## バイナリ書き換えを用いたシステムコールフック機構

SNIA 日本支部 次世代メモリ & ストレージ分科会 2023 年 6 月 2 日

安形憲一、田崎創、オブランピエールルイ、石黒健太

¹IIJ 技術研究所 ²法政大学

#### About this Work

- zpoline is a system call hook mechanism for x86-64 CPUs
- We made this for transparently applying user-space OS subsystems to existing applications

The source code of zpoline has been publicly available at <a href="https://github.com/yasukata/zpoline">https://github.com/yasukata/zpoline</a> since October 2021

USENIX ATC 2023 で発表予定の内容です

https://www.usenix.org/conference/atc23/presentation/yasukata

 System calls are the primary interface for user-space programs to communicate with OS kernels

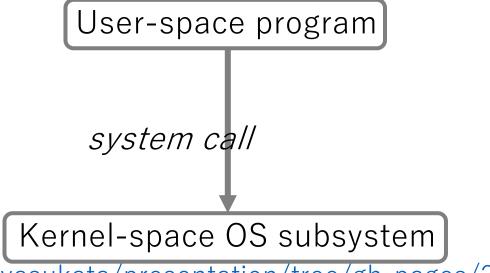
 System calls are the primary interface for user-space programs to communicate with OS kernels

User-space program

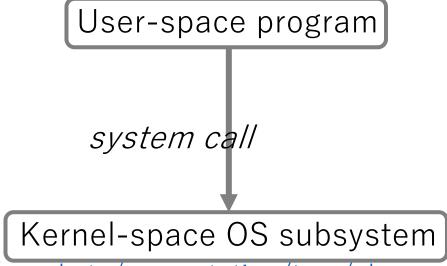
 System calls are the primary interface for user-space programs to communicate with OS kernels

User-space program

 System calls are the primary interface for user-space programs to communicate with OS kernels



- System calls are the primary interface for user-space programs to communicate with OS kernels
- A system call hook mechanism intercepts a system call



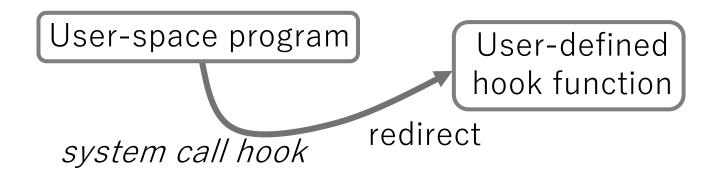
- System calls are the primary interface for user-space programs to communicate with OS kernels
- A system call hook mechanism intercepts a system call

```
User-space program intercept system call hook
```

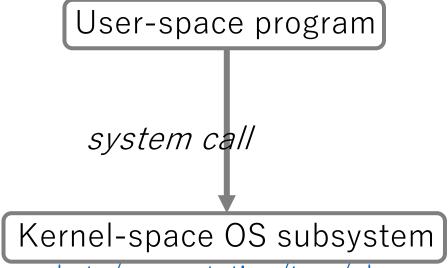
- System calls are the primary interface for user-space programs to communicate with OS kernels
- A system call hook mechanism intercepts a system call, and redirects the execution to a user-defined hook function

```
User-space program intercept system call hook
```

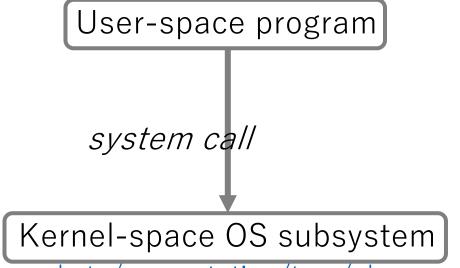
- System calls are the primary interface for user-space programs to communicate with OS kernels
- A system call hook mechanism intercepts a system call, and redirects the execution to a user-defined hook function



 System call hook mechanisms allow us to <u>transparently</u> apply user-space OS subsystems to existing applications

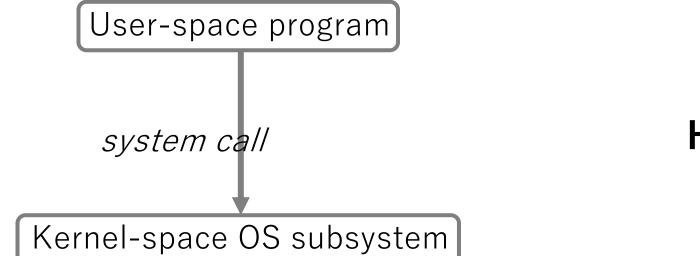


 System call hook mechanisms allow us to <u>transparently</u> apply <u>user-space OS subsystems</u> to existing applications



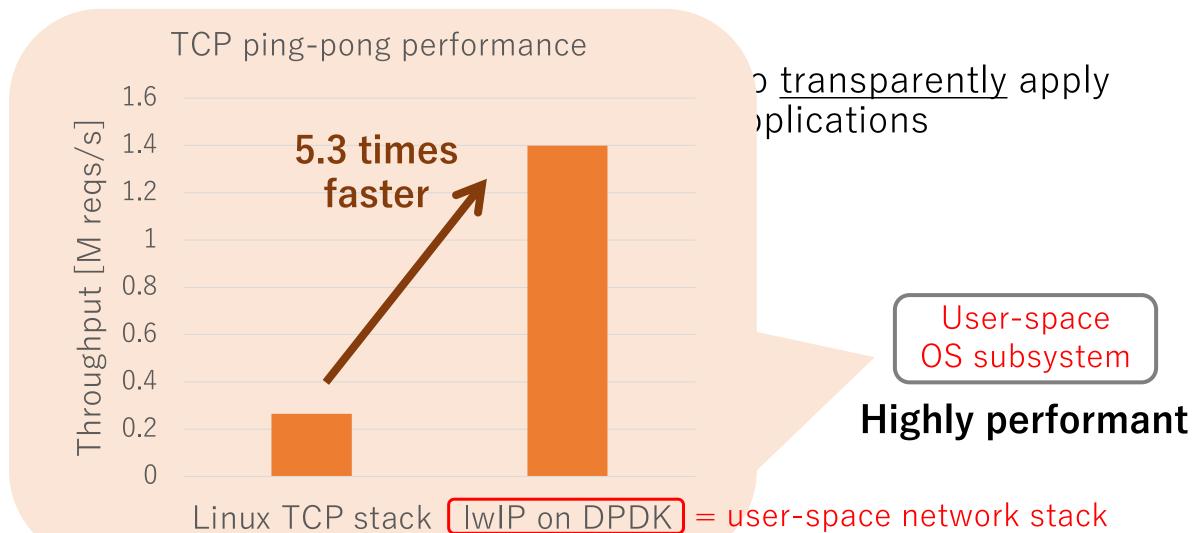
User-space OS subsystem

 System call hook mechanisms allow us to <u>transparently</u> apply <u>user-space OS subsystems</u> to existing applications

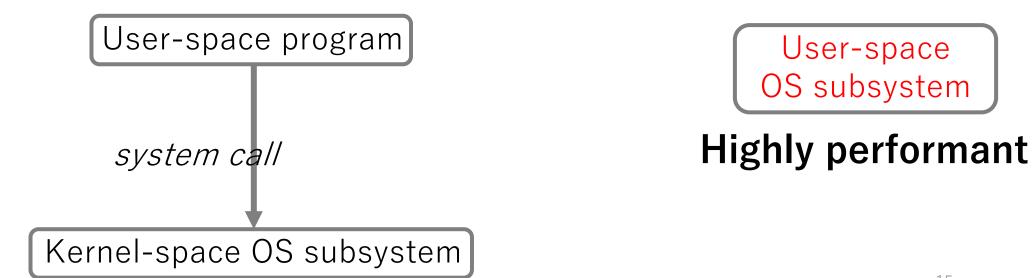


User-space OS subsystem

**Highly performant** 

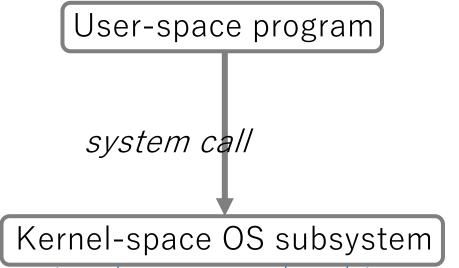


 System call hook mechanisms allow us to <u>transparently</u> apply <u>user-space OS subsystems</u> to existing applications



 System call hook mechanisms allow us to <u>transparently</u> apply <u>user-space OS subsystems</u> to existing applications

Normally, adaptation requires changes of a user-space program to apply a specific API of a user-space OS subsystem



User-space OS subsystem

**Highly performant** 

 System call hook mechanisms allow us to <u>transparently</u> apply <u>user-space OS subsystems</u> to existing applications

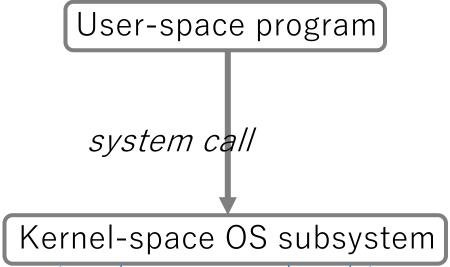
Normally, adaptation requires changes of a user-space program to apply a specific API of a user-space OS subsystem



**Highly performant** 

 System call hook mechanisms allow us to <u>transparently</u> apply <u>user-space OS subsystems</u> to existing applications

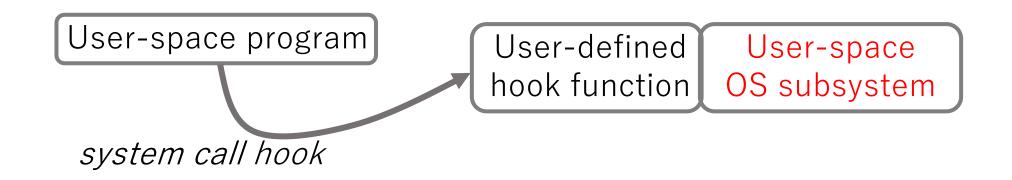
If we use a system call hook mechanism, ...



User-space OS subsystem

 System call hook mechanisms allow us to <u>transparently</u> apply <u>user-space OS subsystems</u> to existing applications

If we use a system call hook mechanism, ...



 System call hook mechanisms allow us to <u>transparently</u> apply <u>user-space OS subsystems</u> to existing applications

If we use a system call hook mechanism, ...

no modification of the user-space program is necessary



 System call hook mechanisms allow us to <u>transparently</u> apply <u>user-space OS subsystems</u> to existing applications

If we use a system call hook mechanism, ...

no modification of the user-space program is necessary



#### **Existing Mechanisms**

- ptrace
- Syscall User Dispatch (SUD)
- int3 signaling technique
- LD\_PRELOAD trick
- Binary rewriting techniques

• • • •

ms allow us to <u>transparently</u> apply to existing applications

ok mechanism, ...

user-space program is necessary

am

User-defined User-space hook function OS subsystem

. . . .

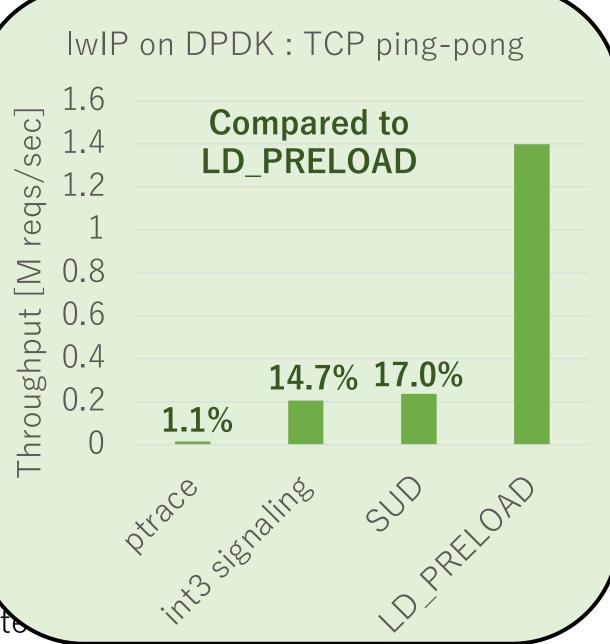
system call hook ← There is no perfect hook mechanism

#### **Existing Mechanisms**

- ptrace
- Syscall User Dispatch (SUD)
- int3 signaling technique
- LD\_PRELOAD trick
- Binary rewriting techniques

