


TU Graz
SCIENCE
PASSION
TECHNOLOGY

# Derandomizing Kernel Object Locations with Software- and Hardware-Induced Side Channels

**Lukas Maar**

September 4-5, 2025

Nullcon Berlin

› isec.tugraz.at

# Who Am I?

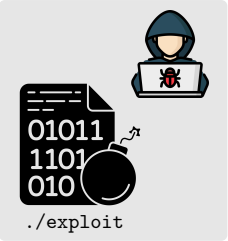
**Lukas Maar**

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

# Motivation

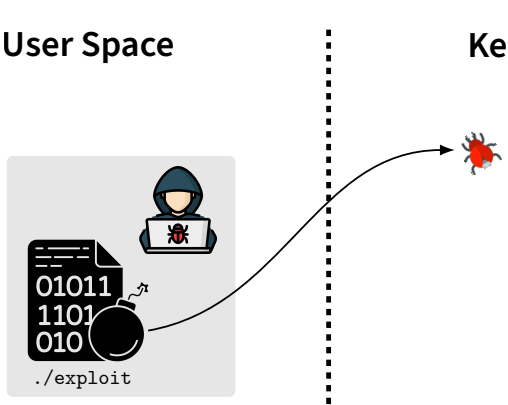
# Kernel Exploitation

## User Space

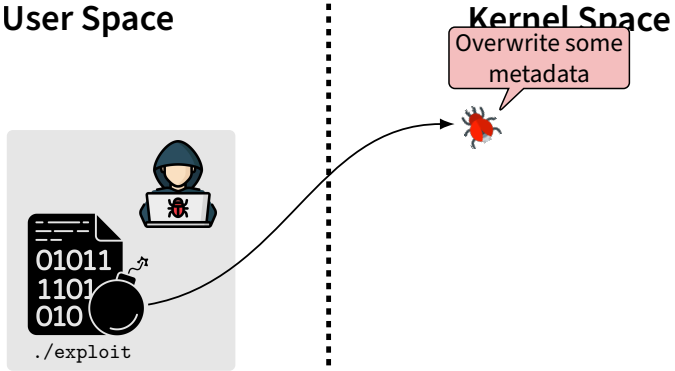



