SEVerity: Code Injection Attacks against Encrypted Virtual Machines

Mathias Morbitzer, Sergej Proskurin, Martin Radev, Marko Dorfhuber and Erick Quintanar Salas, 27th May 2021
AMD SEV
AMD SEV

VM

hypervisor
AMD SEV

![Diagram of a cloud with a hypervisor and a VM.]
AMD SEV

Evil VM

hypervisor

SEVerity | Morbitzer, Proskurin, Radev, Dorfhuber and Quintanar Salas | 27th May 2021 | 2
An (incomplete) timeline on AMD SEV

SEV
(Memory confidentiality)

New CPUs

Software patches

Code Execution

Control flow modification

Memory extraction

Code Execution
An (incomplete) timeline on AMD SEV

SEVerity | Morbitzer, Proskurin, Radev, Dorfhuber and Quintanar Salas | 27th May 2021 | 3
© Fraunhofer
An (incomplete) timeline on AMD SEV

SEV (Memory confidentiality)
SEV-ES (Register protection)
Control flow modification (Hetzelt & Buhren)

SEVerity | Morbitzer, Proskurin, Radev, Dorfhuber and Quintanar Salas | 27th May 2021 | 3
© Fraunhofer
An (incomplete) timeline on AMD SEV

- SEV (Memory confidentiality)
- SEV-ES (Register protection)
- Control flow modification (Hetzelt & Buhren)
- Memory extraction (Morbitzer et al.)

SEVerity | Morbitzer, Proskurin, Radev, Dorfhuber and Quintanar Salas | 27th May 2021 | 3
An (incomplete) timeline on AMD SEV

SEV
(Memory confidentiality)

SEV-ES
(Register protection)

Control flow modification
(Hetzelt & Buhren)

Memory extraction
(Morbitzer et al.)

Code Execution
(Wilke et al.)

© Fraunhofer
An (incomplete) timeline on AMD SEV

- SEV (Memory confidentiality)
- SEV-ES (Register protection)
- Control flow modification (Hetzelt & Buhren)
- Memory extraction (Morbitzer et al.)
- Code Execution (Wilke et al.)
- New CPUs
An (incomplete) timeline on AMD SEV

- **SEV** (Memory confidentiality)
- **SEV-ES** (Register protection)
- Control flow modification (Hetzelt & Buhren)
- Memory extraction (Morbitzer et al.)
- Code Execution (Wilke et al.)

New CPUs
An (incomplete) timeline on AMD SEV

- SEV (Memory confidentiality)
- SEV-ES (Register protection)
- Control flow modification (Hetzelt & Buhren)
- Memory extraction (Morbitzer et al.)
- Code Execution (Radev & Morbitzer)
- Code Execution (Wilke et al.)
- New CPUs

SEVerity | Morbitzer, Proskurin, Radev, Dorfhuber and Quintanar Salas | 27th May 2021 | 3
© Fraunhofer
An (incomplete) timeline on AMD SEV

- SEV (Memory confidentiality)
- SEV-ES (Register protection)
- Control flow modification (Hetzelt & Buhren)
- Memory extraction (Morbitzer et al.)
- Code Execution (Wilke et al.)
- Code Execution (Radev & Morbitzer)

New CPUs
Software patches
An (incomplete) timeline on AMD SEV

- SEV (Memory confidentiality)
- SEV-ES (Register protection)
- Control flow modification (Hetzelt & Buhren)
- Memory extraction (Morbitzer et al.)
- Code Execution (Wilke et al.)
- Code Execution (Radev & Morbitzer)
- Code Execution with SEVerity
- New CPUs
- Software patches

SEVerity | Morbitzer, Proskurin, Radev, Dorfhuber and Quintanar Salas | 27th May 2021 | 3
Attack Overview

Guest Physical Memory

SLAT

Host Physical Memory
Attack Overview

Guest Physical Memory

trigger

SLAT

Host Physical Memory

1
Attack Overview

Guest Physical Memory

trigger

payload

SLAT

Host Physical Memory

1

2
Attack Overview

Guest Physical Memory

- Trigger
- Payload

SLAT

Host Physical Memory

1

2

3
Identifying the trigger

- Use of Non Maskable Interrupts (NMIs)
Identifying the trigger

- Use of Non Maskable Interrupts (NMIs)