• • • •

system call hook

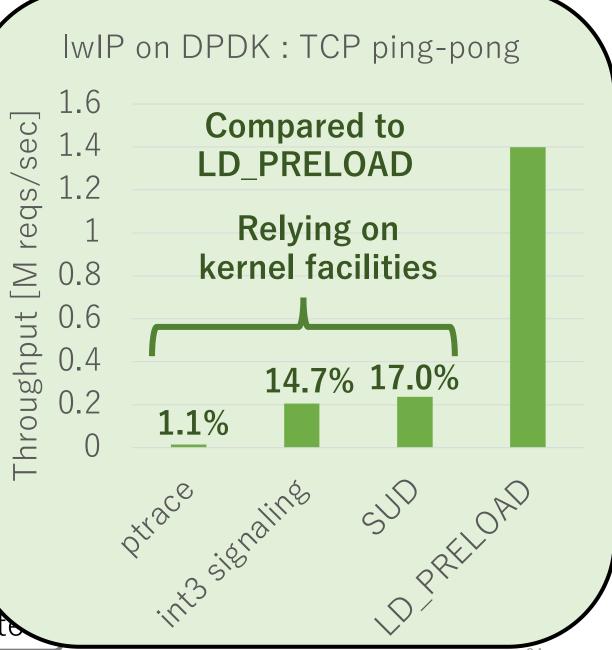


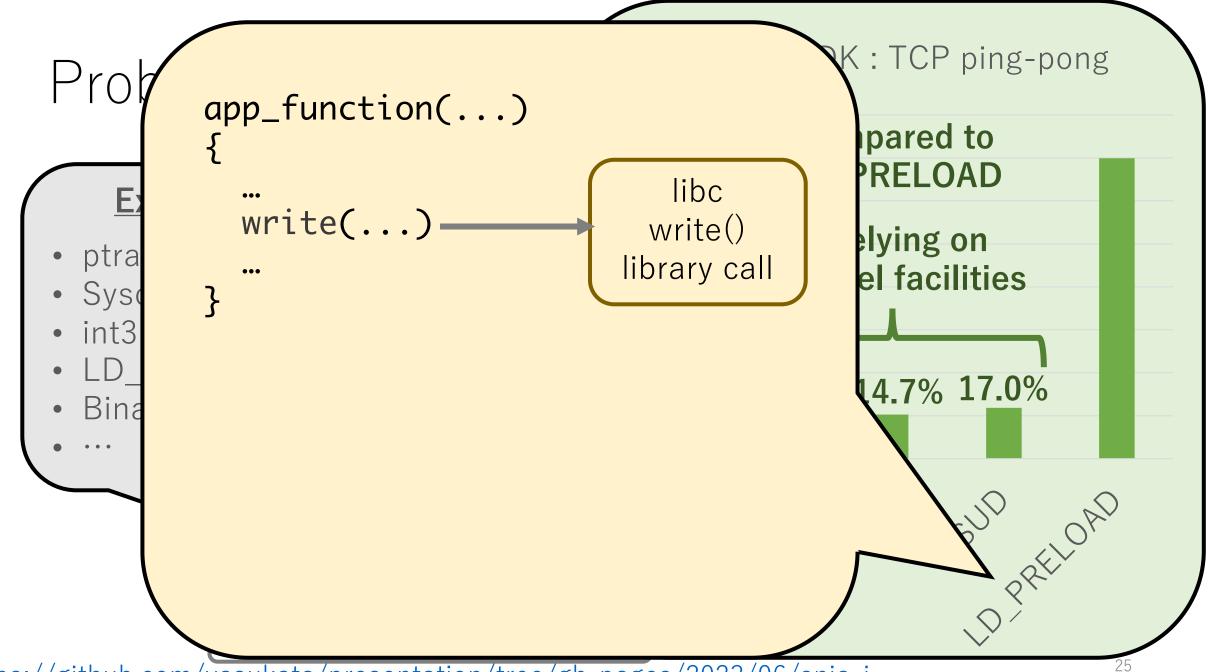
#### **Existing Mechanisms**

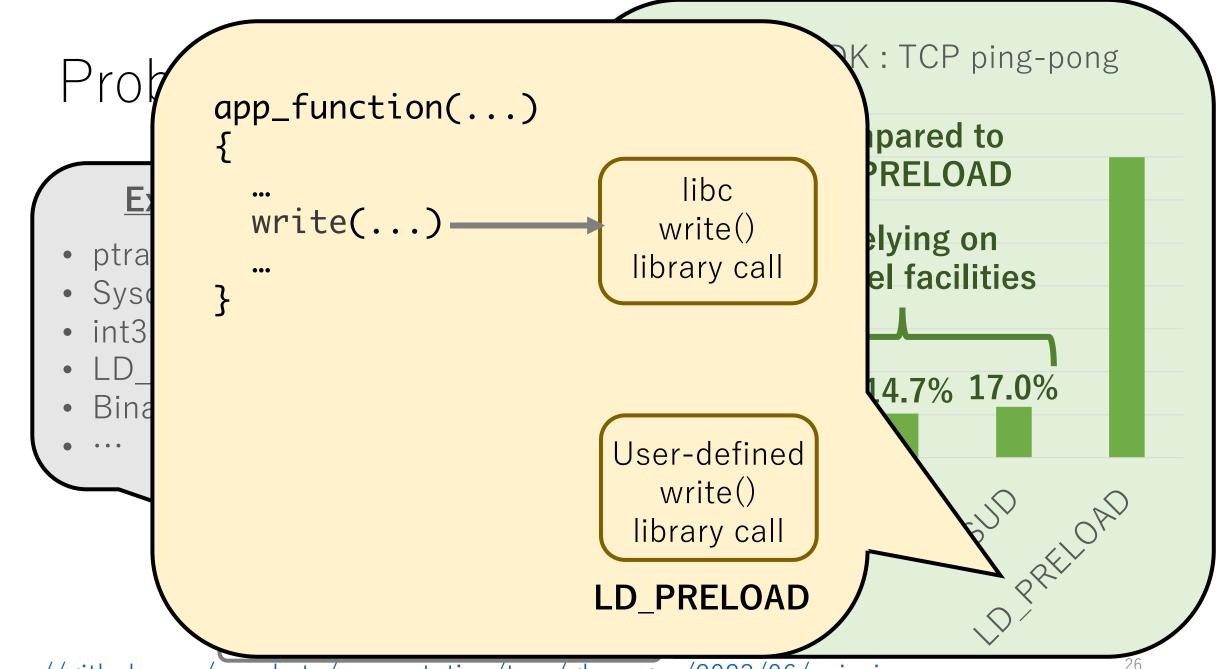
- ptrace
- Syscall User Dispatch (SUD)
- int3 signaling technique
- LD\_PRELOAD trick
- Binary rewriting techniques