## Kernel Space

# Kernel Exploitation

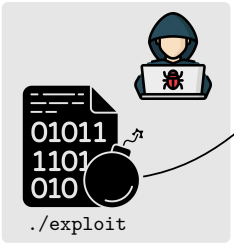


# Kernel Exploitation



# Kernel Exploitation

## User Space




## Kernel Space

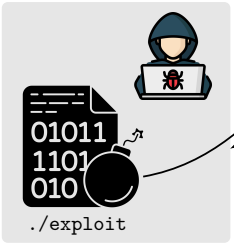
**Read primitive**

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

# Kernel Exploitation

## User Space

## Kernel Space



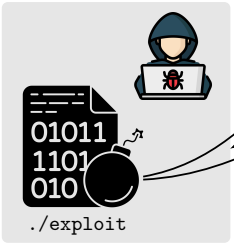

**Read primitive**

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

# Kernel Exploitation

## User Space

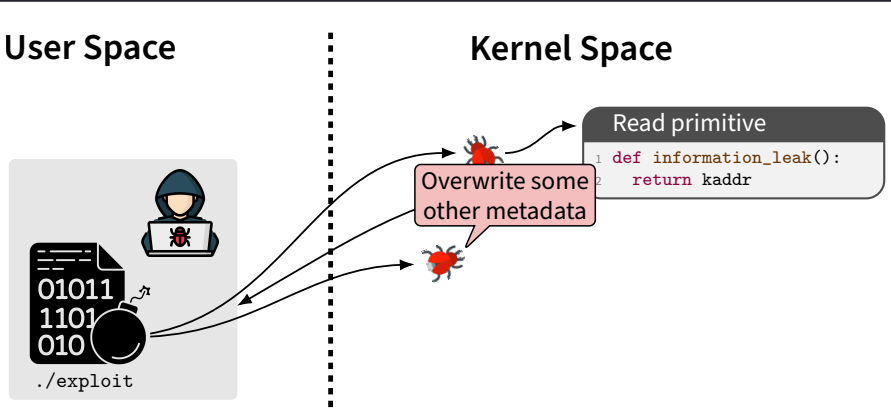
