バイナリ書き換えを用いたシステムコールフック機構

第 159 回 システムソフトウェアとオペレーティング・システム研究会
2023 年 5 月 16 日

安形 憲一[1]、田崎 創[1]、オブラン ピエールルイ[1]、石黒 健太[2]

[1] IIJ 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 https://github.com/yasukata/zpoline since October 2021

USENIX ATC 2023 で発表予定の内容です

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

https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

System Call

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

https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

System Call

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

User-space program

https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

System Call

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

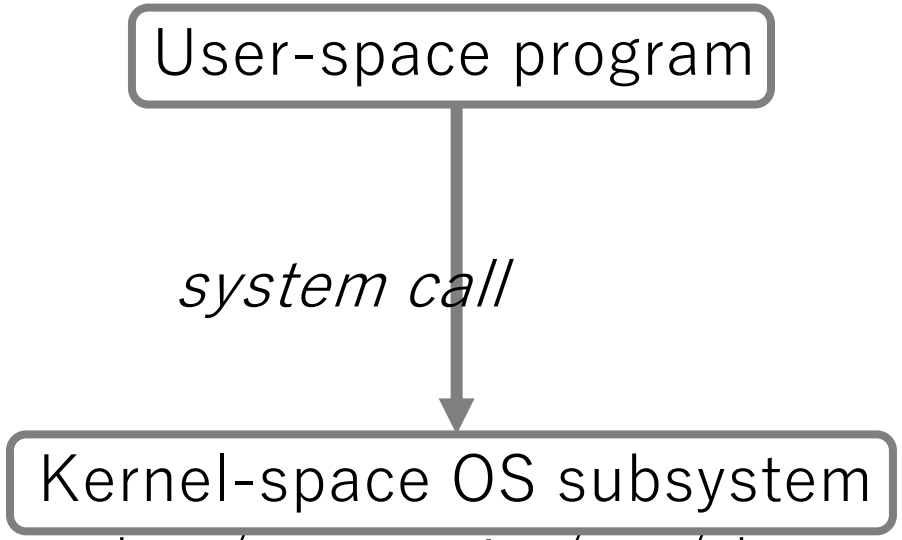
User-space program

Kernel-space OS subsystem

https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

System Call

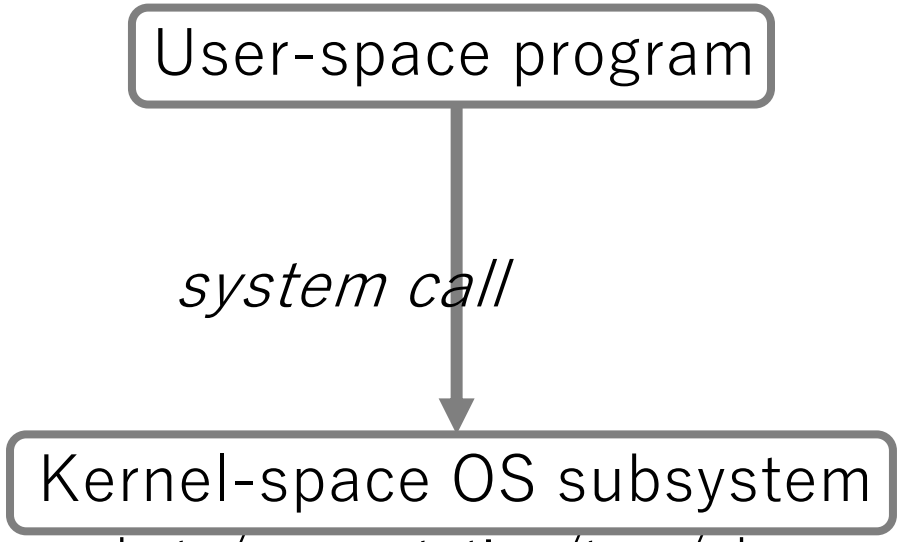
- System calls are the primary interface for user-space programs to communicate with OS kernels



https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

System Call Hook

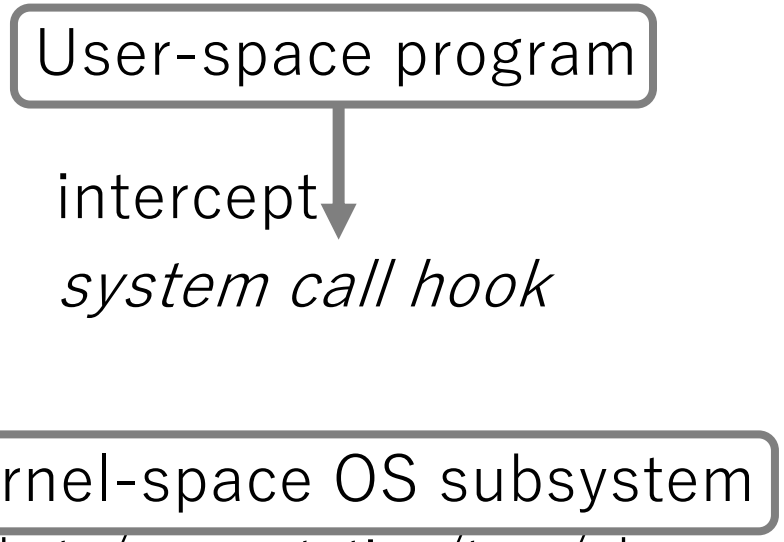
- System calls are the primary interface for user-space programs to communicate with OS kernels
- A system call hook mechanism intercepts a system call



https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

System Call Hook

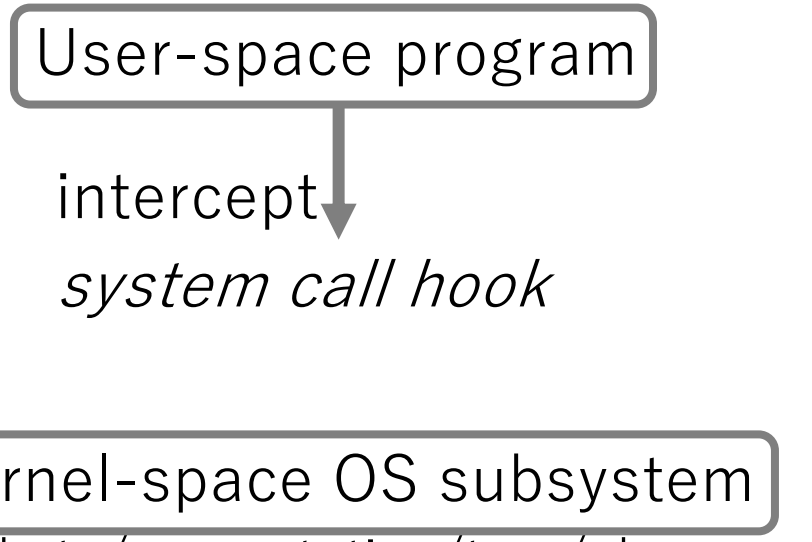
- System calls are the primary interface for user-space programs to communicate with OS kernels
- A system call hook mechanism intercepts a system call



https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

System Call Hook

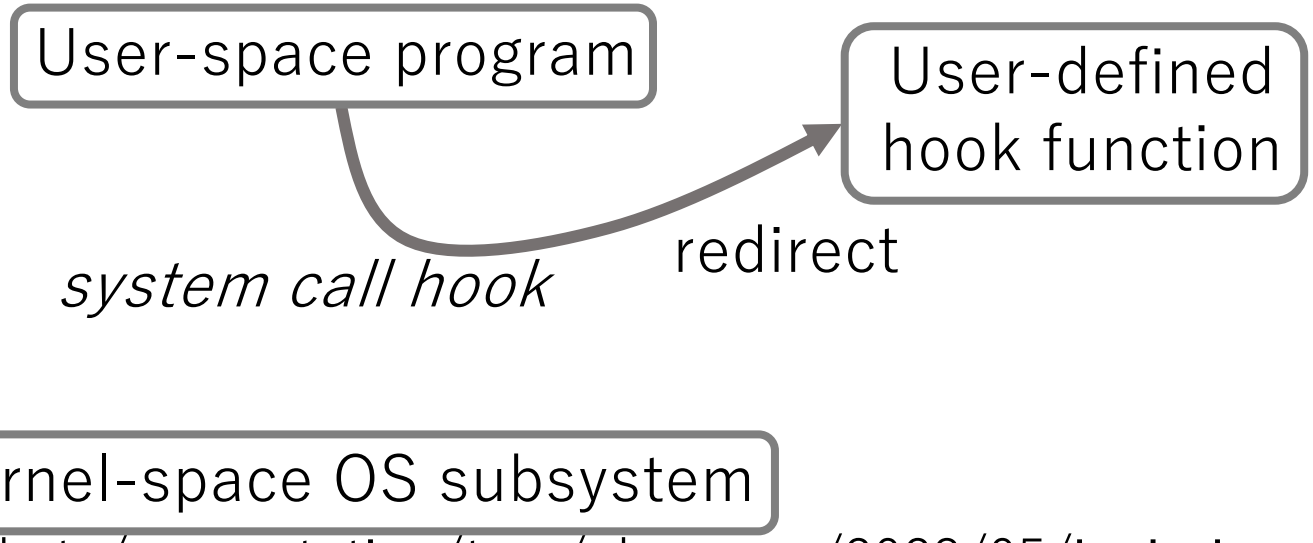
- System calls are the primary interface for user-space programs to communicate with OS kernels
- A system call hook mechanism intercepts a system call, and redirects the execution to a user-defined hook function



https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

System Call Hook

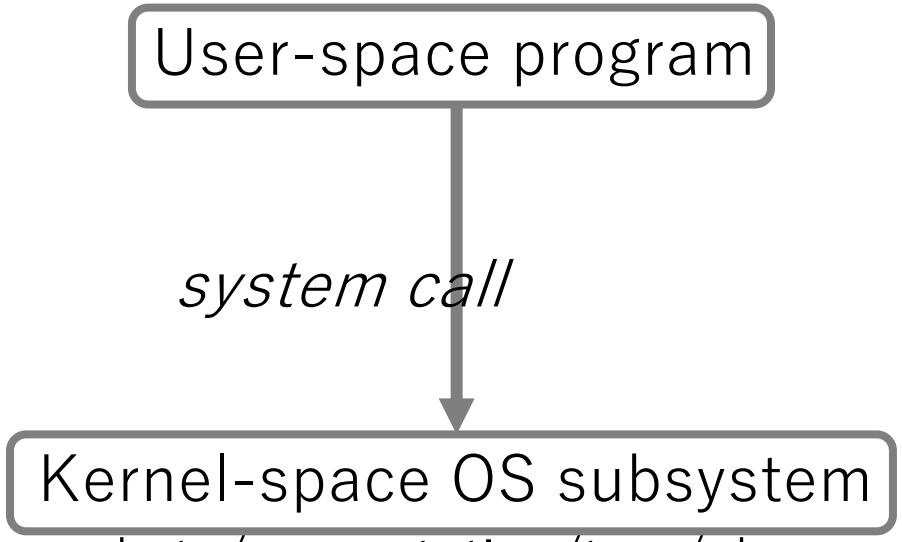
- System calls are the primary interface for user-space programs to communicate with OS kernels
- A system call hook mechanism intercepts a system call, and redirects the execution to a user-defined hook function



https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

Motivating Use Case

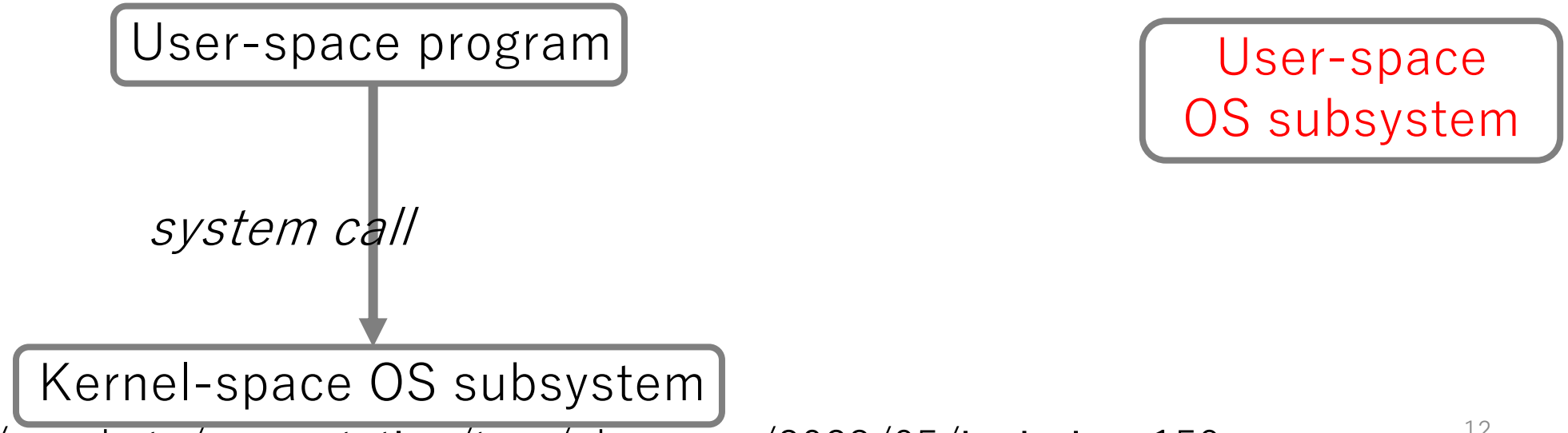
- System call hook mechanisms allow us to transparently apply user-space OS subsystems to existing applications



https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

Motivating Use Case

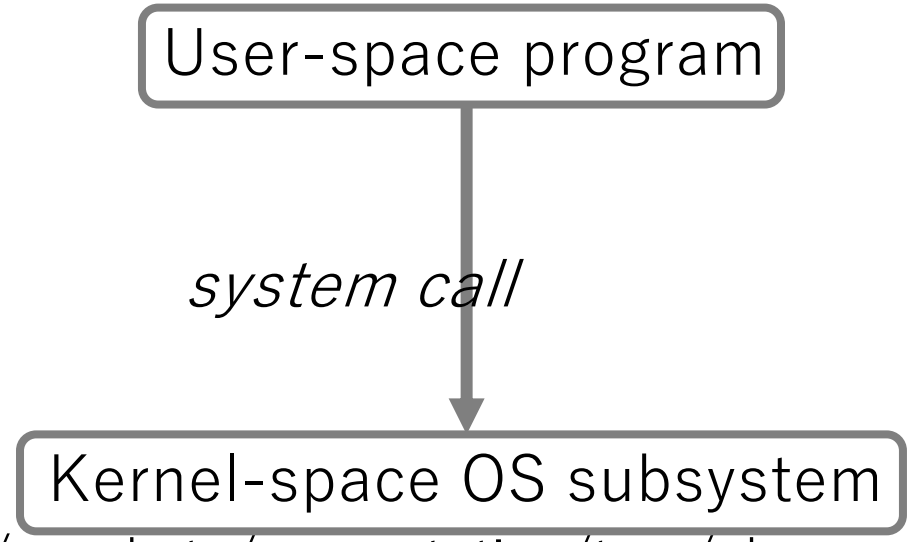
- System call hook mechanisms allow us to transparently apply user-space OS subsystems to existing applications



https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

Motivating Use Case

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

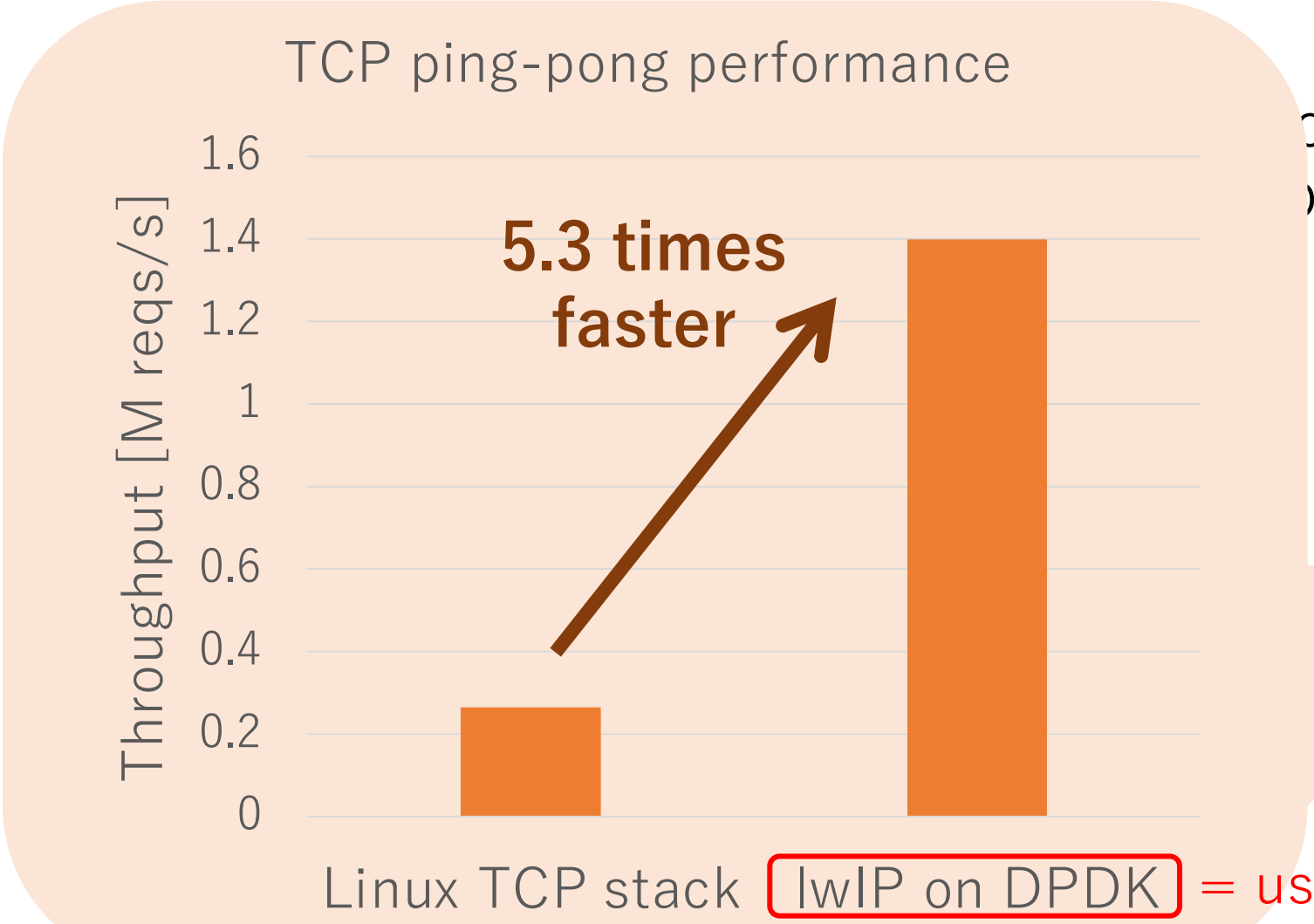


User-space
OS subsystem

Highly performant

https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

Motivating Use Case



= user-space network stack

o transparently apply
pplications

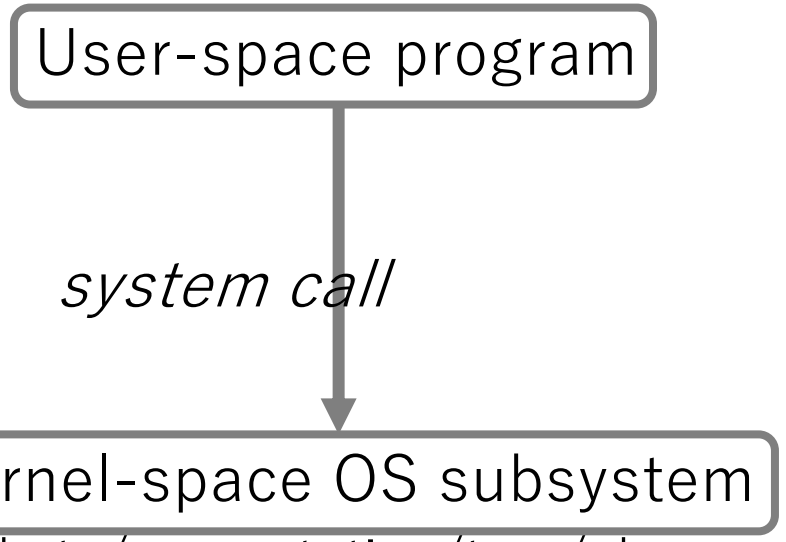
User-space
OS subsystem

Highly performant

https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

Motivating Use Case

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



User-space
OS subsystem

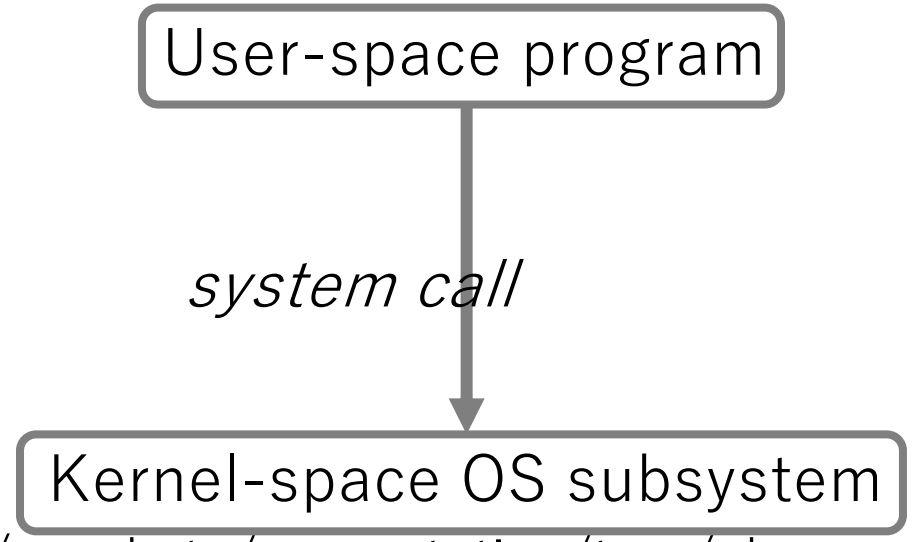
Highly performant

https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

Motivating Use Case

- System call hook mechanisms allow us to transparently apply user-space OS subsystems 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

https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

Motivating Use Case

- System call hook mechanisms allow us to transparently apply user-space OS subsystems 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

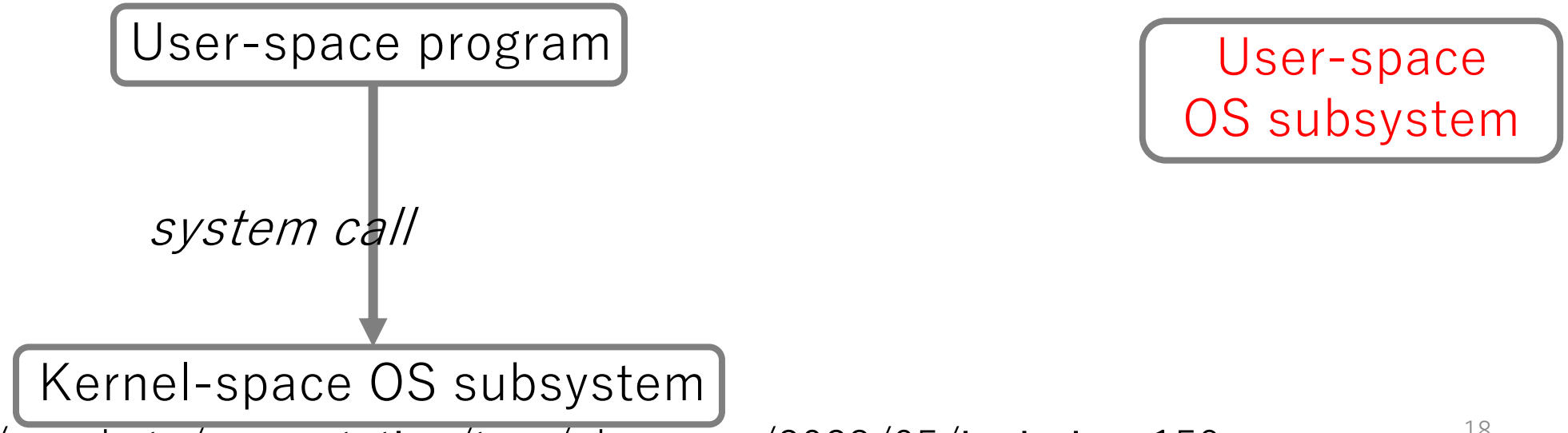
Kernel-space OS subsystem

https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

Motivating Use Case

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

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

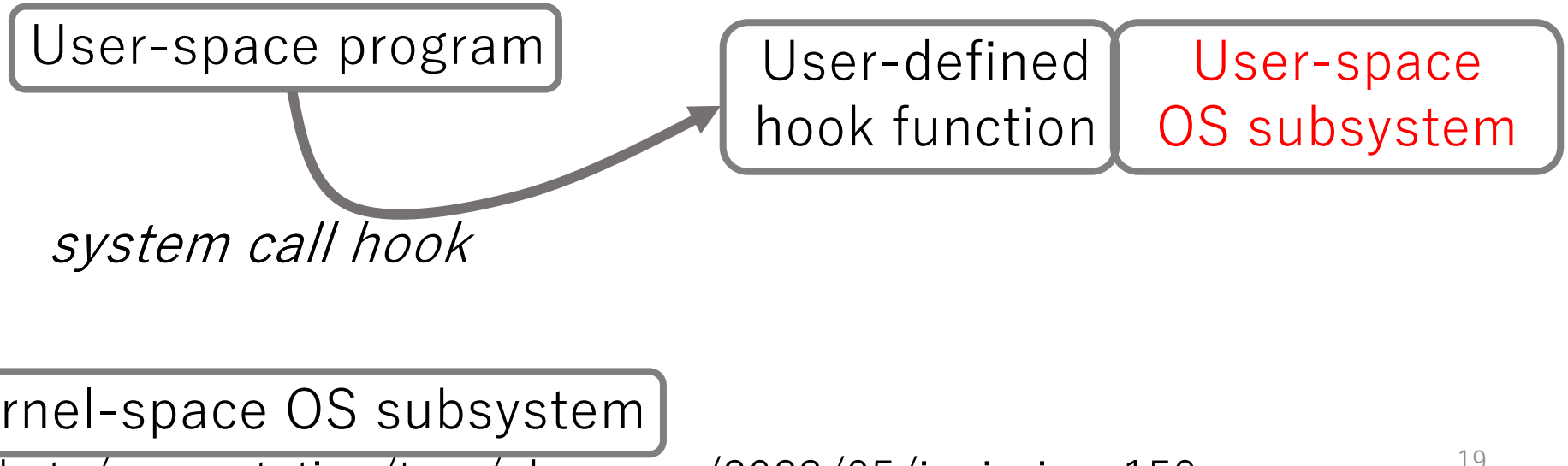


https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

Motivating Use Case

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

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



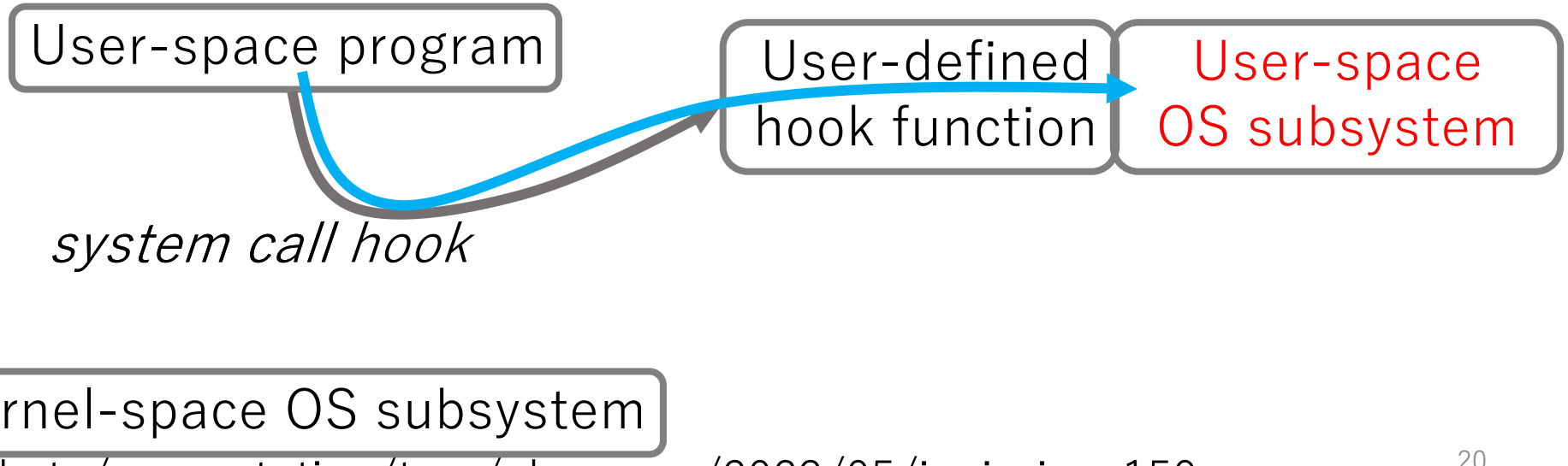
https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

Motivating Use Case

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

If we use a system call hook mechanism, …

no modification of the user-space program is necessary



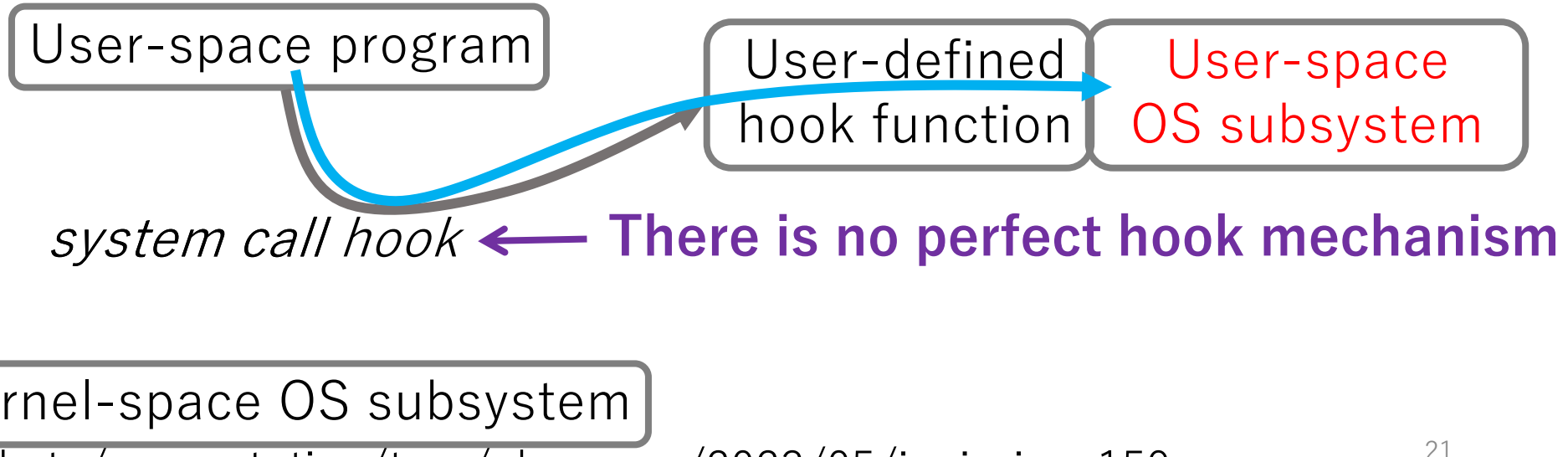
https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

Problem

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

If we use a system call hook mechanism, …

no modification of the user-space program is necessary



https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

Problem

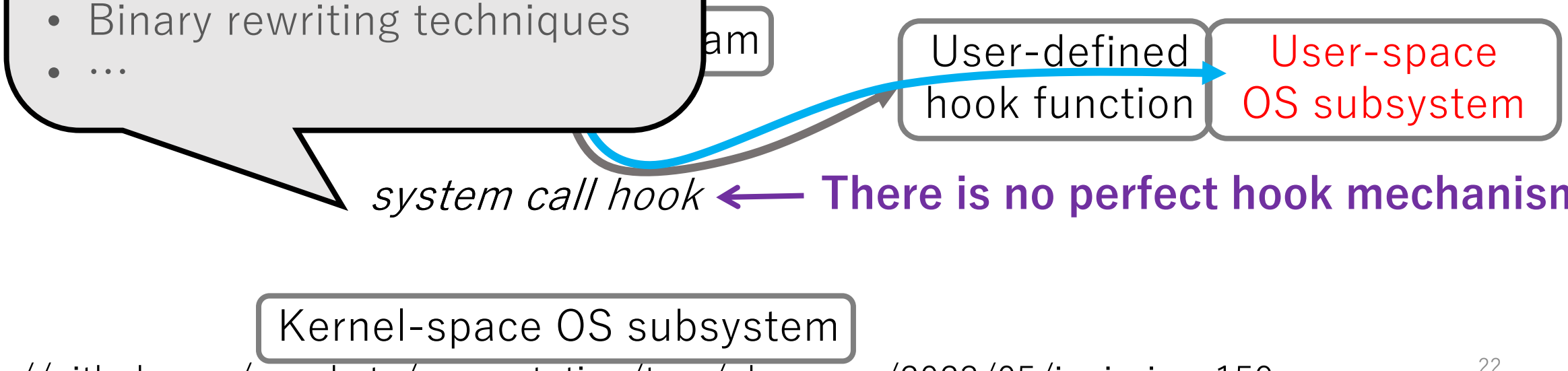
Existing Mechanisms

- ptrace
- Syscall User Dispatch (SUD)
- int3 signaling technique
- LD_PRELOAD trick
- Binary rewriting techniques
- …

ms allow us to transparently apply
to existing applications

ook mechanism, …

user-space program is necessary



https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

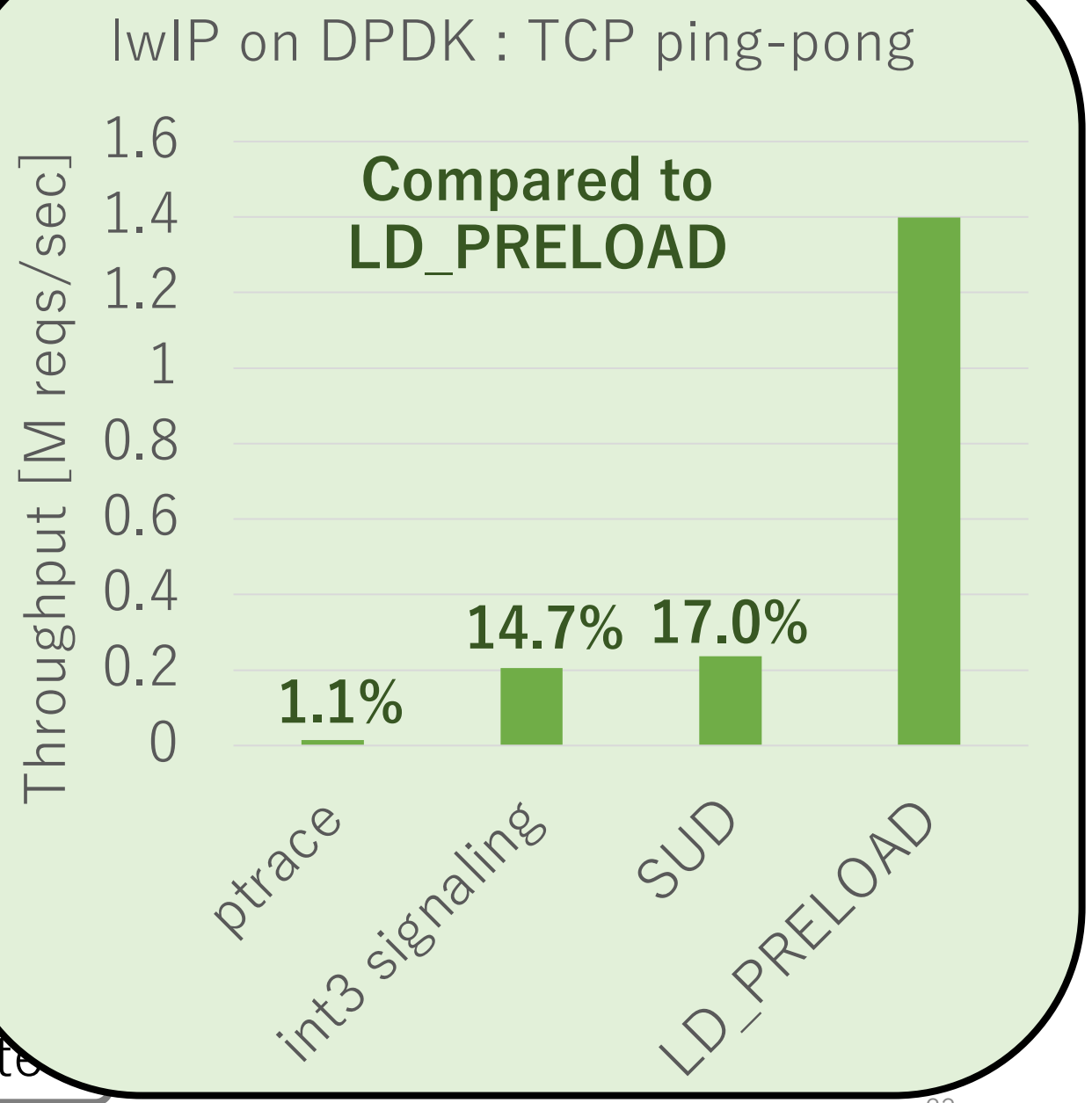
Problem

Existing Mechanisms

- ptrace
- Syscall User Dispatch (SUD)
- int3 signaling technique
- LD_PRELOAD trick
- Binary rewriting techniques
- …

system call hook

Kernel-space OS subsyste



https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

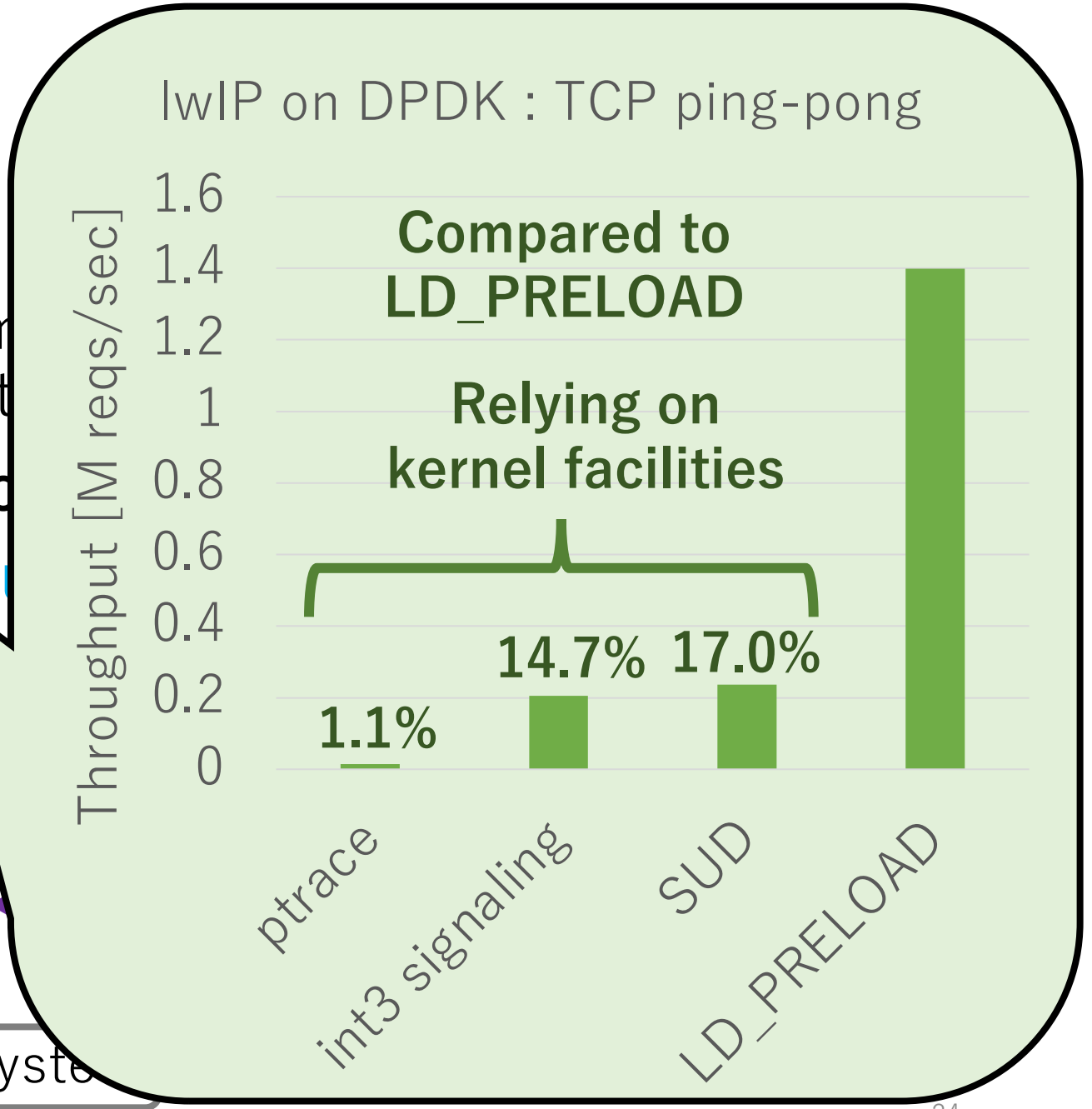
Problem

Existing Mechanisms

- ptrace
- Syscall User Dispatch (SUD)
- int3 signaling technique
- LD_PRELOAD trick
- Binary rewriting techniques
- …