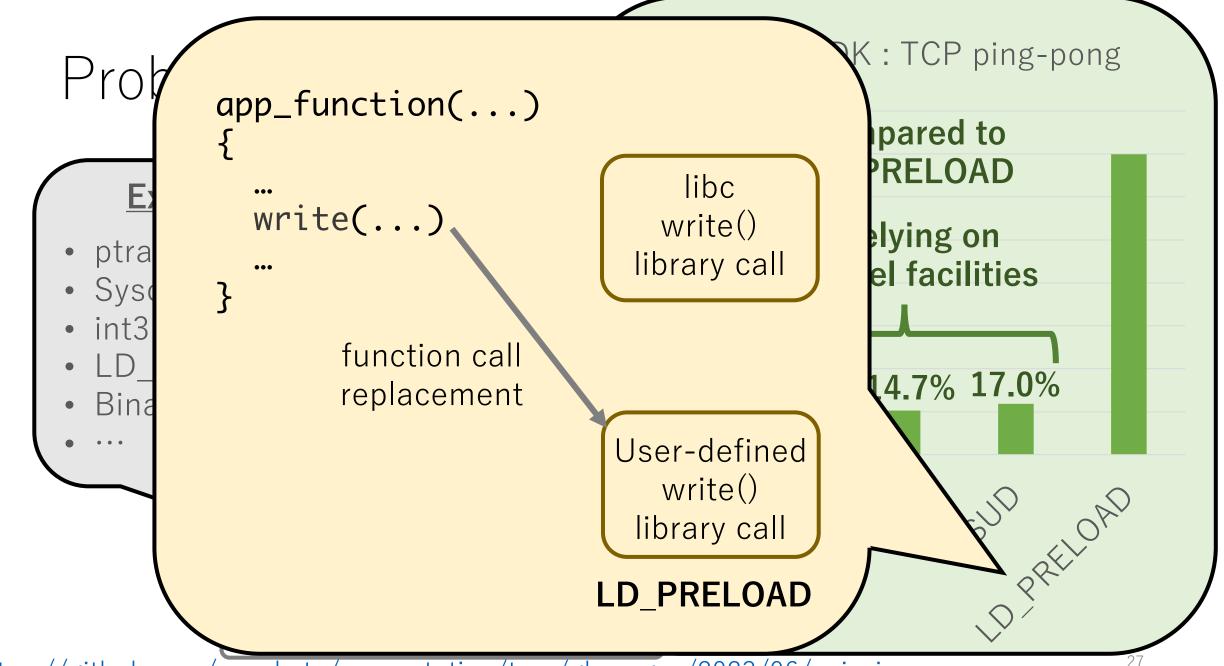
• • •

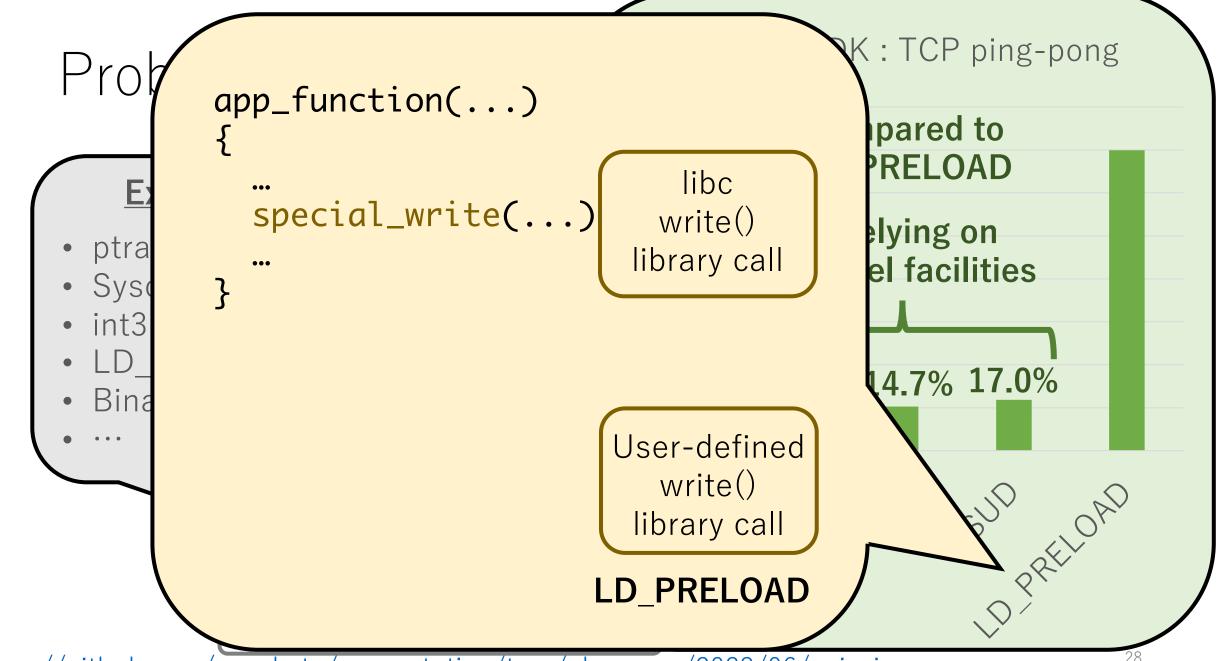
system call hook

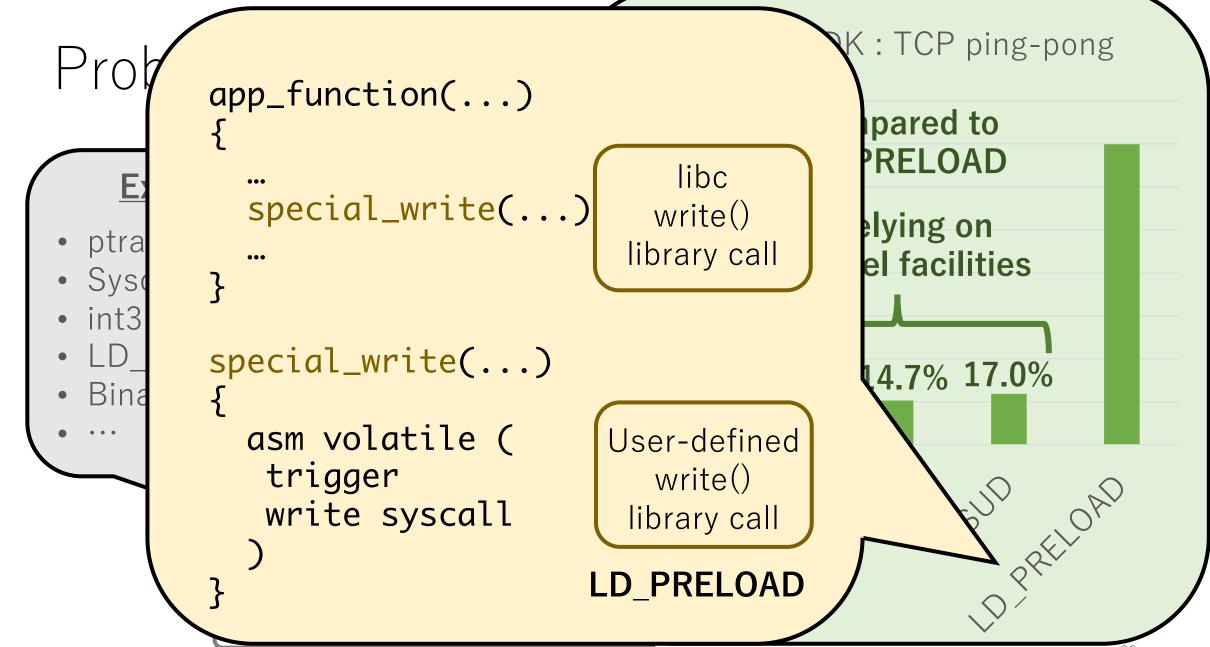


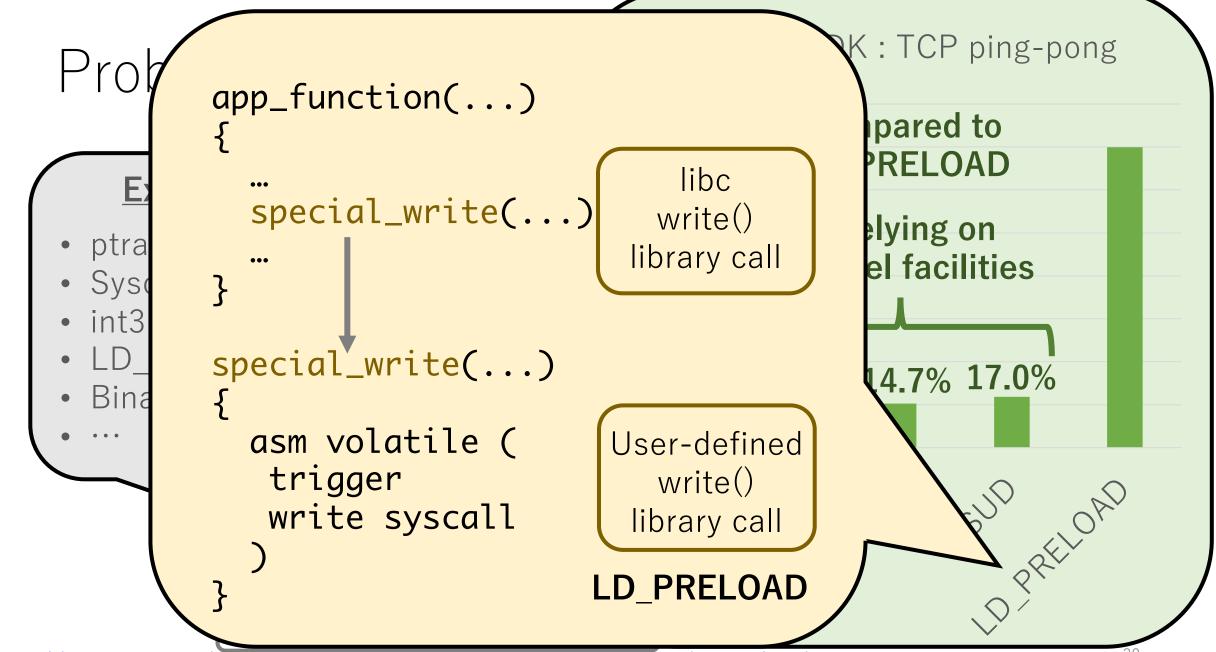


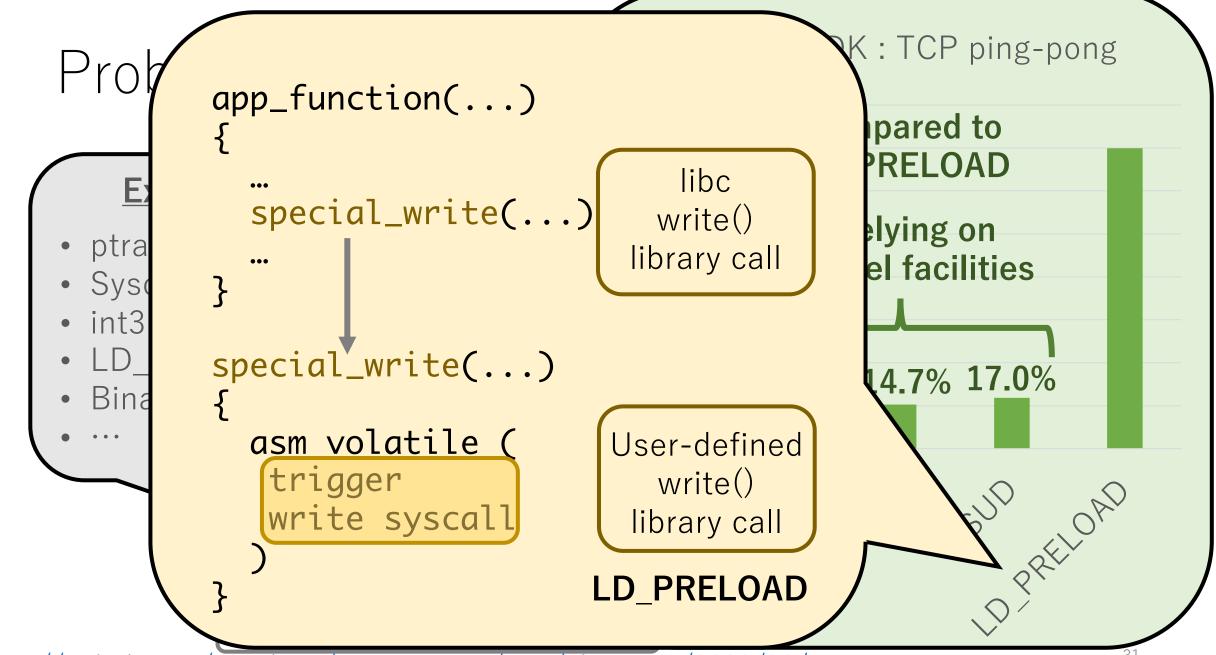


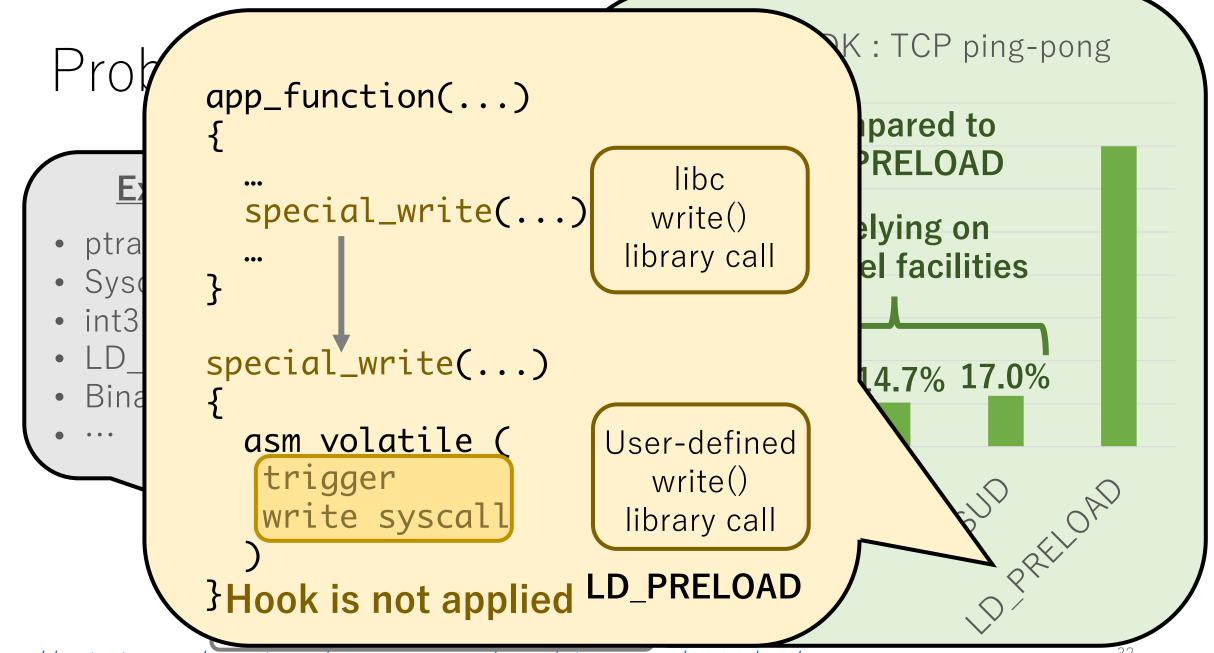


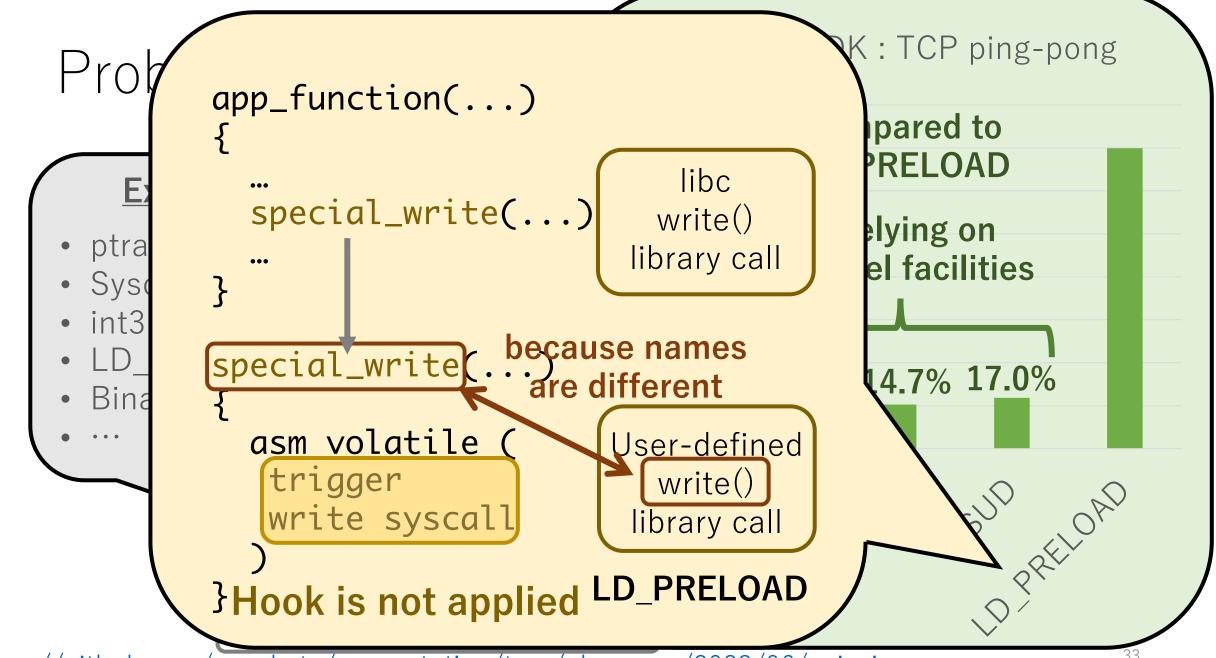


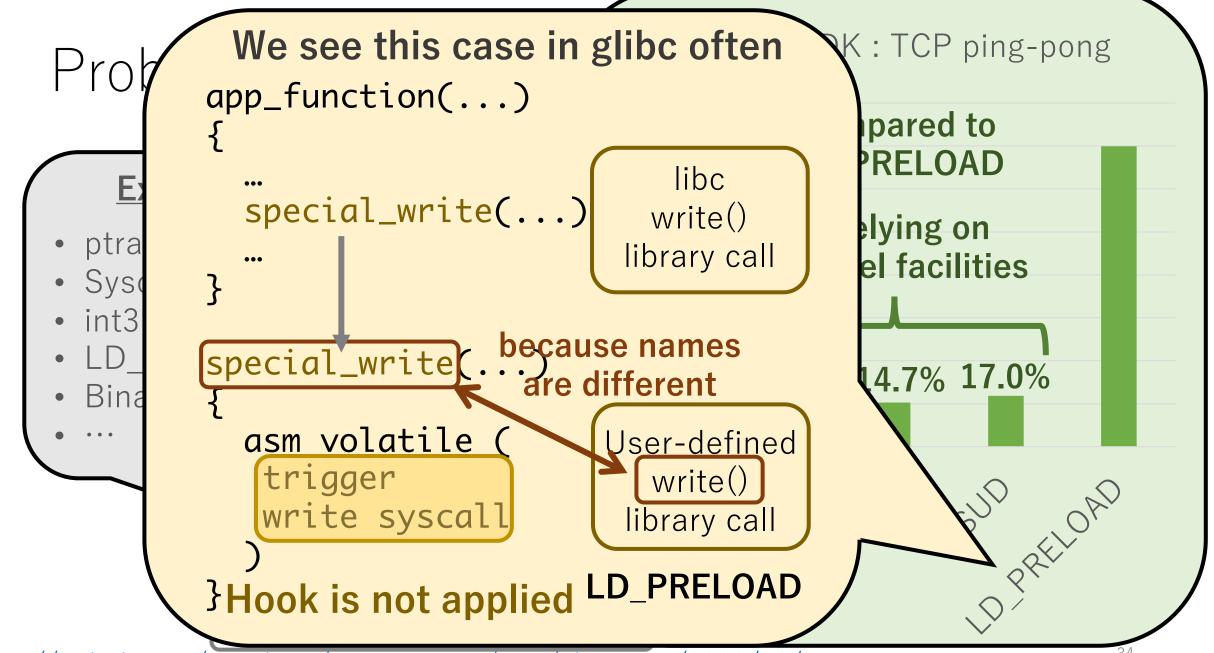












#### **Existing Mechanisms**

- ptrace
- Syscall User Dispatch (SUD)
- int3 signaling technique
- LD\_PRELOAD trick
- Binary rewriting techniques