## Kernel Space



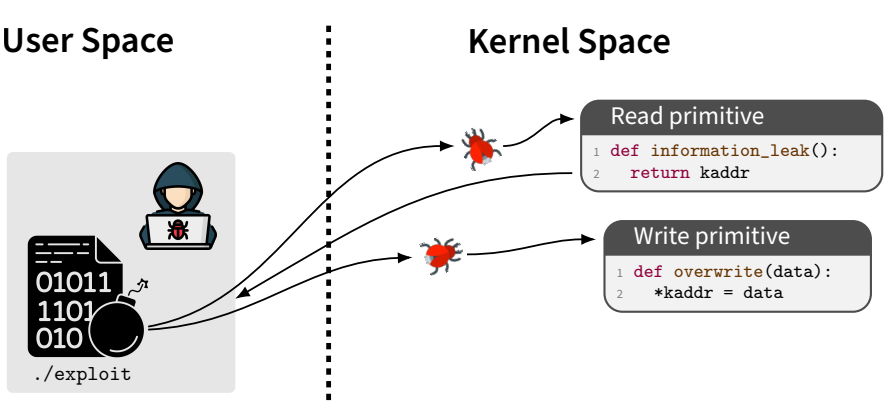

**Read primitive**

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

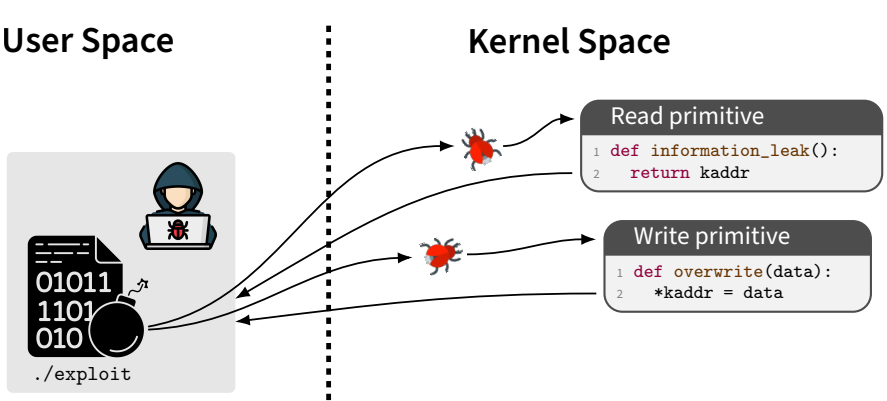
# Kernel Exploitation



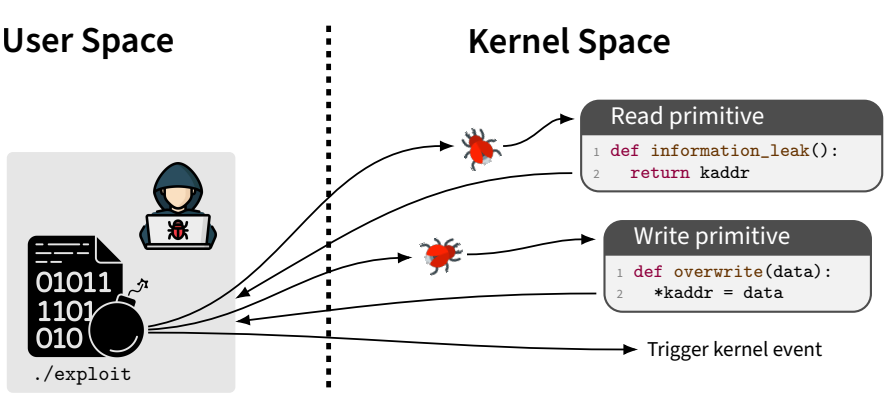
# Kernel Exploitation



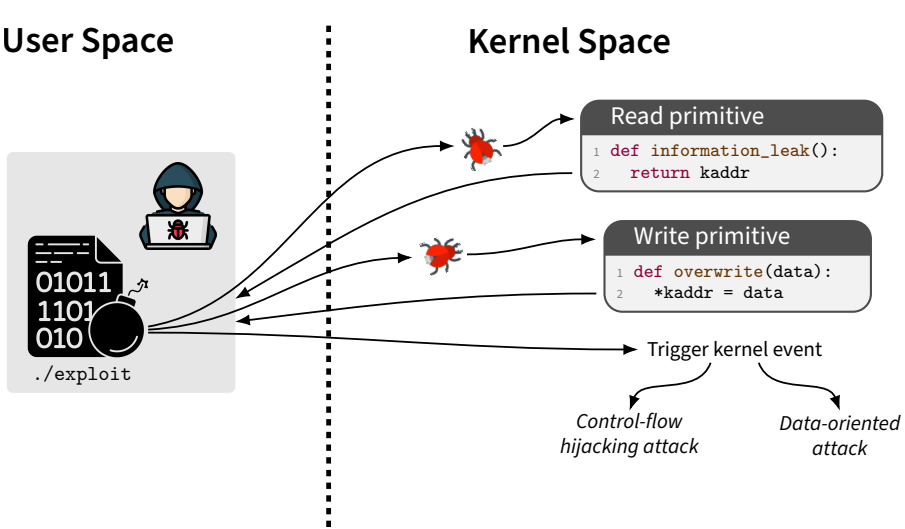
# Kernel Exploitation



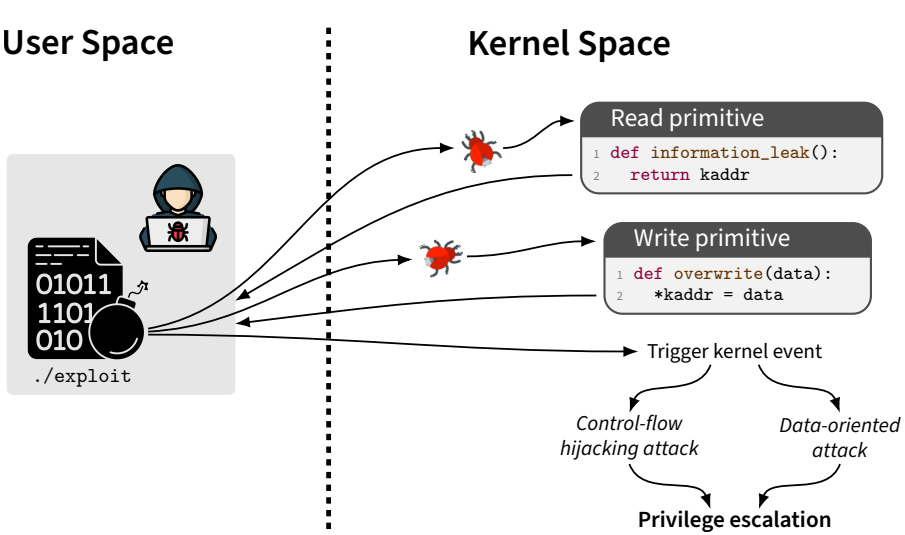
# Kernel Exploitation



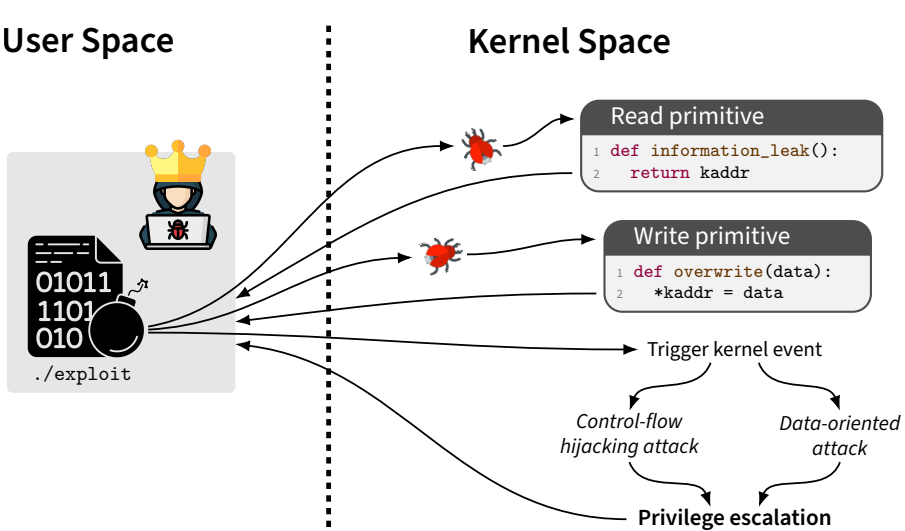