system call hook

Kernel-space OS subsyste



https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

Prob

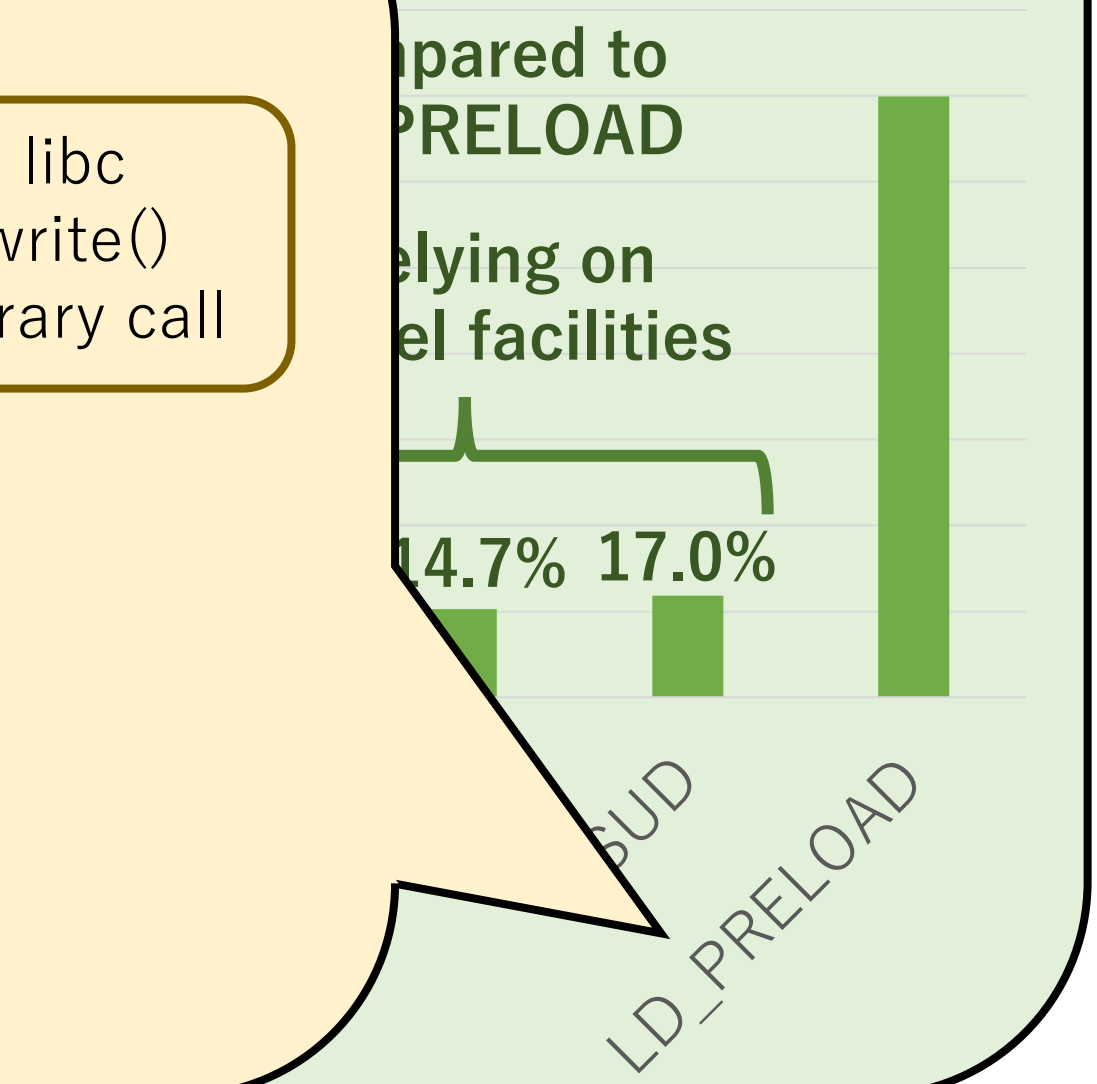
Ex

- ptra
- Sysc
- int3
- LD_
- Bina
- …

```
app_function(...)
{
  …
  write(...)
  …
}
```

libc
write()
library call

K : TCP ping-pong



https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

Prob

Ex

- ptra
- Sysc
- int3
- LD_
- Bina
- …

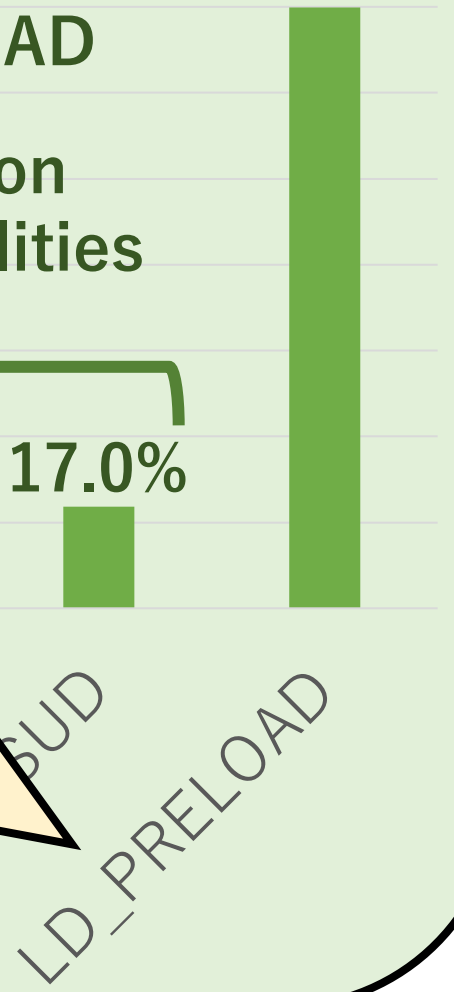
```
app_function(...)
{
  …
  write(...)
  …
}
```

libc
write()
library call

User-defined
write()
library call

LD_PRELOAD

K : TCP ping-pong



https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

Prob

Ex

- ptra
- Sysc
- int3
- LD_
- Bina
- …

```
app_function(...)
{
  …
  write(...)
  …
}
```

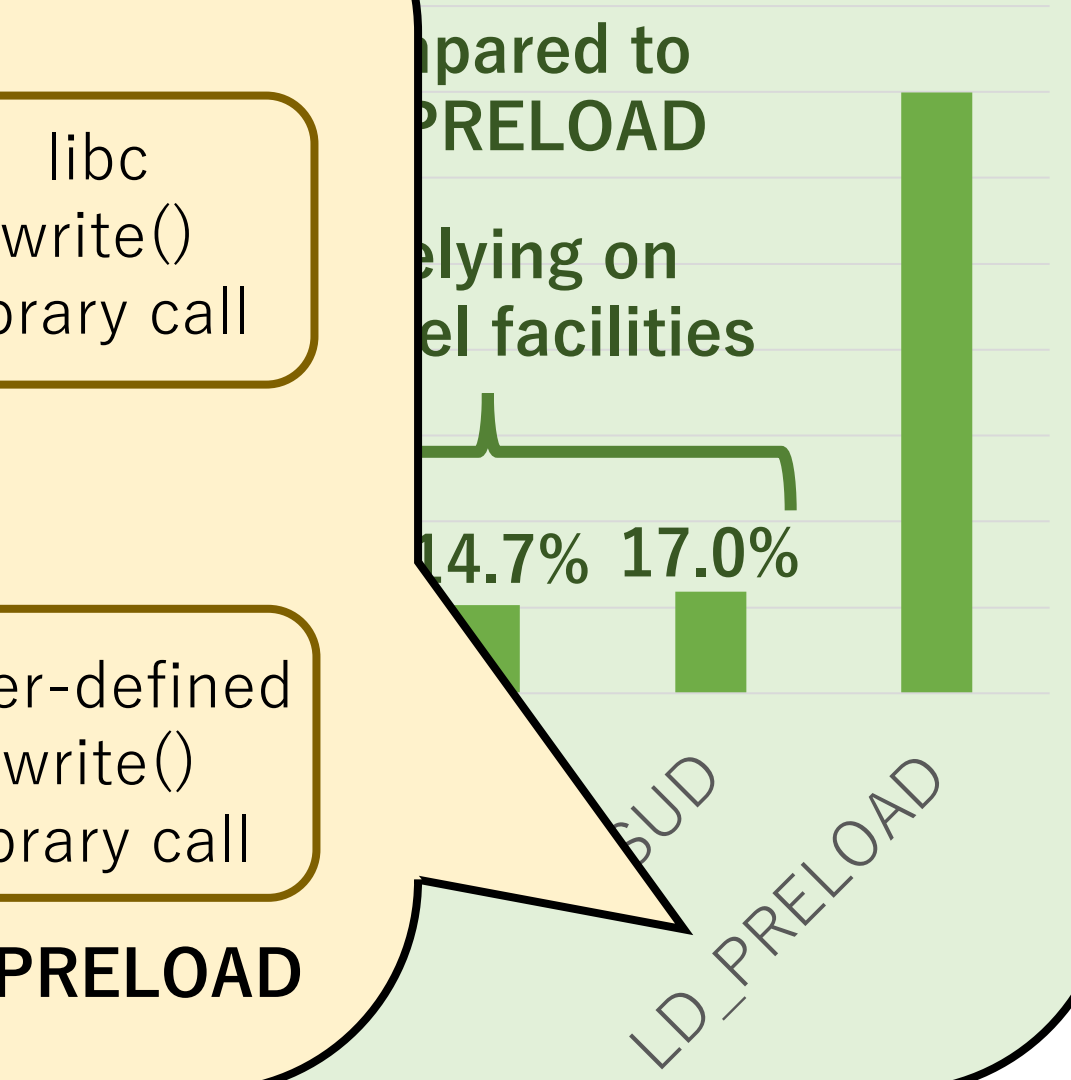
function call replacement

libc
write()
library call

User-defined
write()
library call

LD_PRELOAD

OK : TCP ping-pong



https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

Prob

Ex

- ptra
- Sysc
- int3
- LD_
- Bina
- …

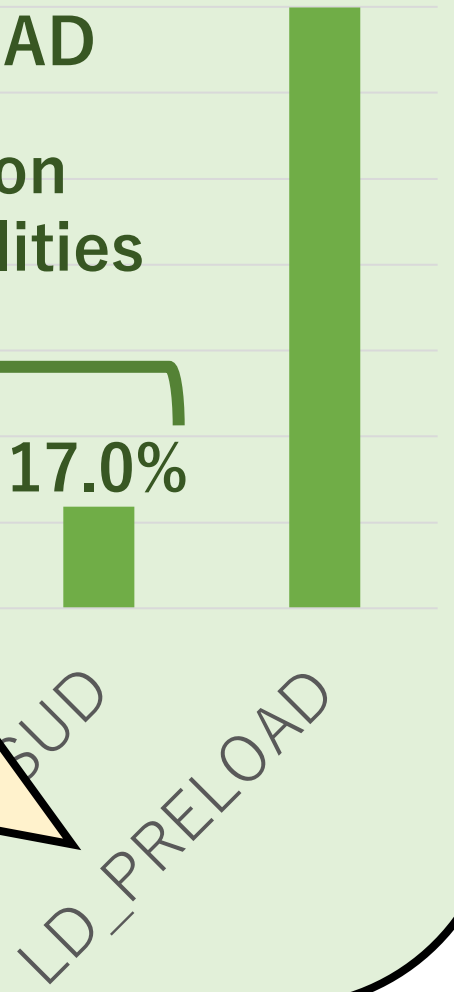
```
app_function(...)
{
  …
  special_write(...)
  …
}
```

libc
write()
library call

User-defined
write()
library call

LD_PRELOAD

OK : TCP ping-pong



https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

Prob

Ex

- ptra
- Sysc
- int3
- LD_
- Bina
- …

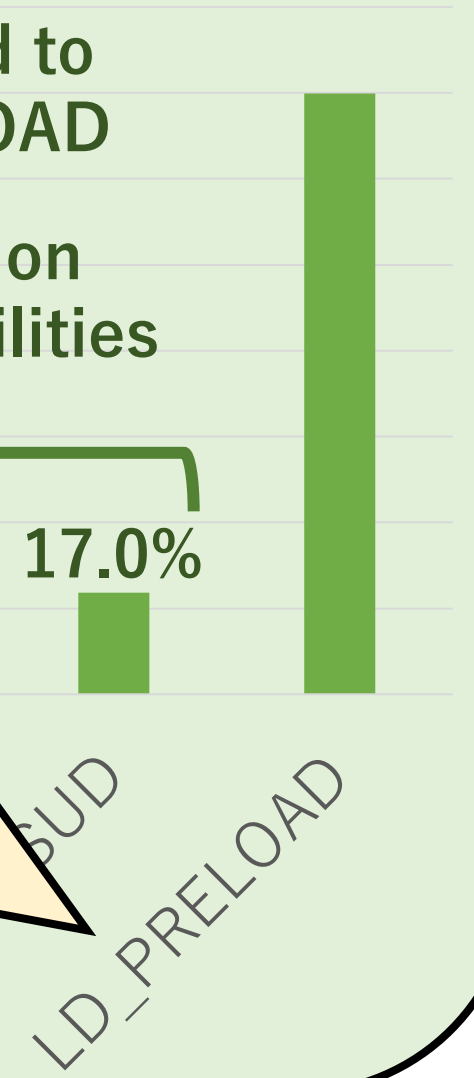
```
app_function(...)
{
  …
  special_write(...)
  …
}

special_write(...)
{
  asm volatile (
   trigger
   write syscall
  )
}
```

libc
write()
library call

User-defined
write()
library call

LD_PRELOAD

K : TCP ping-pong



https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

Prob

Ex

- ptra
- Sysc
- int3
- LD_
- Bina
- …

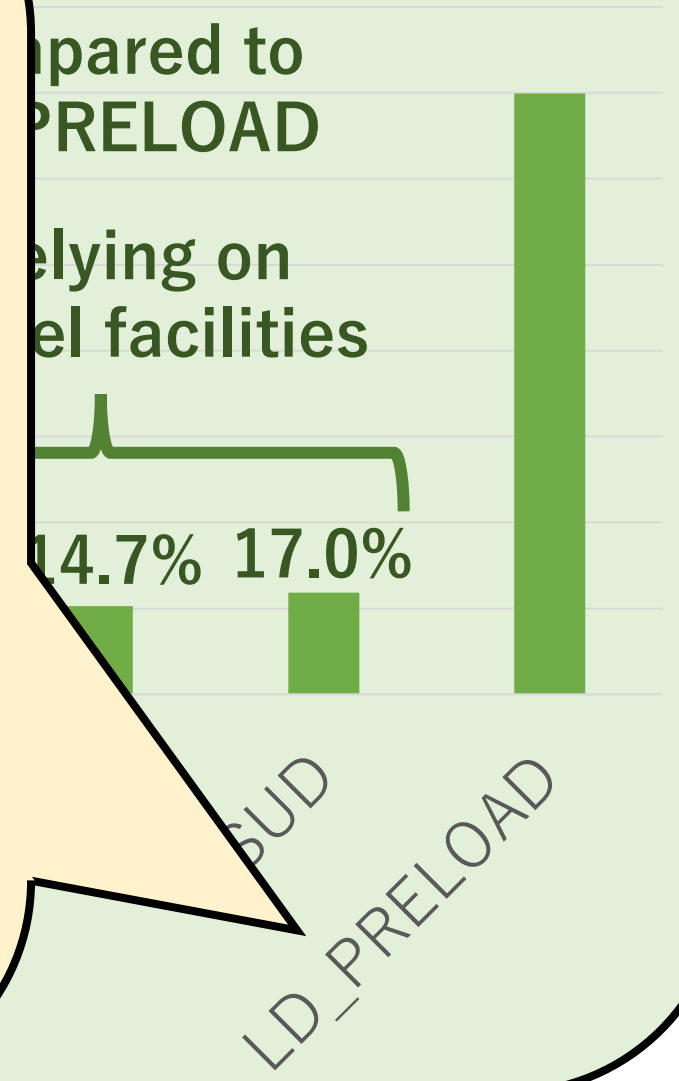
```
app_function(...)
{
  …
  special_write(...)
  …
}

special_write(...)
{
  asm volatile (
   trigger
   write syscall
  )
}
```

libc
write()
library call

User-defined
write()
library call

LD_PRELOAD

K : TCP ping-pong



https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

Prob

Ex

- ptra
- Sysc
- int3
- LD_
- Bina
- …

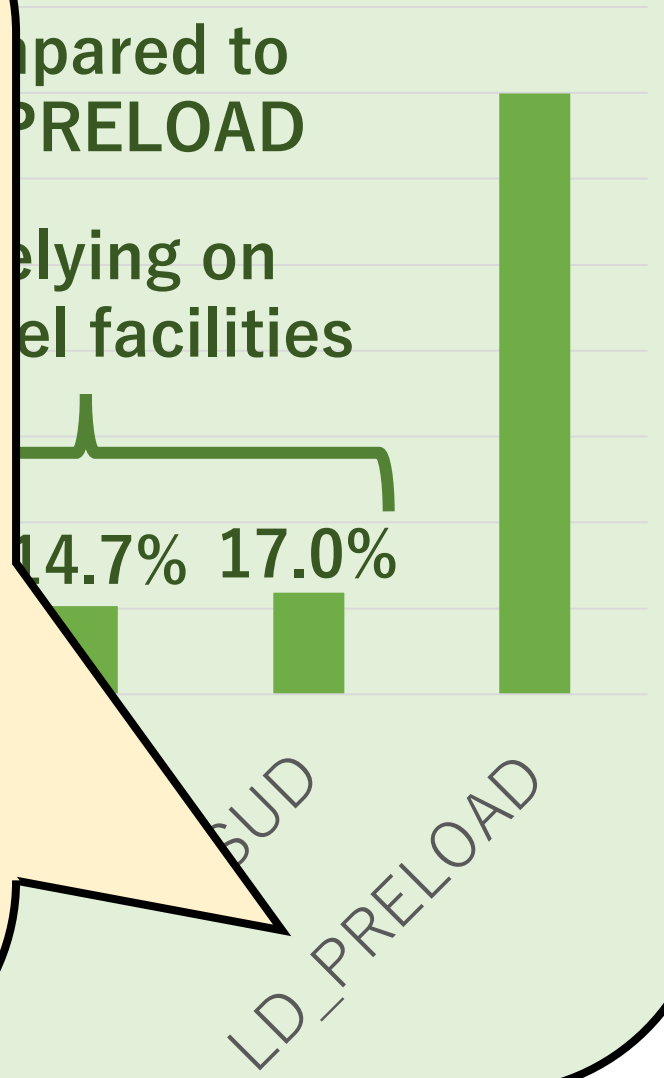
```
app_function(...)
{
  …
  special_write(...)
  …
}

special_write(...)
{
  asm volatile (
    trigger
    write syscall
  )
}
```

libc
write()
library call

User-defined
write()
library call

LD_PRELOAD

K : TCP ping-pong



https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

Prob

Ex

- ptra
- Sysc
- int3
- LD_
- Bina
- …

```
app_function(...)
{
  …
  special_write(...)
  …
}

special_write(...)
{
  asm volatile (
    trigger
    write syscall
  )
}
```

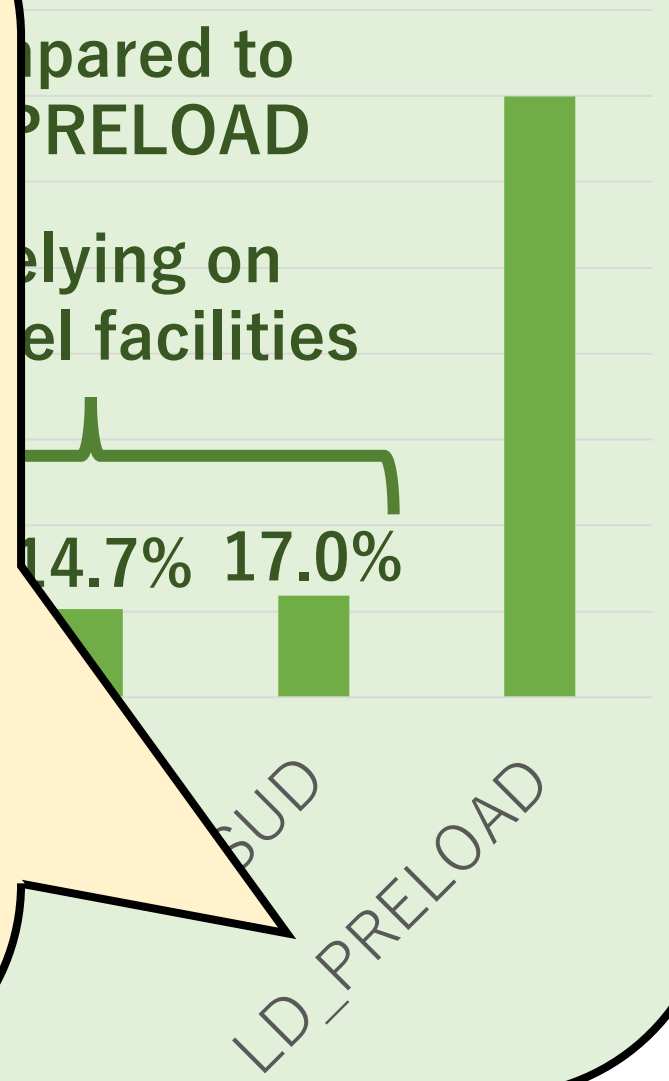
libc
write()
library call

User-defined
write()
library call

Hook is not applied

LD_PRELOAD

OK : TCP ping-pong



https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

Prob

Ex

- ptra
- Sysc
- int3
- LD_
- Bina
- …

```
app_function(...)
{
  …
  special_write(...)
  …
}
```

libc
write()
library call

```
special_write(...)
{
  asm volatile (
    trigger
    write syscall
  )
}
```

because names are different

User-defined
write()
library call

Hook is not applied

LD_PRELOAD

K : TCP ping-pong

pared to
PRELOAD

lying on
el facilities

14.7% 17.0%

SUD

LD_PRELOAD

https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

Prob

Ex

- ptra
- Sysc
- int3
- LD_
- Bina
- …

We see this case in glibc often

```
app_function(...)
{
  …
  special_write(...)
  …
}
```

libc
write()
library call

```
special_write(...)
{
  asm volatile (
    trigger
    write syscall
  )
}
```

because names are different

User-defined
write()
library call

Hook is not applied **LD_PRELOAD**

K : TCP ping-pong

pared to
PRELOAD

lying on
el facilities

14.7% 17.0%

SUD

LD_PRELOAD

https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

Problem

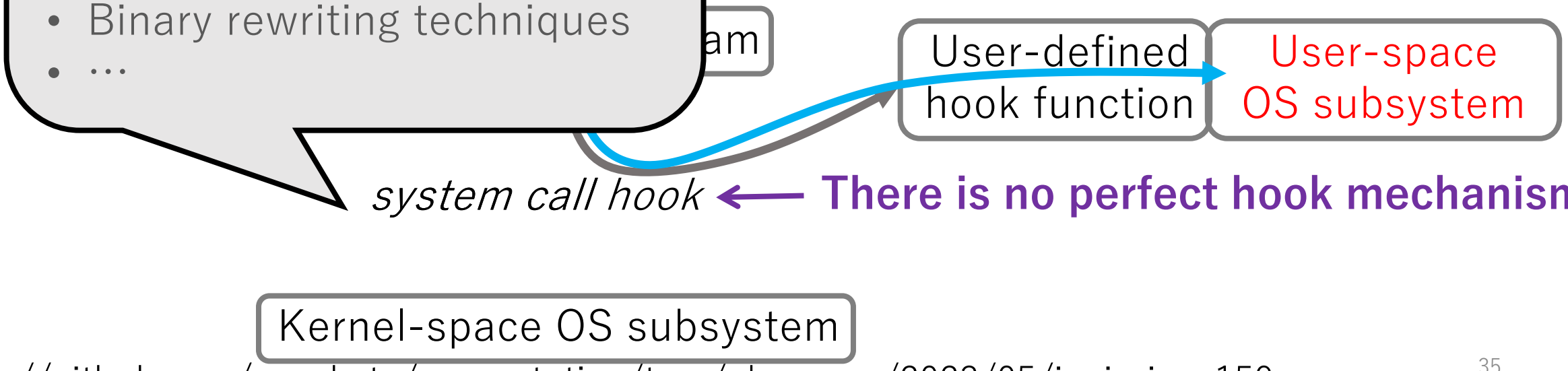
Existing Mechanisms

- ptrace
- Syscall User Dispatch (SUD)
- int3 signaling technique
- LD_PRELOAD trick
- Binary rewriting techniques
- …

ms allow us to transparently apply
to existing applications

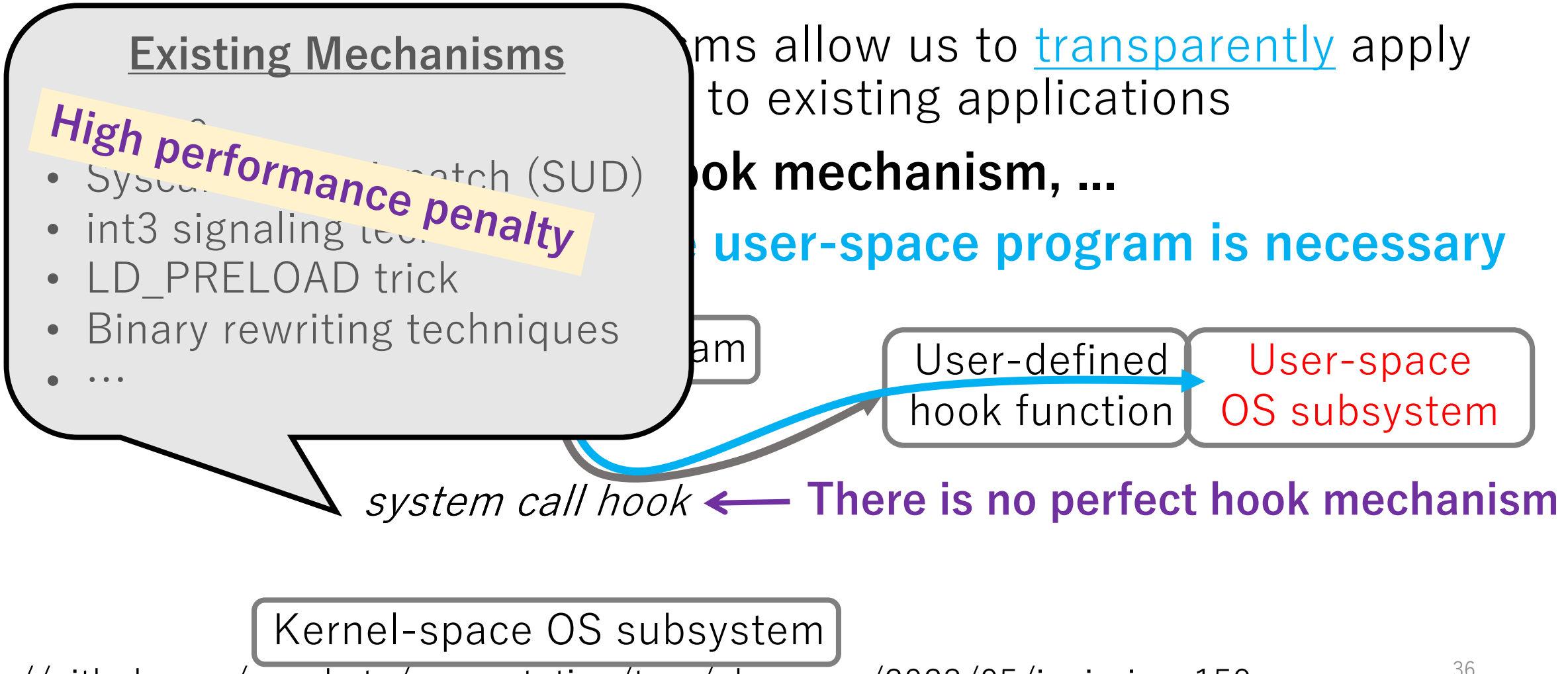
ook mechanism, …

user-space program is necessary



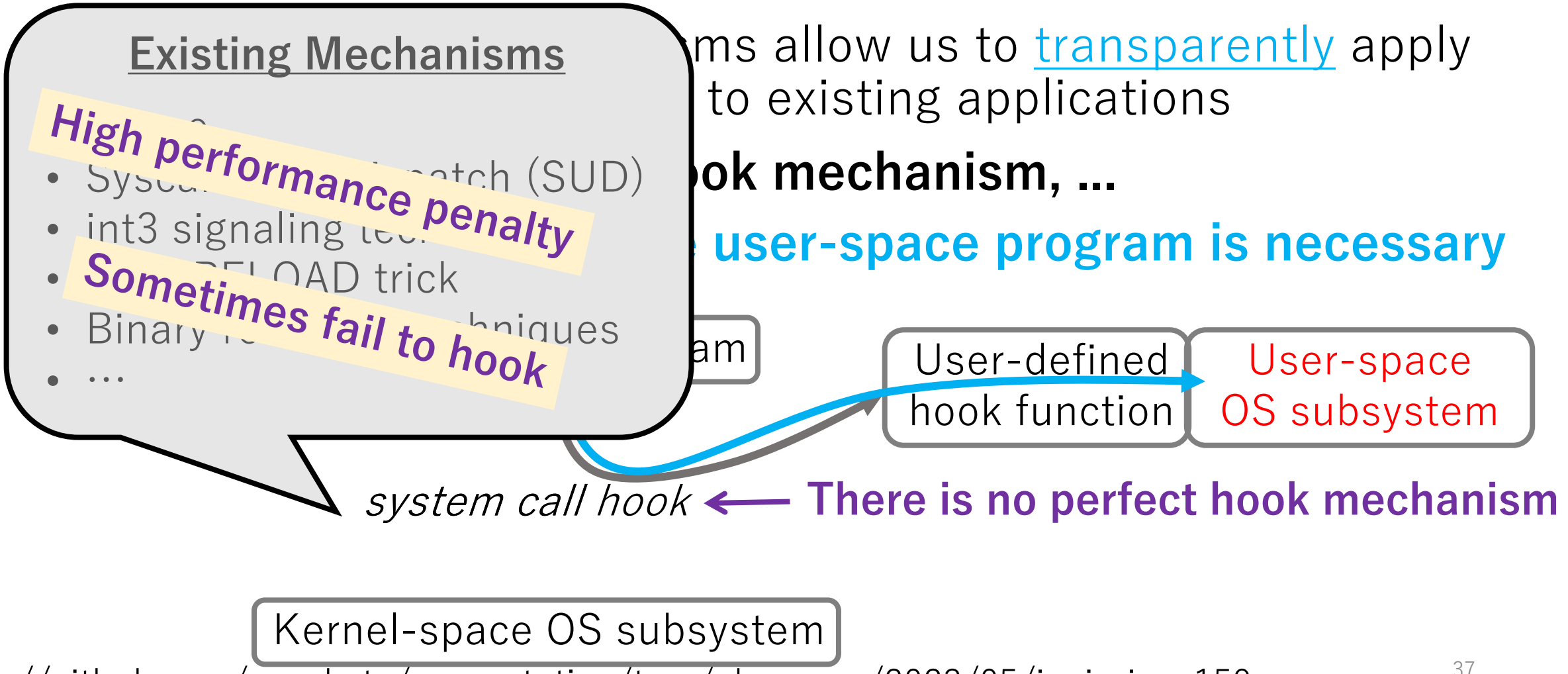
https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

Problem



https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

Problem



https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

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

Existing Mechanisms

- Sysca… patch (SUD)
- int3 signaling tec…
- …ELOAD trick
- Binary re… …hniques
- …

High performance penalty

Sometimes fail to hook

…ms allow us to transparently apply … to existing applications

…ook mechanism, …

… user-space program is necessary

https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

Contribution

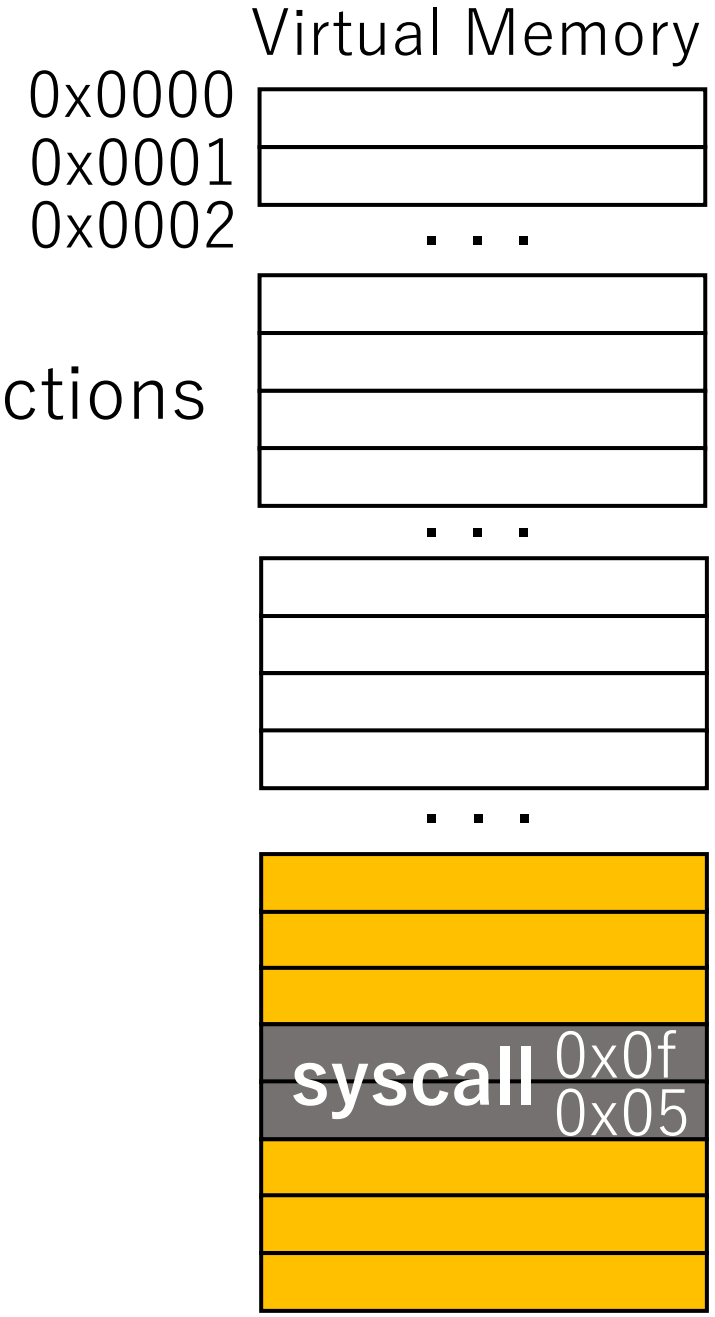
- zpoline employs binary rewriting and offers 6 advantages
  1. Low hook overhead
  2. Exhaustive hooking
  3. No breakage of user-space program logic
  4. No kernel change and no additional kernel module are necessary
  5. No source code of a user-space program is needed
  6. It can be used for emulating system calls

