: 31500727 elapsed time: 1d 19h 20m 31s
Creating VM with same ASID
[03, 00, 00, 00, 00, 00, 00, 00, 11, 0f, a0, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00,
00, 00, 00, f0, 51, a5, 03, 3f, 69, 6b, 93, e8, d8, 61, 0d, 2e, 5a, 45, f1, ea, 6d, bf, 49, fe, e4, a9, 2d, 8d, af, 7
6, 5e, 2e, 56, e0, fa, a9, b3, a7, e0, bc, 09, d9, 4f, 28, 5c, 9f, 84, d2, 7e, 34, eb, ea, 3f, 29, 88, 30, 01, 28, 65
, 8b, 73, 3c, 84, 00, ae, 4a, 74, a2, 7a, d1, c7, 4f, 63, 7f, 72, 7b, 3b, 2f, 08, b3, 1a, 8c, 99, 1b, ad, b5, 1d, 42,
 0b, 4d, 98, d4, 7d, c1, 0b, d6, 2f, b4, 6c, 6b, 51, a2, 92, 17, 3b, 01, e8, 82, 11, 1e, cb, cb, a2, 8f, c9, b0, 52,
1d, 1d, b7, d2, 25, 8d, 32, a9, 7a, 6f, 86, e4, 40, 44, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 0
0, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00
, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00,
 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 80, 00,
88, 00, 00, 00, 00, ee, ff, 00, 00, f0, ff, ff, ff, ff, ff, ff, ff, ff, 3f, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 0
0, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00
, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00,
 00, 00, 00, 00, 00, 00, 00, 00, 00, 00,00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 0
0, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00,00, 00, 00, 00, 00, 00, 00, 00, 00, 00,
 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00,0
0, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00
, 00, 00, 00, 00, 00, 00, 00, 00, 00,00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00,
00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00,00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00
, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00,00,
00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 0
0, 00, 00, 00, 00, 00, 00, 00, 00,00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00,
 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00,00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 0
```

Reusability of Exploits

- The exploits assume very little about the memory corruption:
- Fixed and random writes to RMP-protected memory are exploitable.
- Completely workload-independent
- A third bug I discovered, CVE-2023-31355, can be exploited using strategy #1 with very few changes.

Take-Aways

1. The hypervisor is very powerful:
Even very simple bugs can have a large security impact.

2. The firmware used by SEV (and other TEEs) deserves more attention from the researcher community.

3. Demand as much transparency as possible in all parts of the stack.

Thanks & Q/A

- Proof of Concepts are available on GitHub
  - github.com/freax13/cve-2024-21980-poc
  - github.com/freax13/cve-2024-21978-poc
  - github.com/freax13/cve-2023-31355-poc
- Follow me on Twitter: @13erbse