• • • •

ms allow us to <u>transparently</u> apply to existing applications

ok mechanism, ...

user-space program is necessary

am

User-defined hook function O

User-space OS subsystem

system call hook ← There is no perfect hook mechanism

#### **Existing Mechanisms**

# High performance atch (SUD) System of the SUD) int3 signaling technology

- LD PRELOAD trick
- Binary rewriting techniques

ms allow us to <u>transparently</u> apply to existing applications

ok mechanism, ...

user-space program is necessary

am

User-defined hook function

User-space OS subsystem

system call hook There is no perfect hook mechanism

#### Problem

#### **Existing Mechanisms**

High performance atch (SUD)

• Systemaling Letter Penalty

- Sometimes fail to hook

ms allow us to <u>transparently</u> apply to existing applications

ok mechanism, ...

user-space program is necessary

am

User-defined User-space hook function OS subsystem

system call hook There is no perfect hook mechanism

Kernel-space OS subsystem

#### Problem •



#### Applicability of user-space OS subsystems has been limited regardless of their benefits

#### **Existing Mechanisms**

# High performance atch (SUD) • Systemaling Let Penalty

- int3 signaling icc
- Sometimes fail to hook

ms allow us to transparently apply to existing applications

ok mechanism, ...

user-space program is necessary

am

User-defined hook function

User-space OS subsystem

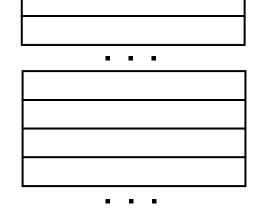
system call hook — There is no perfect hook mechanism

Kernel-space OS subsystem

#### Contribution

- zpoline employs binary rewriting and offers 6 advantages
  - 1. Low hook overhead
  - 2. Exhaustive hooking
  - 3. No breakage of user-space program logic
  - 4. No kernel change and no additional kernel module are necessary
  - 5. No source code of a user-space program is needed
  - 6. It can be used for emulating system calls
- None of existing mechanisms achieve them simultaneously

- 0x0000 0x0001 0x0002
- Virtual Memory
- On x86-64 CPUs, syscall and sysenter instructions trigger a system call
  - syscall: 0x0f 0x05, sysenter: 0x0f 0x34



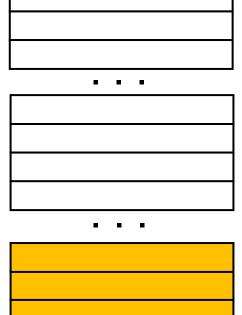
syscall 0x0f 0x05

Virtual Memory 0x0000 0x0001 0x0002

• On x86-64 CPUs, syscall and sysenter instructions trigger a system call

• syscall: 0x0f 0x05, sysenter: 0x0f 0x34

- 0x0000 0x0001 0x0002
- Virtual Memory
- On x86-64 CPUs, syscall and sysenter instructions trigger a system call
  - syscall: 0x0f 0x05, sysenter: 0x0f 0x34
- What we wish to achieve



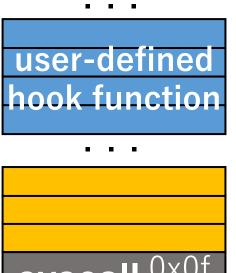
0x0000 0x0001 0x0002

Virtual Memory

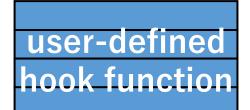
• On x86-64 CPUs, syscall and sysenter instructions trigger a system call

• syscall: 0x0f 0x05, sysenter: 0x0f 0x34

What we wish to achieve

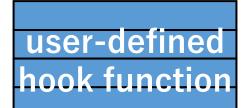


- 0x0000 0x0001 0x0002
- Virtual Memory
- On x86-64 CPUs, syscall and sysenter instructions trigger a system call
  - syscall: 0x0f 0x05, sysenter: 0x0f 0x34
- What we wish to achieve
  - replace syscall/sysenter instruction with something.





- 0x0000 0x0001 0x0002
- Virtual Memory
- On x86-64 CPUs, syscall and sysenter instructions trigger a system call
  - syscall: 0x0f 0x05, sysenter: 0x0f 0x34
- What we wish to achieve
  - replace syscall/sysenter instruction with something



????

- 0x0000 0x0001 0x0002
- On x86-64 CPUs, syscall and sysenter instructions trigger a system call
  - syscall: 0x0f 0x05, sysenter: 0x0f 0x34

jump user-de

- What we wish to achieve
  - replace syscall/sysenter instruction with something
  - to jump to a user-defined hook function

????

- 0x0000 0x0001 0x0002
- On x86-64 CPUs, syscall and sysenter instructions trigger a system call
  - syscall: 0x0f 0x05, sysenter: 0x0f 0x34

jump

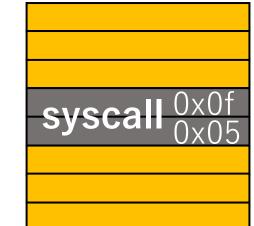
- What we wish to achieve
  - replace syscall/sysenter instruction with something
  - to jump to a user-defined hook function
- Question: what should we put here?

????

- 0x0000 0x0001 0x0002
- Virtual Memory

- On x86-64 CPUs, syscall and sysenter instructions trigger a system call
  - syscall: 0x0f 0x05, sysenter: 0x0f 0x34





- 0x0000 0x0001 0x0002
- Virtual Memory
- On x86-64 CPUs, syscall and sysenter instructions trigger a system call
  - syscall: <u>0x0f 0x05</u>, sysenter: <u>0x0f 0x34</u>

syscall and sysenter are 2-byte instructions

user-defined hook function

syscall 0x0f 0x05

0x0000 0x0001 0x0002

Virtual Memory

• On x86-64 CPUs, syscall and sysenter instructions trigger a system call

• syscall: <u>0x0f 0x05</u>, sysenter: <u>0x0f 0x34</u>

user-defined hook function

• syscall and sysenter are 2-byte instructions

- - -

 Specification for a jump destination address needs more than 2 bytes

syscall 0x0f 0x05

Virtual Memory 0x0000 0x0001 0x0002

• On x86-64 CPUs, syscall and sysenter instructions trigger a system call

• syscall: 0x0f 0x05, sysenter: 0x0f 0x34

**ADDR** user-defined hook function

• syscall and sysenter are **2-byte** instructions

 Specification for a jump destination address needs more than 2 bytes

Virtual Memory 0x0000 0x0001 0x0002

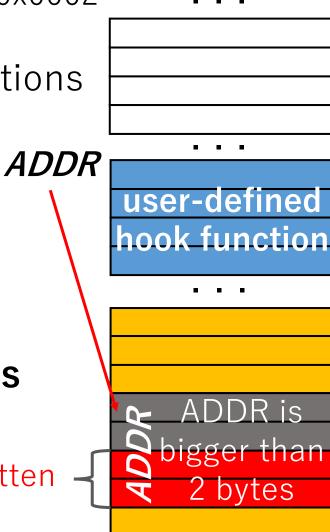
• On x86-64 CPUs, syscall and sysenter instructions trigger a system call

• syscall: 0x0f 0x05, sysenter: 0x0f 0x34

syscall and sysenter are 2-byte instructions

 Specification for a jump destination address needs more than 2 bytes

If we put ADDR, subsequent instructions are overwritten -



0x0000 0x0001 0x0002

• On x86-64 CPUs, syscall and sysenter instructions trigger a system call

<del>some</del> program

• syscall: 0x0f 0x05, sysenter: 0x0f 0x34

ADDR user-defined hook function

syscall and sysenter are 2-byte instructions

 Specification for a jump destination address needs more than 2 bytes

If we put ADDR, subsequent instructions are overwritten -

ADDR is bigger than 2 bytes

• On x86-64 CPUs, syscall and sysenter instructions trigger a system call

• syscall: 0x0f 0x05, sysenter: 0x0f 0x34

syscall and sysenter are 2-byte instructions

 Specification for a jump destination address needs more than 2 bytes

If we put ADDR, subsequent instructions are overwritten

jump to the overwritten part leads to unexpected behaviors

Virtual Memory 0x0000 0x0001 0x0002 <del>some</del> program user-defined hook function bytes

jump



0x0000 0x0001 0x0002

jump

 On x86-64 CPUs, syscall and sysenter instructions trigger a system call

• syscall: <u>0x0f 0x05</u>, sysenter: <u>0x0f 0x34</u>

• syscall and sysenter are **2-byte** instructions

 Specification for a jump destination address needs more than 2 bytes

If we put ADDR, subsequent instructions are overwritten

jump to the overwritten part leads to unexpected behaviors

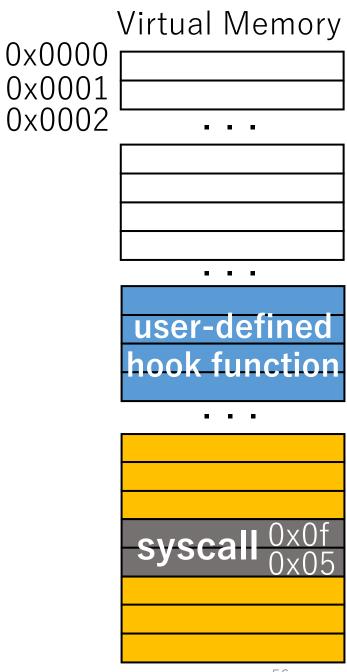
user-defined hook function

<del>some</del>

program

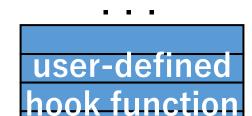
2 bytes

How to invoke a system call



- 0x0000 0x0001 0x0002
- Virtual Memory

- How to invoke a system call
  - A user-space program sets a system call number, predefined by the kenel, to the **rax register** 
    - e.g., 0: read(), 1: write(), 2: open(), ...

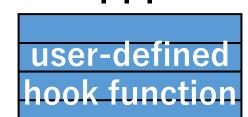


set syscall num to rax register

syscall 0x0f

- 0x0000 [ 0x0001 [ 0x0002
- Virtual Memory

- How to invoke a system call
  - A user-space program sets a system call number, predefined by the kenel, to the rax register
    - e.g., 0: read(), 1: write(), 2: open(), ...
  - The user-space program executes syscall/sysenter

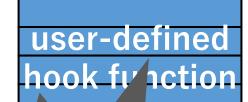


set syscall num to rax register

syscall 0x0f 0x05

- 0x0000 0x0001 0x0002
- Virtual Memory

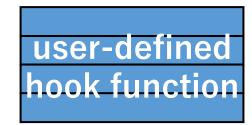
- How to invoke a system call
  - A user-space program sets a system call number, predefined by the kenel, to the rax register
    - e.g., 0: read(), 1: write(), 2: open(), ...
  - The user-space program executes syscall/sysenter
    - ---- the context is switched to the kernel ----



context switch to kernel

- 0x0000 0x0001 0x0002
- Virtual Memory
  - . . .