- None of existing mechanisms achieve them simultaneously

https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

Binary Rewriting Approach

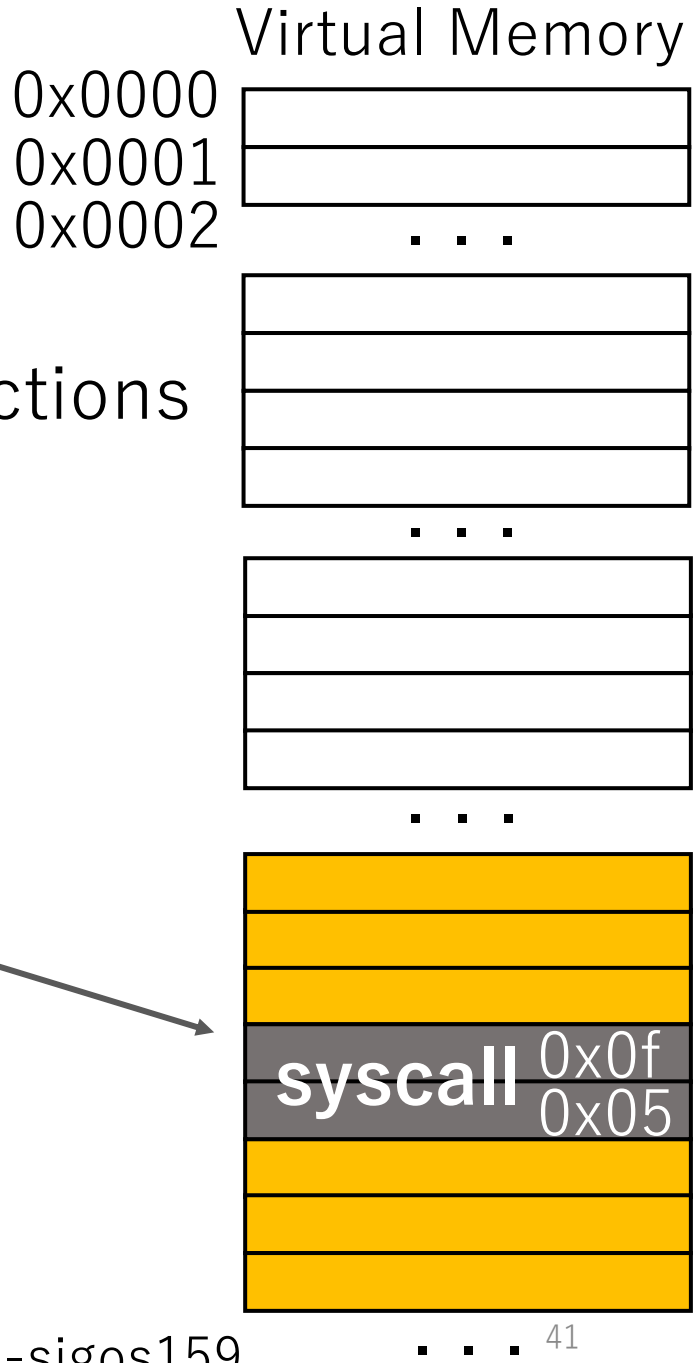
- On x86-64 CPUs, syscall and sysenter instructions trigger a system call
  - syscall: 0x0f 0x05, sysenter: 0x0f 0x34



https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

Binary Rewriting Approach

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

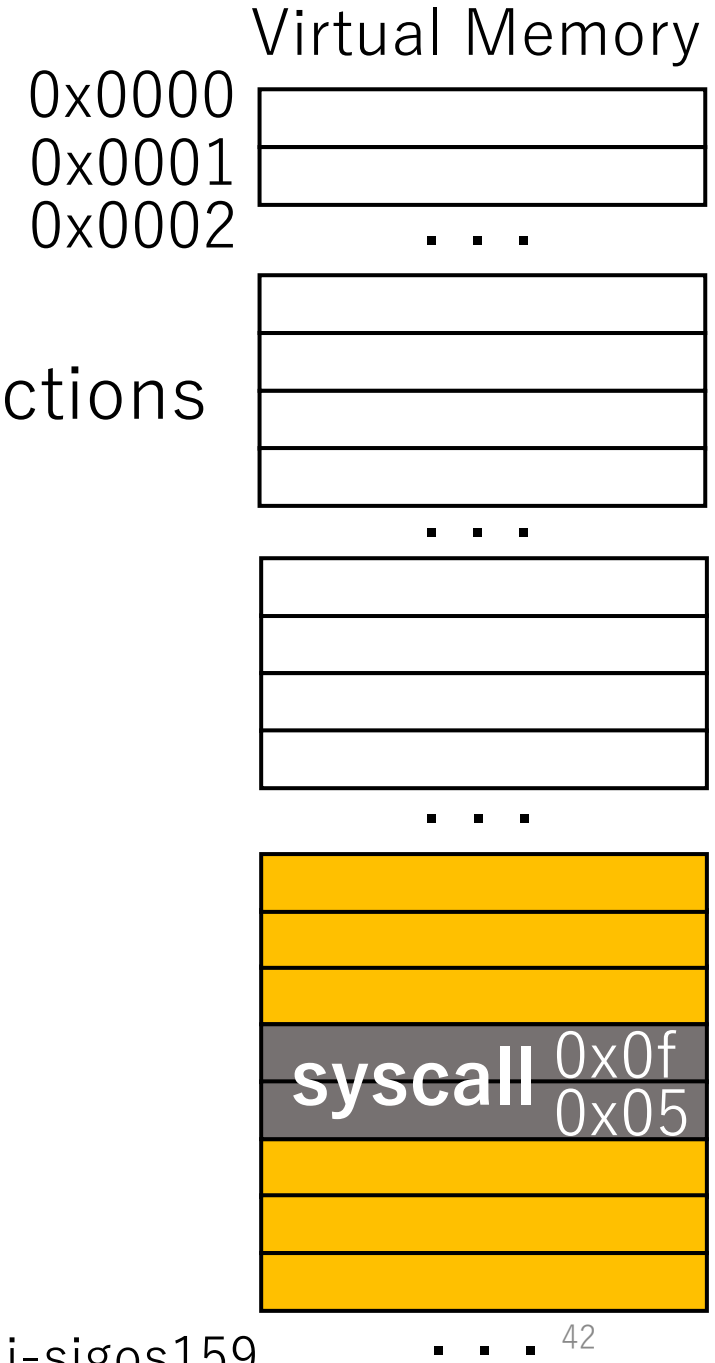


https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

Binary Rewriting Approach

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

- What we wish to achieve

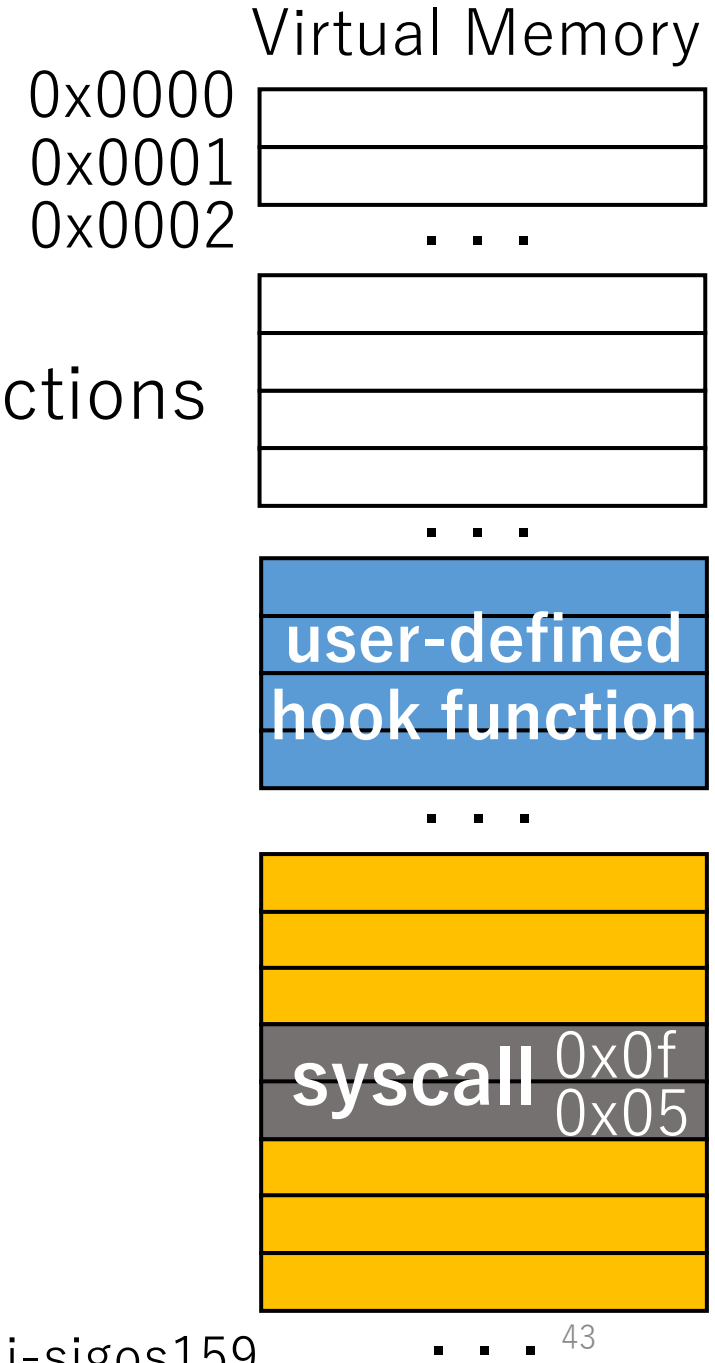


https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

Binary Rewriting Approach

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

- What we wish to achieve

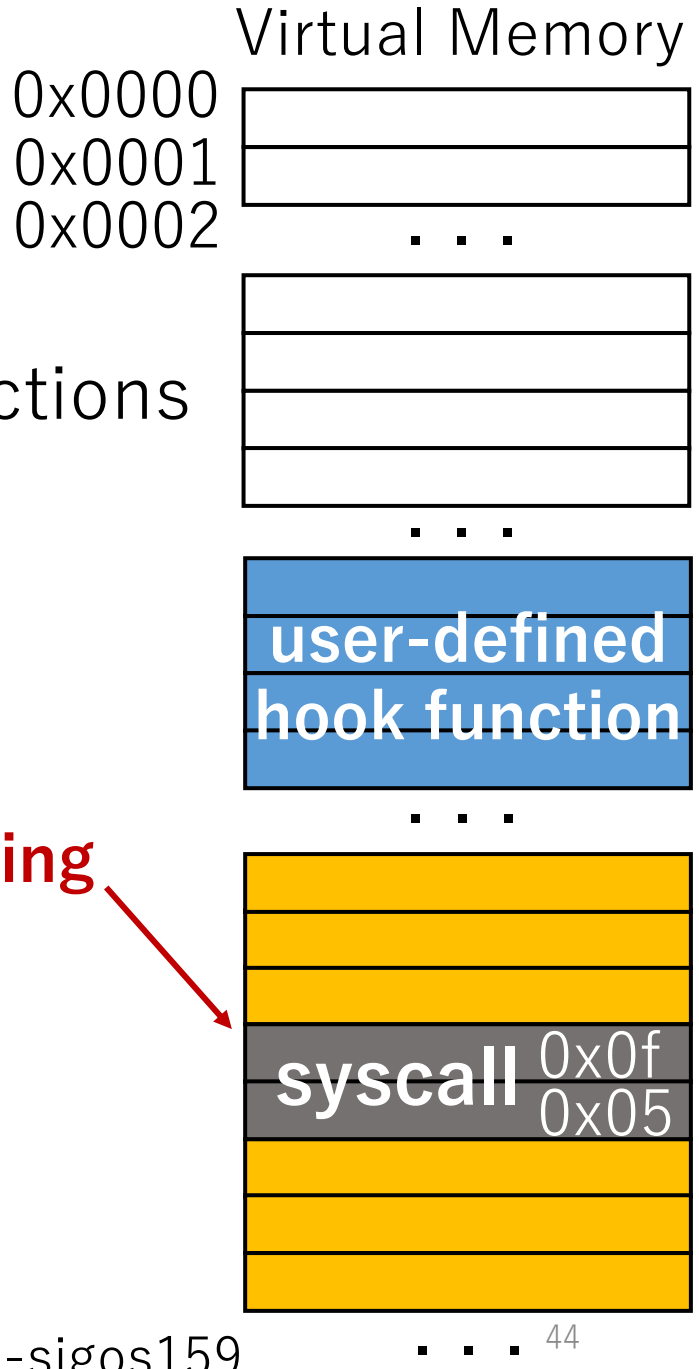


https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

Binary Rewriting Approach

- 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**

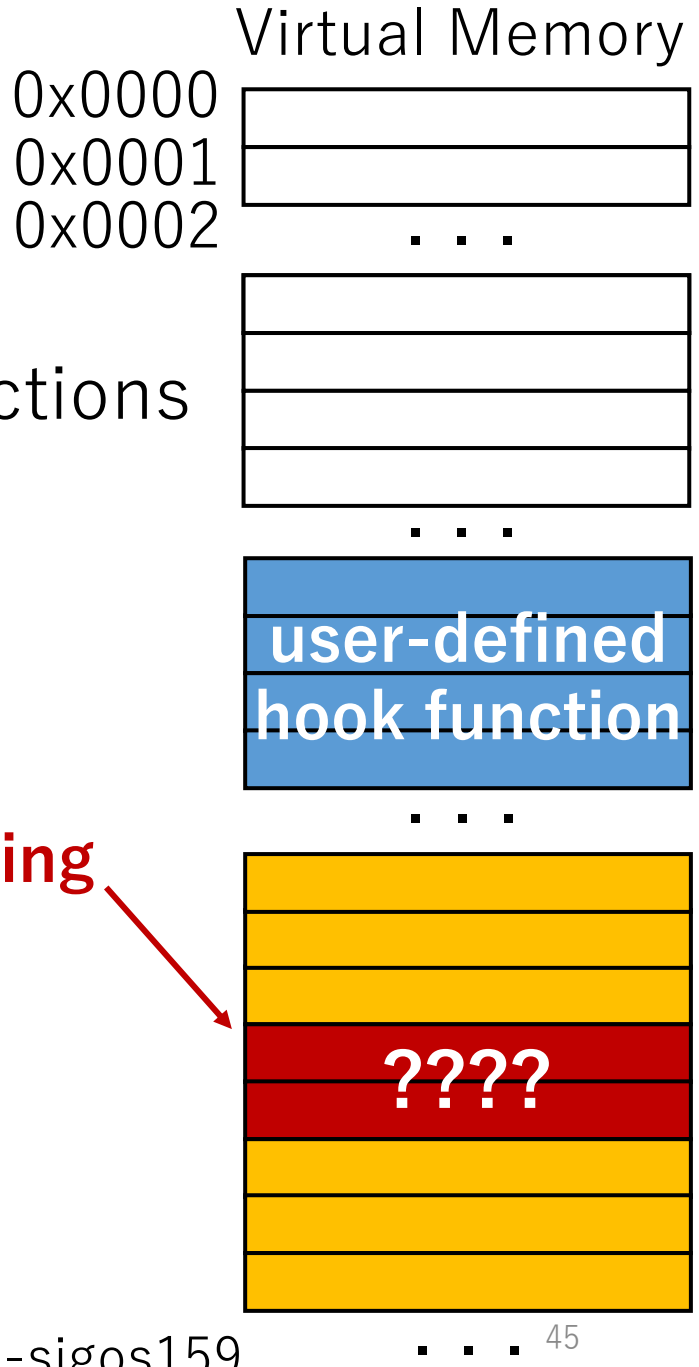


https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

Binary Rewriting Approach

- 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**

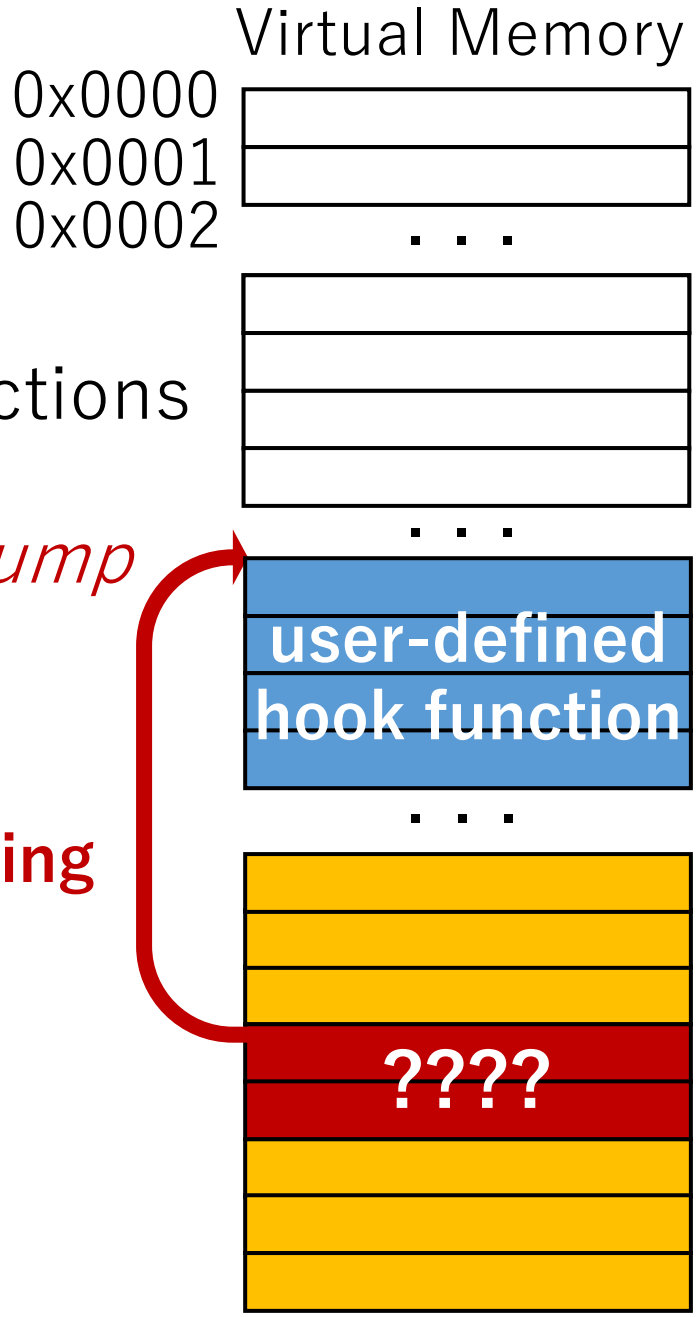


https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

Binary Rewriting Approach

- 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**
  - to jump to a user-defined hook function



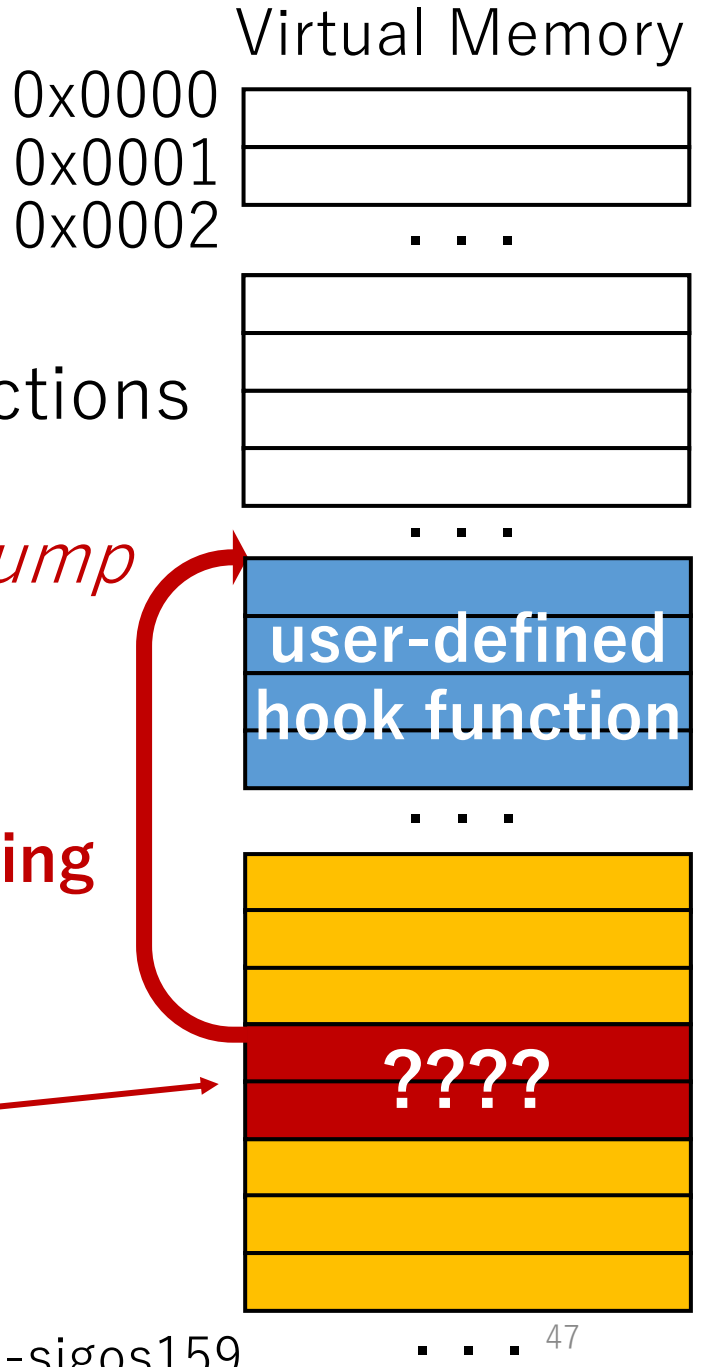
https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

Binary Rewriting Approach

- 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**
  - to jump to a user-defined hook function

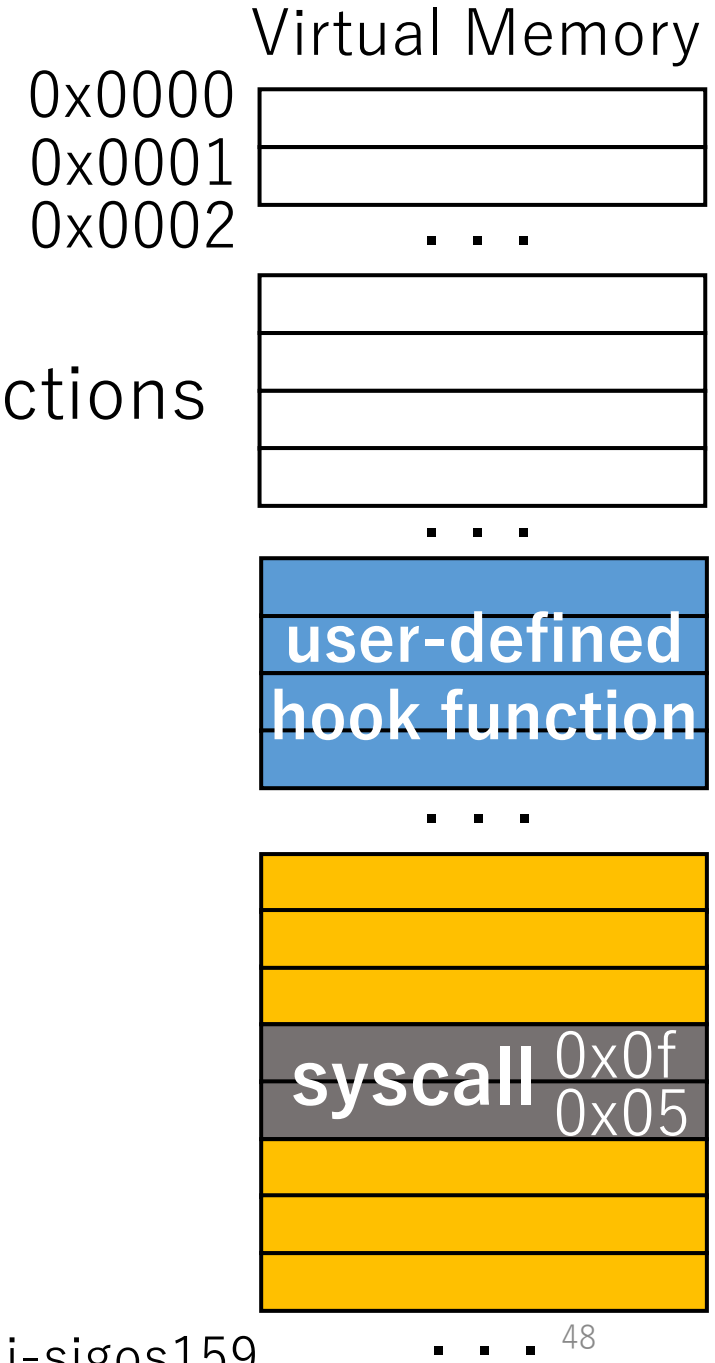
- **Question: what should we put here?**



https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

Challenge

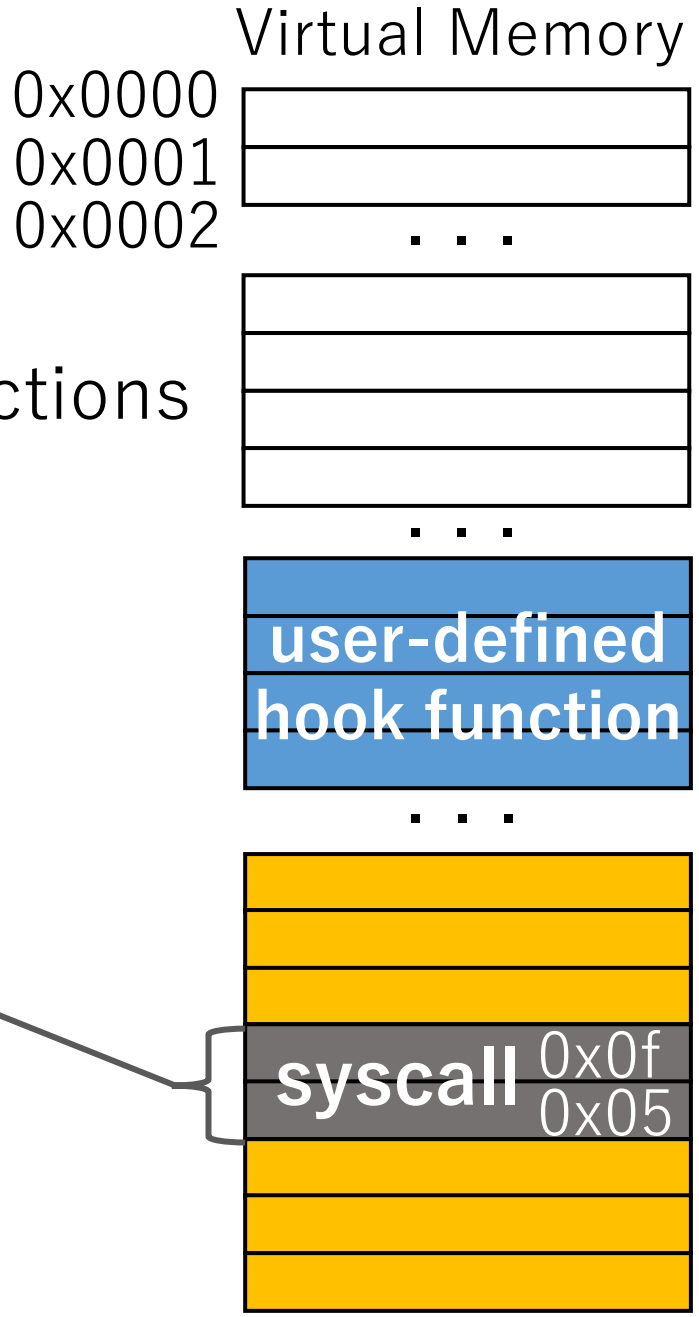
- On x86-64 CPUs, syscall and sysenter instructions trigger a system call
  - syscall: 0x0f 0x05, sysenter: 0x0f 0x34



https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

Challenge

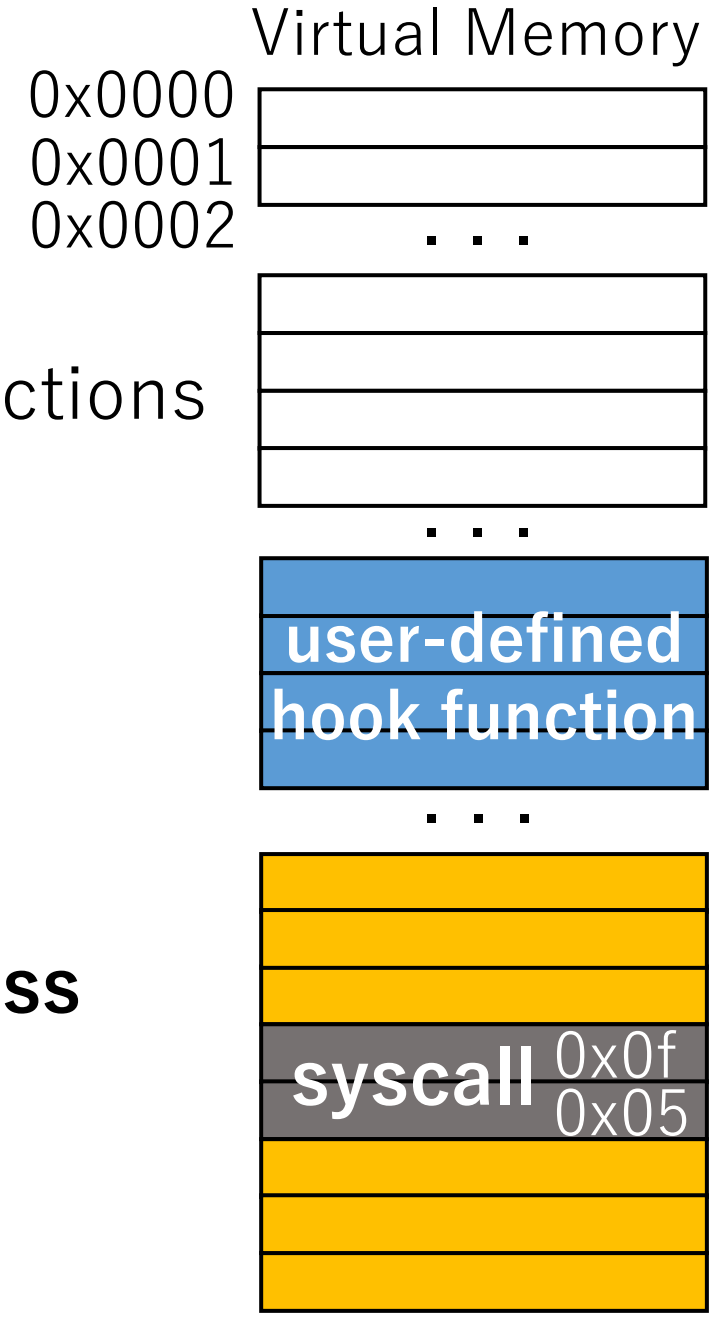
- On x86-64 CPUs, syscall and sysenter instructions trigger a system call
  - syscall: 0x0f 0x05, sysenter: 0x0f 0x34
- syscall and sysenter are **2-byte** instructions



https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

Challenge

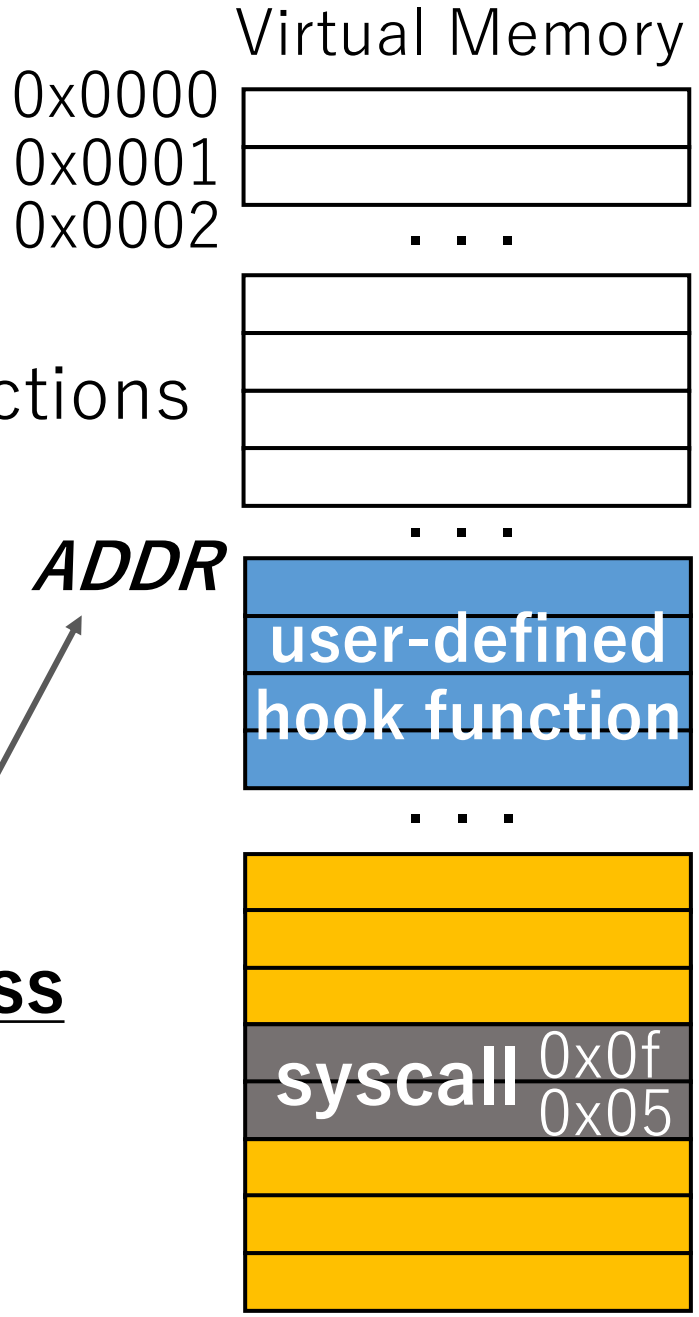
- On x86-64 CPUs, syscall and sysenter instructions trigger a system call
  - syscall: 0x0f 0x05, sysenter: 0x0f 0x34
- syscall and sysenter are **2-byte** instructions
- **Specification for a jump destination address needs more than 2 bytes**



https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

Challenge

- 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**

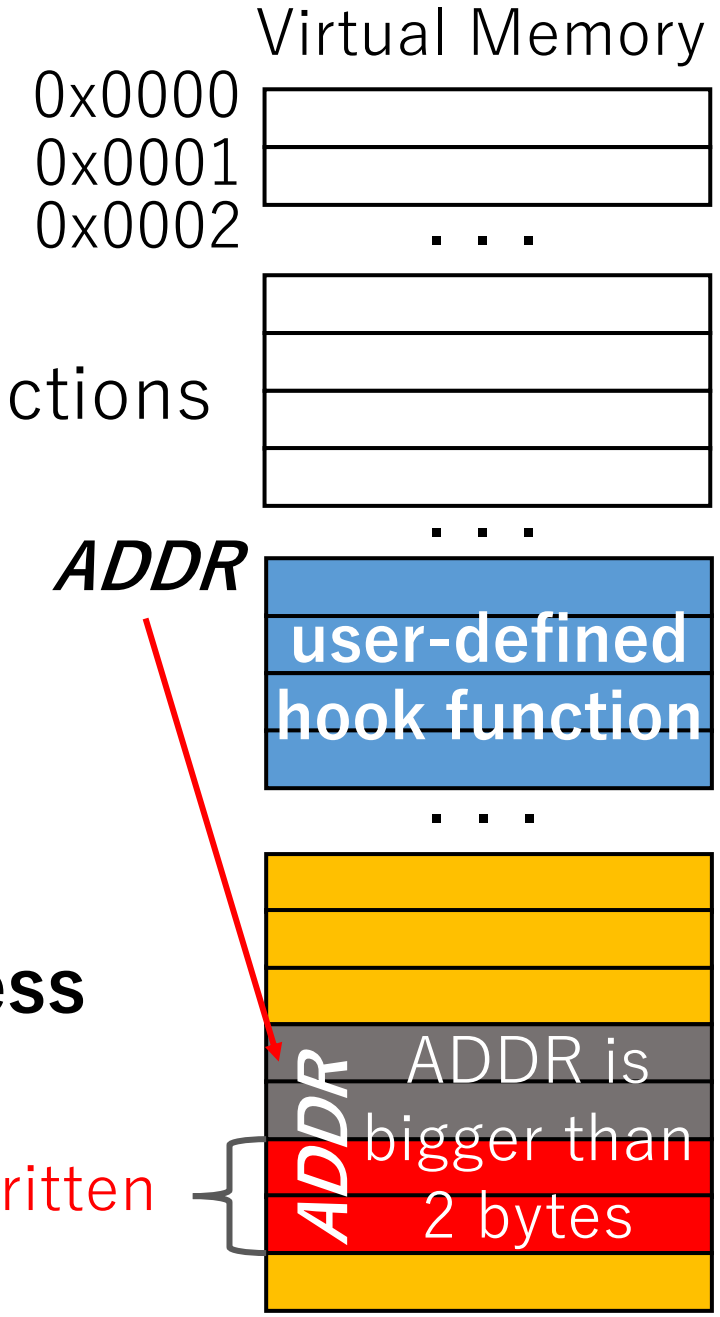


https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

Challenge

- 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

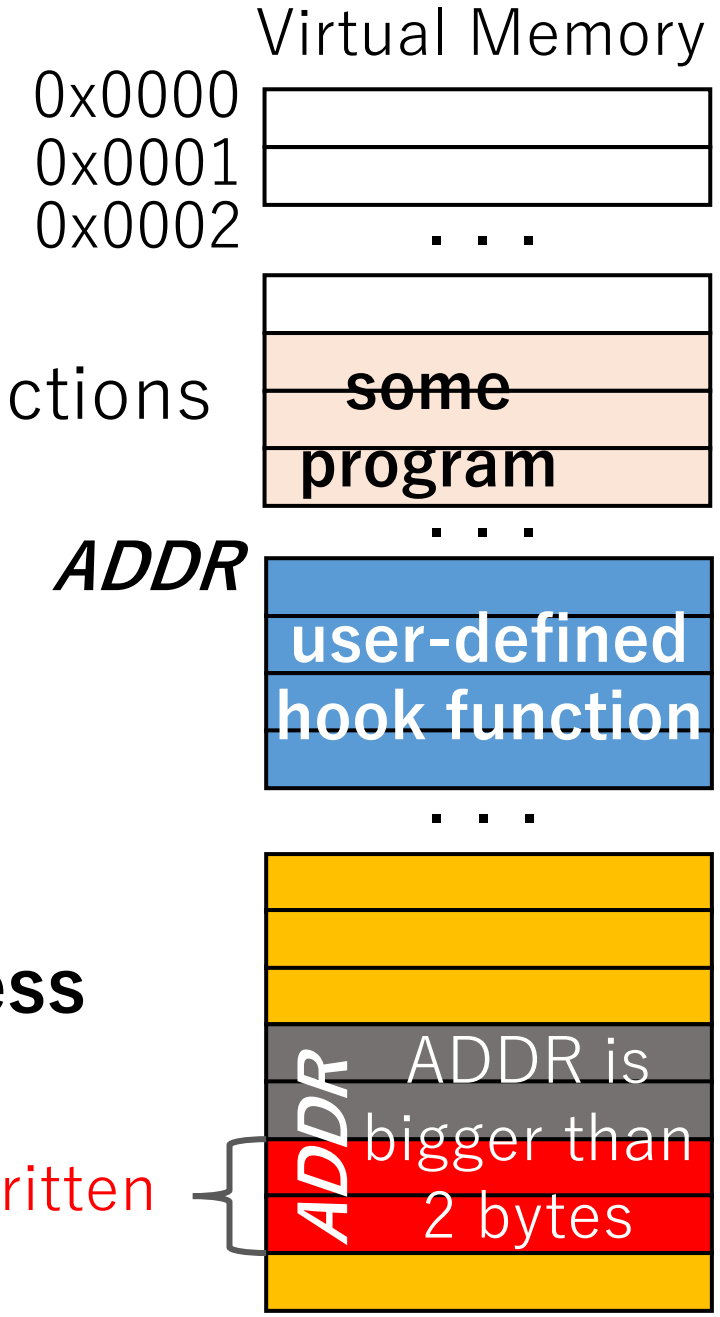


https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

Challenge

- 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



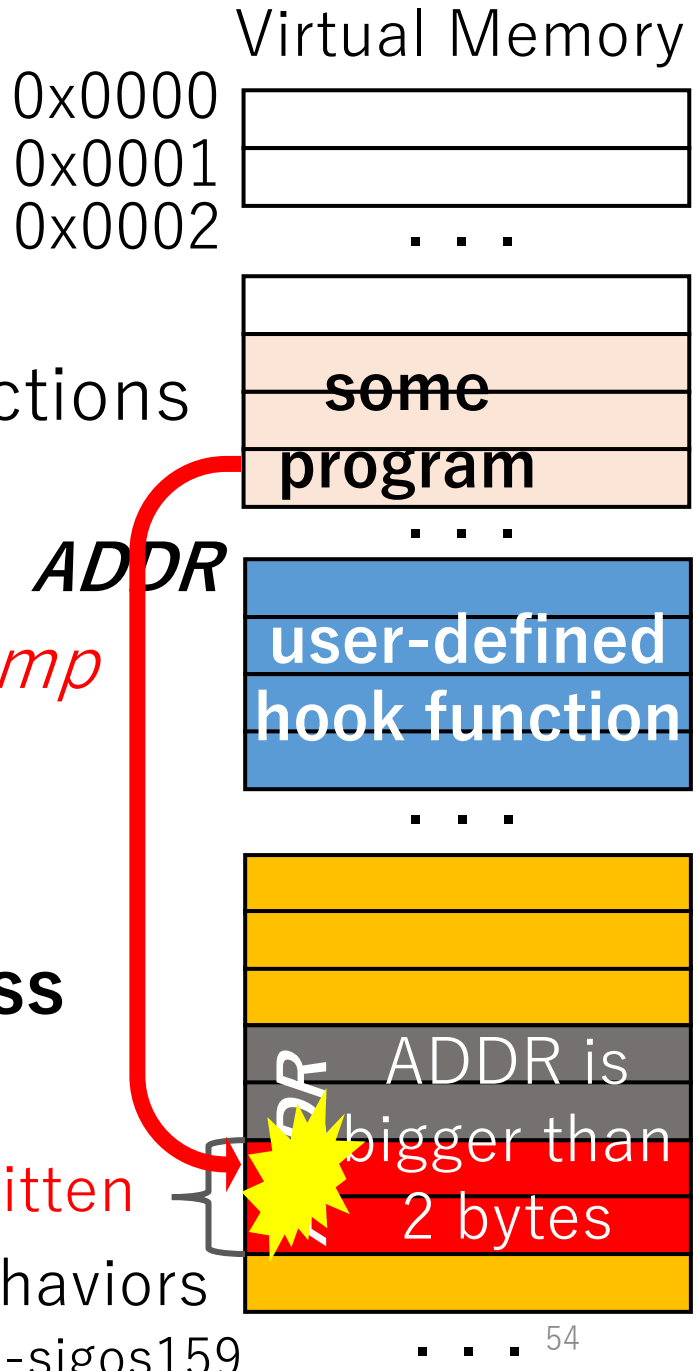
https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