- VM executes NMI handler immediately
Identifying the trigger

- Use of Non Maskable Interrupts (NMIs)
- VM executes NMI handler immediately → perfect trigger
Identifying the trigger

- Use of Non Maskable Interrupts (NMIs)
- VM executes NMI handler immediately → perfect trigger
- Analyze kernel binary to determine location of NMI handler
Identifying the trigger

- Use of Non Maskable Interrupts (NMIs)
- VM executes NMI handler immediately → perfect trigger
- Analyze kernel binary to determine location of NMI handler
- KASLR randomizes the kernel’s offset in the VM’s virtual and physical memory
Identifying the trigger

- Use of Non Maskable Interrupts (NMIs)
- VM executes NMI handler immediately → perfect trigger
- Analyze kernel binary to determine location of NMI handler
- KASLR randomizes the kernel’s offset in the VM’s virtual and physical memory
- Three methods to determine KASLR offset
## Identifying the trigger: probing NMI handler

<table>
<thead>
<tr>
<th>Guest Physical Memory</th>
<th>SLAT</th>
<th>Host Physical Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>trigger</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<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>
1 Identifying the trigger: probing NMI handler

Guest Physical Memory

- trigger

SLAT
- pwx
- pwx
- pwx
- pwx

Host Physical Memory

- locked
- locked
- locked
- locked
Identifying the trigger: probing NMI handler

<table>
<thead>
<tr>
<th>Guest Physical Memory</th>
<th>SLAT</th>
<th>Host Physical Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>trigger</td>
<td>pw-</td>
<td></td>
</tr>
<tr>
<td></td>
<td>pw-</td>
<td></td>
</tr>
<tr>
<td></td>
<td>pw-</td>
<td></td>
</tr>
<tr>
<td></td>
<td>pw-</td>
<td></td>
</tr>
</tbody>
</table>
Identifying the trigger: probing NMI handler

Guest Physical Memory

trigger

Host Physical Memory

SLAT

pw-

pw-

pw-

pw-

© Fraunhofer

SEVerity | Morbitzer, Proskurin, Radev, Dorfhuber and Quintanar Salas | 27th May 2021 | 6
Identifying the trigger: probing NMI handler

Guest Physical Memory

trigger

SLAT

pw-

pw-

pw-

pw-

Host Physical Memory

trigger

© Fraunhofer
Attack Overview

Guest Physical Memory -> SLAT -> Host Physical Memory

- trigger
- payload
Identifying the payload: virtio without SEV

```
VM

available ring
...1 2 3

used ring
...1 2

descriptor table
1 buf1 len1
2 buf2 len2
3 buf3 len3
4 buf4 len4

buffers
buf1
buf2
buf3
buf4

HV

virtual device
```
Identifying the payload: virtio without SEV
Identifying the payload: virtio without SEV

VM

driver

available ring
...1 2 3 ...

used ring
...1 2 ...

descriptor table
1 | buf1 | len1
2 | buf2 | len2
3 | buf3 | len3
4 | buf4 | len4

buffers
buf1
buf2
buf3
buf4

HV

virtual device

a

b

© Fraunhofer
Identifying the payload: virtio without SEV

Diagram:
- VM
  - driver
- HV
  - virtual device
  - available ring
    - ...1 2 3 ...
  - used ring
    - ...1 2 ...
  - descriptor table
    - 1 buf1 len1
    - 2 buf2 len2
    - 3 buf3 len3
    - 4 buf4 len4
  - buffers
    - buf1
    - buf2
    - buf3
    - buf4

Diagram connections:
- A
- B
- C
Identifying the payload: virtio without SEV
Identifying the payload: virtio without SEV