- How to invoke a system call
  - A user-space program sets a system call number, predefined by the kenel, to the rax register
    - e.g., 0: read(), 1: write(), 2: open(), ...
  - The user-space program executes syscall/sysenter
     ---- the context is switched to the kernel ----
  - Kernel executes a system call specified through the system call number set to the rax register



set syscall num
to rax register

syscall 0x0f
0x05

- 0x00000x0001 0x0002
- Virtual Memory

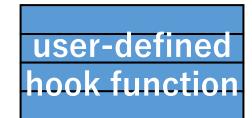
- How to invoke a system call
  - A user-space program sets a system call number, predefined by the kenel, to the rax register
    - e.g., 0: read(), 1: write(), 2: open(), ...
  - The user-space program executes syscall/sysenter ---- the context is switched to the kernel ----
  - Kernel executes a system call specified through the system call number set to the rax register
    - if the rax register has 0, kernel executes read()
    - if the rax register has 1, kernel executes write()
    - if the rax register has 2, kernel executes open()

- 0x0000 0x0001 0x0002
- Virtual Memory

- How to invoke a system call
  - A user-space program sets a system call number, predefined by the kenel, to the rax register
    - e.g., 0: read(), 1: write(), 2: open(), ...
  - The user-space program executes syscall/sysenter
     ---- the context is switched to the kernel ----
  - Kernel executes a system call specified through the system call number set to the rax register

**Point: Calling Convention** 

When syscall/sysenter is executed, the rax register always has a system call number,



set syscall num to rax register

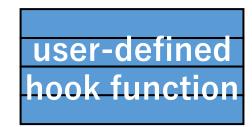
syscall  $\frac{0\times0f}{0\times05}$ 

- 0x0000 [ 0x0001 [ 0x0002
- Virtual Memory

- How to invoke a system call
  - A user-space program sets a system call number, predefined by the kenel, to the rax register
    - e.g., 0: read(), 1: write(), 2: open(), ...
  - The user-space program executes syscall/sysenter
     ---- the context is switched to the kernel ----
  - Kernel executes a system call specified through the system call number set to the rax register

#### **Point: Calling Convention**

When syscall/sysenter is executed, the rax register always has a system call number, which is 0 ~ around 500 (defined in the kernel)



set syscall num to rax register



\_ 63

0x0000 0x0001 0x0002

Virtual Memory

zpoline replaces syscall/sysenter with callq \*%rax

user-defined hook function

. . .

set syscall num to rax register

syscall  $\frac{0\times0}{0\times05}$ 

#### **Point: Calling Convention**

When syscall/sysenter is executed, the rax register always has a system call number, which is 0 ~ around 500 (defined in the kernel)

https://github.com/yasukata/presentation/tree/gh-pages/2023/06/snia-j

- 0x0000 0x0001 0x0002
- zpoline replaces syscall/sysenter with callq \*%rax
  - callq \*%rax is a 2-byte instruction (0xff 0xd0)

#### **Point: Calling Convention**

When syscall/sysenter is executed, the rax register always has a system call number, which is 0 ~ around 500 (defined in the kernel)

https://github.com/yasukata/presentation/tree/gh-pages/2023/06/snia-j

Virtual Memory user-defined hook function set syscall num to rax register

- 0x0000 0x0001 0x0002
- Virtual Memory 000 001
- zpoline replaces syscall/sysenter with callq \*%rax
  - callq \*%rax is a 2-byte instruction (0xff 0xd0)
    - Neighbour instructions are not overwritten

user-defined hook function

set syscall num to rax register

callq \*%rax

#### **Point: Calling Convention**

When syscall/sysenter is executed, the rax register always has a system call number, which is 0 ~ around 500 (defined in the kernel)

\_ 66

- 0x0000 0x0001 0x0002
- Virtual Memory
- zpoline replaces syscall/sysenter with callq \*%rax
  - callq \*%rax is a 2-byte instruction (0xff 0xd0)
    - Neighbour instructions are not overwritten
  - callq \*%rax is an instruction to jump to the address stored in the rax register

user-defined hook function

set syscall num to rax register

callq \*%rax

#### **Point: Calling Convention**

When syscall/sysenter is executed, the rax register always has a system call number, which is 0 ~ around 500 (defined in the kernel)

- 0x0000 0x0001 0x0002
- Virtual Memory

- zpoline replaces syscall/sysenter with callq \*%rax
  - callq \*%rax is a 2-byte instruction (0xff 0xd0)
    - Neighbour instructions are not overwritten
  - callq \*%rax is an instruction to jump to the address stored in the rax register

user-defined hook function

set syscall num to rax register

callq \*%rax

#### **Point: Calling Convention**

When syscall/sysenter is executed, the rax register always has a system call number, which is 0 ~ around 500 (defined in the kernel)

- 0x0000 0x0001 0x0002
- Virtual Memory
- zpoline replaces syscall/sysenter with callq \*%rax
  - callq \*%rax is a 2-byte instruction (0xff 0xd0)
    - Neighbour instructions are not overwritten
  - callq \*%rax is an instruction to jump to the address stored in the rax register

user-defined hook function

set syscall num to rax register

callq \*%rax

#### After the binary rewriting

**Point: Calling Convention** 

When syscall/sysenter is executed, the rax register always has a system call number, which is 0 ~ around 500 (defined in the kernel)

- 0x0000 0x0001 0x0002
- Virtual Memory
- zpoline replaces syscall/sysenter with callq \*%rax
  - callq \*%rax is a 2-byte instruction (0xff 0xd0)
    - Neighbour instructions are not overwritten
  - callq \*%rax is an instruction to jump to the address stored in the rax register

user-defined hook function

After the binary rewriting

**Point: Calling Convention** 

When syscall/sysenter callq \*%rax is executed, the rax register always has a system call number, which is 0 ~ around 500 (defined in the kernel)

set syscall num
to rax register

callq \*%rax

0x0000 0x0001 0x0002

Virtual Memory

- zpoline replaces syscall/sysenter with callq \*%rax
  - callq \*%rax is a 2-byte instruction (0xff 0xd0)
    - Neighbour instructions are not overwritten
  - callq \*%rax is an instruction to jump to the address stored in the rax register

user-defined hook function

After the binary rewriting

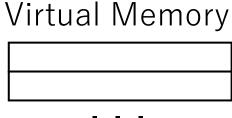
**Point: Calling Convention** 

When syscall/sysenter callq \*%rax is executed, the rax register always has a system call number, which is 0 ~ around 500 (defined in the kernel)

to rax register

callq \*%rax

0x0000 0x0001 0x0002



- zpoline replaces syscall/sysenter with callq \*%rax
  - callq \*%rax is a 2-byte instruction (0xff 0xd0)
    - Neighbour instructions are not overwritten
  - callq \*%rax is an instruction to jump to the address stored in the rax register

user-defined hook function

After the binary rewriting

**Point: Calling Convention** 

When syscall/sysenter callq \*%rax is executed, the rax register always has a system call number, which is 0 ~ around 500 (defined in the kernel)

to rax register

callq \*%rax

72

- 0x0000 0x0001 0x0002
- Virtual Memory
- zpoline replaces syscall/sysenter with callq \*%rax
  - callq \*%rax is a 2-byte instruction (0xff 0xd0)
    - Neighbour instructions are not overwritten
  - callq \*%rax is an instruction to jump to the address stored in the rax register

user-defined hook function

After the binary rewriting

**Point: Calling Convention** 

When syscall/sysenter callq \*%rax is executed, the rax register always has a system call number, which is 0 ~ around 500 (defined in the kernel)

to rax register

callq \*%rax

0x0000 0x0001 0x0002 around 500

- zpoline replaces syscall/sysenter with callq \*%rak
  - callq \*%rax is a 2-byte instruction (0xff 0xd0)
    - Neighbour instructions are not overwritten
  - callq \*%rax is an instruction to jump to the address stored in the rax register
  - replaced callq \*%rax jumps to address 0~around 500

#### After the binary rewriting

**Point: Calling Convention** 

When syscall/sysenter callq \*%rax is executed, the rax register always has a system call number, which is 0 ~ around 500 (defined in the kernel)

Virtual Memory user-defined hook function set syscall num to rax register

address range, potentially replaced "callq \*%rax" jumps to ( *N* is the max syscall number )

- 0x0000 0x0001 0x0002
- zpoline replaces syscall/sysenter with callq \*%ra
  - callq \*%rax is a 2-byte instruction (0xff 0xd0)
    - Neighbour instructions are not overwritten
  - callq \*%rax is an instruction to jump to the address stored in the rax register
  - replaced callq \*%rax jumps to address 0~around 500

#### After the binary rewriting

**Point: Calling Convention** 

When syscall/sysenter callq \*%rax is executed, the rax register always has a system call number, which is 0 ~ around 500 (defined in the kernel)

Virtual Memory user-defined hook function set syscall num to rax register

address range, potentially replaced "callq \*%rax" jumps to ( *N* is the max syscall number )

- Virtual Memory 0x0000 0x0001 0x0002
- zpoline replaces syscall/sysenter with callq
  - callq \*%rax is a 2-byte instruction (0xff 0xd0)
    - Neighbour instructions are not overwritten
  - callq \*%rax is an instruction to jump to the address stored in the rax register
  - replaced callq \*%rax jumps to address 0~around 500

user-defined hook function

address range, potentially replaced "callq \*%rax" jumps to ( *N* is the max syscall number )

- Virtual Memory 0x0000 0x0001 0x0002
- zpoline replaces syscall/sysenter with callq
  - callq \*%rax is a 2-byte instruction (0xff 0xd0)
    - Neighbour instructions are not overwritten
  - callq \*%rax is an instruction to jump to the address stored in the rax register
  - replaced callq \*%rax jumps to address 0~around 500

How to redirect to the user-defined hook function?

user-defined hook function

address range, potentially replaced "callq \*%rax" jumps to ( *N* is the max syscall number )

- Virtual Memory 0x0000 0x0001 0x0002
- zpoline replaces syscall/sysenter with callq
  - callq \*%rax is a 2-byte instruction (0xff 0xd0)
    - Neighbour instructions are not overwritten
  - callq \*%rax is an instruction to jump to the address stored in the rax register
  - replaced callq \*%rax jumps to address 0~around 500

How to redirect to the user-defined hook function?

user-defined hook function

address range, potentially replaced "callq \*%rax" jumps to ( *N* is the max syscall number )

- Virtual Memory 0x0000 0x0001 0x0002

- zpoline replaces syscall/sysenter with callq
  - callq \*%rax is a 2-byte instruction (0xff 0xd0)
    - Neighbour instructions are not overwritten
  - callq \*%rax is an instruction to jump to the address stored in the rax register
  - replaced callq \*%rax jumps to address 0~around 500

#### How to redirect to the user-defined hook function?

zpoline instantiates trampoline code at address 0

user-defined hook function

address range, potentially replaced "callq \*%rax" jumps to ( *N* is the max syscall number )

- Virtual Memory 0x0000 0x0001 0x0002

- zpoline replaces syscall/sysenter with callq
  - callq \*%rax is a 2-byte instruction (0xff 0xd0)
    - Neighbour instructions are not overwritten
  - callq \*%rax is an instruction to jump to the address stored in the rax register
  - replaced callq \*%rax jumps to address 0~around 500

#### How to redirect to the user-defined hook function?

zpoline instantiates trampoline code at address 0

Trampoline code at address 0 (zero) → zpoline

user-defined hook function

address range, potentially replaced "callq \*%rax" jumps to ( *N* is the max syscall number )

- Virtual Memory 0x0000 0x0001 0x0002

- zpoline replaces syscall/sysenter with callq
  - callq \*%rax is a 2-byte instruction (0xff 0xd0)
    - Neighbour instructions are not overwritten
  - callq \*%rax is an instruction to jump to the address stored in the rax register
  - replaced callq \*%rax jumps to address 0~around 500

#### How to redirect to the user-defined hook function?

zpoline instantiates trampoline code at address 0

user-defined hook function

address range, potentially replaced "callq \*%rax" jumps to ( *N* is the max syscall number )

- Virtual Memory 0x0000 0x0001 0x0002
  - nop nop

nop

- zpoline replaces syscall/sysenter with callq
  - callq \*%rax is a 2-byte instruction (0xff 0xd0)
    - Neighbour instructions are not overwritten
  - callq \*%rax is an instruction to jump to the address stored in the rax register
  - replaced callq \*%rax jumps to address 0~around 500

#### How to redirect to the user-defined hook function?

- zpoline instantiates trampoline code at address 0
  - fills address range 0 to N with nop (0x90)

user-defined hook function

address range, potentially replaced "callq \*%rax" jumps to ( *N* is the max syscall number )

- Virtual Memory 0x0000 0x0001 0x0002
  - nop nop

nop

- zpoline replaces syscall/sysenter with callq
  - callq \*%rax is a 2-byte instruction (0xff 0xd0)
    - Neighbour instructions are not overwritten
  - callq \*%rax is an instruction to jump to the address stored in the rax register
  - replaced callq \*%rax jumps to address 0~around 500

#### How to redirect to the user-defined hook function?

- zpoline instantiates trampoline code at address 0
  - fills address range 0 to N with nop (0x90)

user-defined hook function

address range, potentially replaced "callq \*%rax" jumps to ( *N* is the max syscall number )