Challenge

- 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



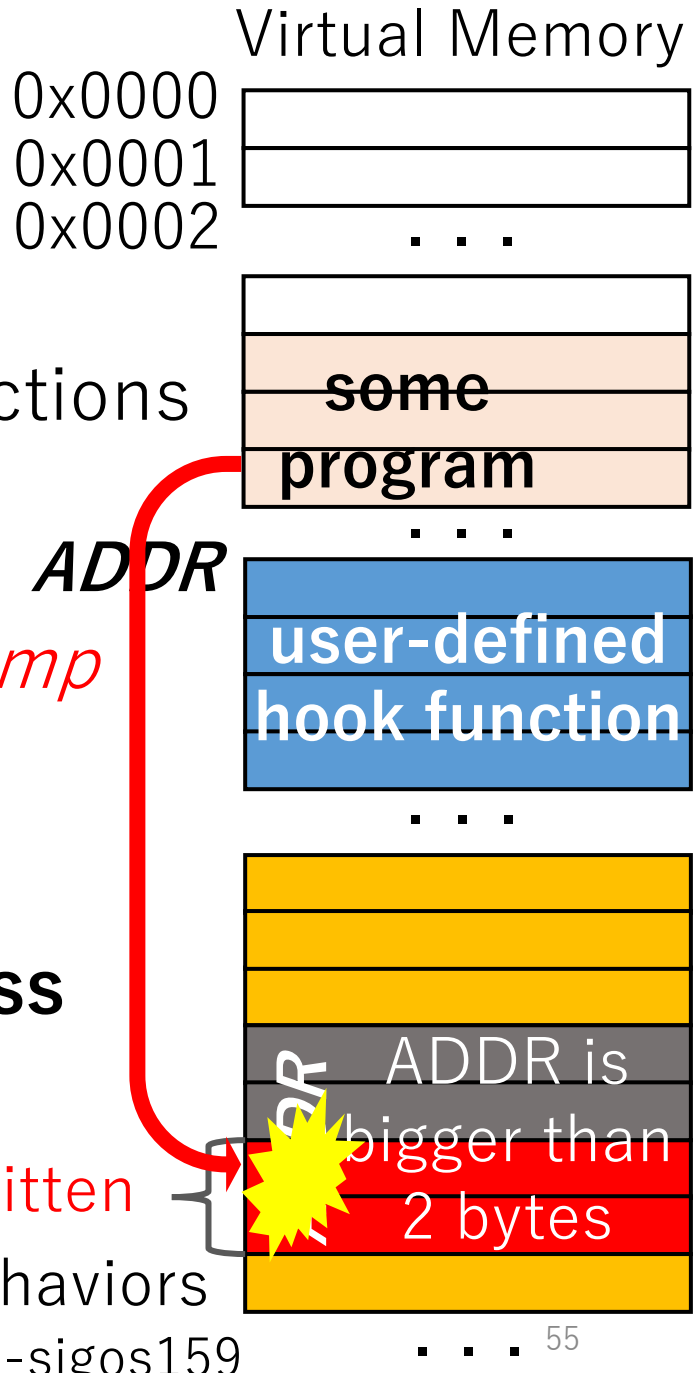
https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

Goal ➡ **jump to a function using only 2 bytes originally occupied by syscall/sysenter**

- 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

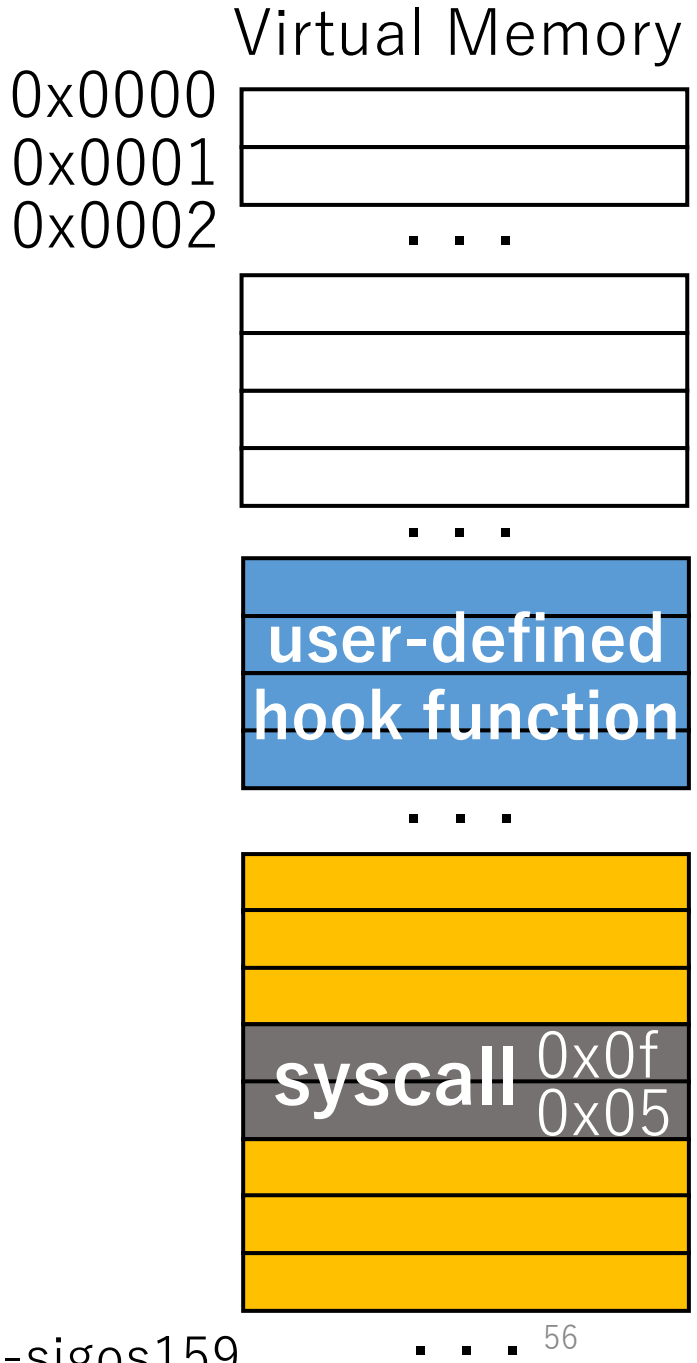
  jump to the overwritten part leads to unexpected behaviors



https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

Calling Convention

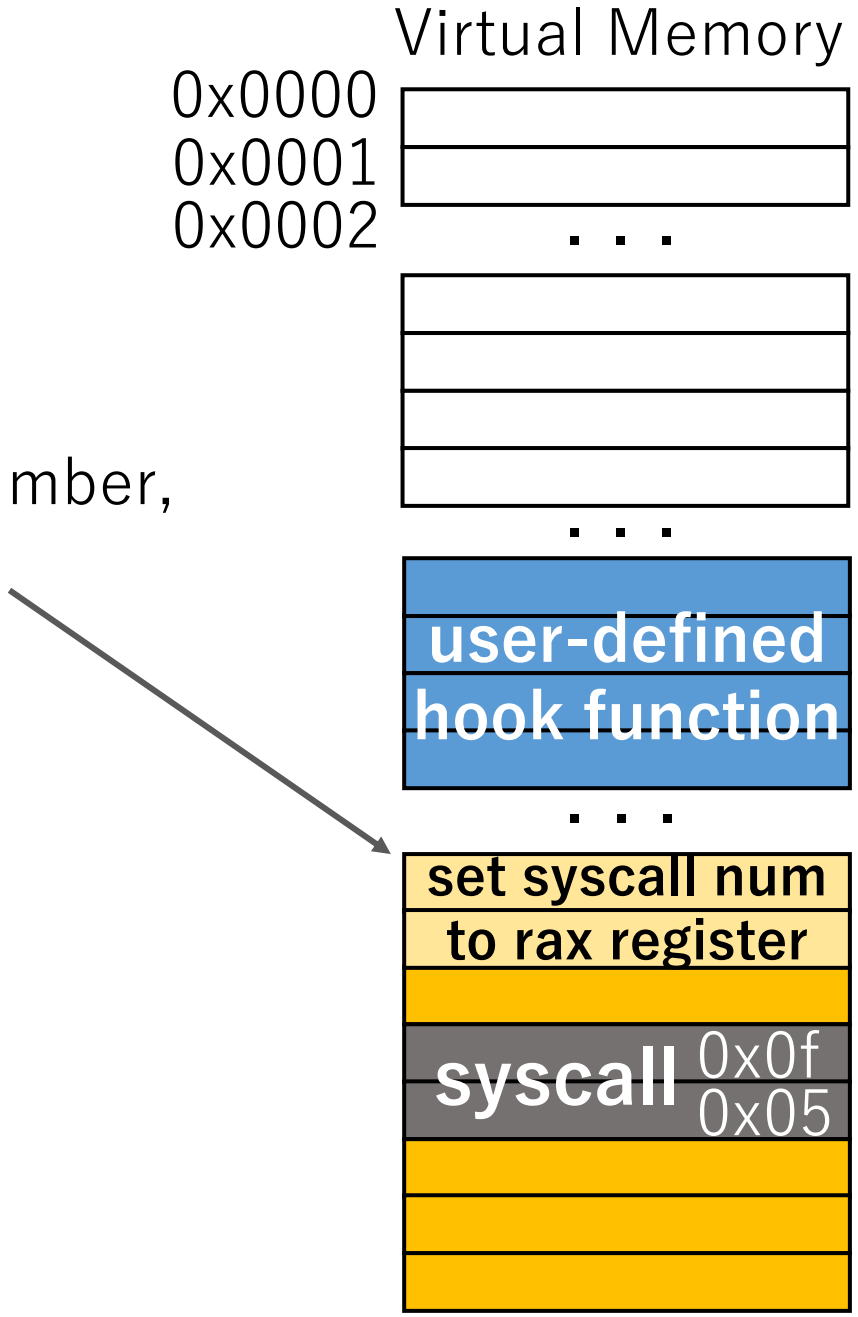
- How to invoke a system call



https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

Calling Convention

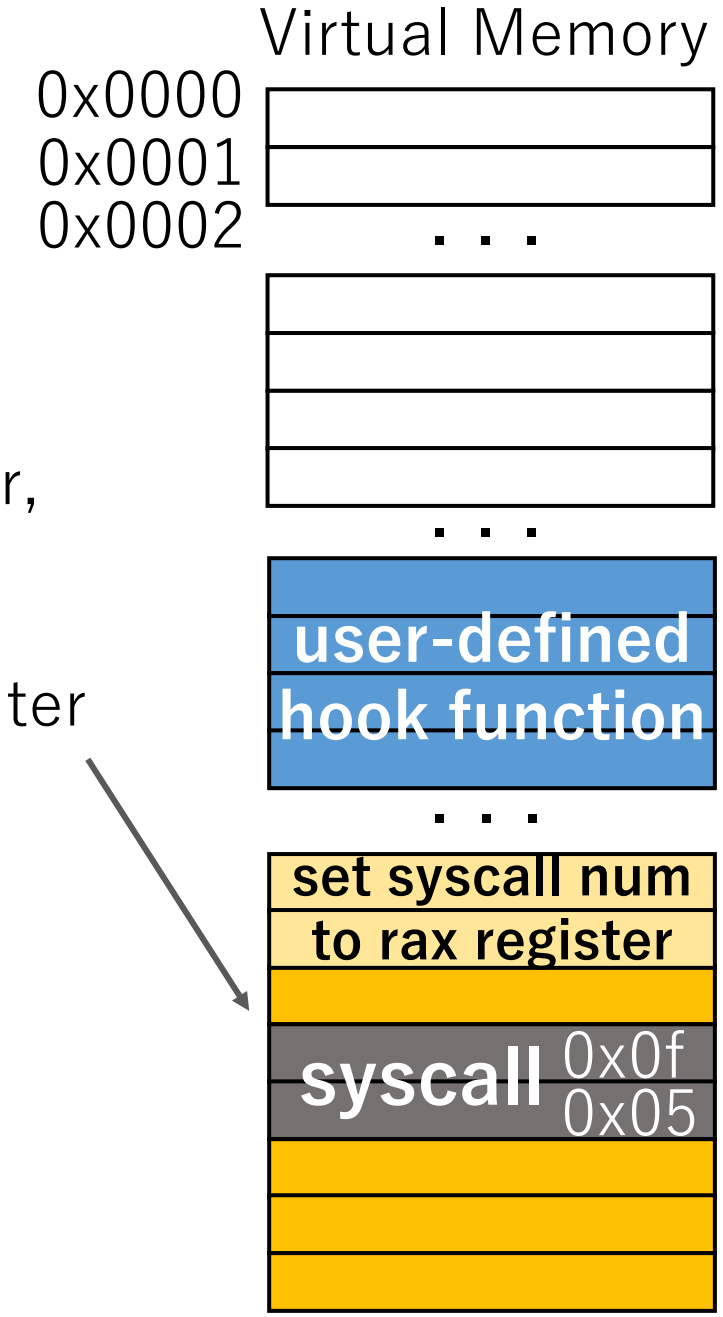
- How to invoke a system call
  - A user-space program sets a system call number, predefined by the kenel, to the **rax register**
    - e.g., 0: read(), 1: write(), 2: open(), ...



https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

Calling Convention

- 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

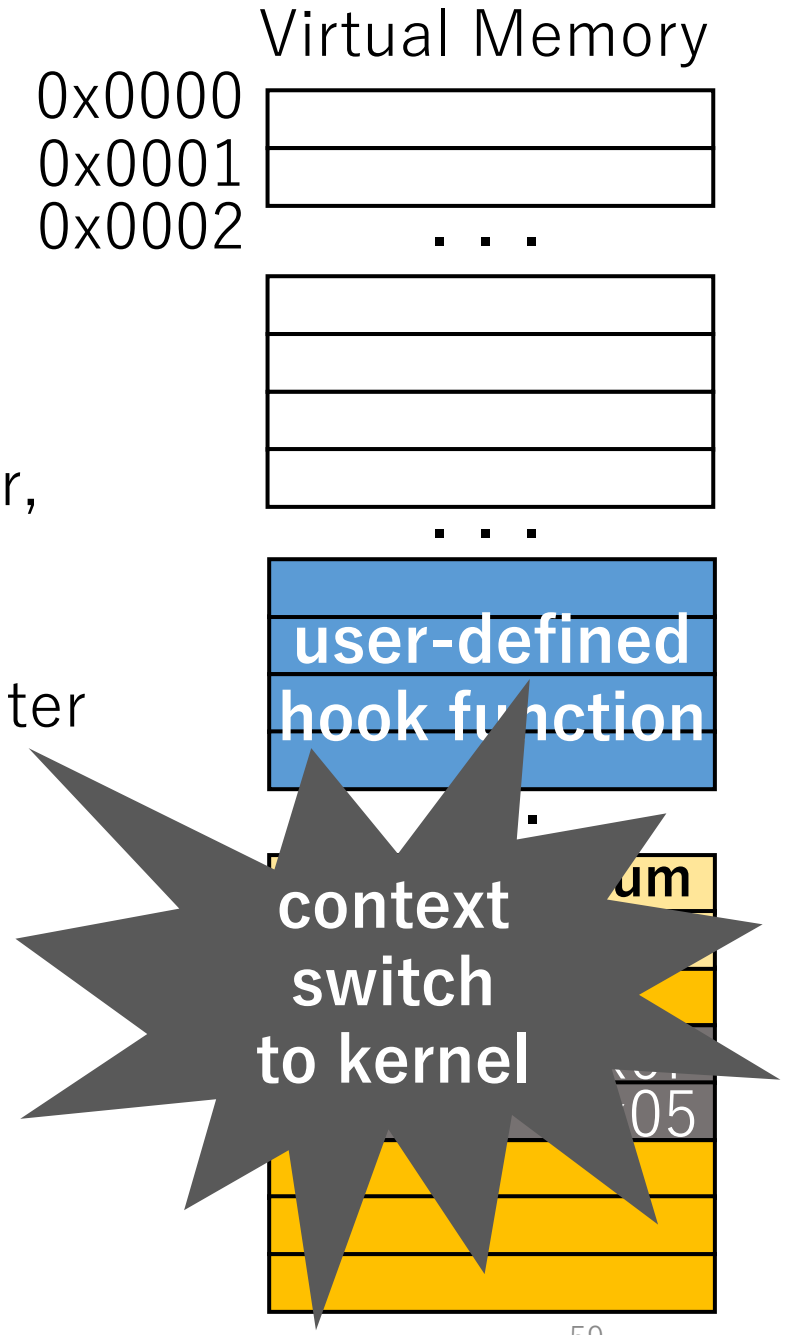


https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

Calling Convention

- 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 ----

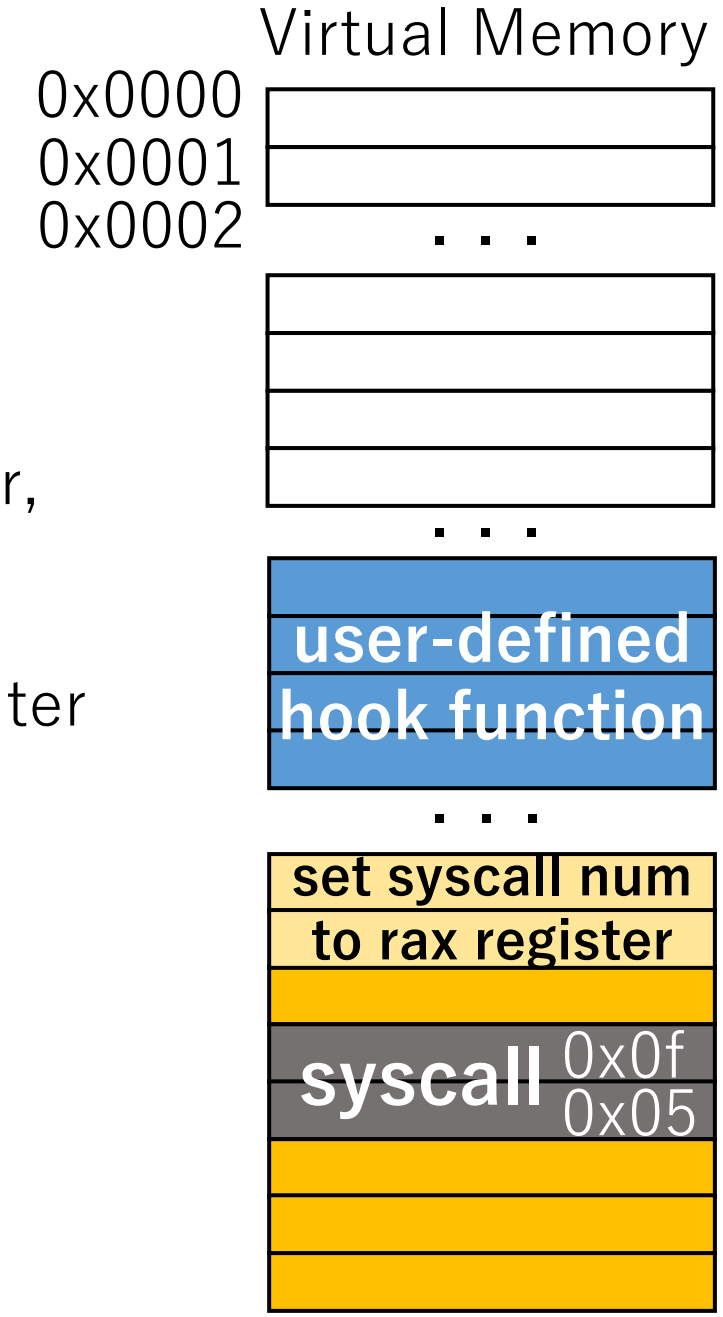


https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

Calling Convention

- 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**

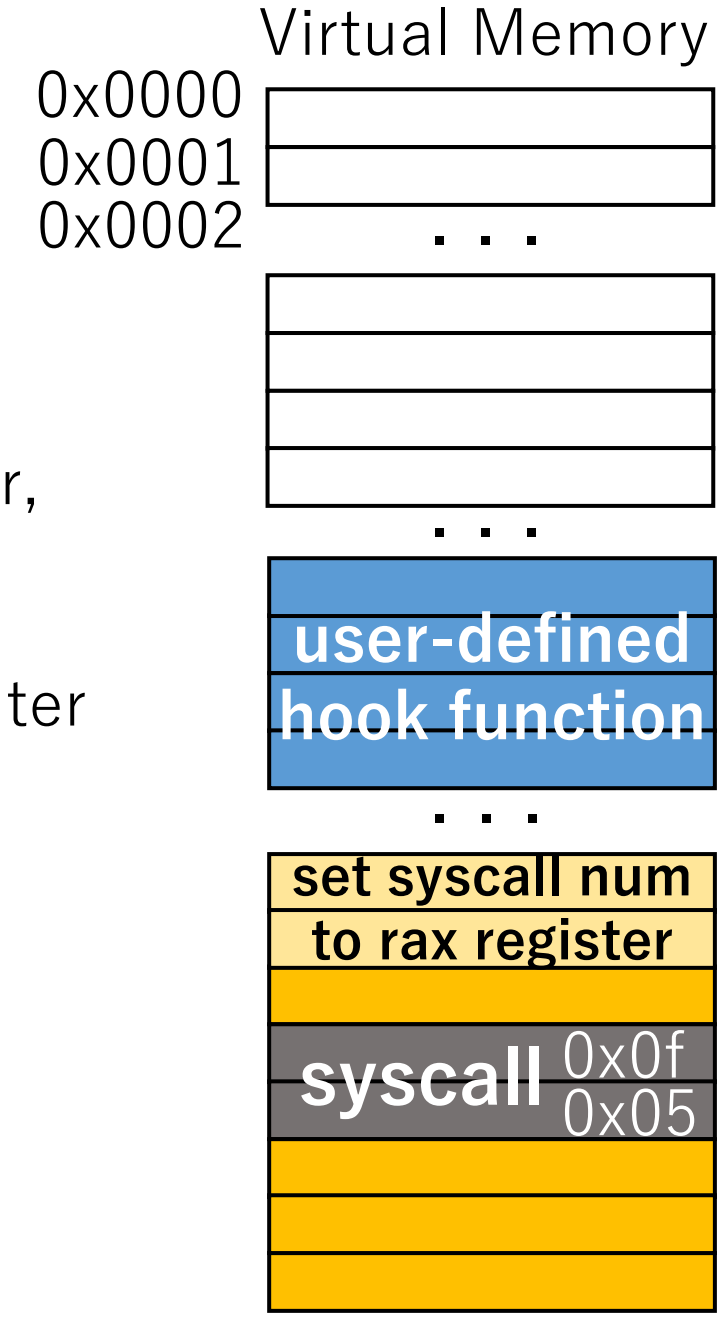


https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

Calling Convention

- 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()



https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

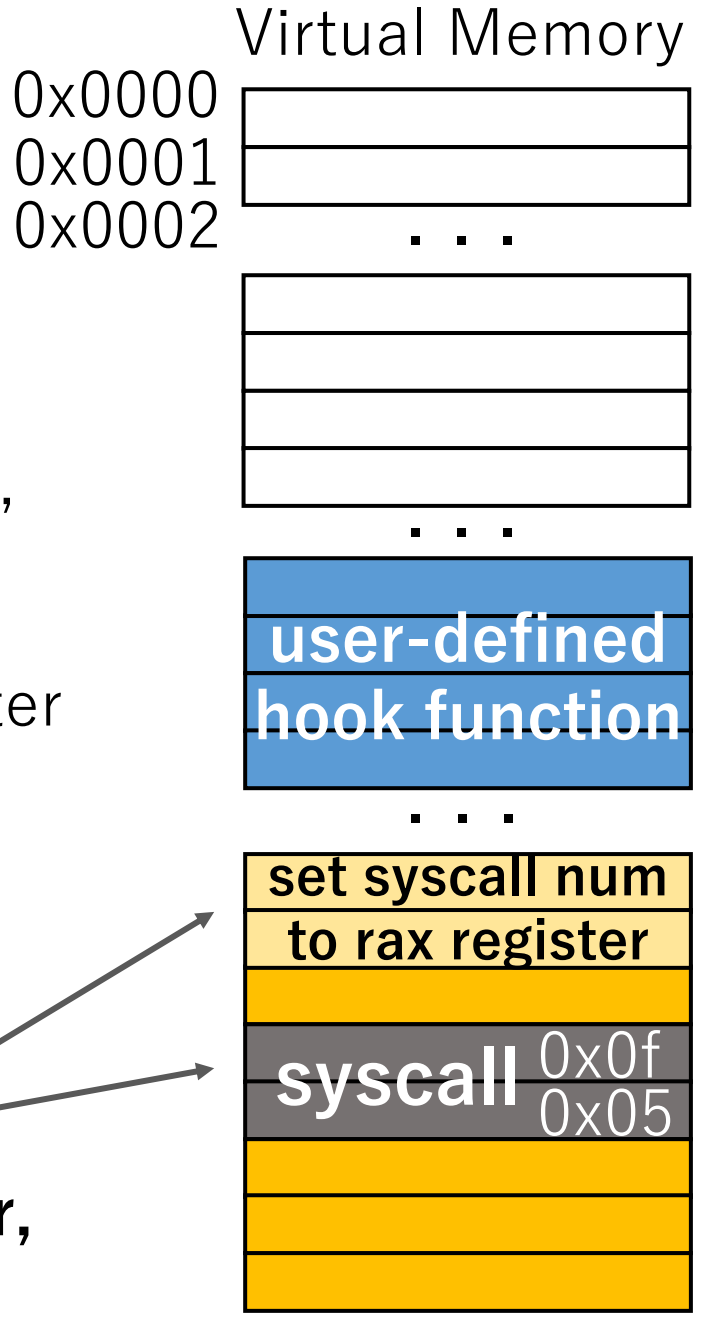
Calling Convention

- 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,



https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

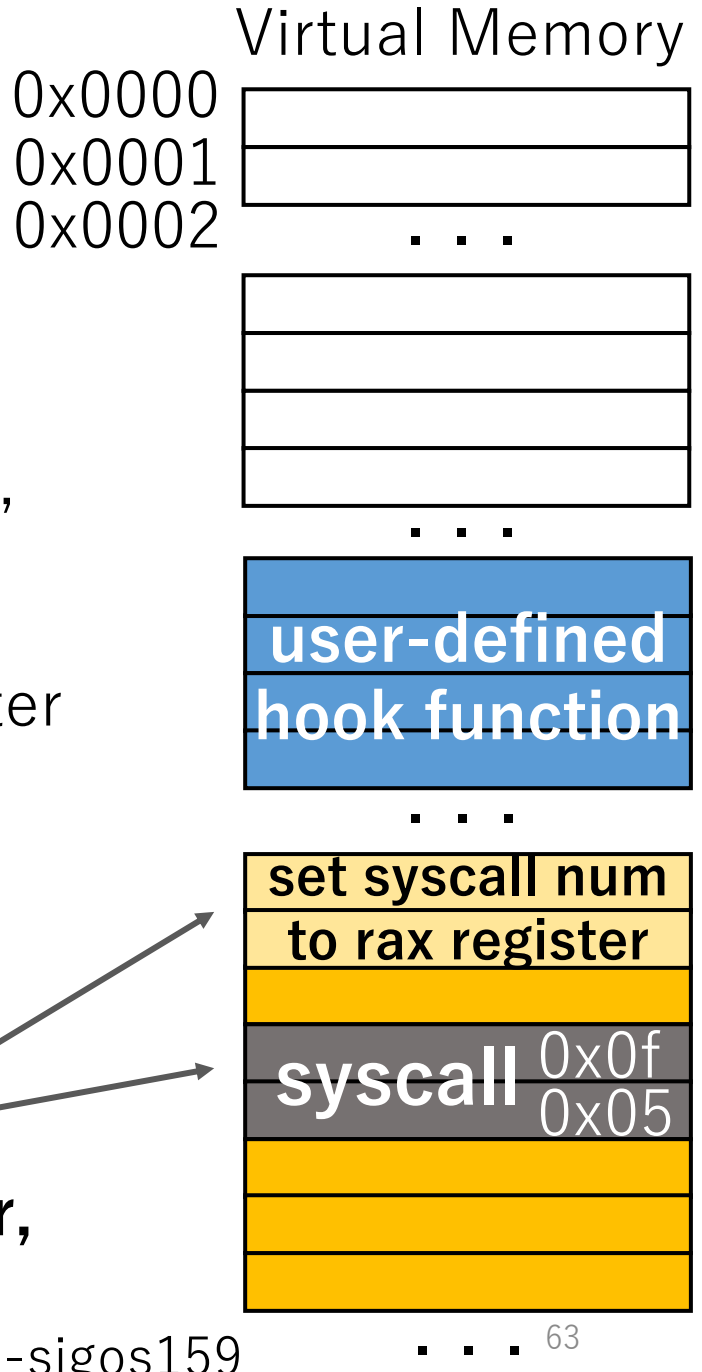
Calling Convention

- 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

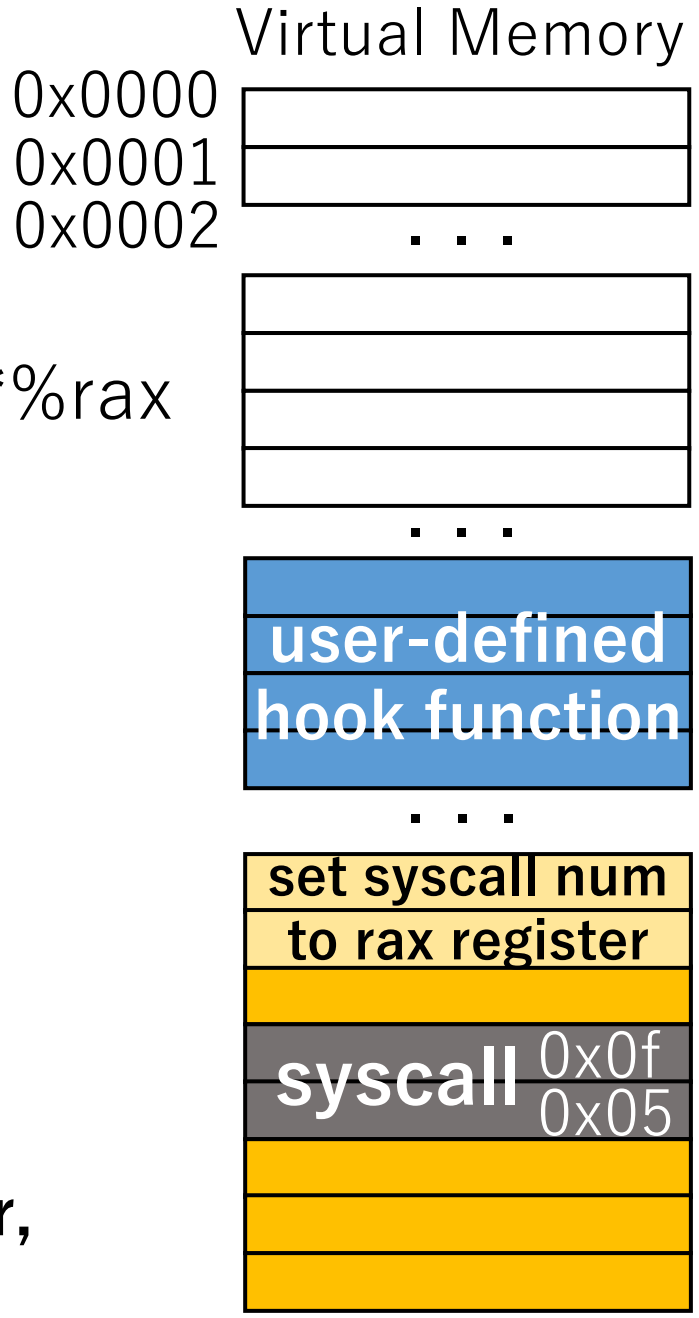
When syscall/sysenter is executed,
the rax register always has a system call number,
which is 0 ~ around 500 (defined in the kernel)



https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

zpoline

- zpoline replaces syscall/sysenter with callq *%rax



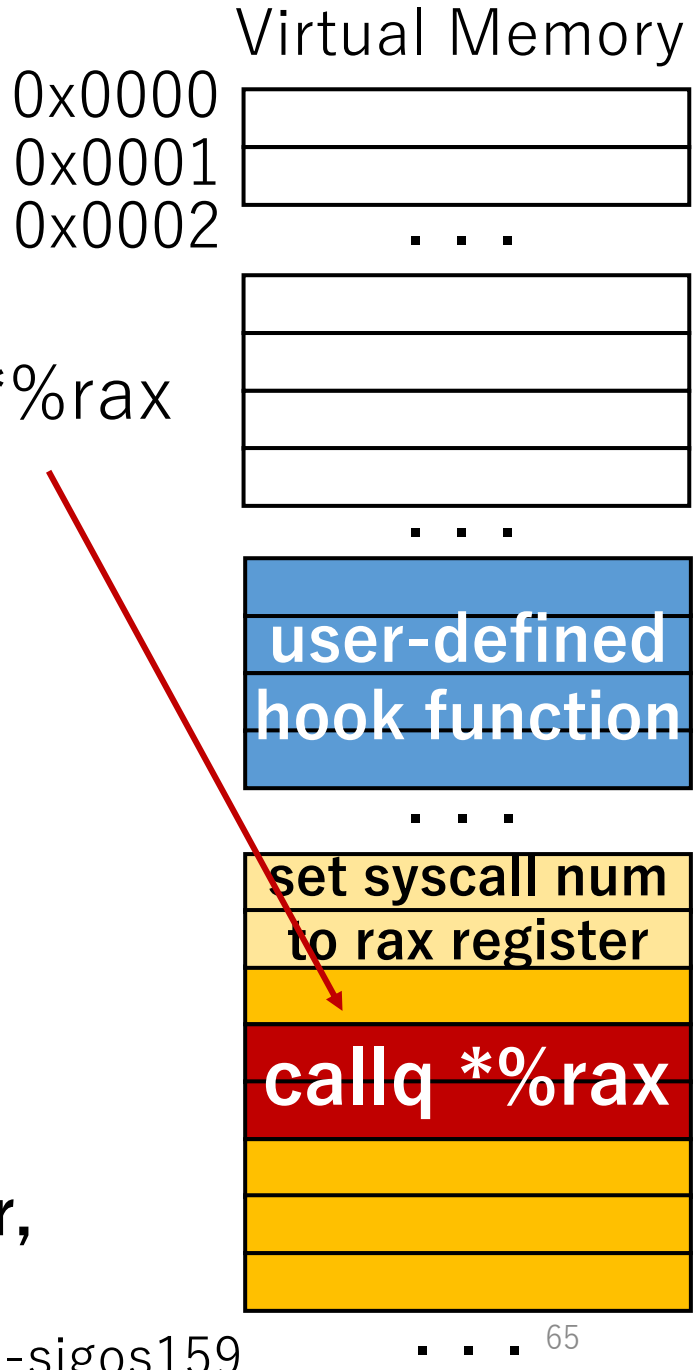
Point: Calling Convention

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

https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

zpoline

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



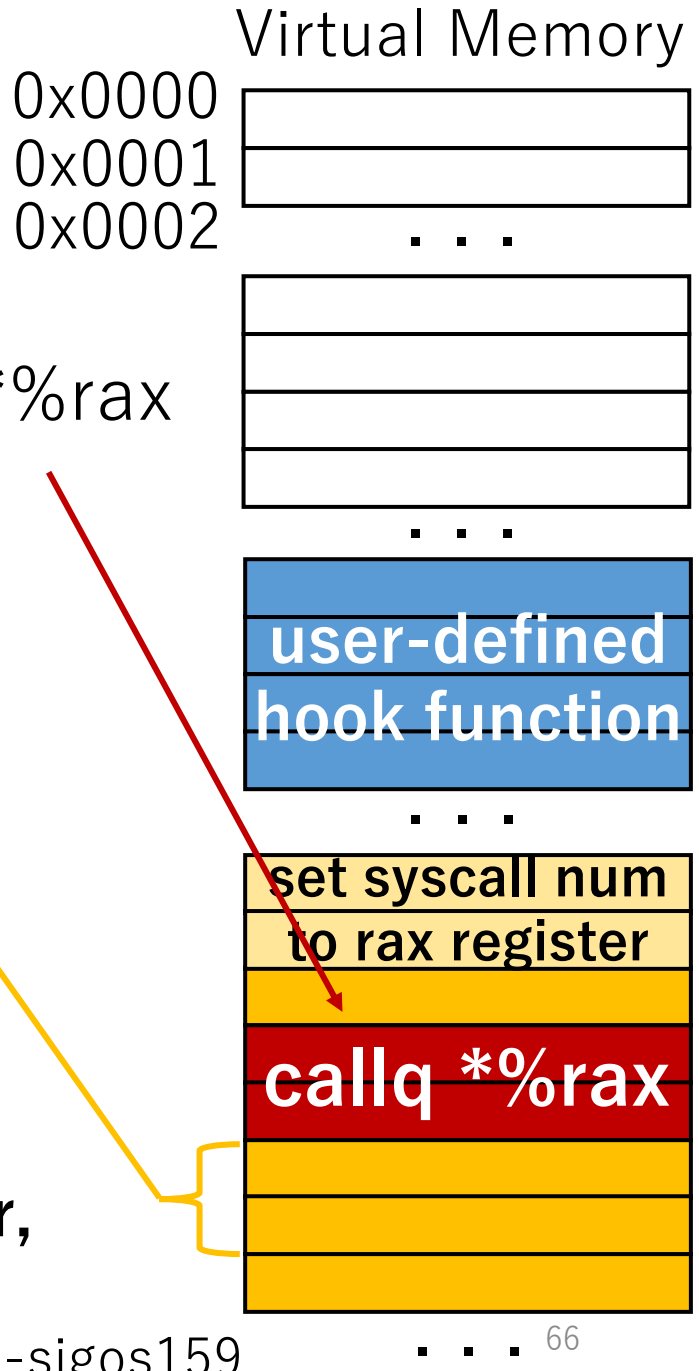
Point: Calling Convention

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/2023/05/ipsj-sigos159

zpoline

- zpoline replaces syscall/sysenter with callq *%rax
  - callq *%rax is a **2-byte** instruction (0xff 0xd0)
    - Neighbour instructions are not overwritten