# Kernel Exploitation



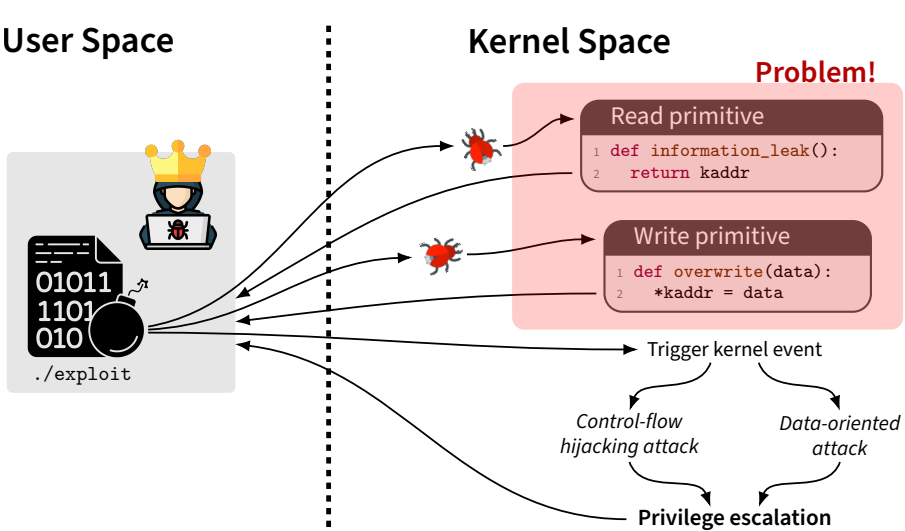
# Kernel Exploitation



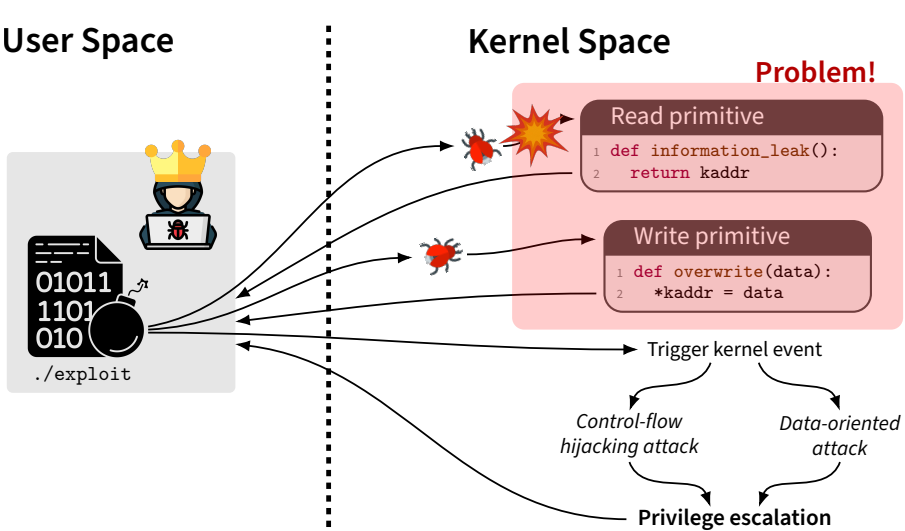
# Kernel Exploitation



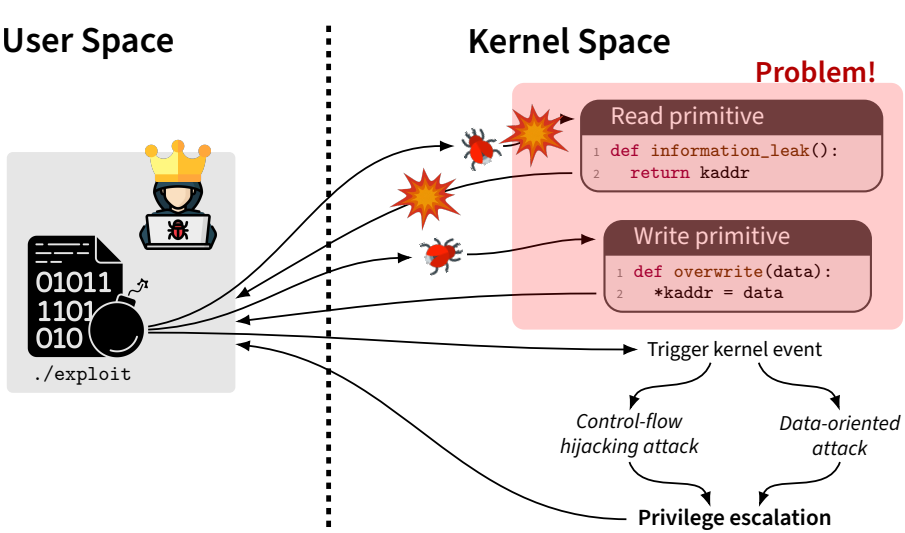
# Kernel Exploitation



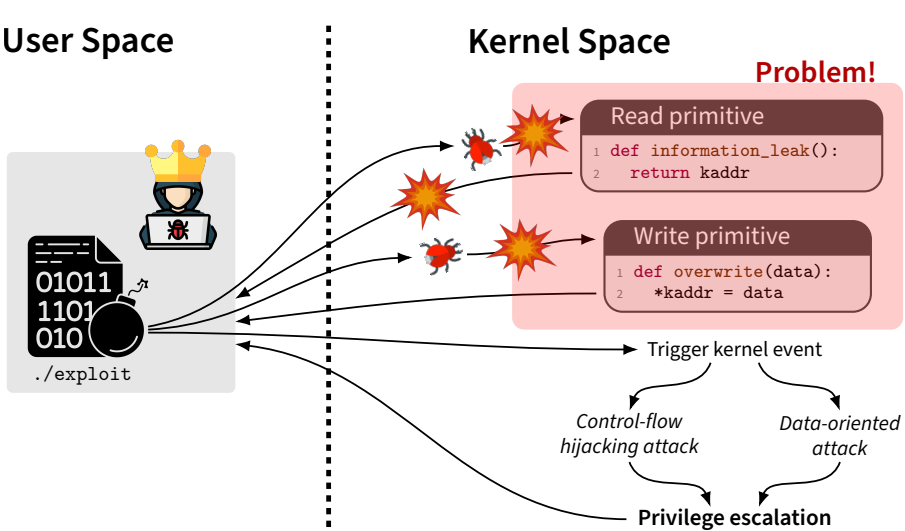
# Kernel Exploitation



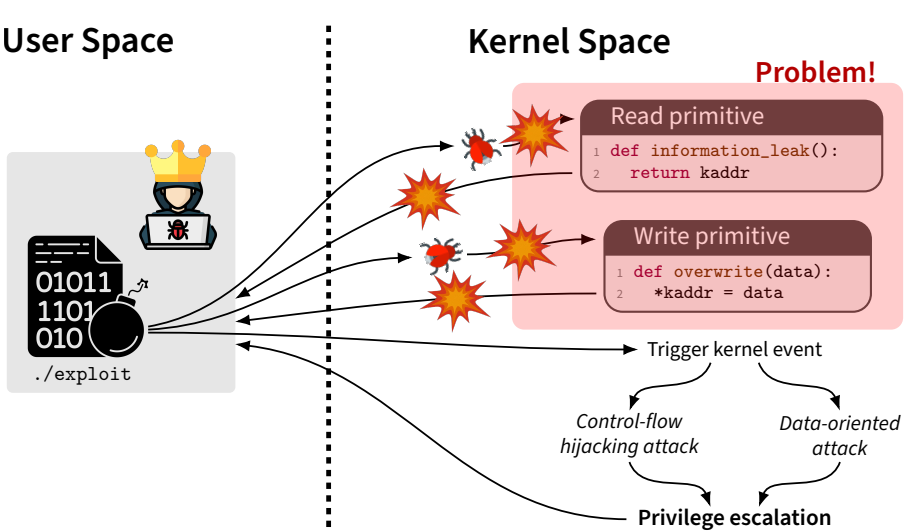
# Kernel Exploitation



# Kernel Exploitation



# Kernel Exploitation



# 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?**

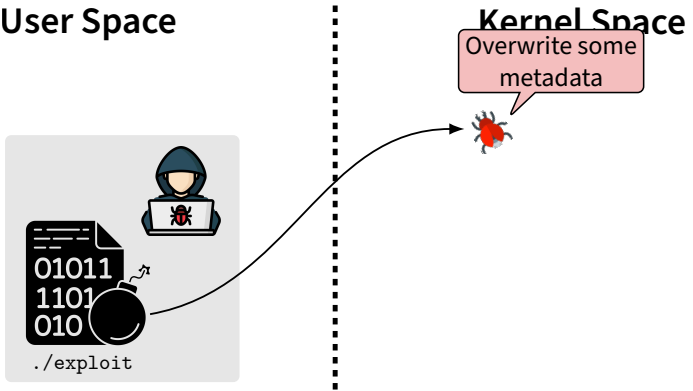