Virtual Memory 0x0000 0x0001 0x0002

nop nop

> nop jump to

hook function

user-defined hook function

set syscall num to rax register

- \*%ra zpoline replaces syscall/sysenter with callq
  - callq \*%rax is a 2-byte instruction (0xff 0xd0)
    - Neighbour instructions are not overwritten
  - callq \*%rax is an instruction to jump to the address stored in the rax register
  - replaced callq \*%rax jumps to address 0~around 500

#### How to redirect to the user-defined hook function?

- zpoline instantiates trampoline code at address 0
  - fills address range 0 to N with nop (0x90)
    - puts code to jump to the hook function next to the last nop

address range, potentially replaced "callq \*%rax" jumps to ( N is the max syscall number )

Virtual Memory 0x0000 0x0001 0x0002

nop nop

nop jump to

hook function

zpoline replaces syscall/sysenter with callq

callq \*%rax is a 2-byte instruction (0xff 0xd0)

• Neighbour instructions are not overwritten

 callq \*%rax is an instruction to jump to the address stored in the rax register

replaced callq \*%rax jumps to address 0~around 500

#### How to redirect to the user-defined hook function?

- zpoline instantiates trampoline code at address 0
  - fills address range 0 to N with nop (0x90)
    - puts code to jump to the hook function next to the last hop

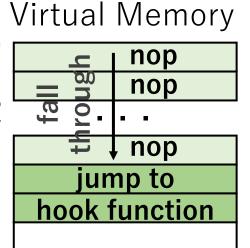
We could reach the user-defined hook function! b.com/yasukata/presentation/tree/gh-pages/2023/06/snia-

user-defined

0x0000

0x0001 0x0002

A buggy program may access NULL (address 0)



user-defined hook function

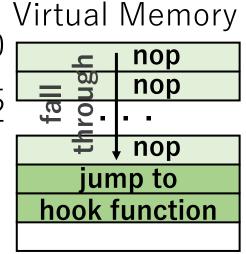
. . .

set syscall num to rax register

callq \*%rax

0x0000 0x0001 0x0002 N

A buggy program may access NULL (address 0)



user-defined hook function

. . .

set syscall num to rax register

callq \*%rax

Bug Access NULL

8/

Virtual Memory

 $0 \times 0000$ 0x0001 0x0**(**02

nop nop

nop jump to

hook function

. . .

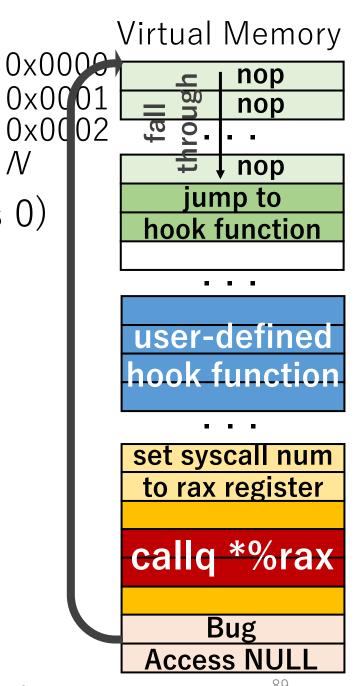
user-defined hook function

set syscall num to rax register

callq \*%rax Bug **Access NULL** 

A buggy program may access NULL (address 0)

- A buggy program may access NULL (address 0)
  - In principle, NULL access has to be terminated



#### Normally, ...

## NULL Access Termination

Virtual Memory

0x00001 0x0001 0x0002 no physical

> memory mapping!

user-defined hook function

set syscall num to rax register

callq \*%rax

Bug Access NULL

- A buggy program may access NULL (address 0)
  - In principle, NULL access has to be terminated
  - Normally, a page fault happens because no physical memory is mapped to virtual address 0

Virtual Memory

0x0009 0x0001  $0 \times 0002$ 

nop nop

nop jump to hook function

user-defined hook function

set syscall num to rax register

callq \*%rax

Bug **Access NULL** 

- A buggy program may access NULL (address 0)
  - In principle, NULL access has to be terminated
  - Normally, a page fault happens because no physical memory is mapped to virtual address 0
  - zpoline uses virtual address 0, therefore, the page fault does not happen

Virtual Memory

0x0000 0x0001 0x0002

= nop

≠ ↓ nop jump to hook function

user-defined hook function

set syscall num to rax register

callq \*%rax

Bug Access NULL

- A buggy program may access NULL (address 0)
  - In principle, NULL access has to be terminated
  - Normally, a page fault happens because no physical memory is mapped to virtual address 0
  - zpoline uses virtual address 0, therefore, the page fault does not happen



The buggy program continues to run

Virtual Memory

nop

0x00001 0x0001 0x0002

= anop to nop jump to

hook function

user-defined hook function

set syscall num to rax register

callq \*%rax

Bug Access NULL

A buggy program may access NULL (address 0)

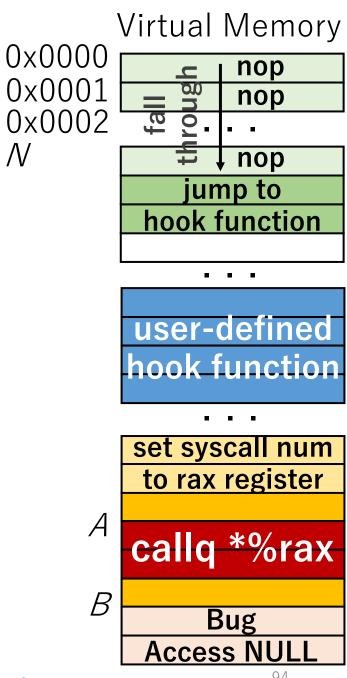
- In principle, NULL access has to be terminated
- Normally, a page fault happens because no physical memory is mapped to virtual address 0
- zpoline uses virtual address 0, therefore, the page fault does not happen



The buggy program continues to run

How can we detect and terminate a buggy NULL access?

Memory access: read / write / execute



- Memory access: read / write / execute
- Solution
  - read/write: configure the trampoline code as XOM

Virtual Memory 0x0000 **eXecute** 0x0001 0x0002 Only Memory (XOM) user-defined hook function set syscall num

set syscall num
to rax register

A callq \*%rax

B Bug
Access NULL

- Memory access: read / write / execute
- Solution
  - read/write: configure the trampoline code as XOM
    - read/write access to the trampoline code causes a fault
      - This can be done by mprotect() system call

> user-defined hook function

set syscall num
to rax register

A callq \*%rax

B Bug
Access NULL

- 0x00000x0001 0x0002
- Virtual Memory

eXecute Only Memory (XOM)

user-defined hook function

set syscall num to rax register callq \*%rax BBug

**Access NULL** 

- Memory access: read / write / execute
- Solution
  - read/write: configure the trampoline code as XOM
    - read/write access to the trampoline code causes a fault
      - This can be done by mprotect() system call
  - execute: check the caller address

- 0x0000 0x0001 0x0002 N
- Virtual Memory

eXecute
Only
Memory
(XOM)

user-defined hook function

. . .

set syscall num
to rax register

Callq \*%rax

Bug
Access NULL

- Memory access: read / write / execute
- Solution
  - read/write: configure the trampoline code as XOM
    - read/write access to the trampoline code causes a fault
      - This can be done by mprotect() system call
  - execute: check the caller address
    - 1. collect the addresses of replaced syscall/sysenter during the binary rewriting phase

- 0x0000 0x0001 0x0002 N
- Virtual Memory

eXecute Only Memory (XOM)

user-defined hook function

set syscall num
to rax register

A callq \*%rax
B

Bug

**Access NULL** 

- Memory access: read / write / execute
- Solution
  - read/write: configure the trampoline code as XOM
    - read/write access to the trampoline code causes a fault
      - This can be done by mprotect() system call
  - execute: check the caller address
    - 1. collect the addresses of replaced syscall/sysenter during the binary rewriting phase

#### During binary rewriting phase ...

List of replaced addresses : [...]

- 0x0000 0x0001 0x0002 N
- Virtual Memory
  - eXecute Only Memory (XOM)
  - user-defined hook function
  - set syscall num to rax register
  - callq \*%rax
  - Bug Access NULL

- Memory access: read / write / execute
- Solution
  - read/write: configure the trampoline code as XOM
    - read/write access to the trampoline code causes a fault
      - This can be done by mprotect() system call
  - execute: check the caller address
    - 1. collect the addresses of replaced syscall/sysenter during the binary rewriting phase

#### During binary rewriting phase ...

List of replaced addresses: [...]

- 0x0000 0x0001 0x0002
- Virtual Memory

**eXecute** Only Memory (XOM)

user-defined hook function

set syscall num to rax register

Bug Access NULL

- Memory access: read / write / execute
- Solution
  - read/write: configure the trampoline code as XOM
    - read/write access to the trampoline code causes a fault
      - This can be done by mprotect() system call
  - execute: check the caller address
    - 1. collect the addresses of replaced syscall/sysenter during the binary rewriting phase

#### During binary rewriting phase ...

List of replaced addresses : [A, ...]

0x0000 0x0001 0x0002 N

- Memory access: read / write / execute
- Solution
  - read/write: configure the trampoline code as XOM
    - read/write access to the trampoline code causes a fault
      - This can be done by mprotect() system call
  - execute: check the caller address
    - 1. collect the addresses of replaced syscall/sysenter during the binary rewriting phase
    - 2. check the caller address is one of the replaced addresses in the hook function

to rax register

Callq \*%rax

Bug

**Access NULL** 

List of replaced addresses : [A, ...]

https://github.com/yasukata/presentation/tree/gh-pages/2023/06/snia-j

eXecute Only Memory (XOM)

user-defined

hook function

set syscall num

Virtual Memory

#### At runtime ...

- Memory access: read / write / execute
- Solution
  - read/write: configure the trampoline code as XOM
    - read/write access to the trampoline code causes a fault
      - This can be done by mprotect() system call
  - execute: check the caller address
    - 1. collect the addresses of replaced syscall/sysenter during the binary rewriting phase
    - 2. check the caller address is one of the replaced addresses in the hook function

List of replaced addresses : [A, ...]

Virtual Memory 0x000 **eXecute** 0x0001 0x0002 Only Memory (XOM) user-defined hook function set syscall num to rax register

Bug

**Access NULL** 

#### At runtime ...

- Memory acces The caller address is A
  Solution A is in the list, so
  - read/write.
    - read/write access to the transport of causes a fault

this is a valid access

- This can be done by mprotect() system call
- execute: check the caller address
  - 1. collect the addresses of replaced syscall/sysenter during the binary rewriting phase
  - 2. check the caller address is one of the replaced addresses in the hook function

List of replaced addresses : [A, ...]

Virtual Memory

user-defined hook function

set syscall num to rax register

callq \*%rax

Bug Access NULL

#### At runtime ...

- Memory access: read / write / execute
- Solution
  - read/write: configure the trampoline code as XOM
    - read/write access to the trampoline code causes a fault
      - This can be done by mprotect() system call
  - execute: check the caller address
    - 1. collect the addresses of replaced syscall/sysenter during the binary rewriting phase
    - 2. check the caller address is one of the replaced addresses in the hook function

List of replaced addresses : [A, ...]

0x0000 0x0001 0x0002 N

B

Virtual Memory

eXecute Only Memory (XOM)

user-defined hook function

to rax register

callq \*%rax

Bug

**Access NULL** 

#### At runtime ...

- Memory access: read / write / execute
- Solution
  - read/write: configure the trampoline code as XOM
    - read/write access to the trampoline code causes a fault
      - This can be done by mprotect() system call
  - execute: check the caller address
    - 1. collect the addresses of replaced syscall/sysenter during the binary rewriting phase
    - 2. check the caller address is one of the replaced addresses in the hook function

List of replaced addresses : [A, ...]

Virtual Memory  $0 \times 0000$ **eXecute** 0×0001 0x0**(**02 Only Memory (XOM) user-defined hook function set syscall num to rax register Bug Access NULL

#### At runtime ...

- Memory acces The caller address is B
- Solution
  - read/write.
- B is NOT in the list, so this is an invalid access
- read/write access to the transde causes a fault
  - This can be done by mprotect() system call
- execute: check the caller address
  - 1. collect the addresses of replaced systall/sysenter during the binary rewriting phase
  - 2. check the caller address is one of the replaced addresses in the hook function

List of replaced addresses : [A, ...]

Virtual Memory

0x0009
0x0001
0x0001
0x0002

Only

Memory
(XOM)

user-defined hook function

set syscall num to rax register

callq \*%rax

Bug Access NULL

Virtual Memory

0x0009 0x0001 \x0002

XOM

**eXecute** Only

Memory (XOM)

# NULL Access Termination

#### At runtime ...

Memory acces

The caller address is B B is NOT in the list, so

this is an invalid access

 Solution read/write.

• read/write access to the transport de causes a fault

This can be done by mprotect() system call

- execute: check the caller address
  - 1. collect the addresses of replaced sysdall/sysenter during the binary rewriting phase
  - 2. check the caller address is one of the replaced addresses in the hook function

List of replaced addresses : [A, ...]

set syscall num to rax register Bug **Access NULL** 

Virtual Memory

0x0000 0x0001 0x0002

eXecute Only

Memory (XOM)

### NULL Access Termination

#### At runtime ...

Memory acces

Solution

The caller address is *B B* is NOT in the list, so this is an invalid access

read/write.