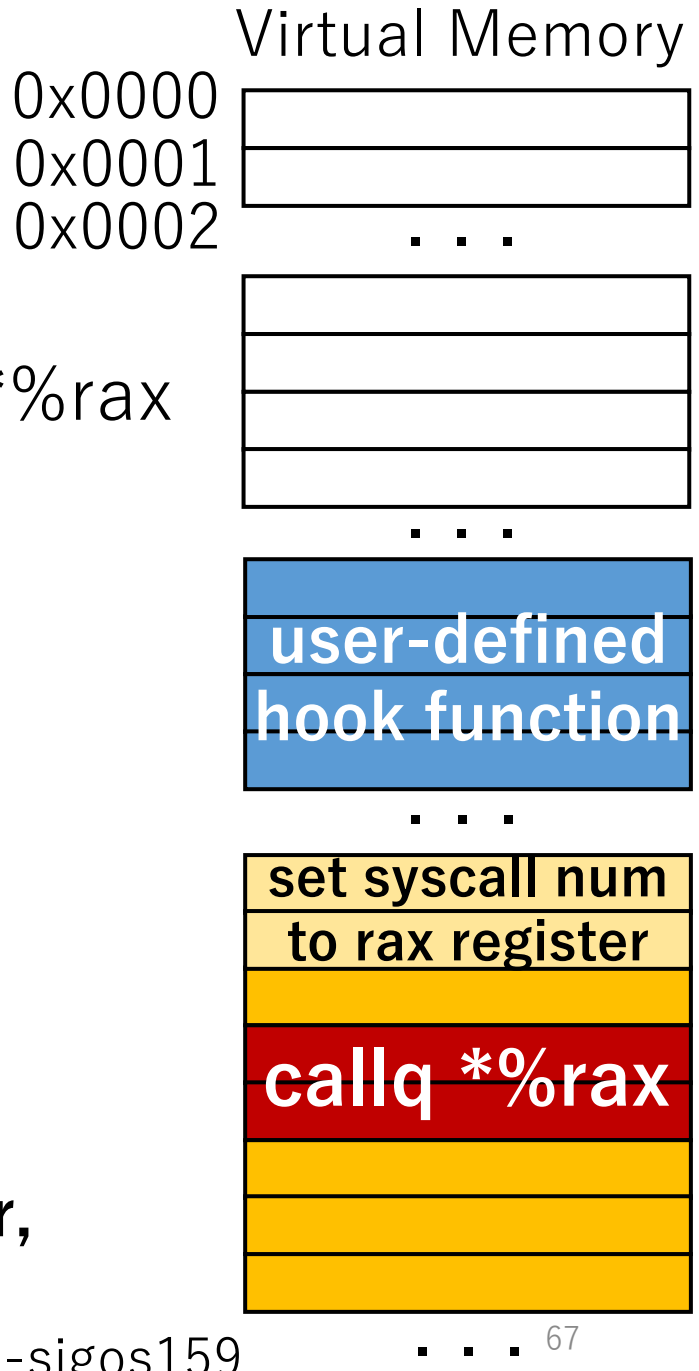
Point: Calling Convention

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/2023/05/ipsj-sigos159

zpoline

- 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



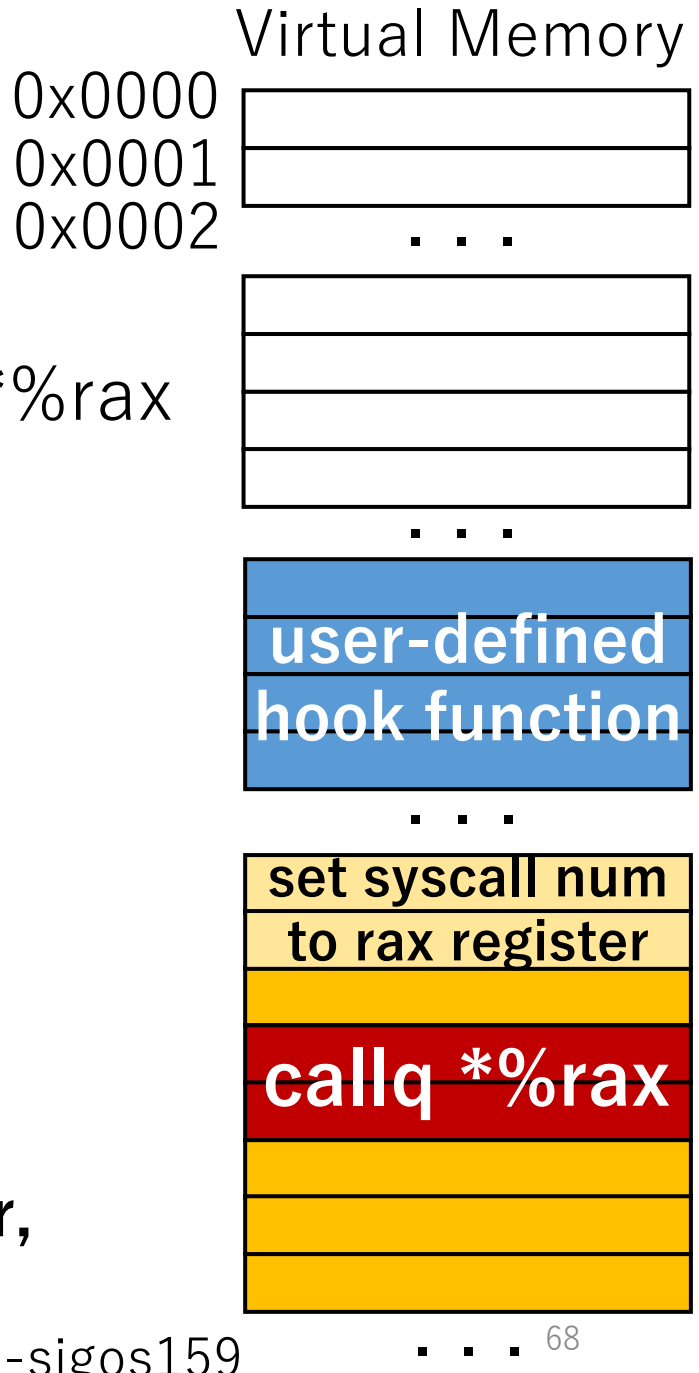
Point: Calling Convention

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/2023/05/ipsj-sigos159

zpoline

- 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



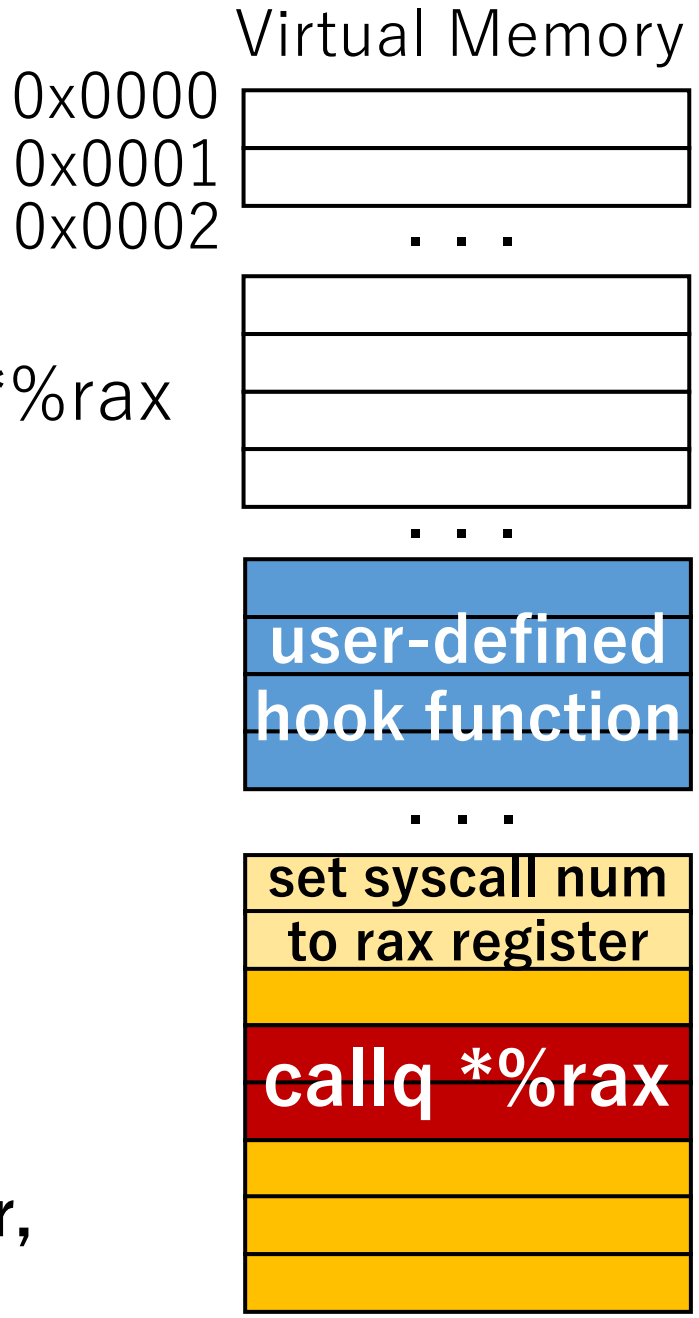
Point: Calling Convention

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/2023/05/ipsj-sigos159

zpoline

- 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



After the binary rewriting

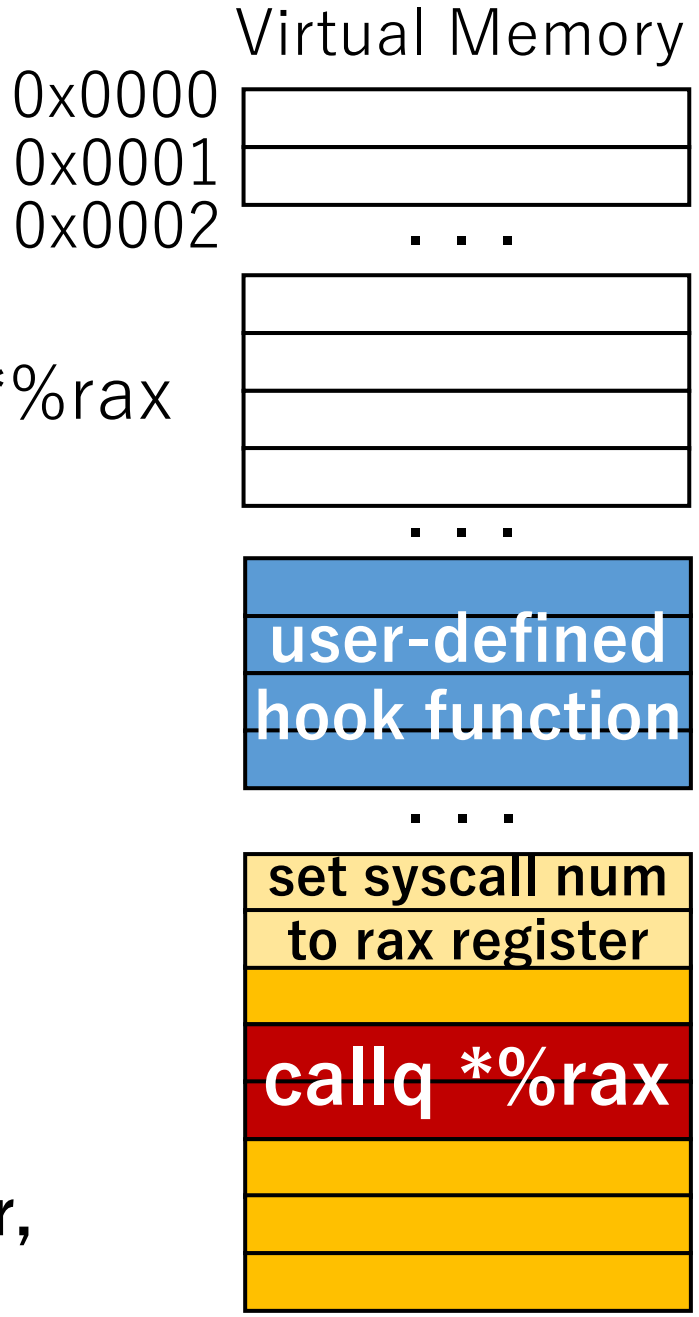
Point: Calling Convention

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

https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

zpoline

- 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



After the binary rewriting

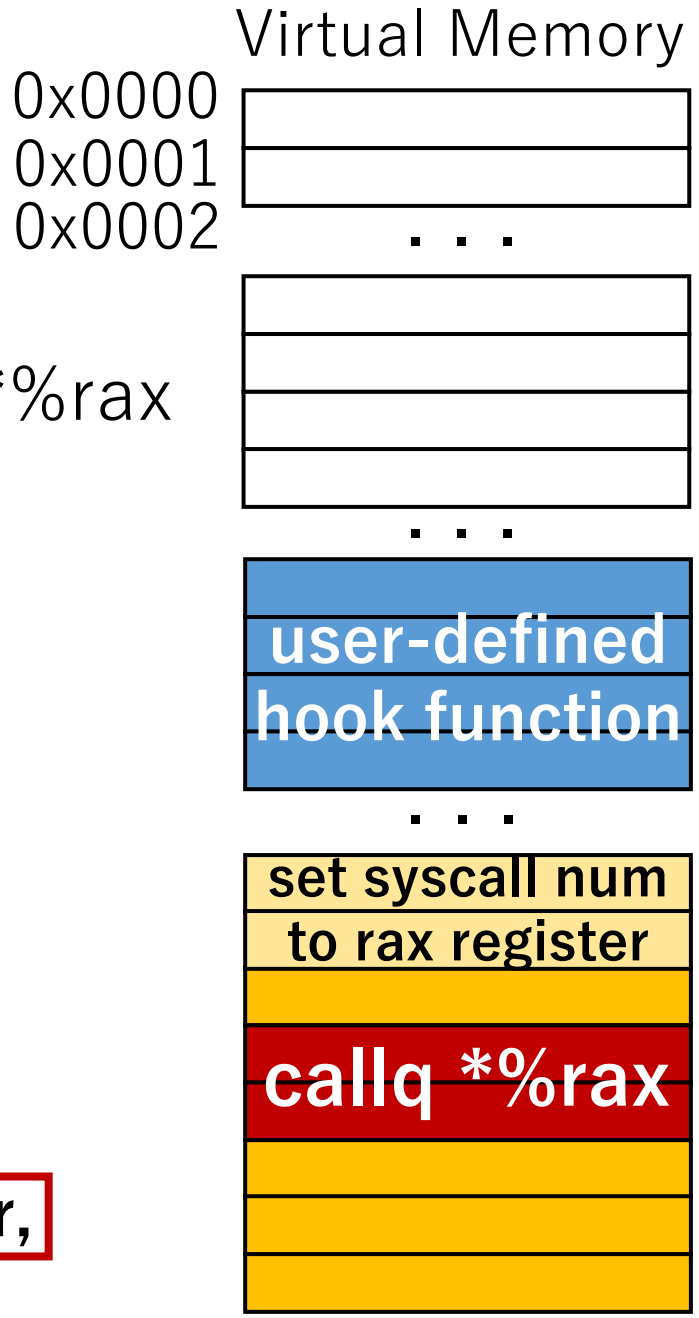
Point: Calling Convention

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)

https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

zpoline

- 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



After the binary rewriting

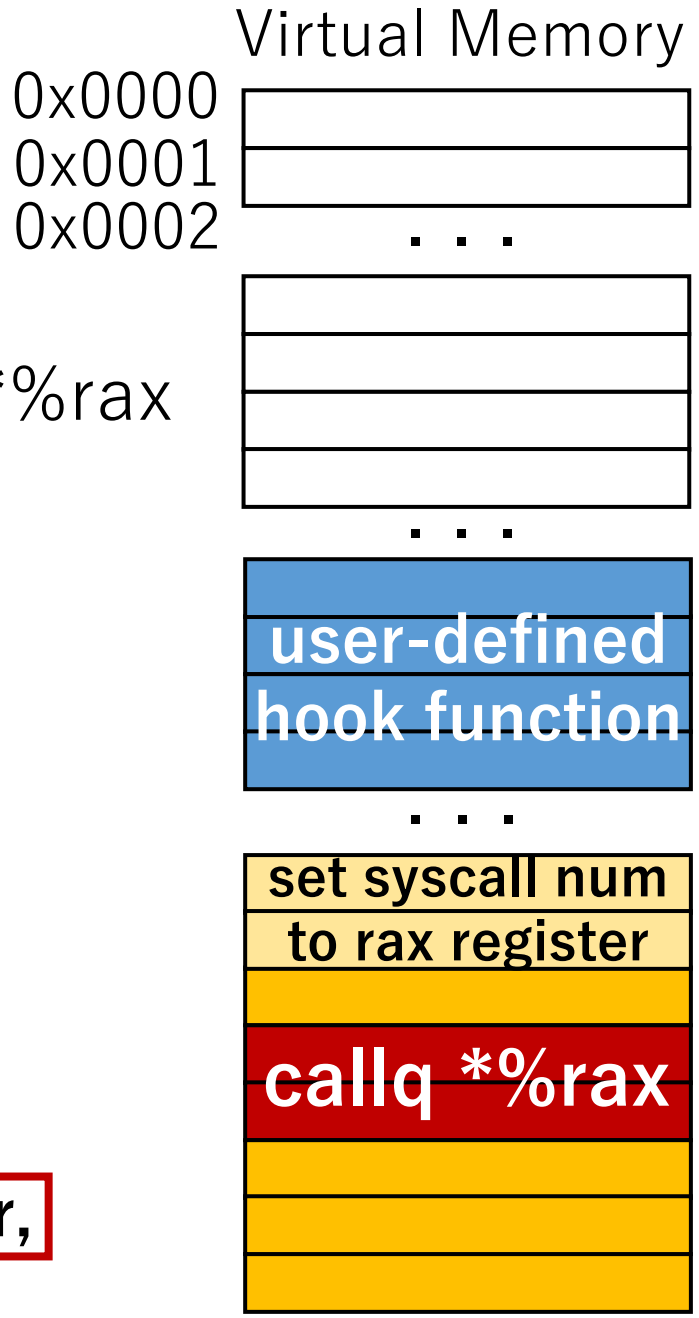
Point: Calling Convention

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)

https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

zpoline

- 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



After the binary rewriting

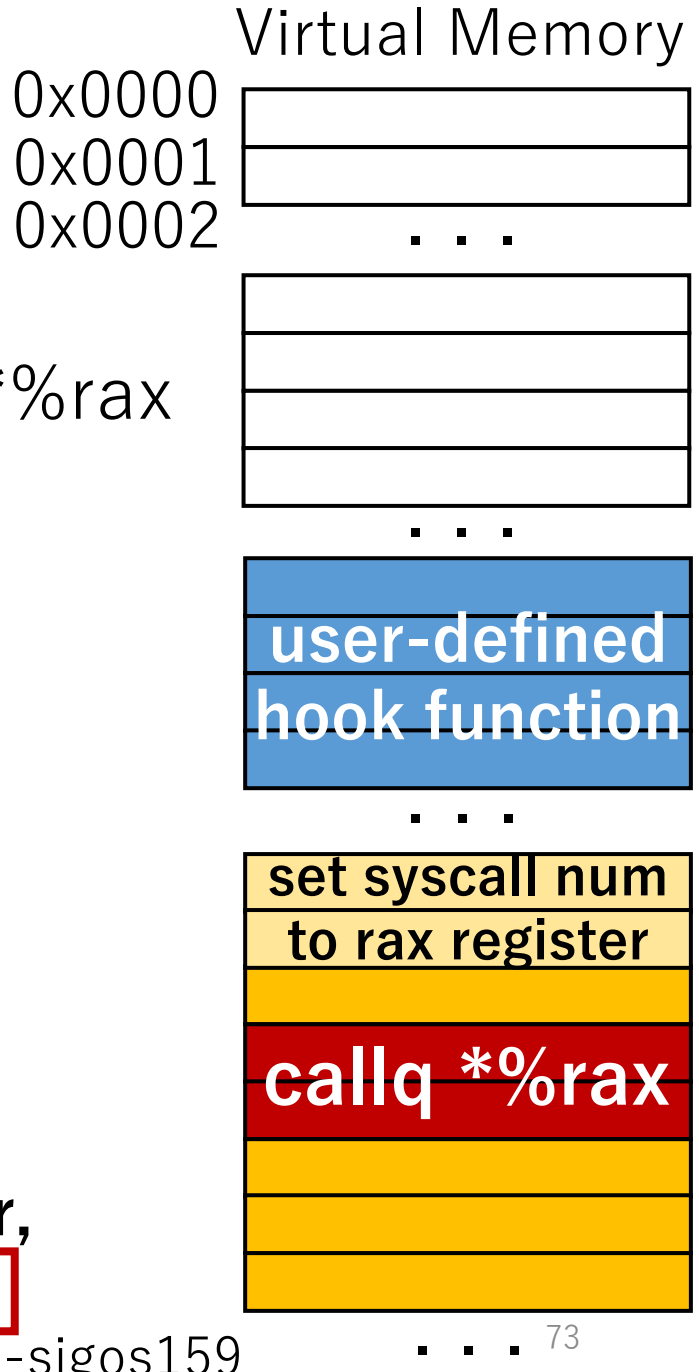
Point: Calling Convention

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)

https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

zpoline

- 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



After the binary rewriting

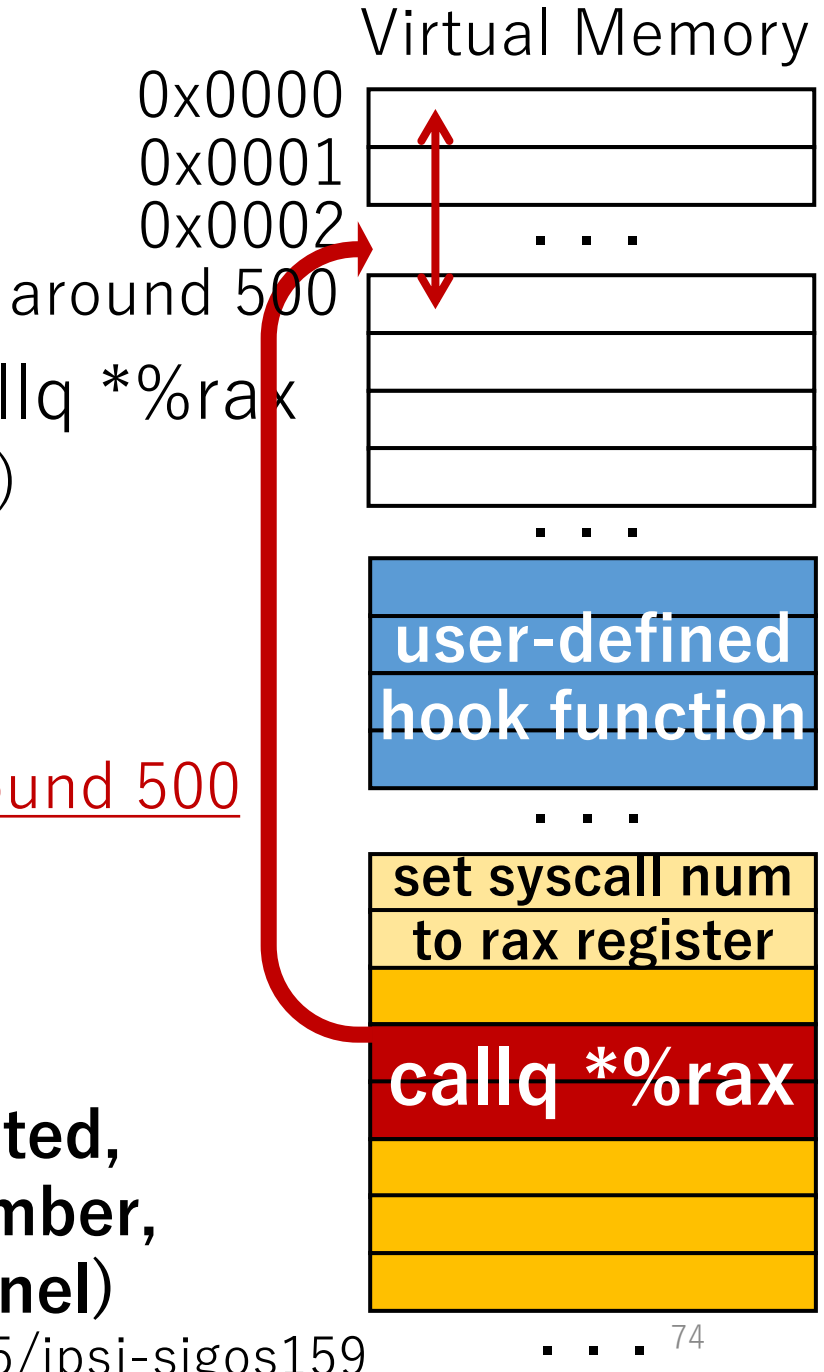
Point: Calling Convention

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)

https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

zpoline

- 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
  - 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)

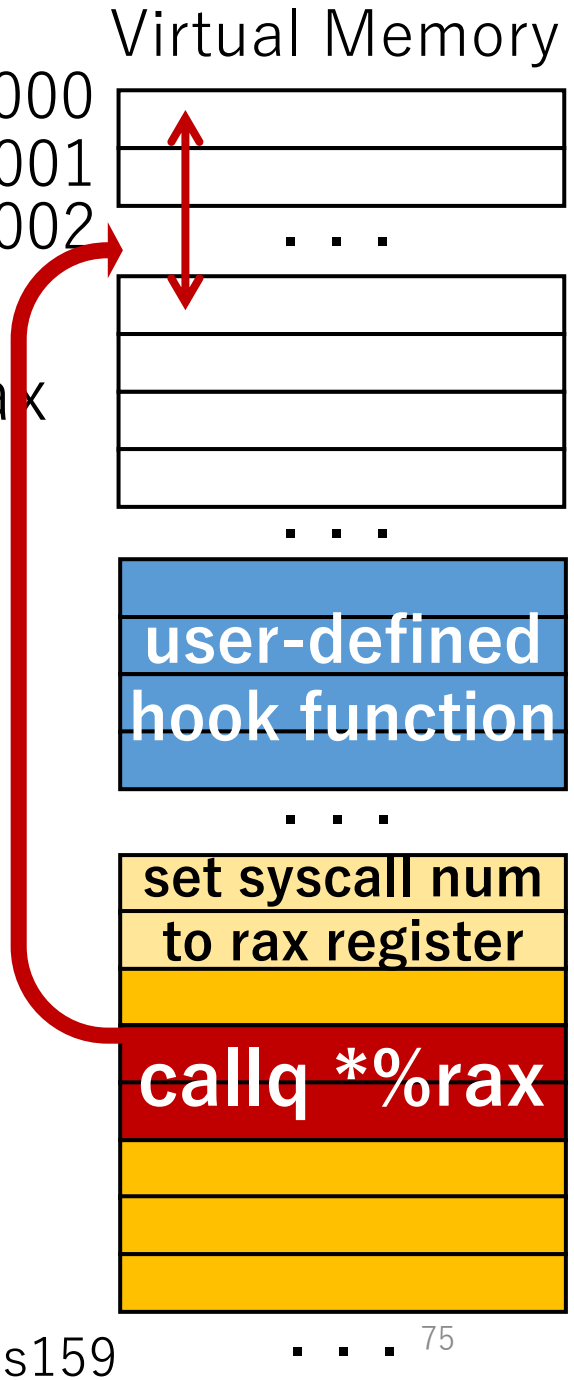
https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

zpoline

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

0x0000
0x0001
0x0002
N

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



After the binary rewriting

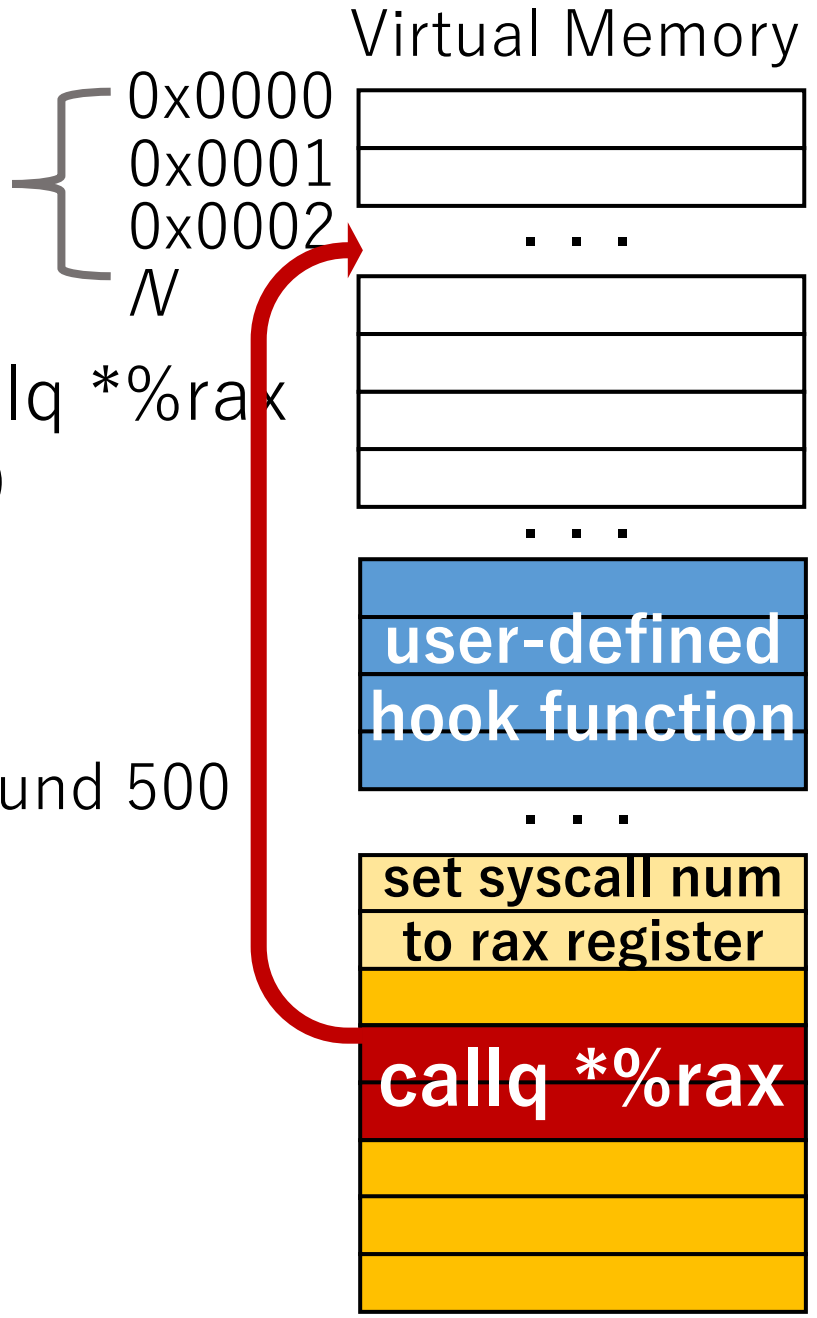
Point: Calling Convention

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)

https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

zpoline

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



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

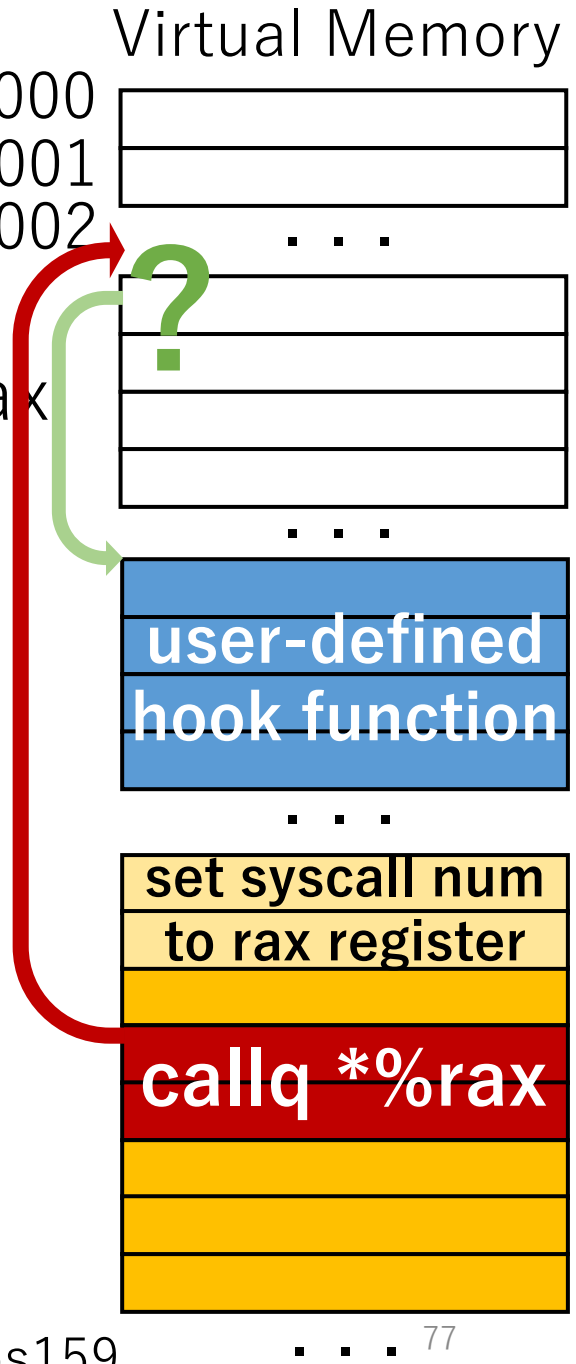
https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

zpoline

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

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

How to redirect to the user-defined hook function?



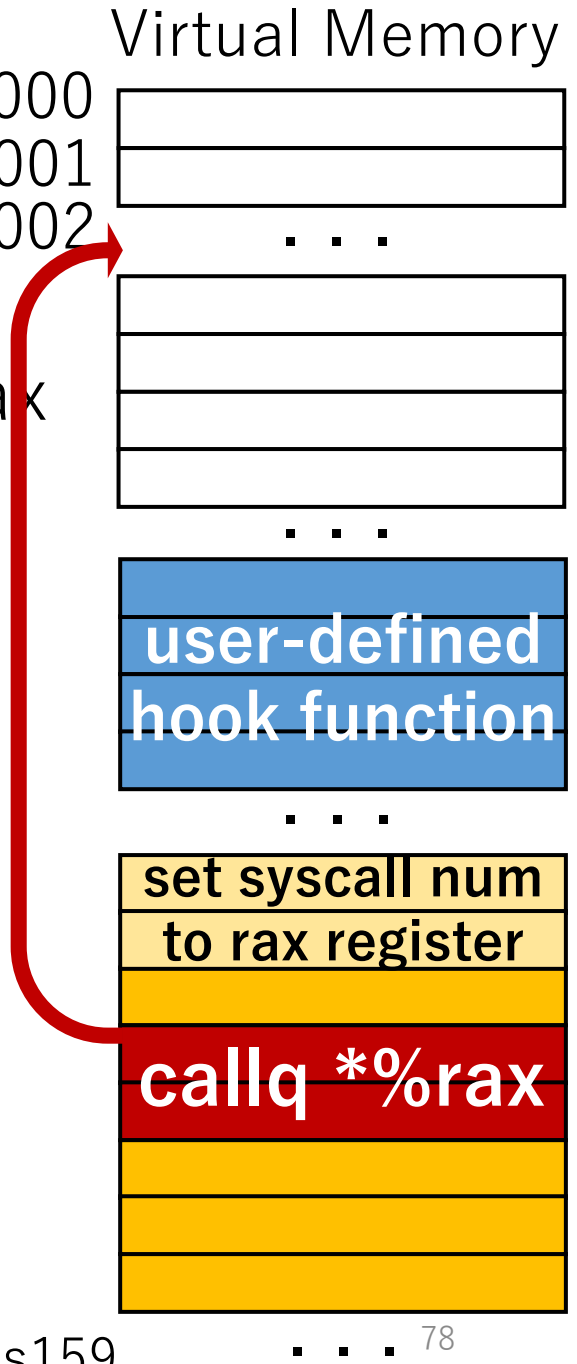
https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

zpoline

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

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

How to redirect to the user-defined hook function?



https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

zpoline

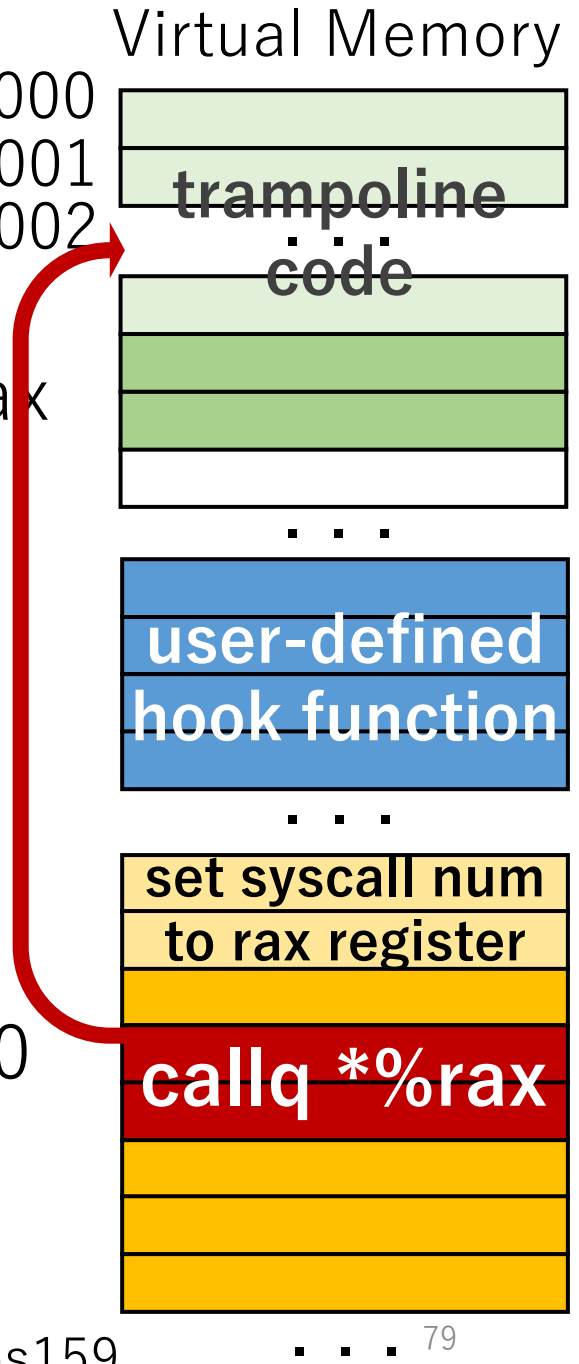
address range, potentially replaced "callq *%rax" jumps to
(*N* is the max syscall number)

0x0000
0x0001
0x0002
N

- 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
  - 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

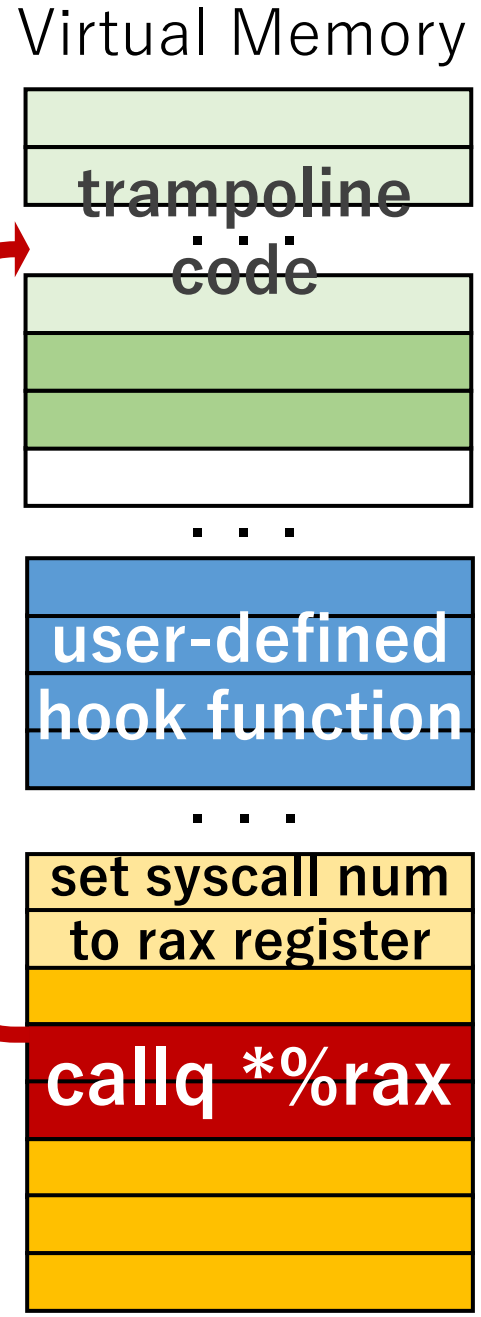


https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

zpoline

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

0x0000
0x0001
0x0002
N



- 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
  - 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

Tram**poline** code at address 0 (**z**ero) ➔ **zpoline**

https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

zpoline

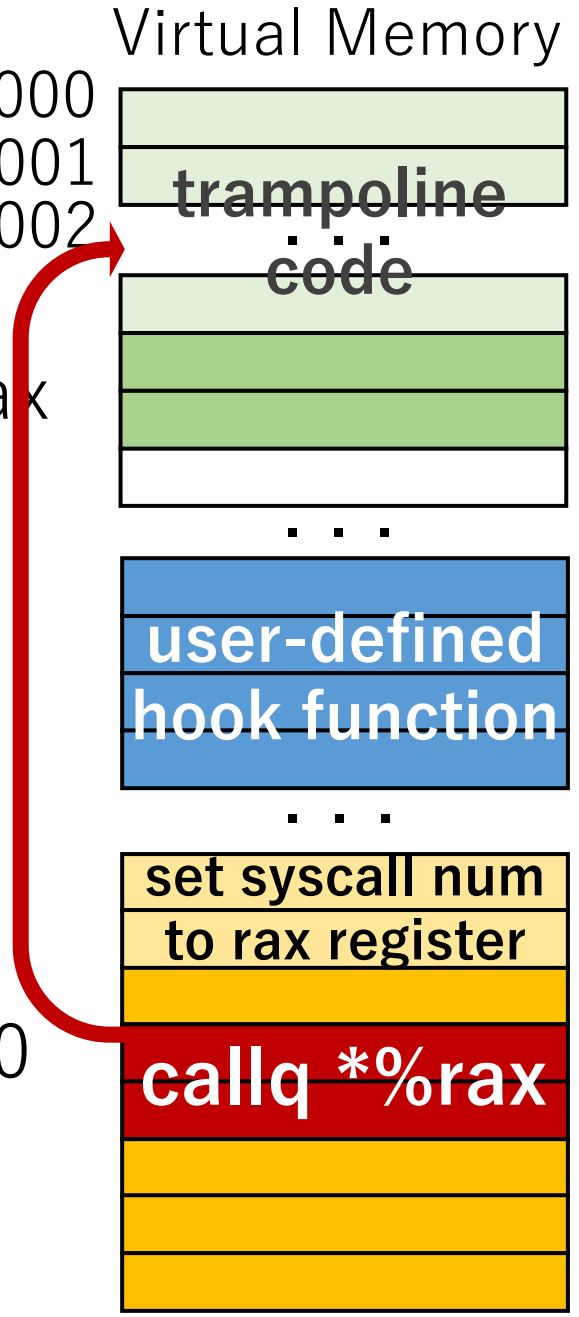
address range, potentially replaced "callq *%rax" jumps to
(*N* is the max syscall number)

0x0000
0x0001
0x0002
N

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

How to redirect to the user-defined hook function?