# Problem

- **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!**

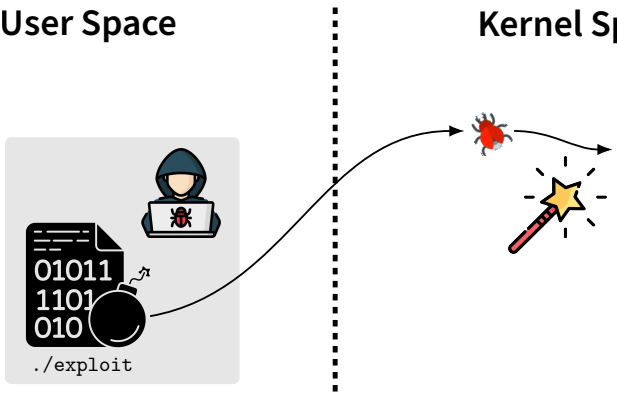
# Problem

- **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

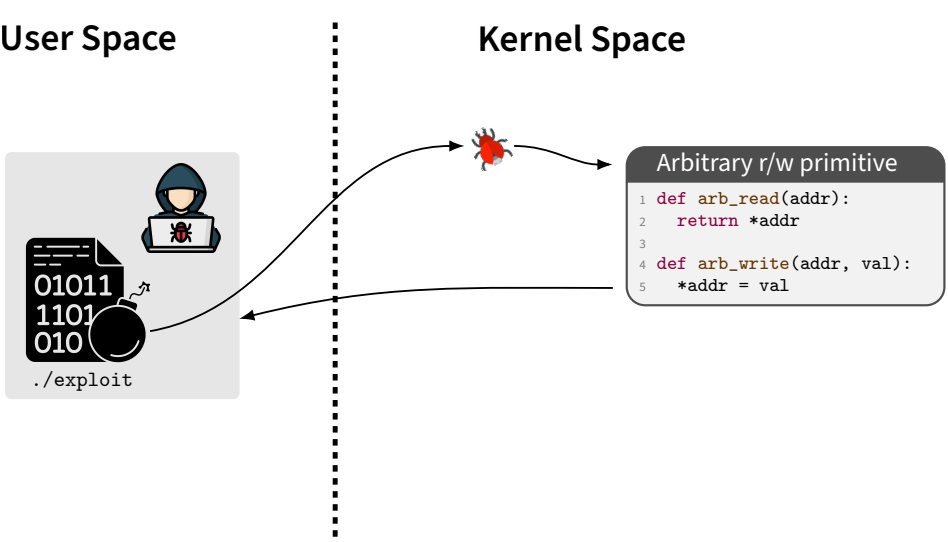
# Magic Wand


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

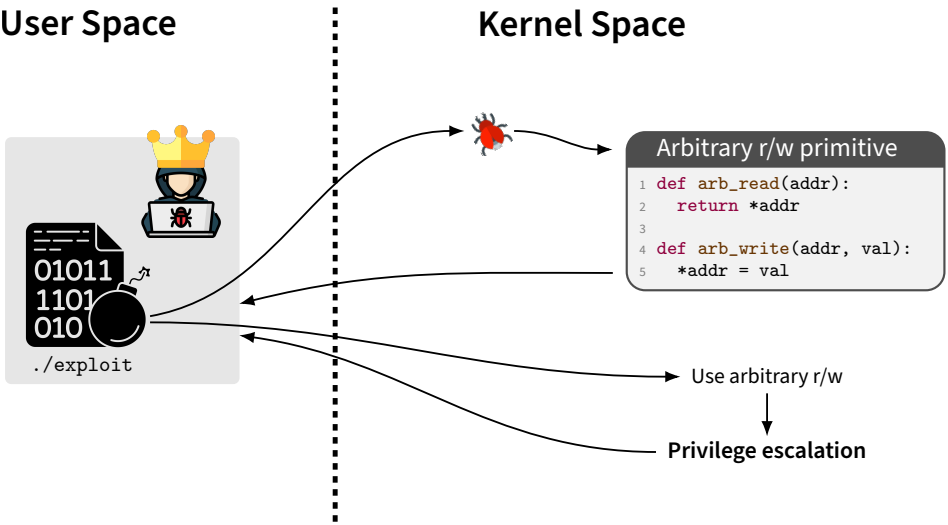
# Magic Wand

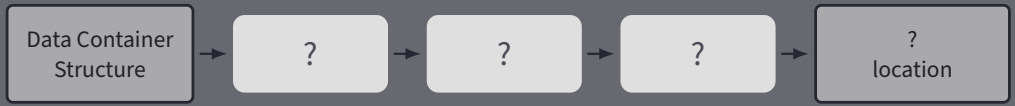


# Magic Wand

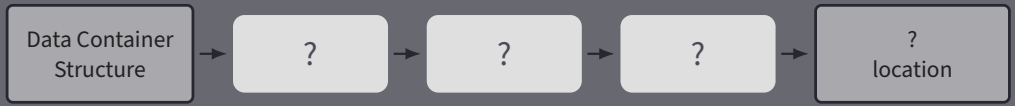

User Space
Kernel Space