• read/write access to the transition ode causes a fault

• This can be done by mprotect() system call

- execute: check the caller address
  - 1. collect the addresses of replaced syscall/sysenter during the binary rewriting phase
  - 2. check the caller address is one of the replaced addresses in the hook function
    - Current prototype uses bitmap to implement this check

List of replaced addresses : [A, ...]

set syscali num
to rax register

A callq \*%rax

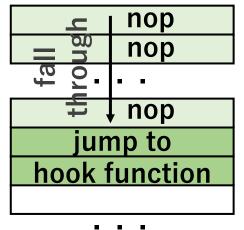
Bug
Access NULL

Time to hook getpid() and return a dummy value

Mechanism	Time [ns]
ptrace	31201
int3 signaling	1342
SUD	1156
zpoline	41

LD_PRELOAD	6
------------	---





user-defined hook function

to rax register

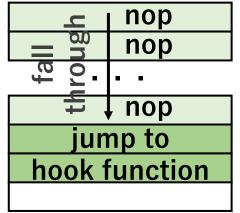
callq \*%rax

Time to hook getpid() and return a dummy value

Mechanism	Time [ns]	
ptrace	31201 7	′16x
int3 signaling	1342	32.7x
SUD	1156 2	28.1x
zpoline	41 improv	/ement

6

Virtual Memory



user-defined hook function

. . .

set syscall num to rax register

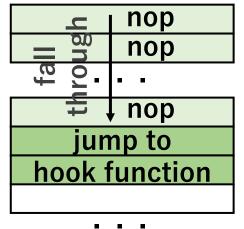
callq \*%rax

LD PRELOAD

Time to hook getpid() and return a dummy value

Mechanism	Time [ns]
ptrace	31201
int3 signaling	1342
SUD	1156
zpoline	41
zpoline (w/o NULL exec check)	40
LD_PRELOAD	6





user-defined hook function

set syscall num to rax register

callq \*%rax

additional overhead

Virtual Memory

= nop

. . .

user-defined hook function

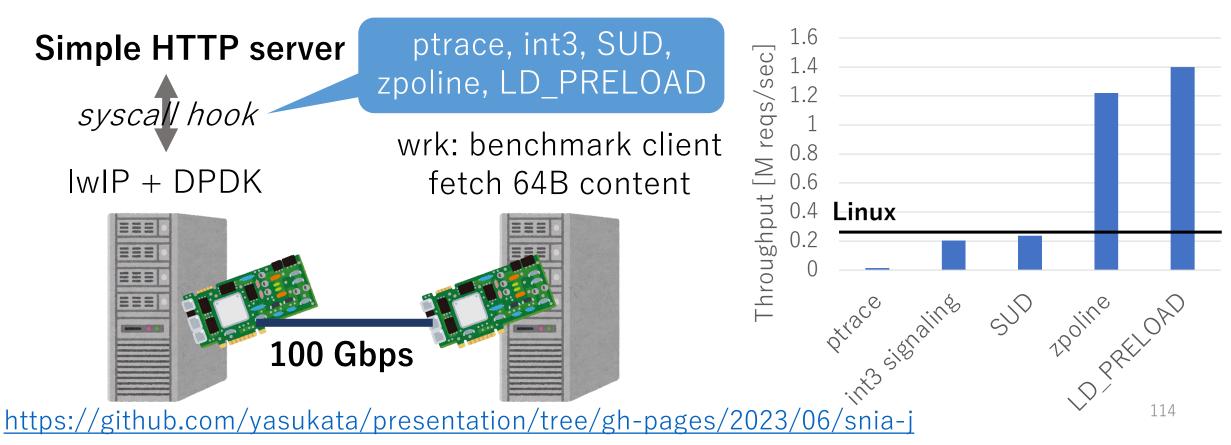
set syscall num

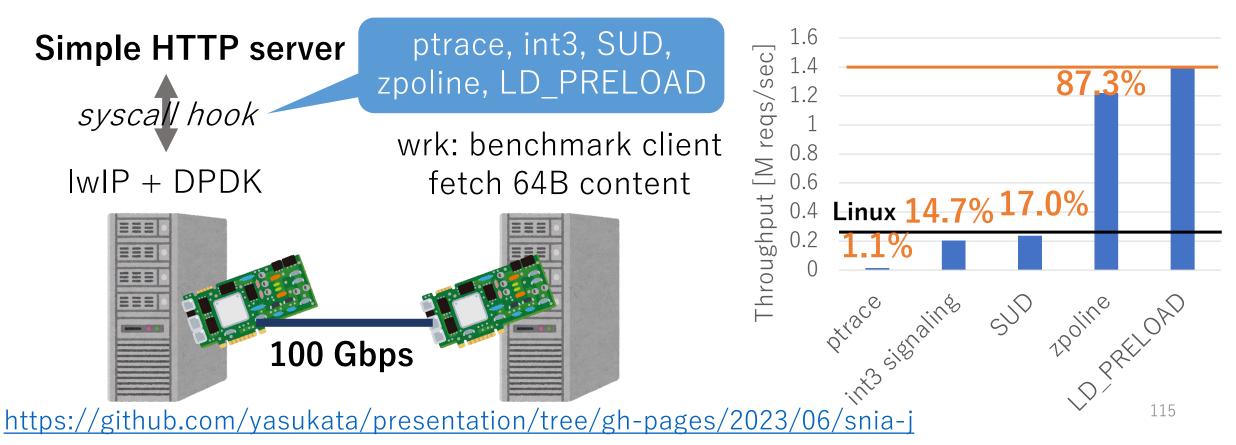
to rax register

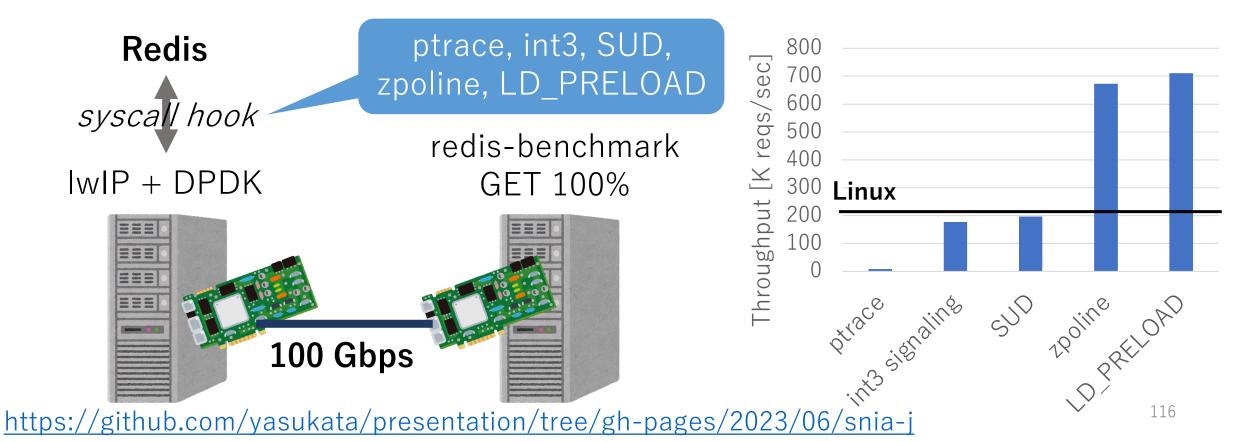
callq \*%rax

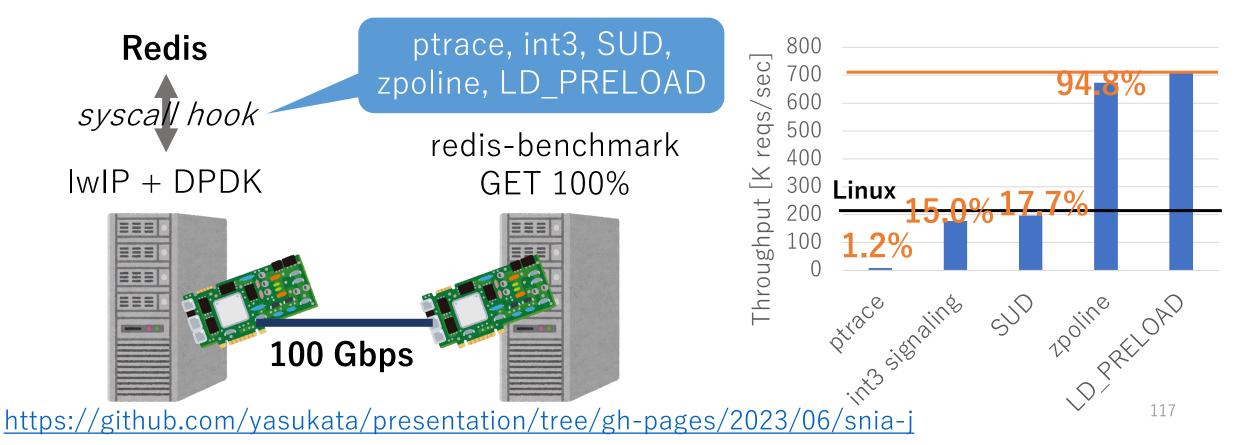
Time to hook getpid() and return a dummy value

Mechanism	Time [ns]
ptrace	31201
int3 signaling	1342
SUD	1156
zpoline	41 /
zpoline (w/o NULL exec check)	40
LD_PRELOAD	6









# Summary

- zpoline: a system call hook mechanism
  - replaces syscall/sysenter with callq \*%rax
  - instantiates the trampoline code at virtual address 0 (zero)
- 6 advantages: good for transparently applying user-space OS subsystems to existing applications
  - 1. Low hook overhead
  - 2. Exhaustive hooking
  - 3. No breakage of user-space program logic
  - 4. No kernel change and no additional kernel module are necessary
  - 5. No source code of a user-space program is needed
  - 6. It can be used for emulating system calls

### 最適化

additional overhead Virtual Memory

unop nop

hook function

- - -

user-defined hook function

set syscall num to rax register

callq \*%rax

Time to hook getpid() and return a dummy value

Mechanism	Time [ns]
ptrace	31201
int3 signaling	1342 /
SUD	1156
zpoline	41 /
zpoline (w/o NULL exec check)	40
LD_PRELOAD	6

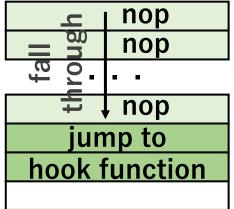
### 最適化

- nop を敷き詰めている理由
  - jump してきたアドレスを命令の先頭と解釈する

- 目標
  - 0 ~ N どこに jump してもフック関数を呼び出す コードへ辿り着けるようにしたい
  - nop のように逐一実行する必要がなければなお良い



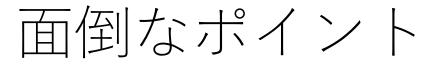
Virtual Memory



user-defined hook function

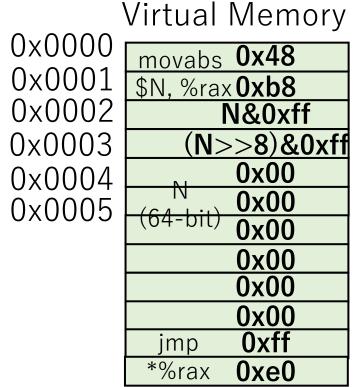
set syscall num to rax register

callq \*%rax



- 例えば、
  - rax に N を入れて
    - N < 512 として見てください
  - rax の値をアドレスとして jump

movabs \$N, %rax jmp \*%rax



N-106

 $\mathbb{N}$ 

jump to hook function

https://github.com/yasukata/presentation/tree/gh-pages/2023/06/snia-j

\_ \_ 121

Virtual Memory

ジャンプ先

### 面倒なポイント

- 例えば、
  - rax に N を入れて
    - N < 512 として見てください
  - rax の値をアドレスとして jump

movabs \$N, %rax jmp \*%rax

	r
0×0001	\$
0x0002	<u> </u>
0x0003	
0x0004	
0x0005	(
	_

novabs **0x48** SN, %rax **0xb8** N&0xff (N>>8)&0xff0x00 0x000x00 0x000x000x000xff imp \*%rax 0xe0