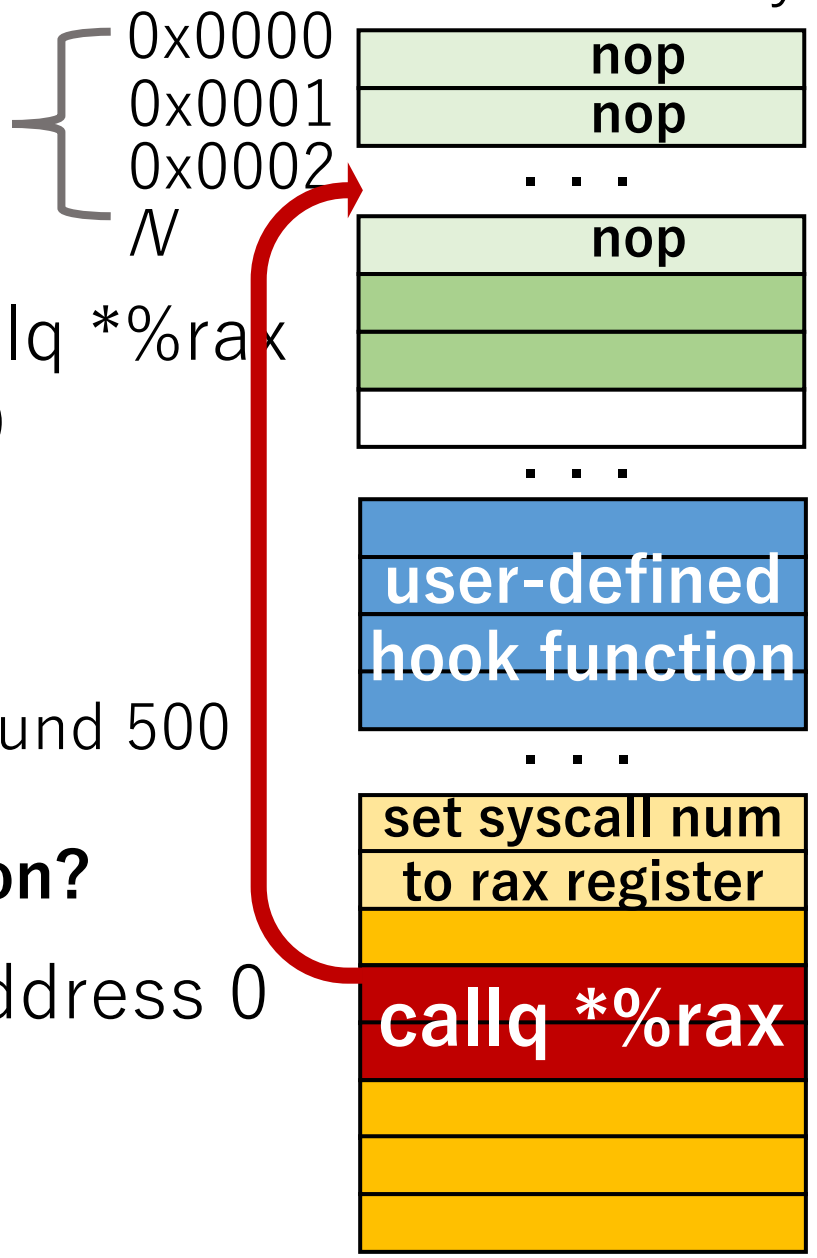
- zpoline instantiates trampoline code at address 0



https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

zpoline

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



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

How to redirect to the user-defined hook function?

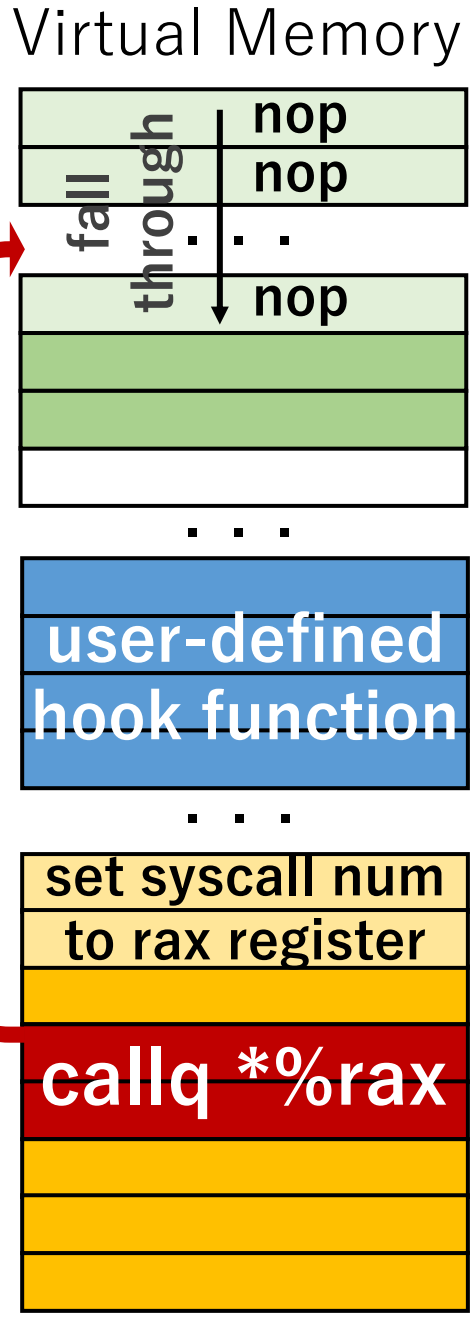
- zpoline instantiates trampoline code at address 0
  - fills address range 0 to *N* with nop (0x90)

https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

zpoline

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

0x0000
0x0001
0x0002
N



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

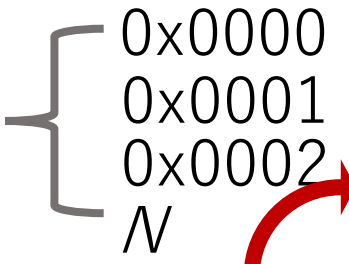
How to redirect to the user-defined hook function?

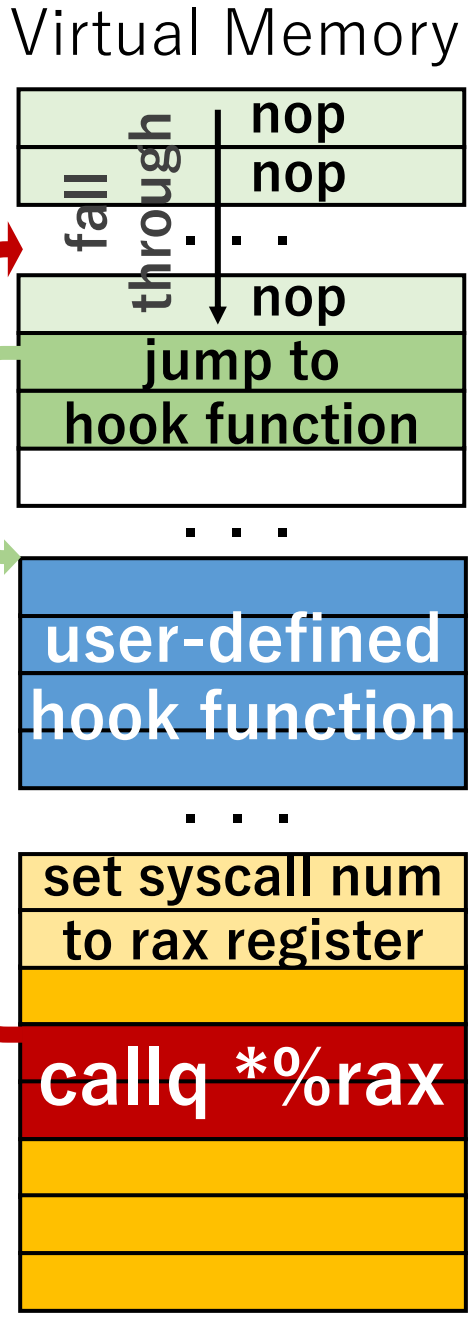
- zpoline instantiates trampoline code at address 0
  - fills address range 0 to *N* with nop (0x90)

https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

zpoline

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





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

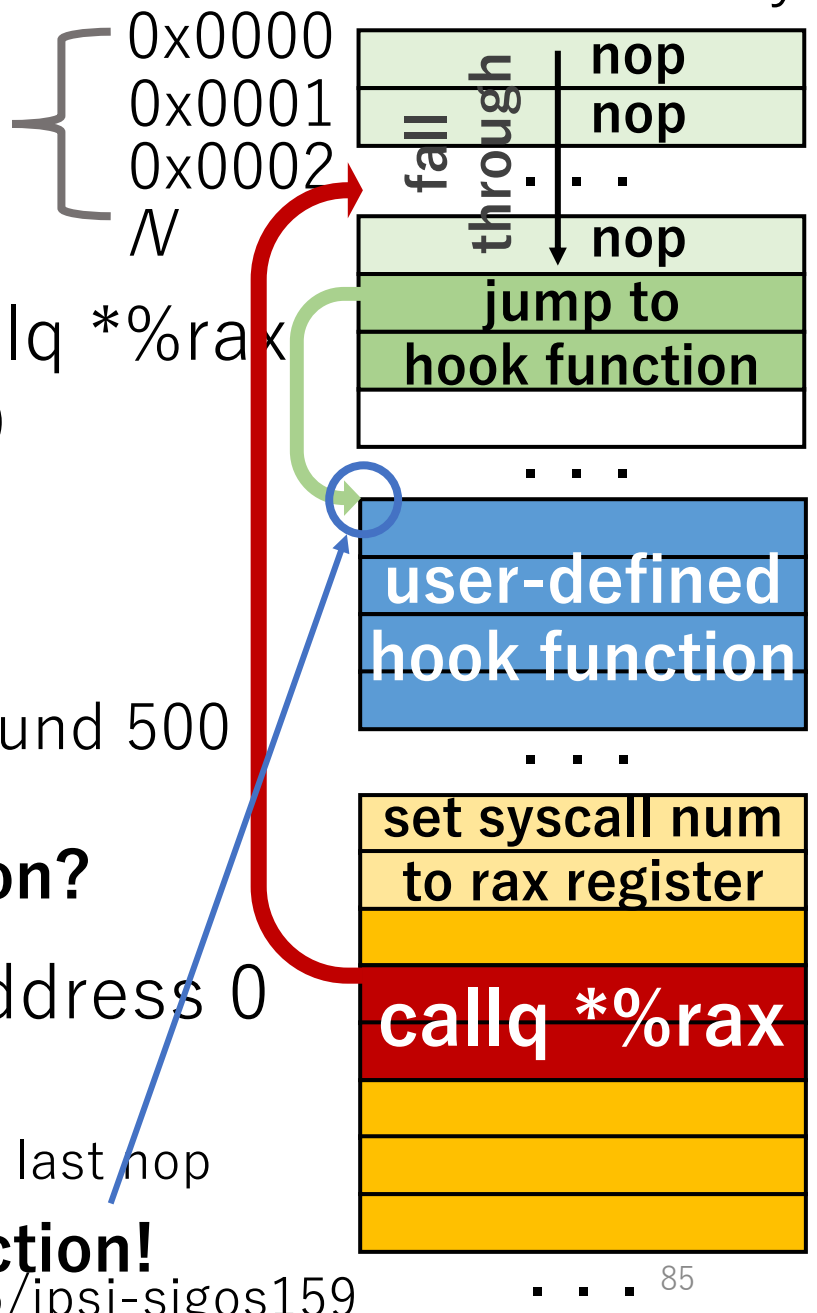
How to redirect to the user-defined hook function?

- 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

https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

zpoline

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



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

How to redirect to the user-defined hook function?

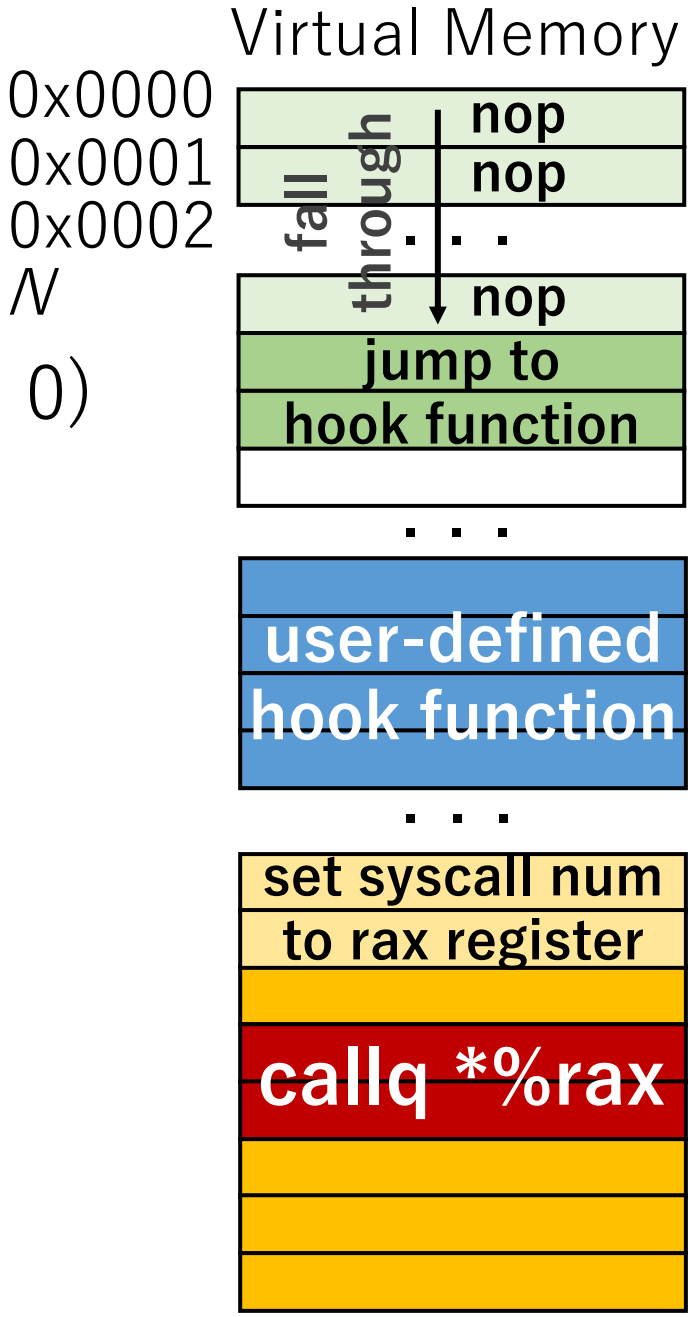
- zpoline instantiates trampoline code at address 0
  - fills address range 0 to *N* with nop (0x90)
    - puts code to jump to the hook function next to the last nop

We could reach the user-defined hook function!

https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

NULL Access Termination

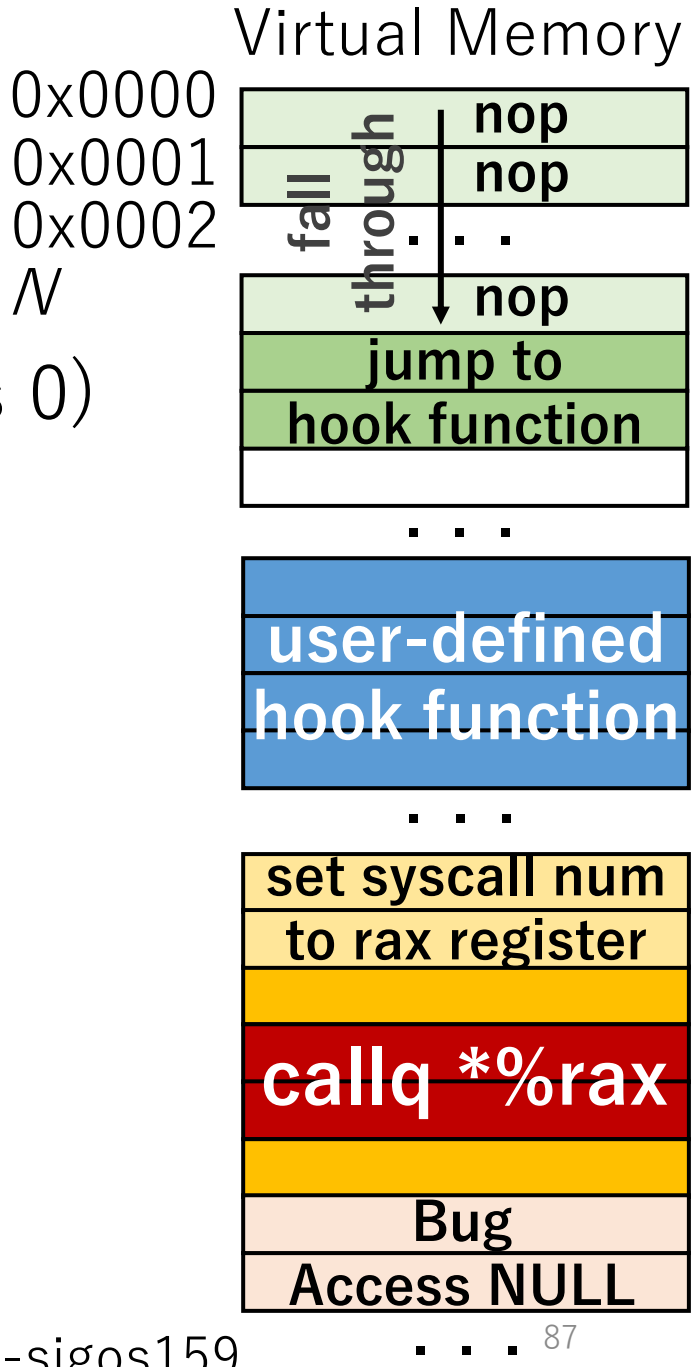
- A buggy program may access NULL (address 0)



https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

NULL Access Termination

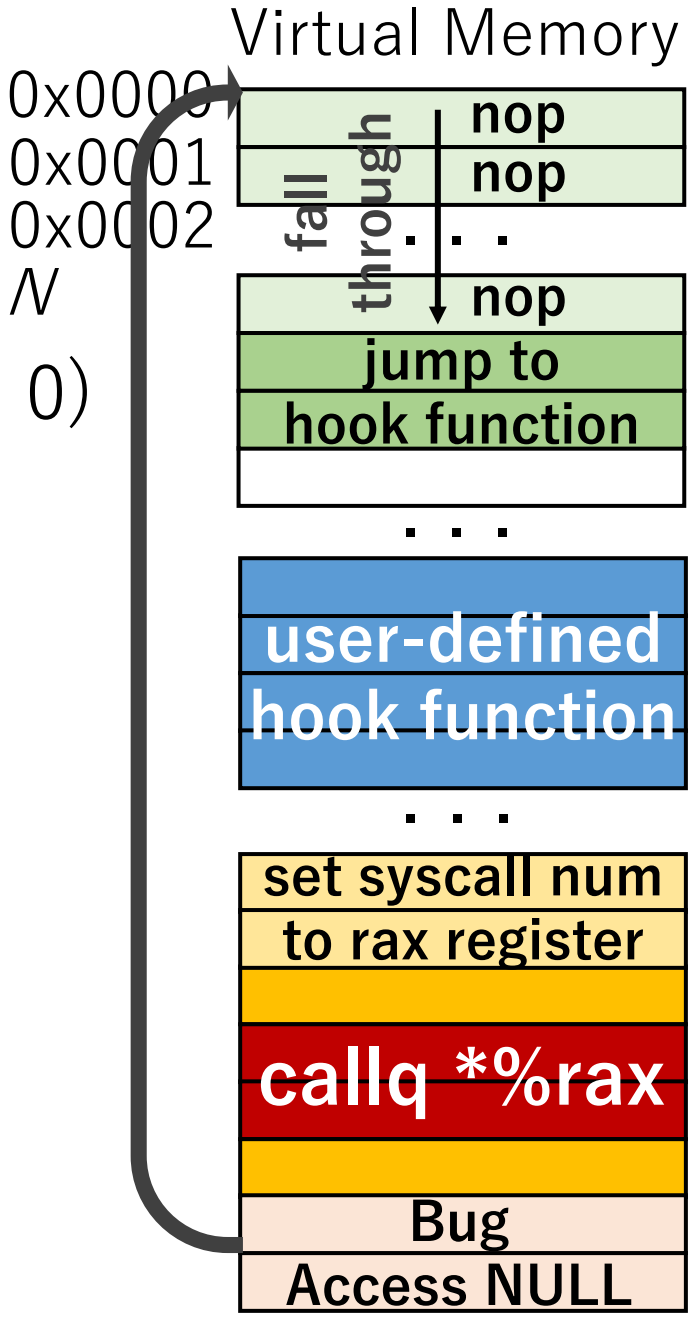
- A buggy program may access NULL (address 0)



https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

NULL Access Termination

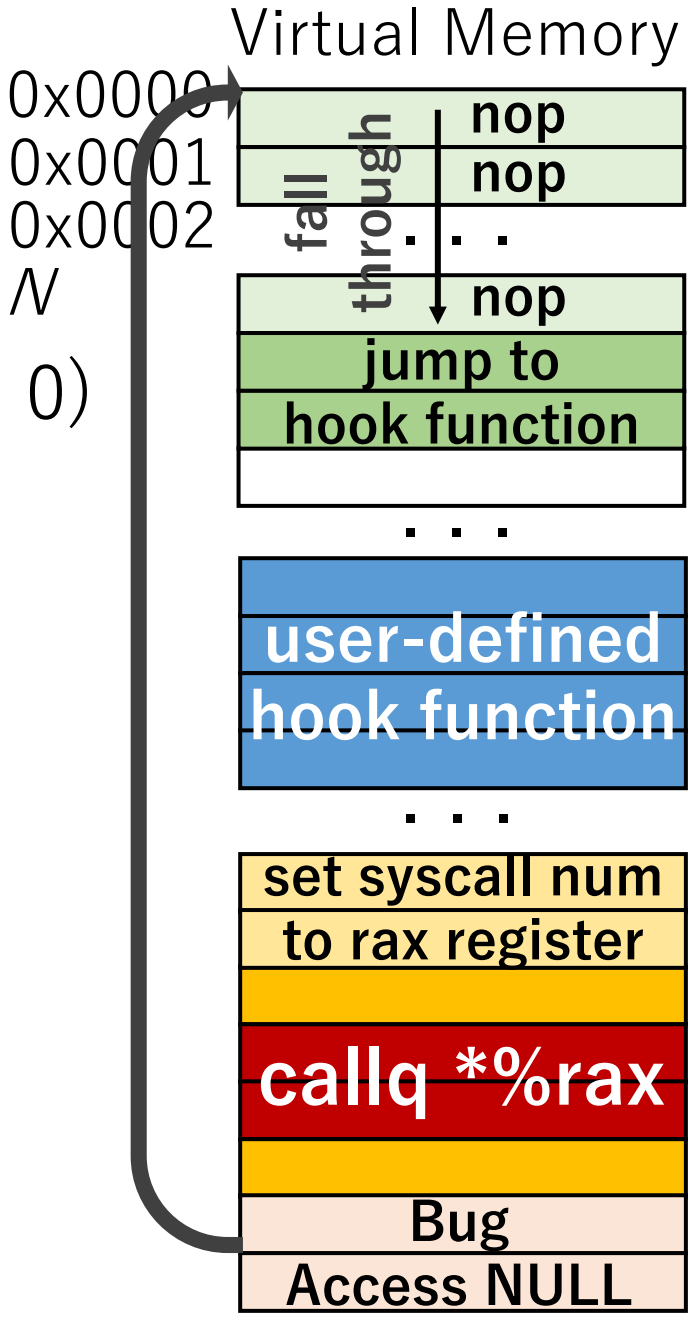
- A buggy program may access NULL (address 0)



https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

NULL Access Termination

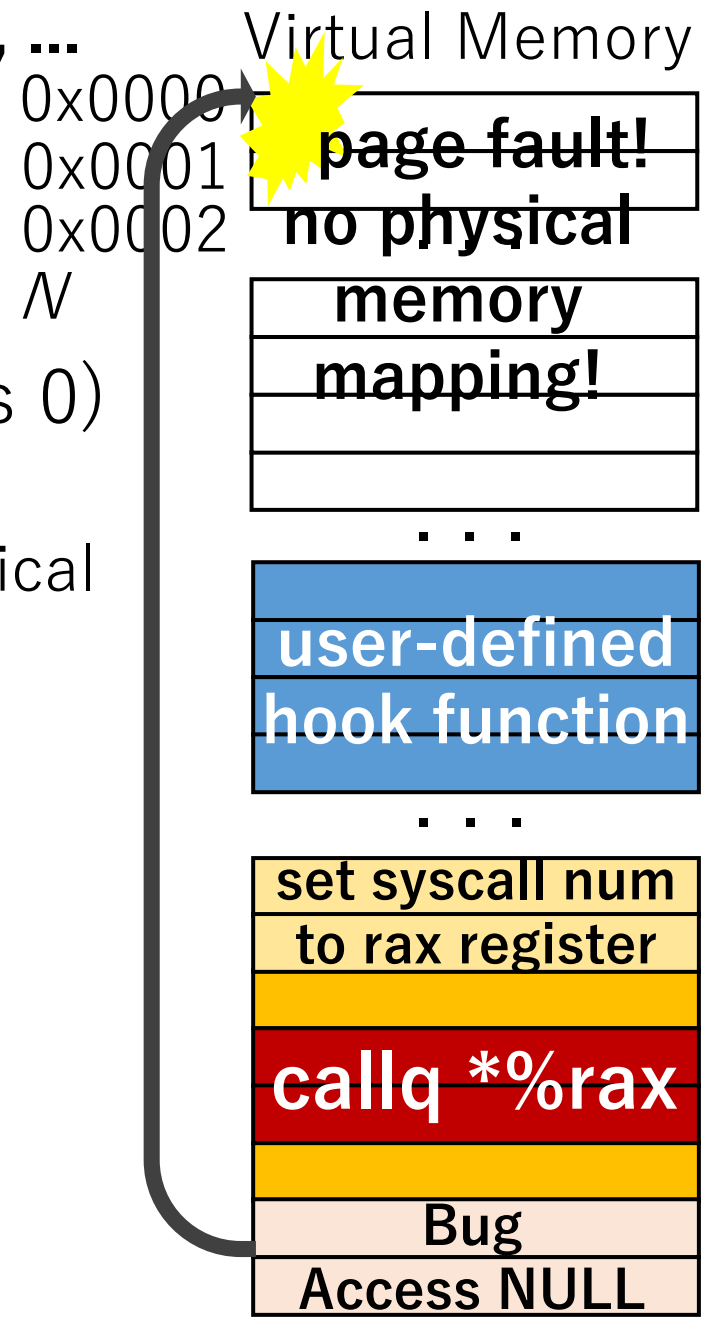
- A buggy program may access NULL (address 0)
  - In principle, NULL access has to be terminated



https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

NULL Access Termination

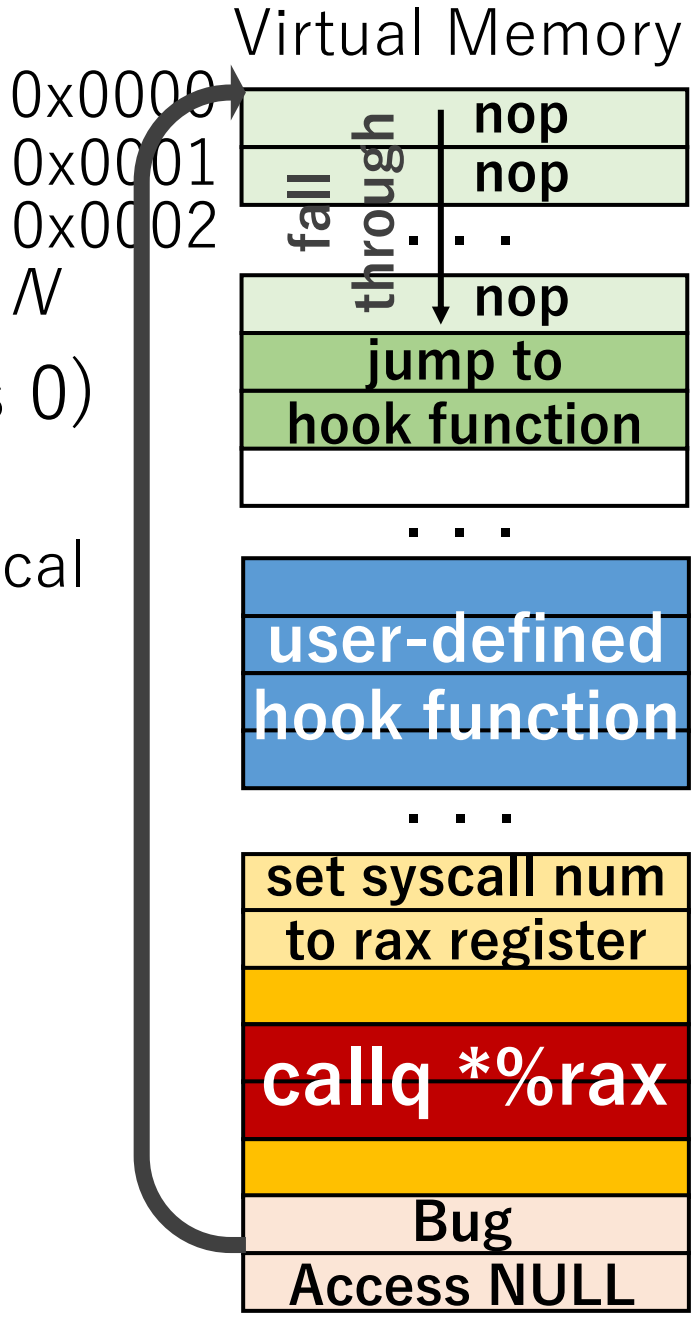
- A buggy program may access NULL (address 0)
  - In principle, NULL access has to be terminated
  - Normally, a page fault happens because no physical memory is mapped to virtual address 0



https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

NULL Access Termination

- 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



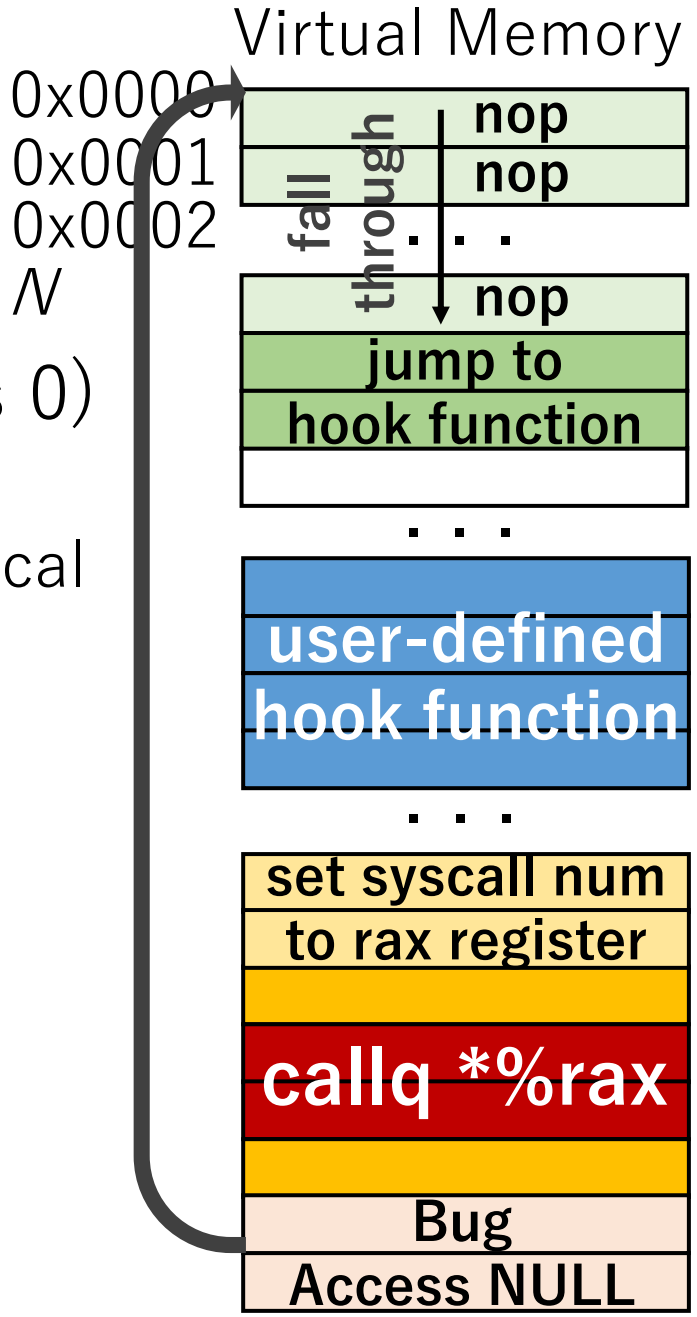
https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

NULL Access Termination

- 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



https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

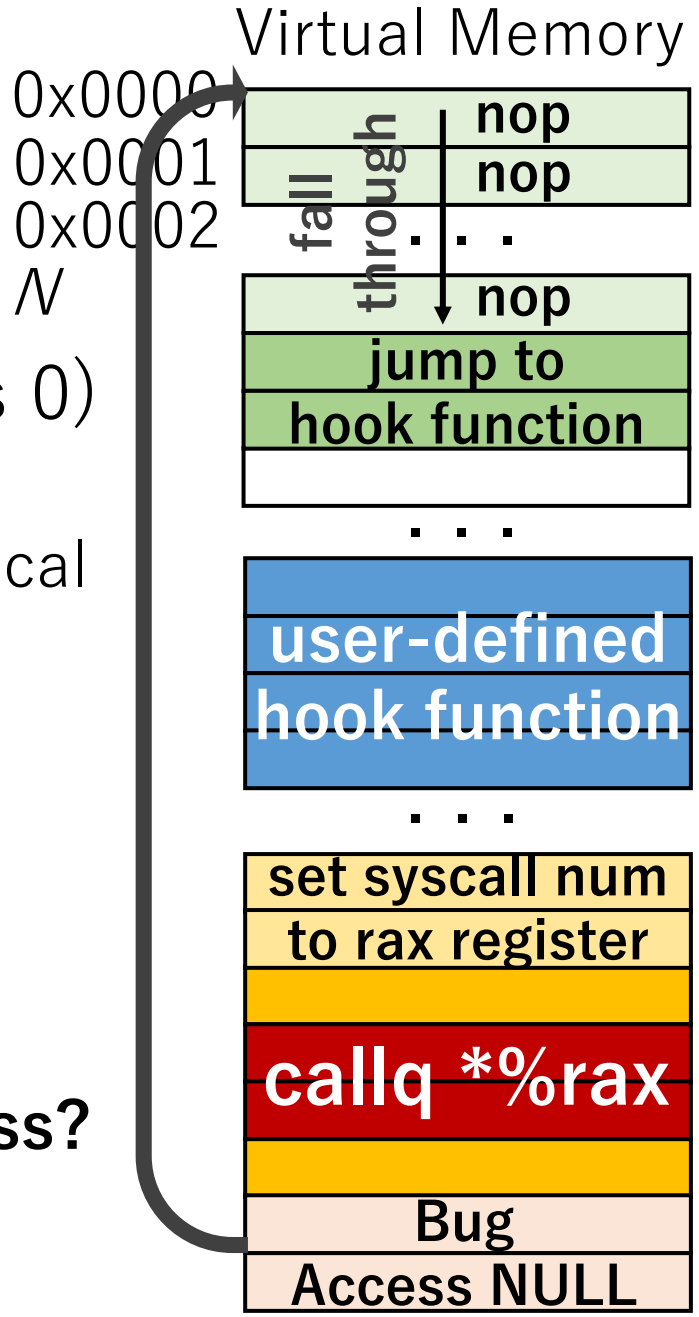
NULL Access Termination

- 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

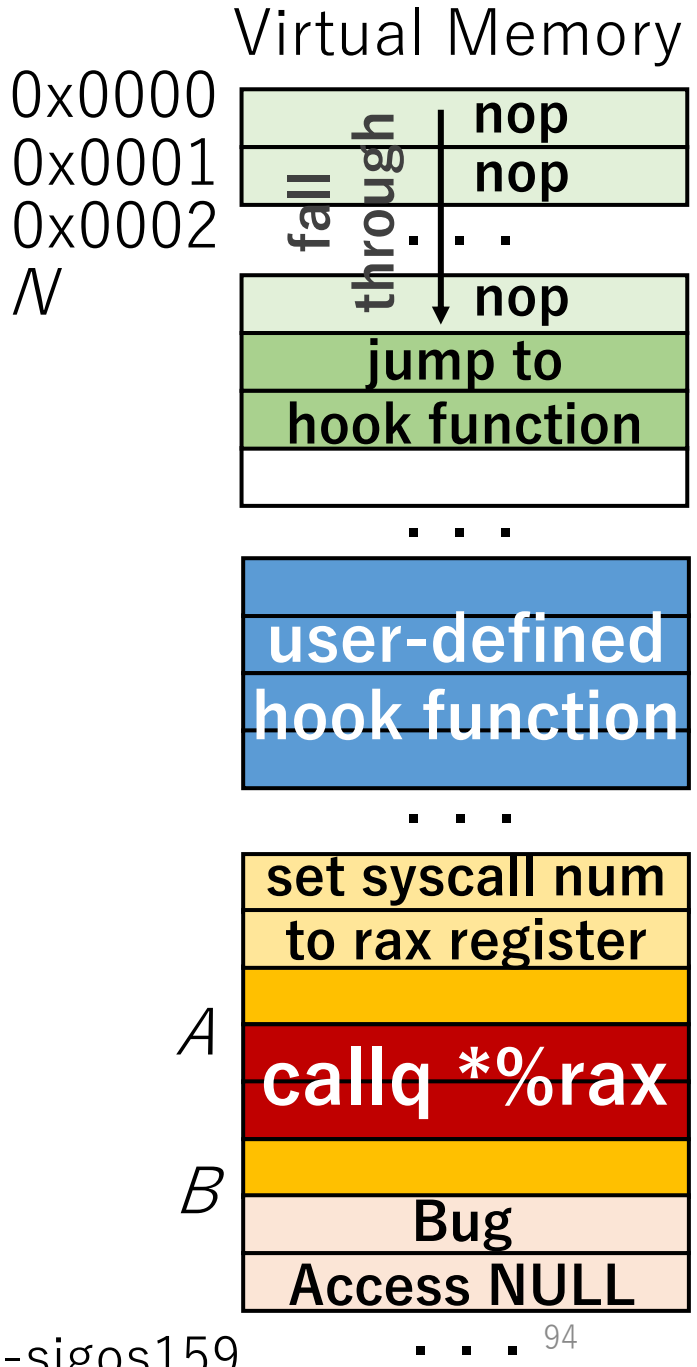
How can we detect and terminate a buggy NULL access?