Diagram:
- VM
  - Driver
  - Available ring:
    - 1
    - 2
    - 3
  - Used ring:
    - 1
    - 2
- HV
  - Virtual device
  - Descriptor table:
    - buf1 len1
    - buf2 len2
    - buf3 len3
    - buf4 len4
  - Buffers:
    - buf1
    - buf2
    - buf3
    - buf4
Identifying the payload: virtio with SEV
Identifying the payload: virtio with SEV

VM

Shared

available ring

used ring

descriptor table

buffers

HV

driver

packet buffer

virtual device

buf1

buf2

buf3

buf4

buf1 len1

buf2 len2

buf3 len3

buf4 len4
Identifying the payload: virtio with SEV

VM

Shared

HV

driver

packet buffer

available ring

... 1 2 3

used ring

... 1 2

descriptor table

1 buf1 | len1
2 buf2 | len2
3 buf3 | len3
4 buf4 | len4

buffers

buf1
buf2
buf3
buf4

a
b
c
d

SEVerity | Morbitzer, Proskurin, Radev, Dorfhuber and Quintanar Salas | 27th May 2021 | 9
Identifying the payload: virtio with SEV
Identifying the payload: virtio with SEV

**Diagram:**

- **VM:**
  - driver
  - packet buffer

- **Shared:**
  - available ring: ...1 2 3 ...
  - used ring: ...1 2 ...
  - descriptor table:
    - 1: buf1 len1
    - 2: buf2 len2
    - 3: buf3 len3
    - 4: buf4 len4

- **HV:**
  - virtual device

**Connections:**

- a: available ring to virtual device
- b: used ring to virtual device
- c: descriptor table to virtual device
- d: virtual device to driver
- e: virtual device to packet buffer
- f: driver to packet buffer
Identifying the payload: virtio with SEV

VM

Shared

HV

driver

available ring

... 1 2 3

used ring

... 1 2

descriptor table

1 buf1 len1
2 buf2 len2
3 buf3 len3
4 buf4 len4

buffers

buf1
buf2
buf3
buf4

packet buffer
Identifying the payload: virtio with SEV

VM

Shared

HV

driver

available ring

used ring

descriptor table

buffers

packet buffer

virtual device

buf1 len1
buf2 len2
buf3 len3
buf4 len4

pwx
pwx
pwx
pwx
Identifying the payload: virtio with SEV

VM

Shared

HV

driver

available ring

used ring

descriptor table

buffers

packet buffer

virtual device

SEVerity | Morbitzer, Proskurin, Radev, Dorfhuber and Quintanar Salas | 27th May 2021 | 9
© Fraunhofer
Identifying the payload: virtio with SEV

- VM
- Shared
  - available ring
  - used ring
  - descriptor table
    - buf1 len1
    - buf2 len2
    - buf3 len3
    - buf4 len4
- HV
  - virtual device

buffers
- wx
- wx
- wx
- wx
Identifying the payload: virtio with SEV

- **VM**
  - p-x

- **Shared**
  - Available ring:
    - 1
    - 2
    - 3
  - Used ring:
    - 1
    - 2
  - Descriptor table:
    - buf1 | len1
    - buf2 | len2
    - buf3 | len3
    - buf4 | len4
  - Buffers:
    - wx
    - wx
    - wx

- **HV**
  - Virtual device
Identifying the payload: virtio with SEV

VM

p-x

Shared

available ring
... 1 2 3 ...

used ring
... 1 2 ...

descriptor table
1 buf1 len1
2 buf2 len2
3 buf3 len3
4 buf4 len4

buffers
-wx
-wx
-wx
-wx

HV

virtual device

SEVerity | Morbitzer, Proskurin, Radev, Dorfhuber and Quintanar Salas | 27th May 2021 | 9
© Fraunhofer
Attack Overview

Guest Physical Memory

trigger

payload

SLAT

Host Physical Memory

1

2

© Fraunhofer
Remapping

Instruction fetch
trigger page

SLAT

Trigger page

Payload page

0x700
0xf00
0x600
0xc00
Remapping

Instruction fetch trigger page

SLAT

Trigger page

Payload page

0x700

0xf00

0x600

0xc00

Trigger

Arbitrary data
3 Remapping

- Instruction fetch
- Trigger page
- SLAT

Diagram:
- Trigger page
  - 0x700
  - 0xf00
- Payload page
  - 0x600
  - 0x700
  - 0xc00
Remapping