N-106

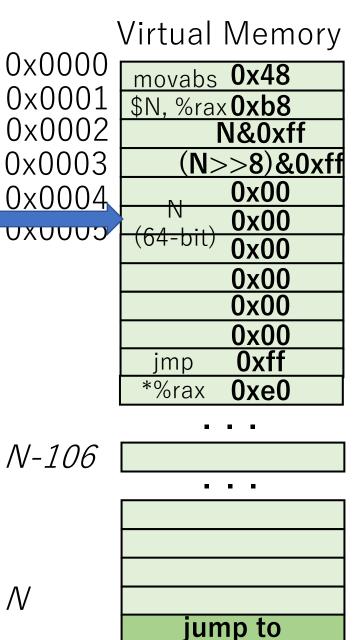
 $\mathbb{N}$ 

jump to hook function

例えば、

ジャンプ先

- rax に N を入れて
  - N < 512 として見てください
- rax の値をアドレスとして jump



https://github.com/yasukata/presentation/tree/gh-pages/2023/06/snia-j

123

hook function

例えば、

ジャンプ先

- rax に N を入れて
  - N < 512 として見てください
- rax の値をアドレスとして jump

add %al, (%rax)

Virtual Memory 0x0000 0x48 0x0001 0xb8 0x0002 N&0xff 0x0003 (N>>8)&0xff0x000x0004 0x00add CUUUXU %al, (%rax**)0x00** add 0x00%al, (%rax)**0x00** 0x000xff 0xe0

N-106

N

jump to hook function

- 例えば、
  - rax に N を入れて
    - N < 512 として見てください
  - rax の値をアドレスとして jump

add %al, (%rax)

ジャンプ先のアドレスが命令の先頭と解釈される

ジャンプ先

Virtual Memory 0x00000x48 0x0001 0xb80x0002 N&0xff 0x0003 (N>>8)&0xff0x000x0004 0x00add CUUUXU %al, (%rax**)0x00** add 0x00%al, (%rax)**0x00** 0x000xff 0xe0

N-106

 $\mathcal{N}$ 

jump to hook function

例えば、

ジャンプ先

- rax に N を入れて
  - N < 512 として見てください
- rax の値をアドレスとして jump

add %al, (%rax)

ジャンプ先のアドレスが命令の先頭と解釈される

1 byte の nop 命令で埋めておけばどこに着地しても同じ命令と解釈され、フックへ飛ぶ処理へ辿り着ける

0x0000 0x0001 0x0002 0x0003 0x0004

4	nop
II ug	nop
fa	nop
th	nop
	nop
•	

Virtual Memory

N-106

 $\mathcal{N}$ 

nop nop nop nop nop jump to

nop

\_ \_ 126

- 匿名レビュアー C
  - 使われていないシステムコール番号に short jump 命令を入れたら良いのでは?
    - 0xeb 127:+127 バイトジャンプ

Virtual Memory 0x0000nop 0x0001 nop 0x0002 nop 0x0003 nop 0x0004 nop nop 0x0005 nop **J**mp +127nop N-106 nop nop nop nop  $\mathcal{N}$ nop jump to hook function

epoll\_ctl\_old 214 epoll wait old 215

匿名レビュアー C

• 使われていないシステムコール番号に short jump 命令を入れたら良いのでは?

• 0xeb 127:+127 バイトジャンプ

• epoll\_ctl\_old / epoll\_wait\_old に short jump 命令を埋め込むとフックのコストが 31 ns まで削減できた(これまで 41 ns)

epoll\_ctl\_old 214 epoll\_wait\_old 215

Virtual Memory 0x0000nop 0x0001 nop 0x0002 nop 0x0003 nop 0x0004 nop nop 0x0005 nop **J**mp +127

N-106

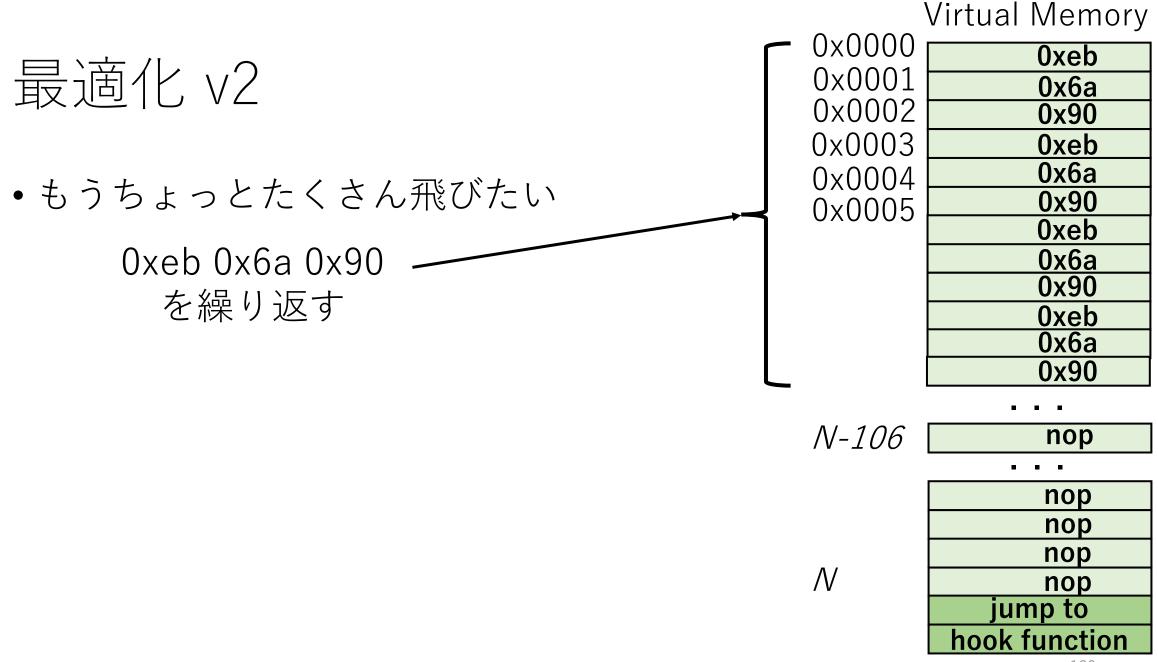
 $\mathcal{N}$ 

nop
nop
nop
nop
nop
jump to
hook function

nop

nop

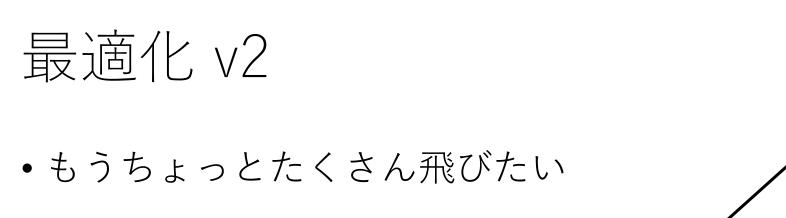
128



https://github.com/yasukata/presentation/tree/gh-pages/2023/06/snia-j

129

Virtual Memory



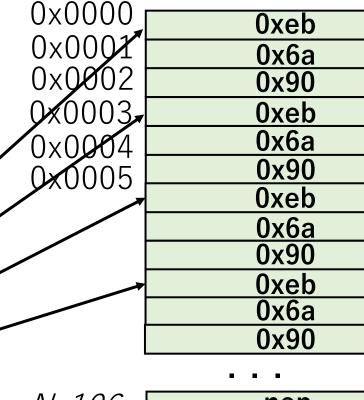
0xeb 0x6a 0x90 を繰り返す



• n \* 3 + 0 : 0xeb 0x6a 0x90 •

• n \* 3 + 1 : 0x6a 0x90 0xeb 0x90

• n \* 3 + 2 : 0x90 0xeb 0x6a



*N-106* **nop** 

nop
nop
nop
nop
jump to
hook function

\_ 130

 $\mathbb{N}$ 

• もうちょっとたくさん飛びたい

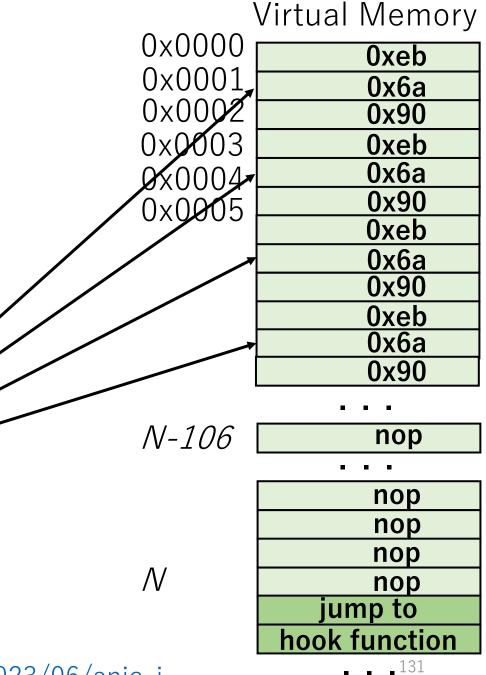
0xeb 0x6a 0x90 を繰り返す

・この場合、命令の解釈は3通り

• n \* 3 + 0 : 0xeb 0x6a 0x90

• n \* 3 + 1 : 0x6a 0x90 0xeb 0x90

• n \* 3 + 2 : 0x90 0xeb 0x6a



• もうちょっとたくさん飛びたい

0xeb 0x6a 0x90 を繰り返す

• この場合、命令の解釈は3通り

• n \* 3 + 0 : 0xeb 0x6a 0x90

• n \* 3 + 1 : 0x6a 0x90 0xeb 0

• n \* 3 + 2 : 0x90 0xeb 0x6a

Virtual Memory 0x00000xeb 0x0001 0x6a 0x0002 0x90 0xeb 0x6a 0x**2**004 0x90 X0005 0xeb 0x6a 0x90 0xeb 0x6a 0x90N-106 nop nop nop nop  $\mathcal{N}$ nop jump to hook function \_ 132

もうちょっとたくさん飛びたい

0xeb 0x6a 0x90 を繰り返す

・この場合、命令の解釈は3通

• n \* 3 + 0 : 0xeb 0x6a 0x90

• n \* 3 + 1 : 0x6a 0x90 0xeb 0x90

• n \* 3 + 2 : 0x90 0xeb 0x6a

Virtual Memory 0x0000 jmp 0x000x0 106 0x0002 nop 0xeb 0x6a 0x0004 0x90 0x0005 0xeb 0x6a 0x90 0xeb 0x6a 0x90

N-106

nop

. . .

nop nop

 $\mathcal{N}$ 

nop nop jump to

hook function

https://github.com/yasukata/presentation/tree/gh-pages/2023/06/snia-j

\_ 133

もうちょっとたくさん飛びたい

0xeb 0x6a 0x90 を繰り返す

・この場合、命令の解釈は3通り

• n \* 3 + 0 : 0xeb 0x6a 0x90

• n \* 3 + 1 : 0x6a 0x90 0xeb 0x90

• n \* 3 + 2 : 0x90 0xeb 0x6a

	Virtual Memory
0x0000	0xeb
0x0001	push
0x0002	0x90
0x <b>0</b> 003	jmp
<b>2</b> ×0004	106
0x0005	0x90
07.0000	0xeb
	0x6a
	0x90
	0xeb
1	0x6a
	0x90
N-106	nop
	nop
	nop
Λ./	nop
/V	nop
	jump to

• もうちょっとたくさん飛びたい

0xeb 0x6a 0x90 を繰り返す

• この場合、命令の解釈は3通り

• n \* 3 + 0 : 0xeb 0x6a 0x90

• n \* 3 + 1 : 0x6a 0x90 0xeb 0x90

• n \* 3 + 2 : 0x90 0xeb 0x6a

	Virtual Memory
0x0000	0xeb
0x0001	0x6a
0x0002	nop
0x0093	jmp
0x <b>0</b> 004	106
<b>2</b> ×0005	0x90
	0xeb
	<u>0x6a</u>
	0x90
	<u>0xeb</u>
,	0x6a
	0x90
N-106	nop
	nop
	nop
0.7	nop
$\mathcal{N}$	nop
	jump to
	hook function

• もうちょっとたくさん飛びたい

0xeb 0x6a 0x90 を繰り返す

• この場合、命令の解釈は3通り

• n \* 3 + 0 : 0xeb 0x6a 0x90

n \* 3 + 1 : 0x6a 0x90 0xeb 0x90

• n \* 3 + 2 : 0x90 0xeb 0x6a

この場合だけ 0x90 がスタックに積まれてしまうのでフック関数の中で捨てるようにする

0x0000 0x00001 0x00002 0x00002 0x00003 0x00004 0x00005 0x90 0x6a 0x90

N-106

 $\mathbb{N}$ 

nop

0xeb

0x6a

0x90

. . .

nop
nop
nop
nop
jump to
hook function

\_ \_ 136

もうちょっとたくさん飛びたい 0xeb 0x6a 0x90 を繰り返す

• この最適化を適用後、フックのコストが 10 ns まで減少 (v1 は 31 ns)

0	_
$0 \times 0000$	
0x0001	
0x0002	
)x0003	
0x0004	
0x0005	
7,0000	

 GIGHT TUTOTTION
0xeb
push
0x90
jmp
106
0x90
0xeb
0x6a
0x90
0xeb
0x6a
0x90

Virtual Memory

V- <i>106</i>	

nop

nop

 $\mathcal{N}$ 

nop nop nop jump to hook function