https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

NULL Access Termination

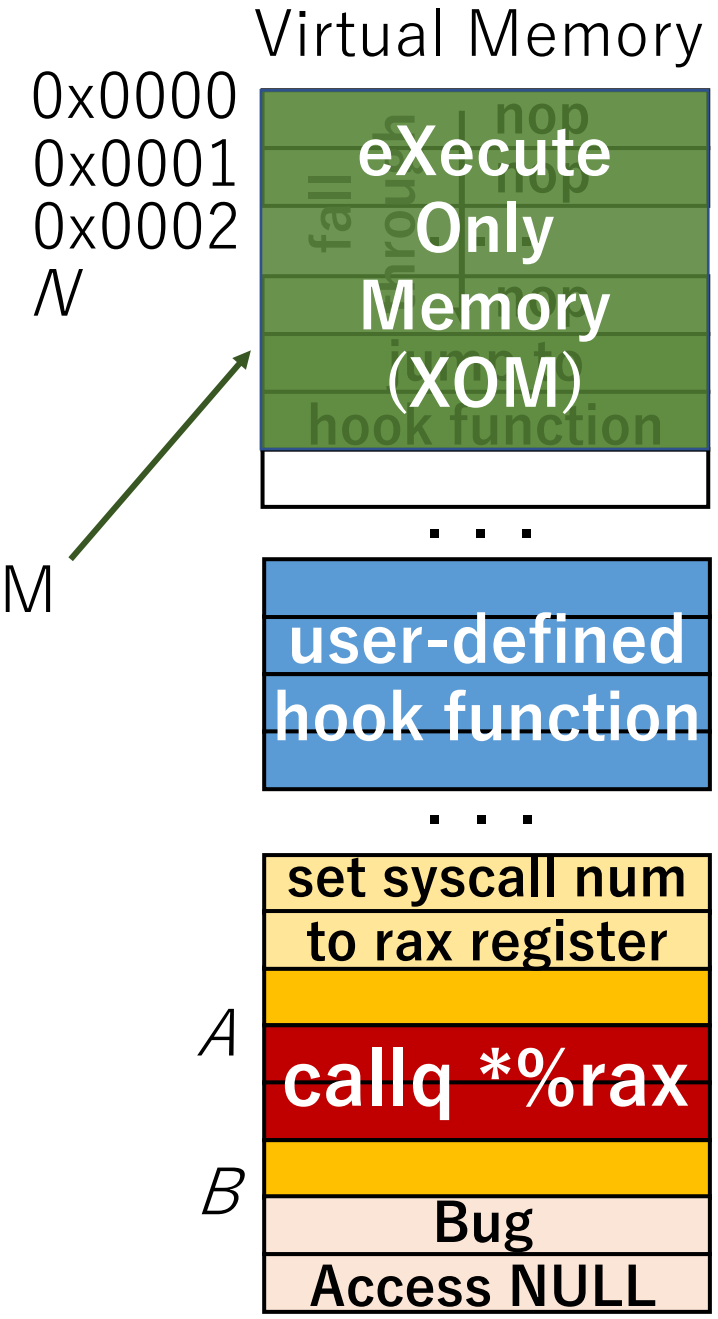
- Memory access: read / write / execute



https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

NULL Access Termination

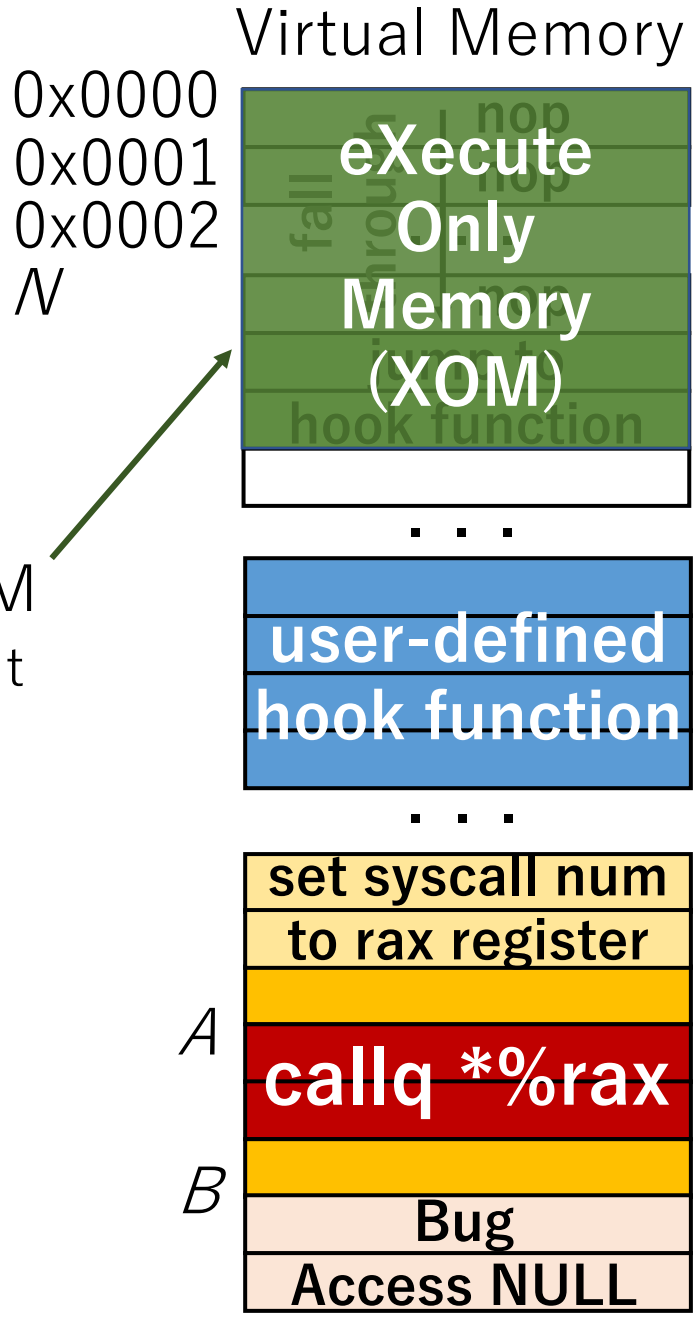
- Memory access: read / write / execute
- Solution
  - read/write: configure the trampoline code as XOM



https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

NULL Access Termination

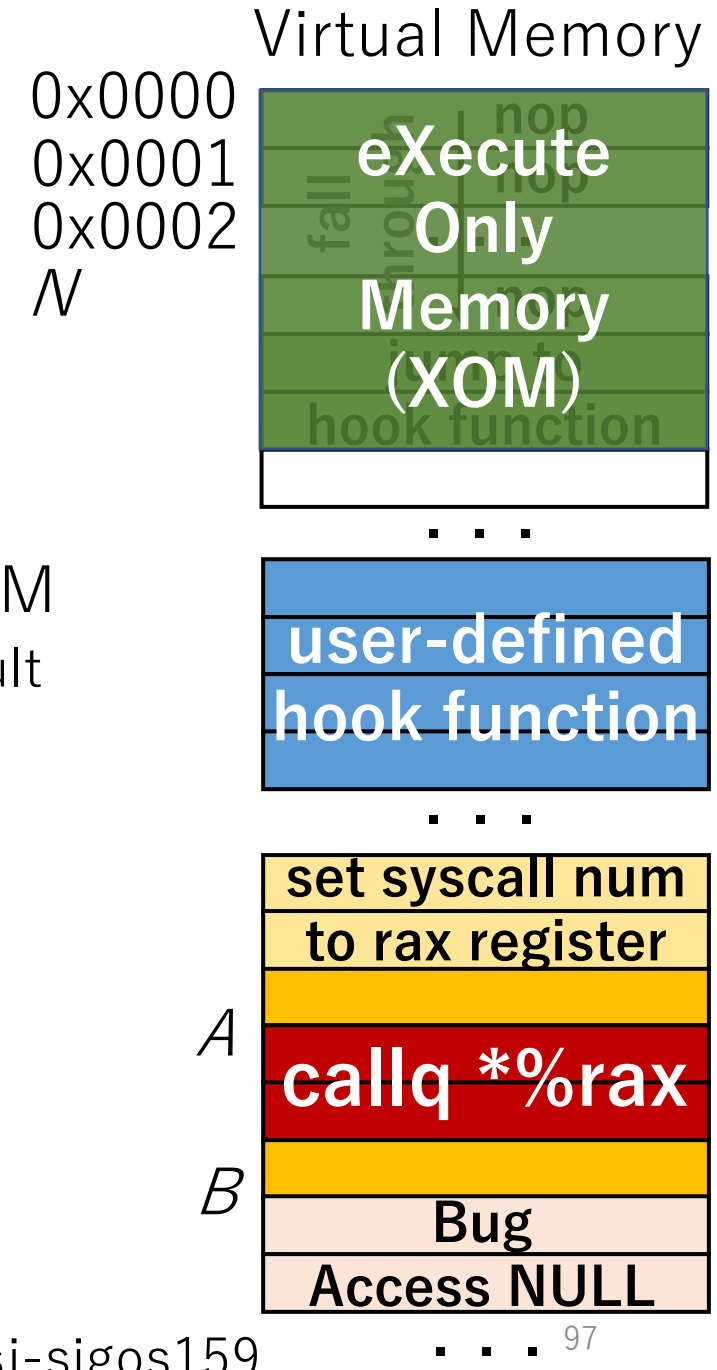
- Memory access: read / write / execute
- Solution
  - read/write: configure the trampoline code as XOM
    - read/write access to the trampoline code causes a fault
      - This can be done by mprotect() system call



https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

NULL Access Termination

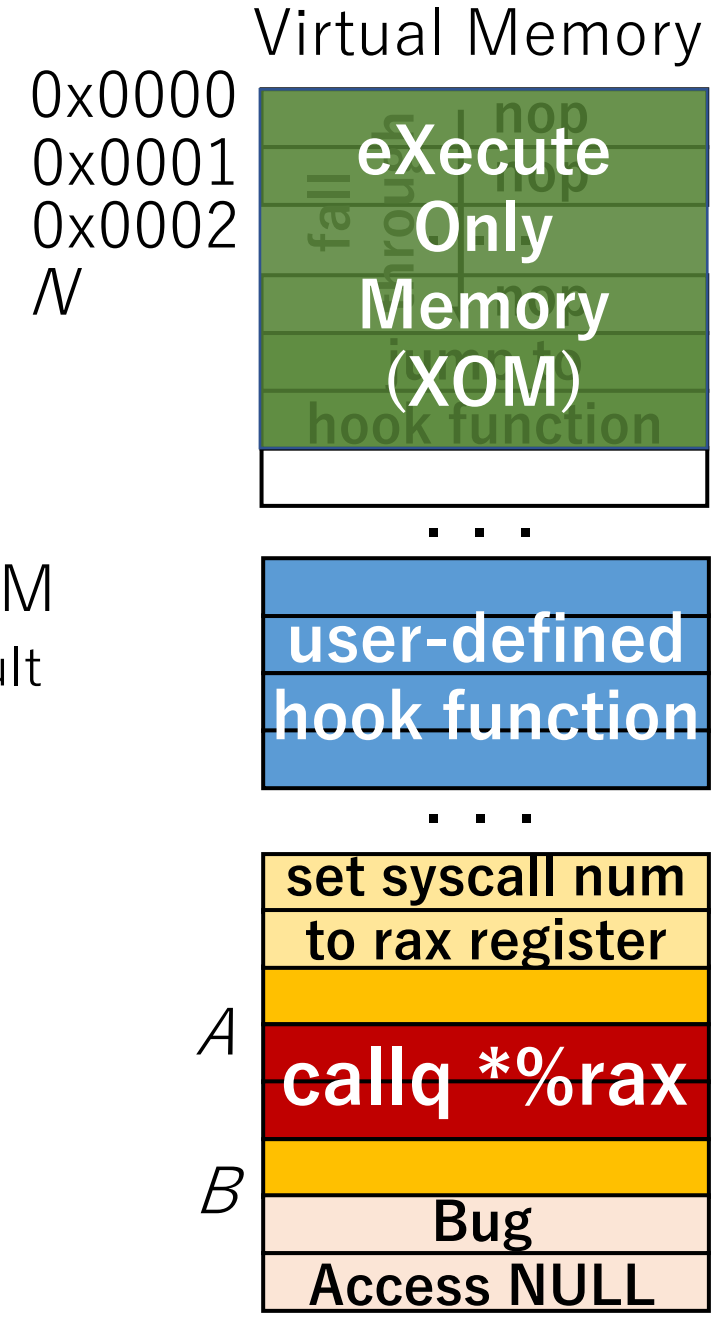
- Memory access: read / write / execute
- Solution
  - read/write: configure the trampoline code as XOM
    - read/write access to the trampoline code causes a fault
      - This can be done by mprotect() system call
  - execute: check the caller address



https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

NULL Access Termination

- 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

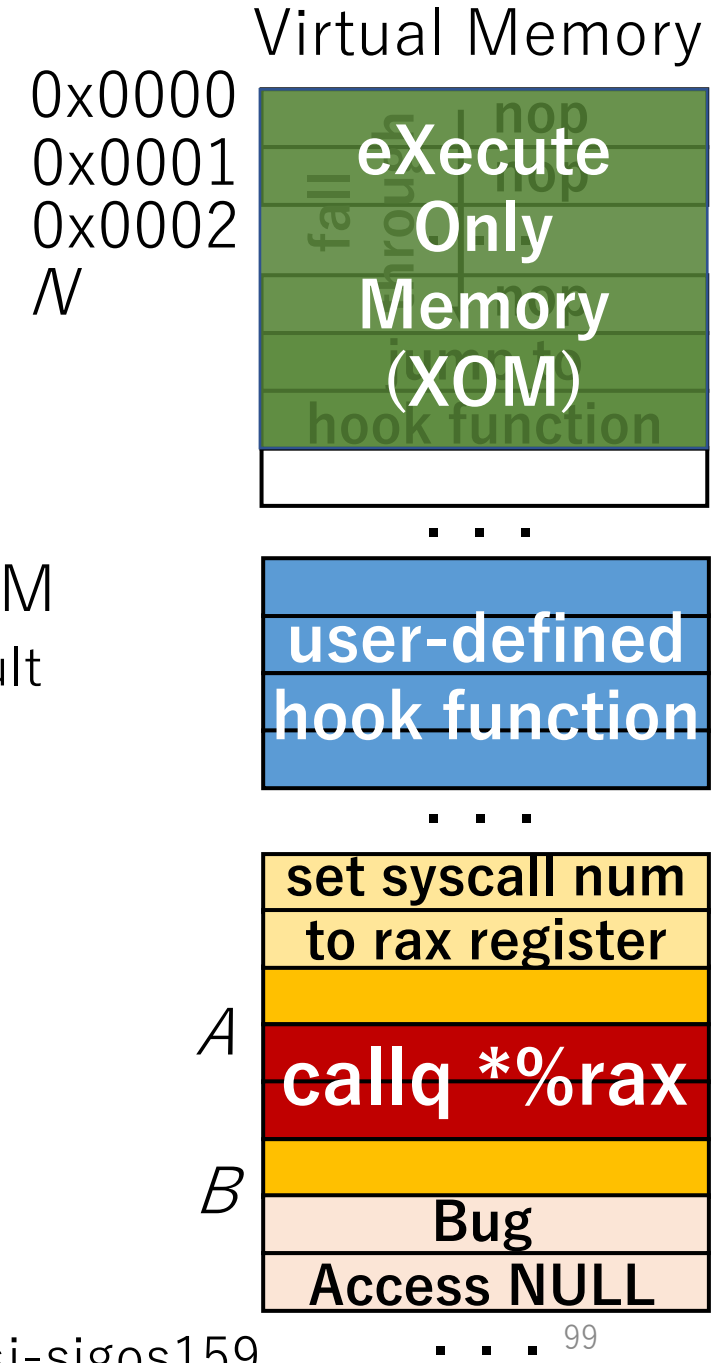


List of replaced addresses : [...]

https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

NULL Access Termination

- 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



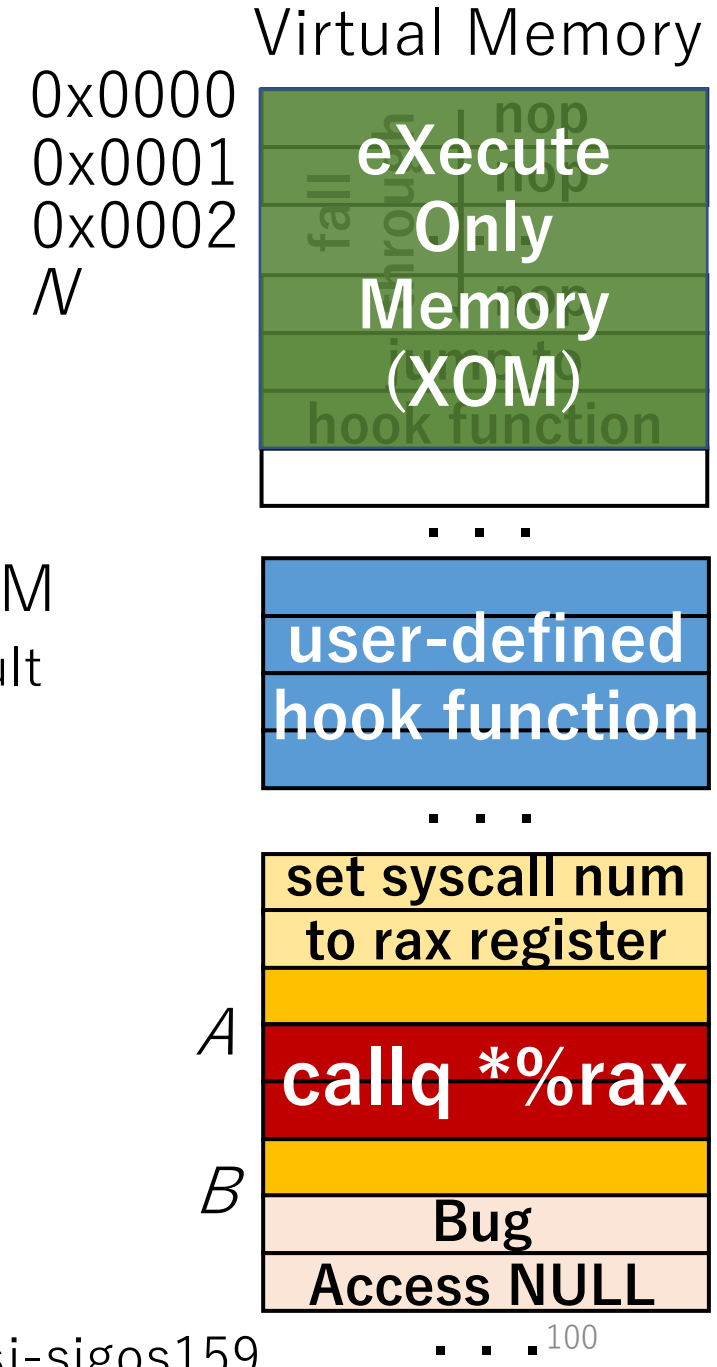
https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

NULL Access Termination

- 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 : [...]



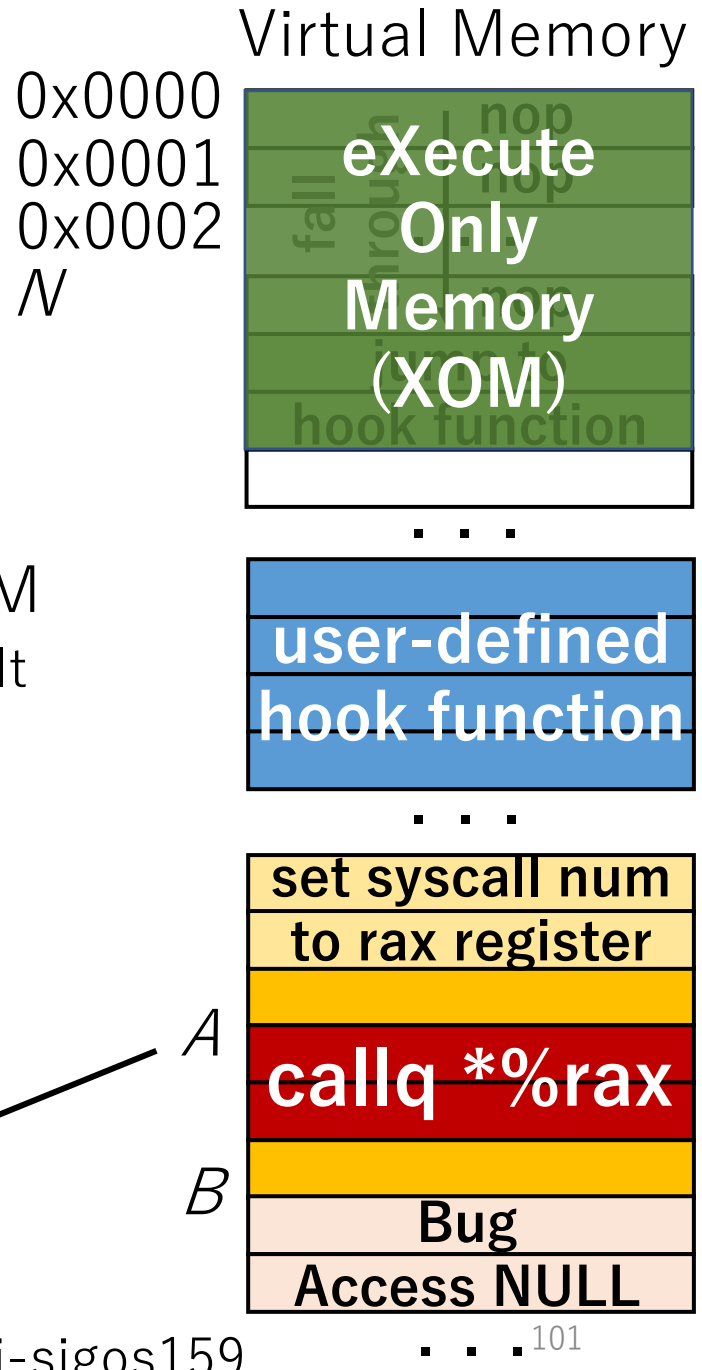
https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

NULL Access Termination

- 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 : [...]

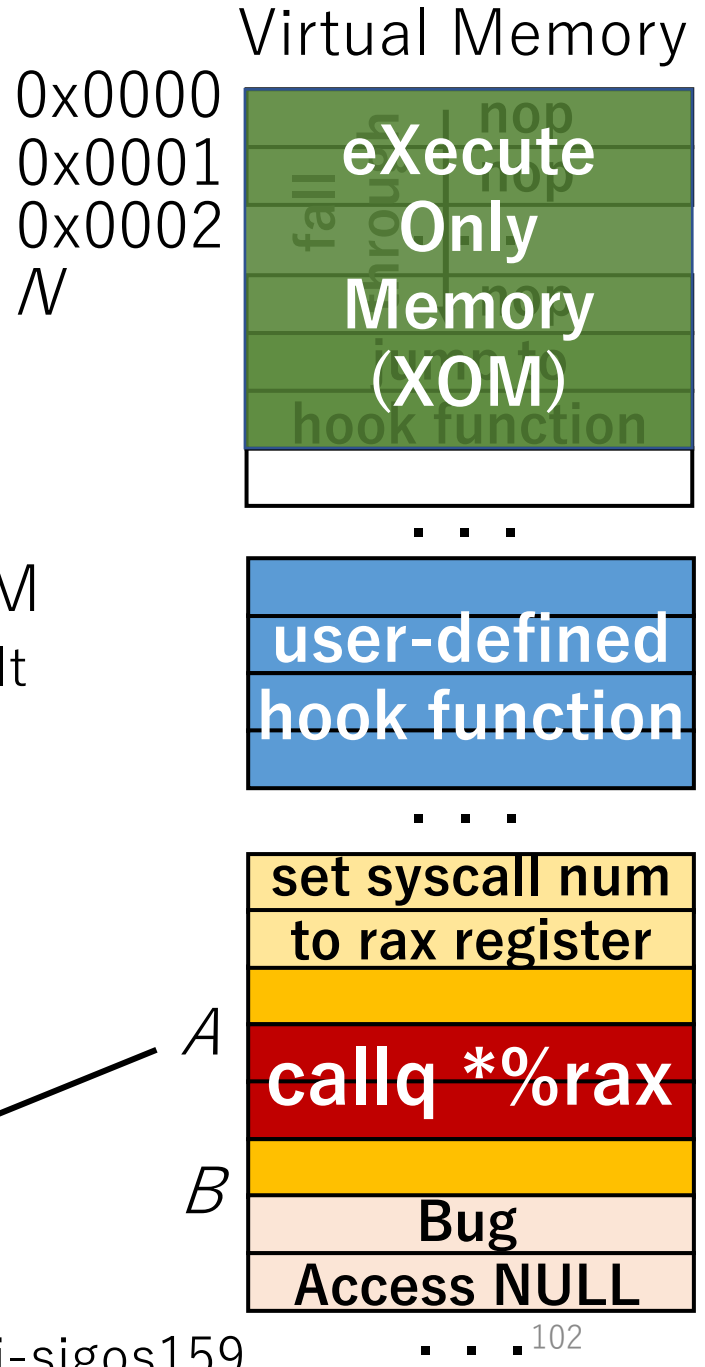


NULL Access Termination

- 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* , …]

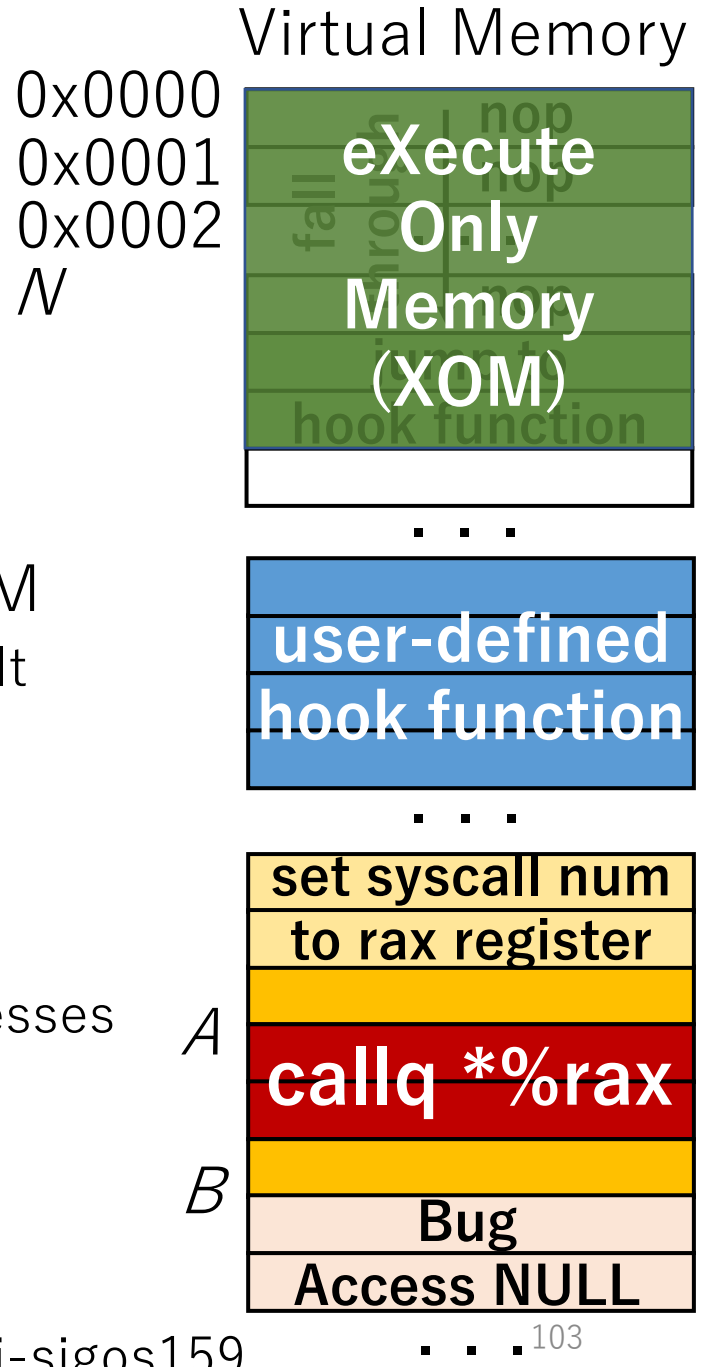


https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

NULL Access Termination

- 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* , ...]



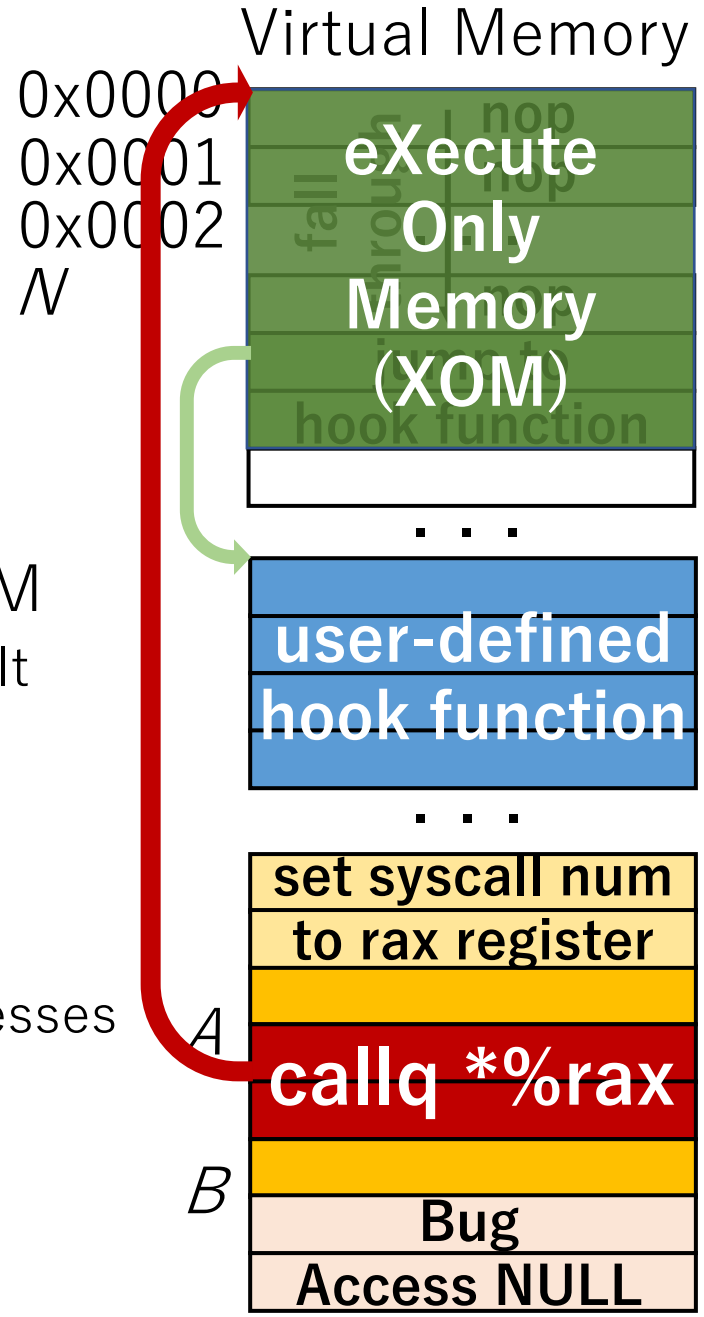
https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

NULL Access Termination

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* , …]



https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

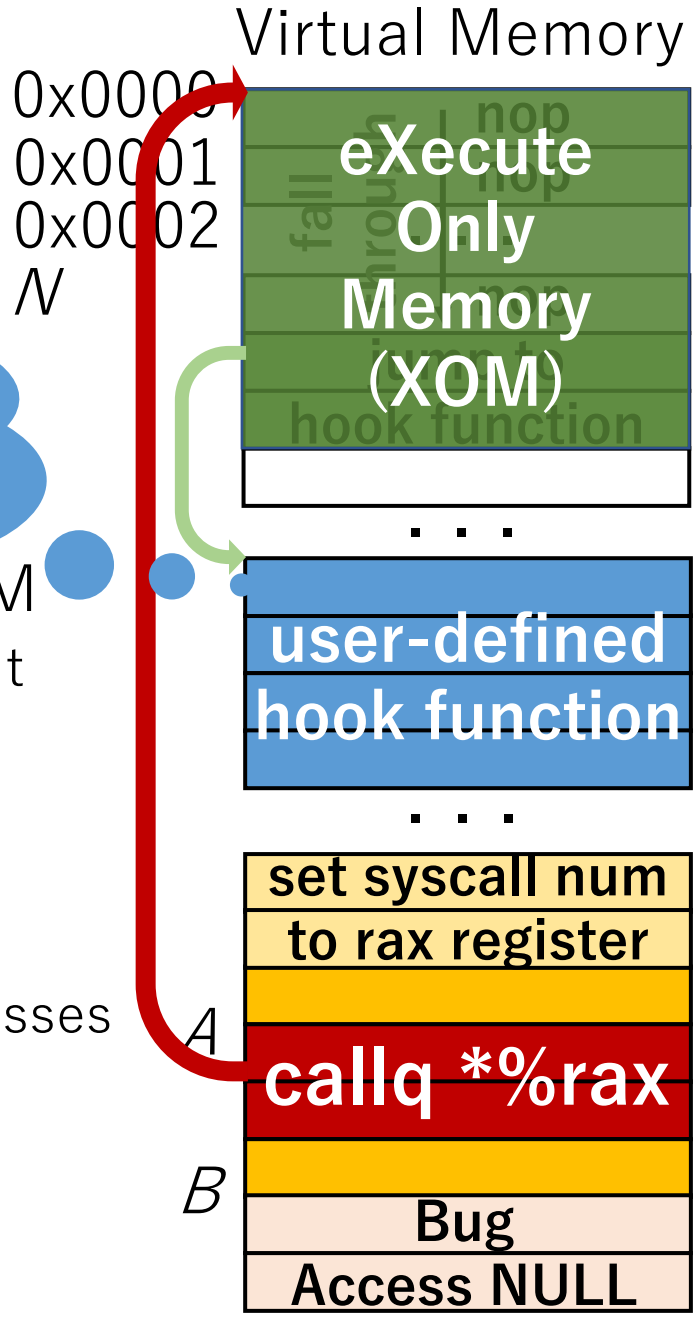
NULL Access Termination

At runtime …

- Memory acces
- Solution
  - read/write: s XOM
    - read/write access to the tra 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

The caller address is *A*
***A* is in the list, so**
this is a valid access

List of replaced addresses : [*A* , …]



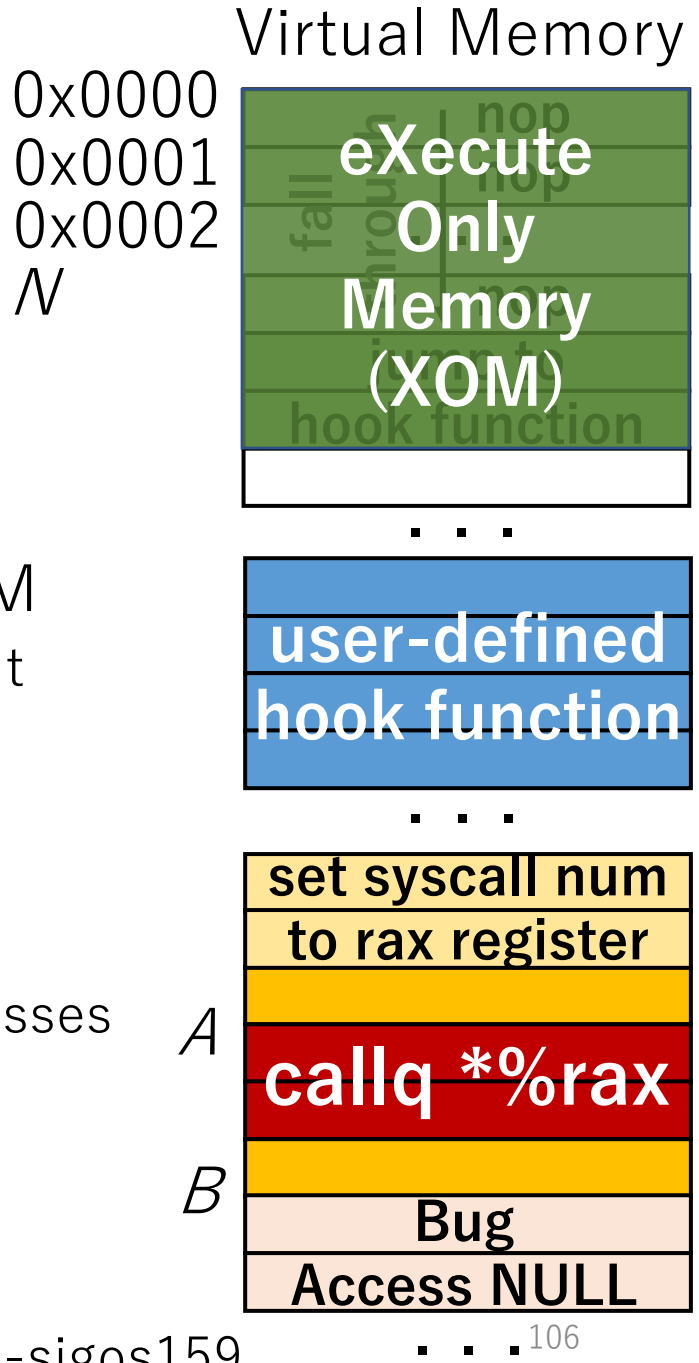
https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

NULL Access Termination

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* , …]



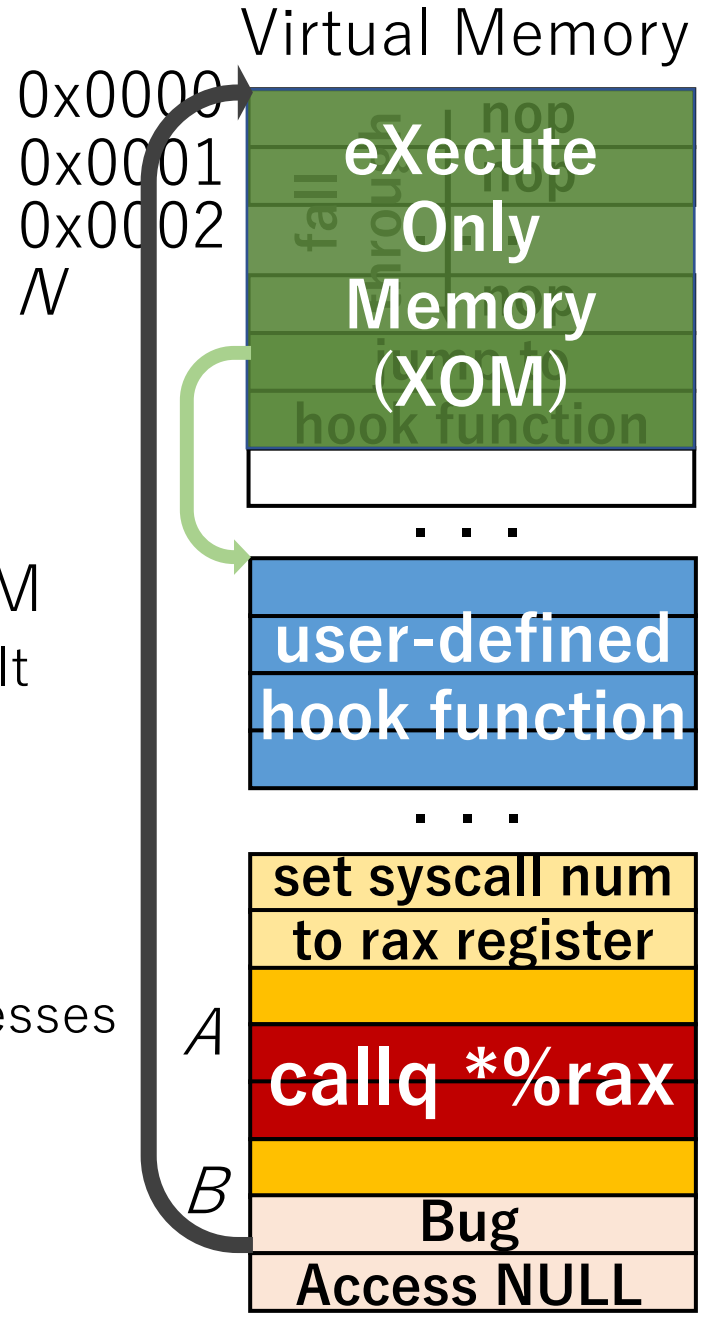
https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

NULL Access Termination

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* , …]



https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

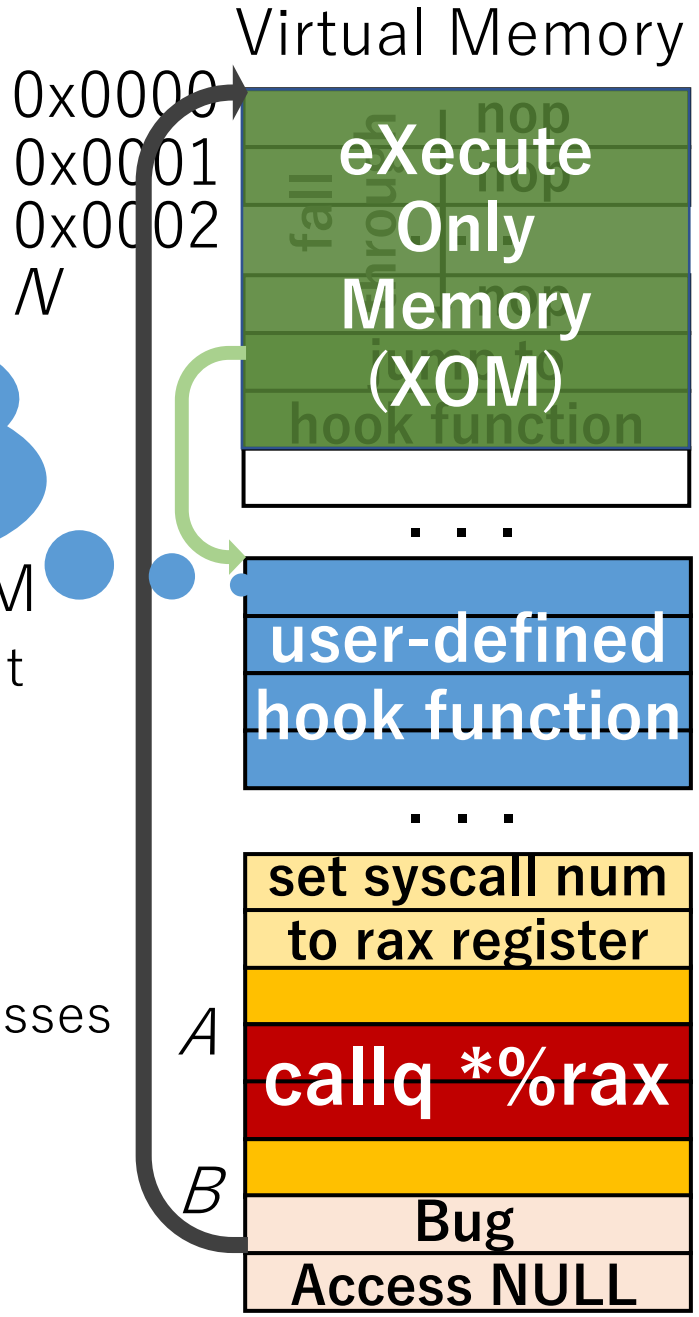
NULL Access Termination

At runtime …

- Memory acces
- Solution
  - read/write:
    - read/write access to the tra… 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