Instruction fetch

Trigger page

SLAT

Remap

Trigger page

Payload page

0x700

0xf00

0x600

0xc00

Trigger

Arbitrary data

Payload
Attack Overview

Guest Physical Memory
- trigger
- payload

SLAT

Host Physical Memory
- 1
- 2
- 3
A (still incomplete) timeline on AMD SEV

- SEV (Memory confidentiality)
- SEV-ES (Register protection)
- Code Execution (Wilke et al.)
- Memory extraction (Morbitzer et al.)
- Code Execution (Radev & Morbitzer)
- Code Execution with SEVerity
- New CPUs
- Software patches

Control flow modification (Hetzelt & Buhren)
A (still incomplete) timeline on AMD SEV

SEV
(Memory confidentiality)

SEV-ES
(Register protection)

New CPUs

Software patches

SEV-SNP
(Integrity protection)

Control flow modification
(Hetzelt & Buhren)

Memory extraction
(Morbitzer et al.)

Code Execution
(Wilke et al.)

Code Execution
(Radev & Morbitzer)

Code Execution with SEVerity

SEVerity | Morbitzer, Proskurin, Radev, Dorfhuber and Quintanar Salas | 27th May 2021 | 13
© Fraunhofer
A (still incomplete) timeline on AMD SEV

SEV
(Memory confidentiality)
SEV-ES
(Register protection)
Code Execution
(Wilke et al.)
New CPUs
Software patches
SEV-SNP
(Integrity protection)
Control flow modification
(Hetzelt & Buhren)
Memory extraction
(Morbitzer et al.)
Code Execution
(Radev & Morbitzer)
Code Execution
with SEVerity

SEVerity | Morbitzer, Proskurin, Radev, Dorfhuber and Quintanar Salas | 27th May 2021 | 13
© Fraunhofer
Attack overview with SEV-SNP

Guest Physical Memory
- trigger
- payload

SLAT

Host Physical Memory

1

2
Attack overview with SEV-SNP

Guest Physical Memory

<table>
<thead>
<tr>
<th>trigger</th>
</tr>
</thead>
<tbody>
<tr>
<td>payload</td>
</tr>
</tbody>
</table>

SLAT

Host Physical Memory

1

2
Conclusion

- SEVerity allows to execute arbitrary code in SEV-protected VMs
Conclusion

- SEVerity allows to execute arbitrary code in SEV-protected VMs
  → Using page tracking and SLAT remapping
Conclusion

- SEVerity allows to execute arbitrary code in SEV-protected VMs → Using page tracking and SLAT remapping

- PoC uses Linux & virtio → but general concept applies to all guest OS
Conclusion

- SEVerity allows to execute arbitrary code in SEV-protected VMs → Using page tracking and SLAT remapping

- PoC uses Linux & virtio → but general concept applies to all guest OS

- SEV and SEV-ES are vulnerable to various attacks
Conclusion

- SEVerity allows to execute arbitrary code in SEV-protected VMs → Using page tracking and SLAT remapping

- PoC uses Linux & virtio → but general concept applies to all guest OS

- SEV and SEV-ES are vulnerable to various attacks

- SEV-SNP adds integrity protection
Conclusion

- SEVerity allows to execute arbitrary code in SEV-protected VMs
  → Using page tracking and SLAT remapping

- PoC uses Linux & virtio
  → but general concept applies to all guest OS

- SEV and SEV-ES are vulnerable to various attacks

- SEV-SNP adds integrity protection
  - SEV-SNP capable CPUs available since Q1 2021
Conclusion

- SEVerity allows to execute arbitrary code in SEV-protected VMs → Using page tracking and SLAT remapping

- PoC uses Linux & virtio → but general concept applies to all guest OS

- SEV and SEV-ES are vulnerable to various attacks

- SEV-SNP adds integrity protection
  - SEV-SNP capable CPUs available since Q1 2021
  - First software patches also available