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

List of replaced addresses : [*A* , …]



https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

NULL Access Termination

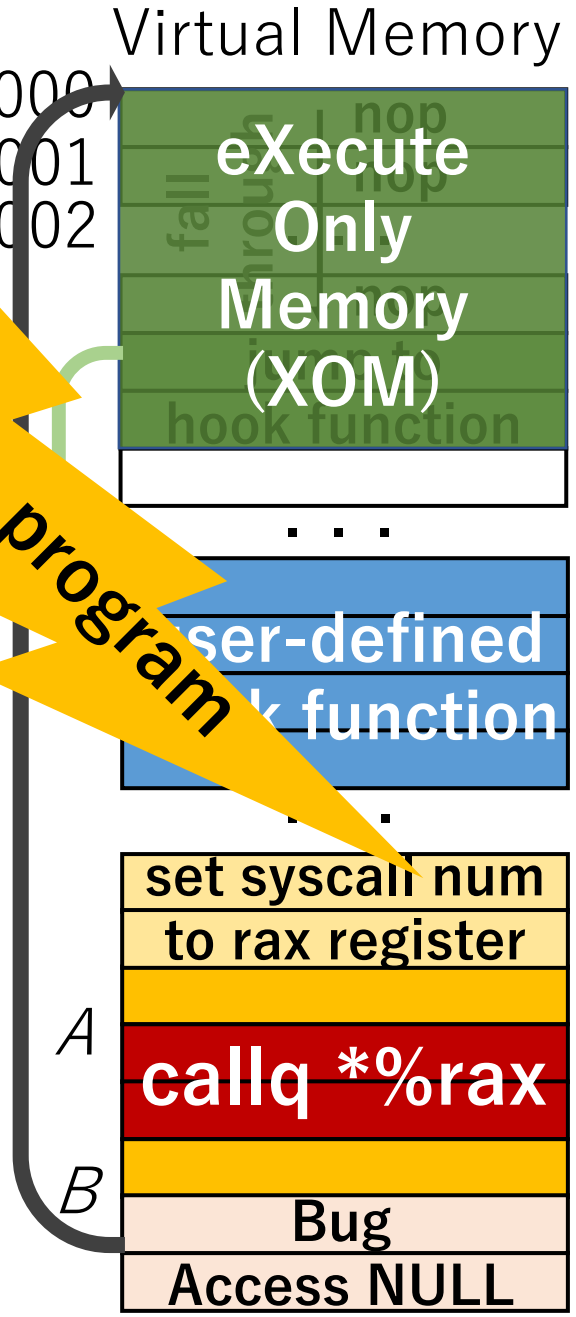
At runtime …

- Memory acces
- Solution
  - read/write: XOM
    - read/write access to the tra 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

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

stop the program

List of replaced addresses : [*A* , …]



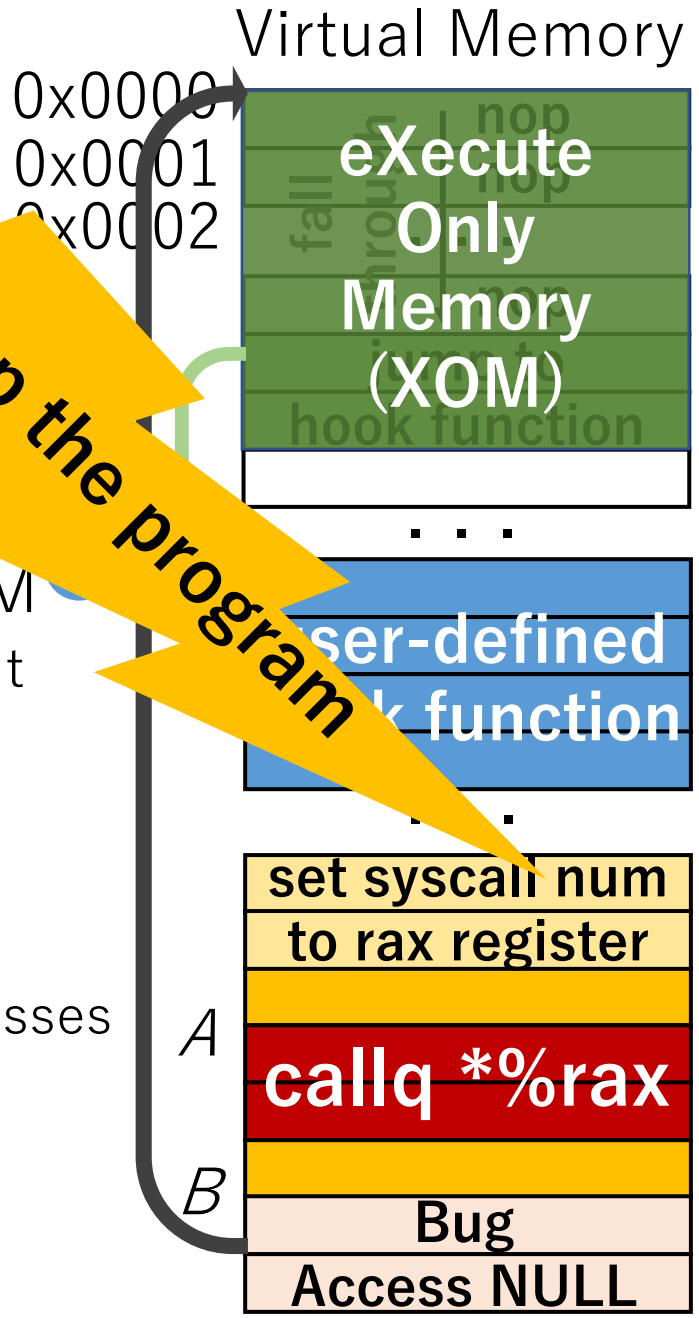
https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

NULL Access Termination

At runtime …

- Memory acces
- Solution
  - read/write: XOM
    - read/write access to the tra 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
       - Current prototype uses bitmap to implement this check

List of replaced addresses : [*A* , …]

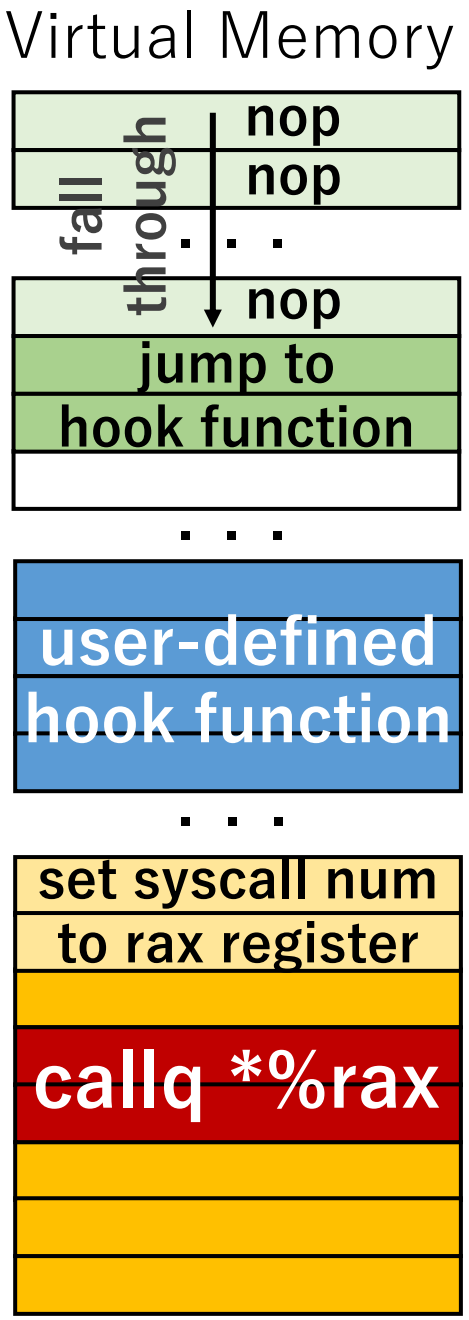


https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

System Call Hook Overhead

- Time to hook getpid() and return a dummy value

Mechanism	Time [ns]
ptrace	31201
int3 signaling	1342
SUD	1156
zpoline	41
LD_PRELOAD	6

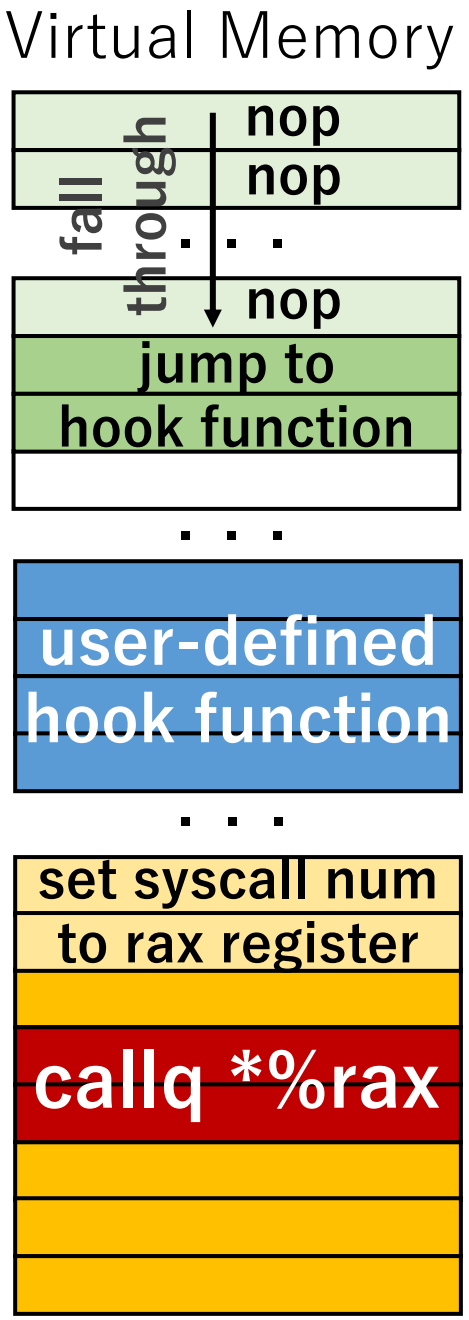


https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

System Call Hook Overhead

- Time to hook getpid() and return a dummy value

Mechanism	Time [ns]	
ptrace	31201	716x
int3 signaling	1342	32.7x
SUD	1156	28.1x
zpoline	41	improvement
LD_PRELOAD	6	

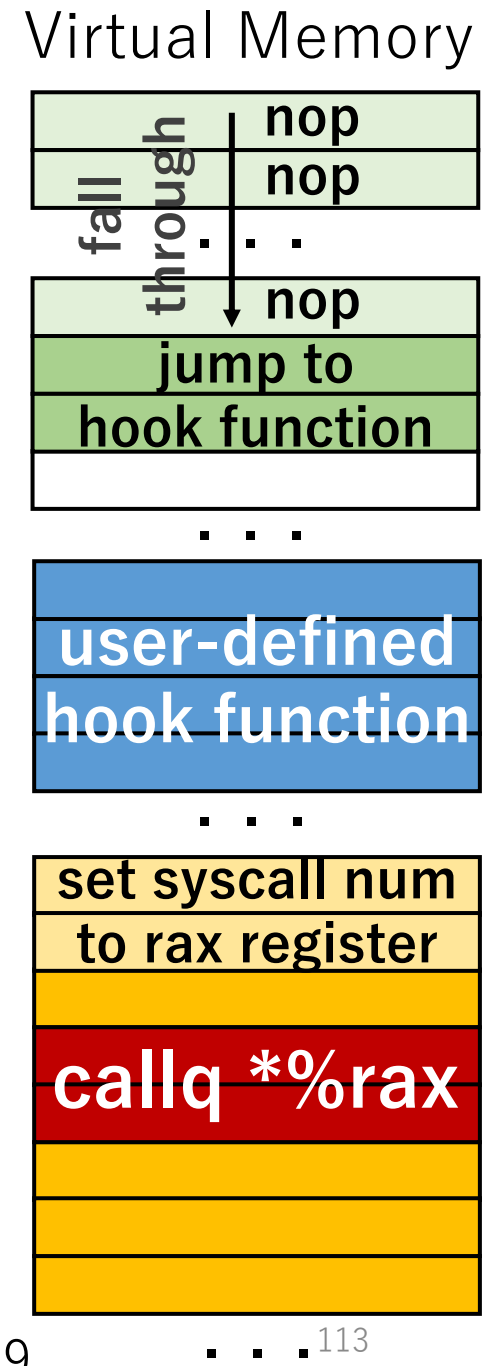


https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

System Call Hook Overhead

- 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

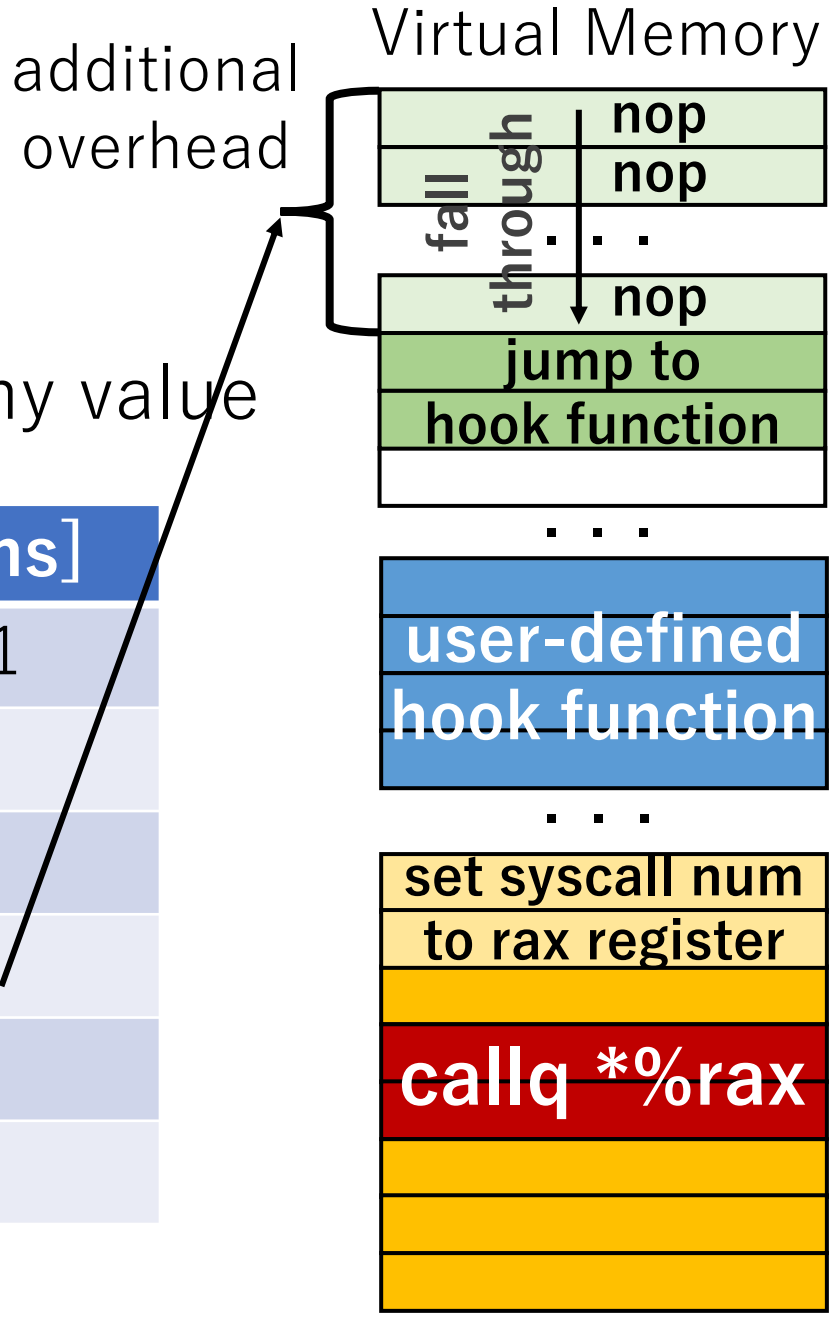


https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

System Call Hook Overhead

- Time to hook getpid() and return a dummy value

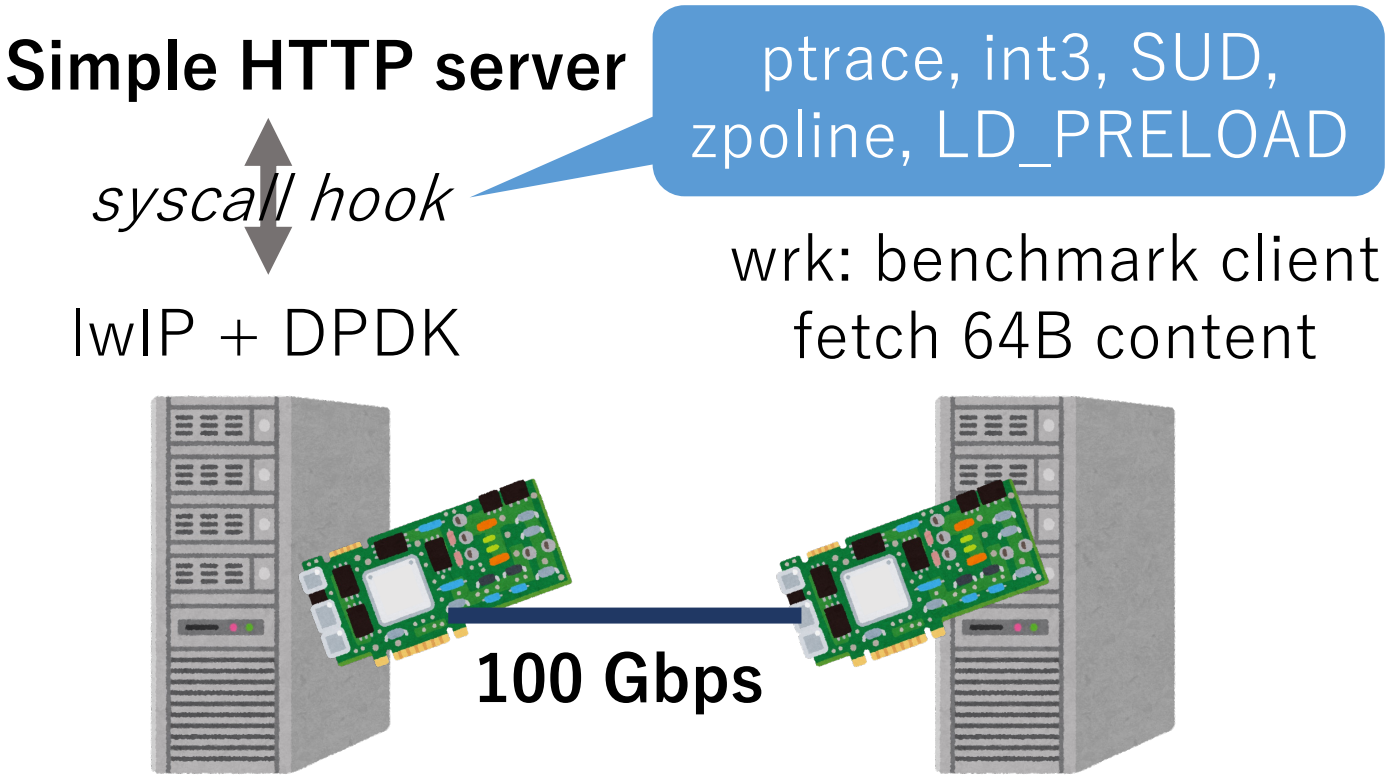
Mechanism	Time [ns]
ptrace	31201
int3 signaling	1342
SUD	1156
zpoline	41
zpoline (w/o NULL exec check)	40
LD_PRELOAD	6

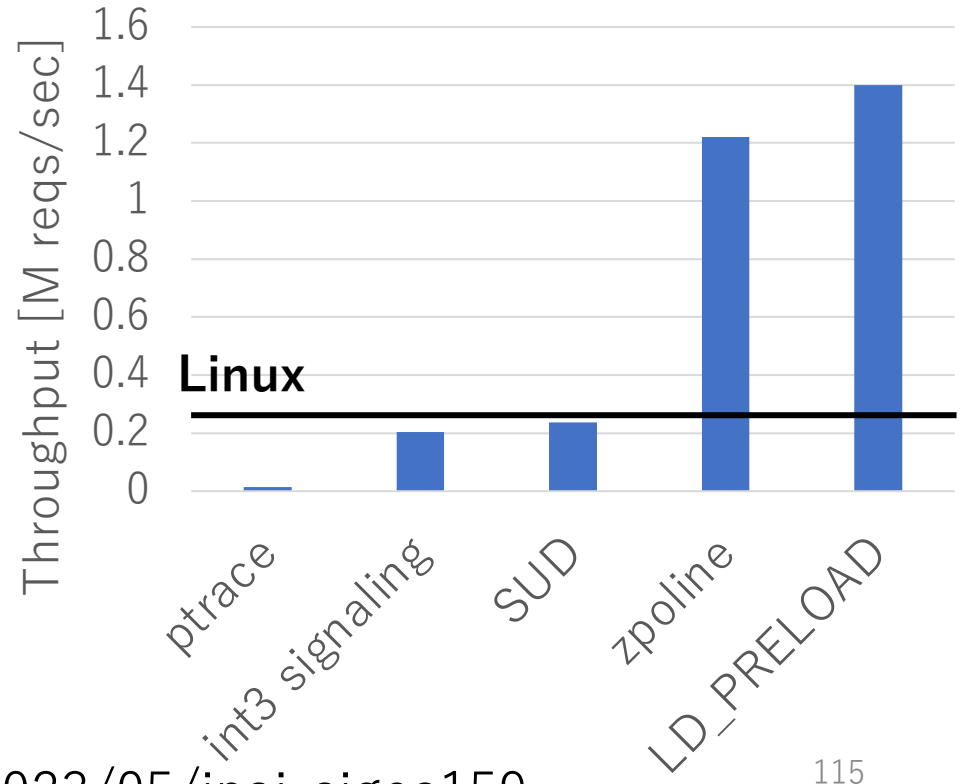


https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

Application Performance

- We **transparently** apply lwIP + DPDK to an application using different system call hook mechanisms

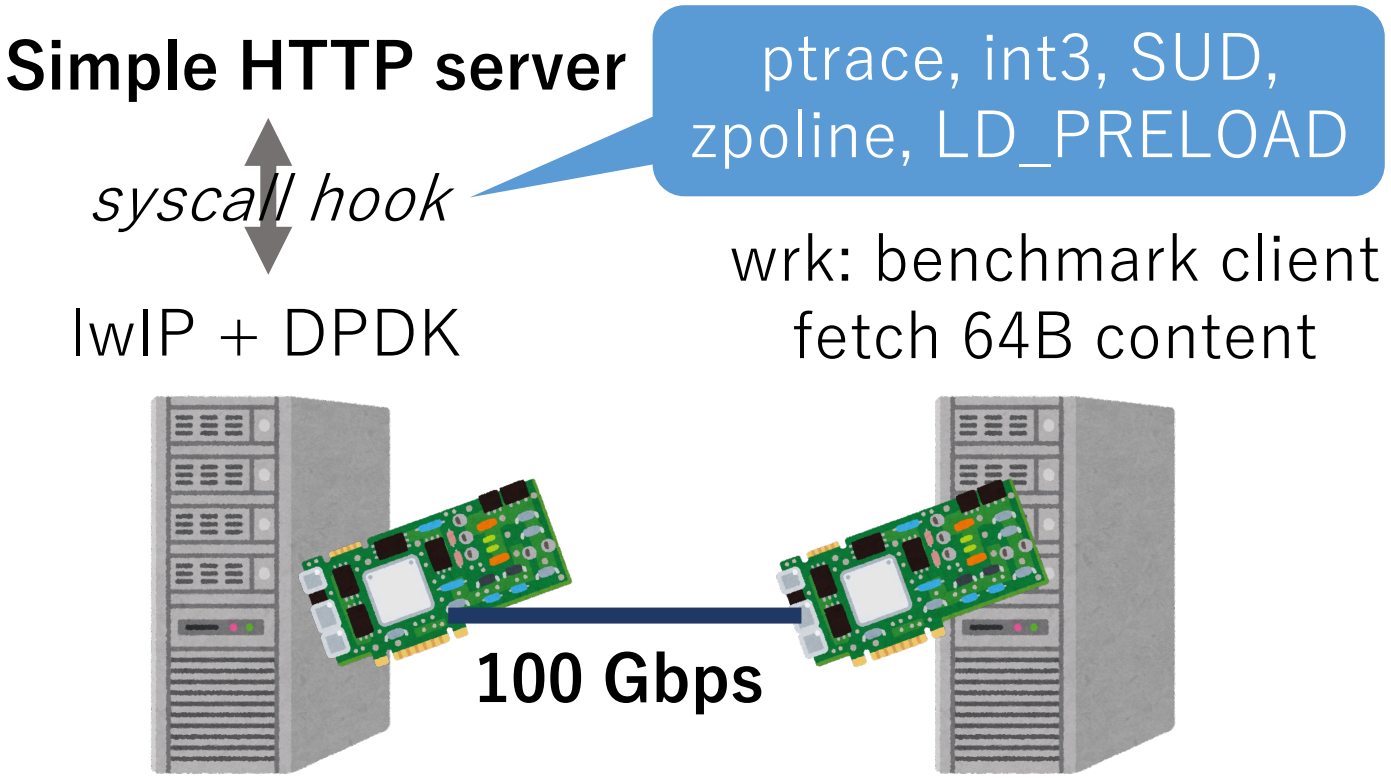


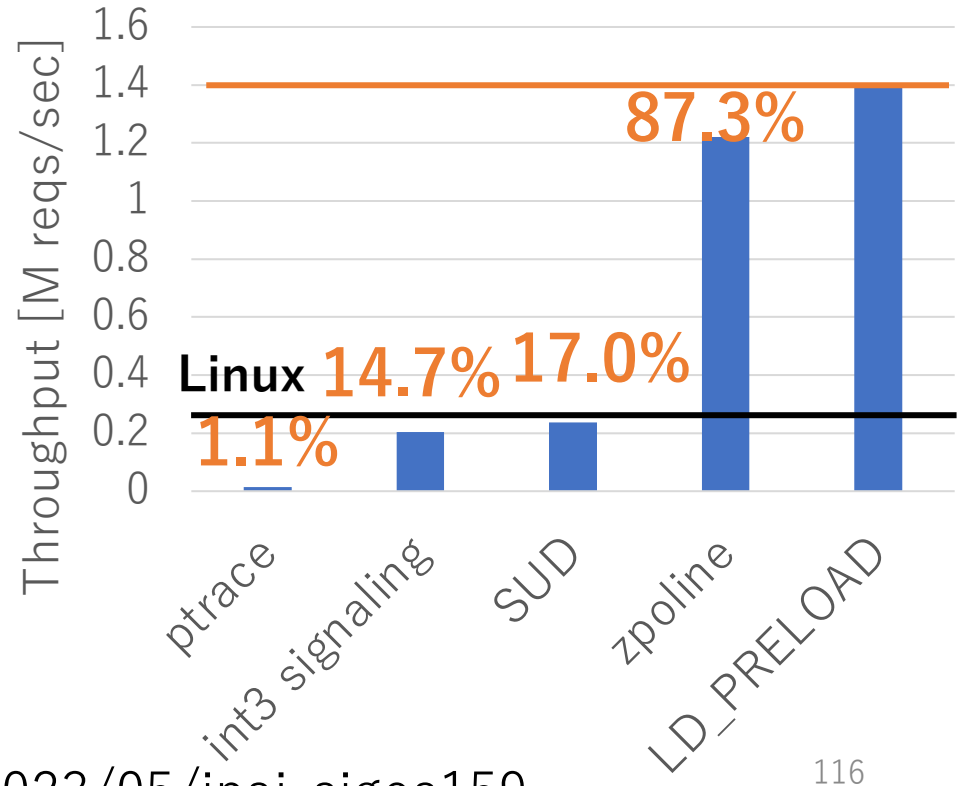


https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

Application Performance

- We **transparently** apply lwIP + DPDK to an application using different system call hook mechanisms

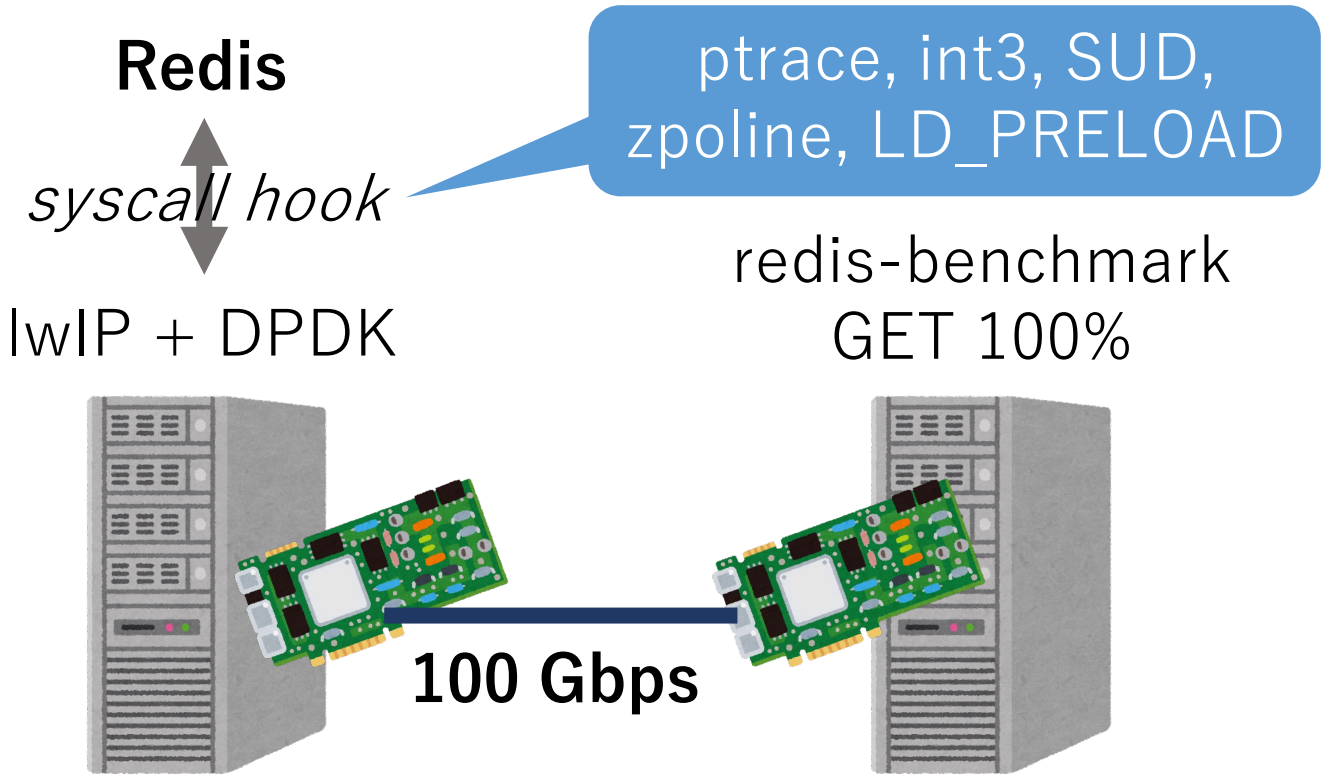


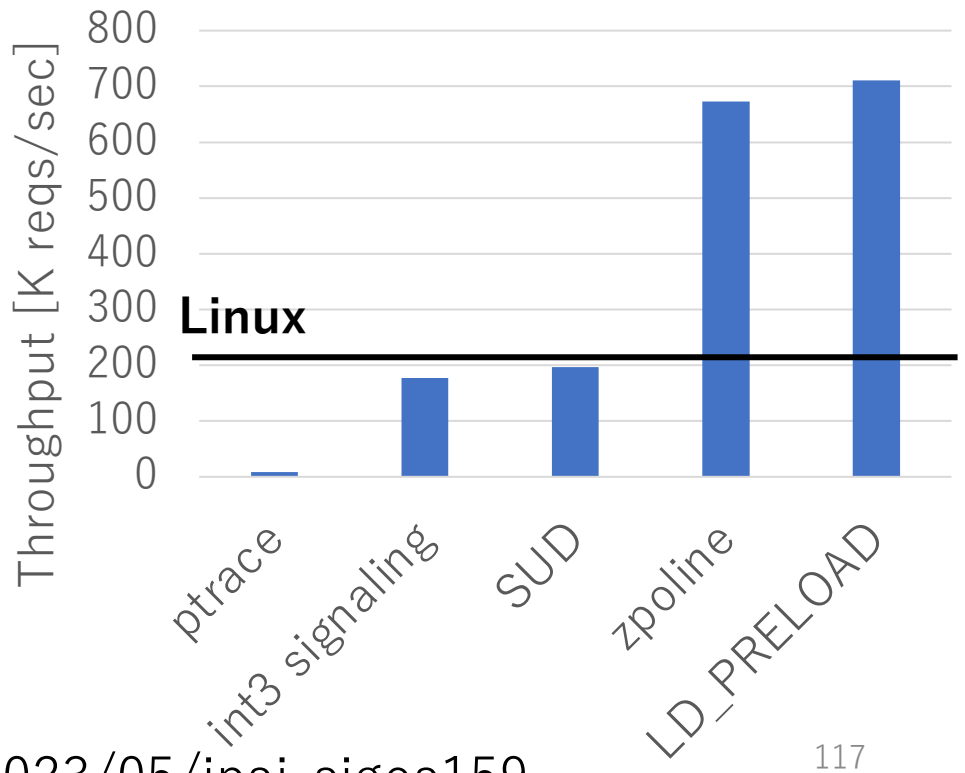


https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

Application Performance

- We **transparently** apply lwIP + DPDK to an application using different system call hook mechanisms

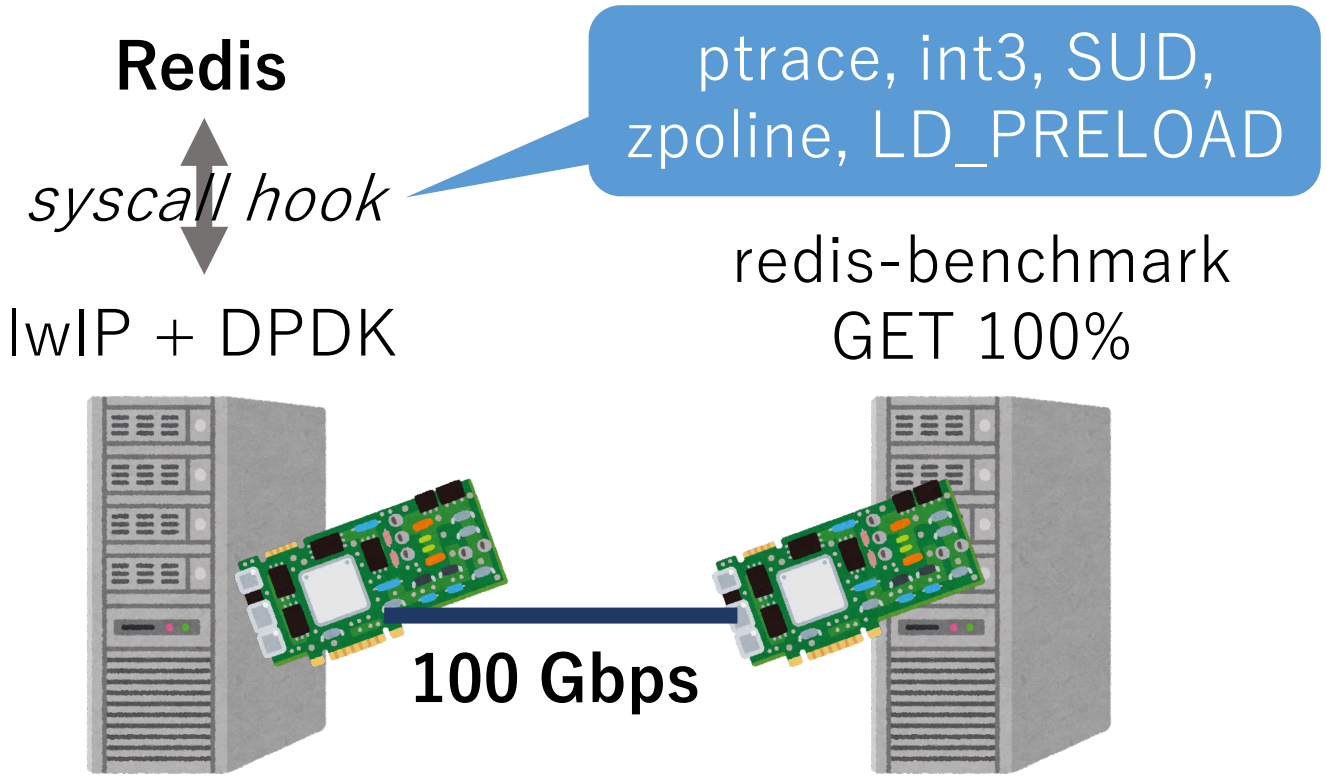


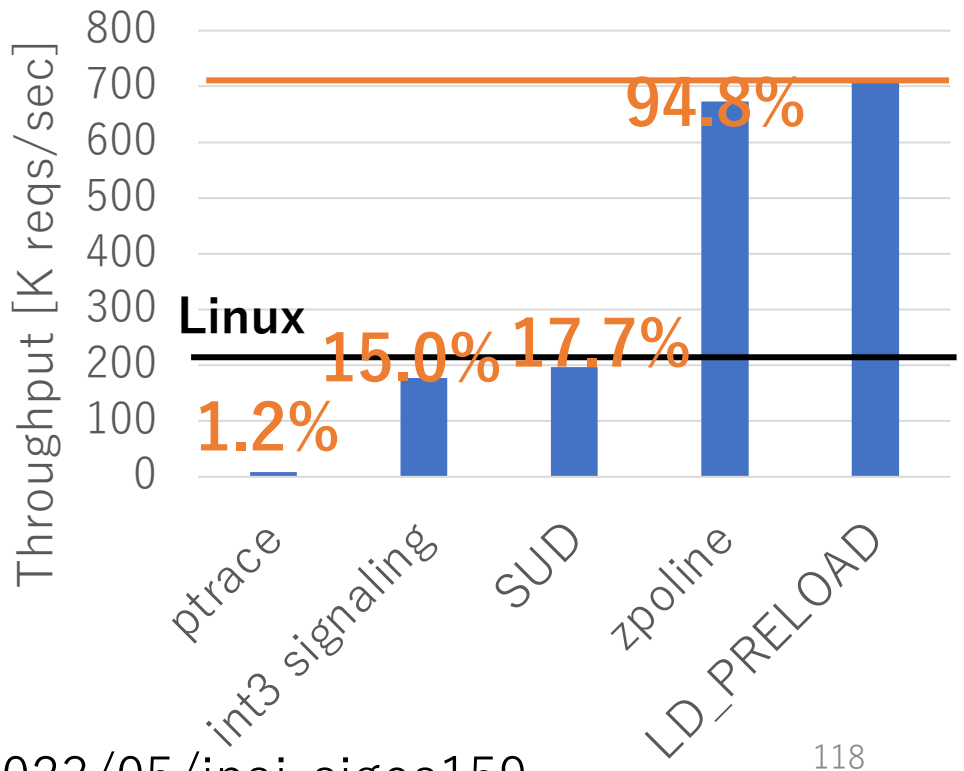


https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

Application Performance

- We **<u>transparently</u>** apply lwIP + DPDK to an application using different system call hook mechanisms





https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159

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

https://github.com/yasukata/presentation/tree/gh-pages/2023/05/ipsj-sigos159