01011
1101
010
./exploit
Arbitrary r/w primitive
1 def arb_read(addr):
2 return *addr
3
4 def arb_write(addr, val):
5 *addr = val

# 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


Data Container Structure
?
?
?
? location


Hardware Cache
?
?
?
? location


Data Container
Structure
?
?
?
?
location


Hardware
Cache
?
?
?
?
location

# Linked Lists

Empty list

List with
2 elements

List with
5 elements

# Linked Lists

Empty list

List with
2 elements

List with
5 elements

*Fast access*

# Linked Lists

Empty list

List with
2 elements

List with
5 elements

*Fast access*

*Medium fast access*

# Linked Lists

Empty list

List with
2 elements

List with
5 elements

*Fast access*

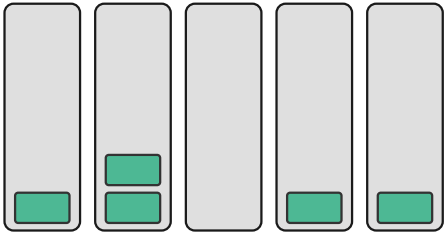
*Medium fast access*

*Slow access*

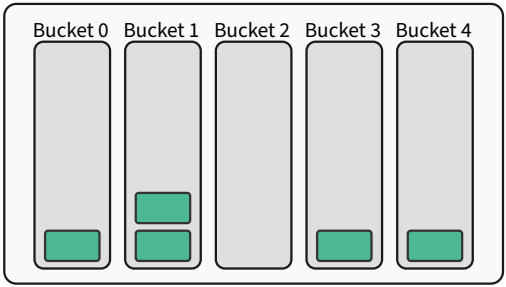
# Hash Tables

# Hash Tables

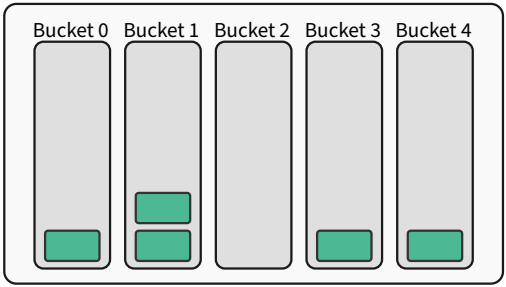
# Hash Tables



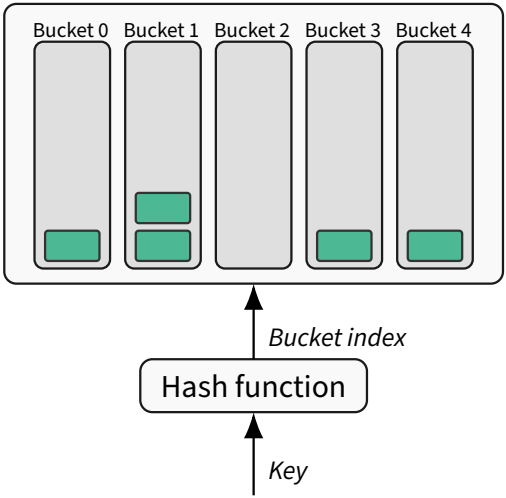
# Hash Tables



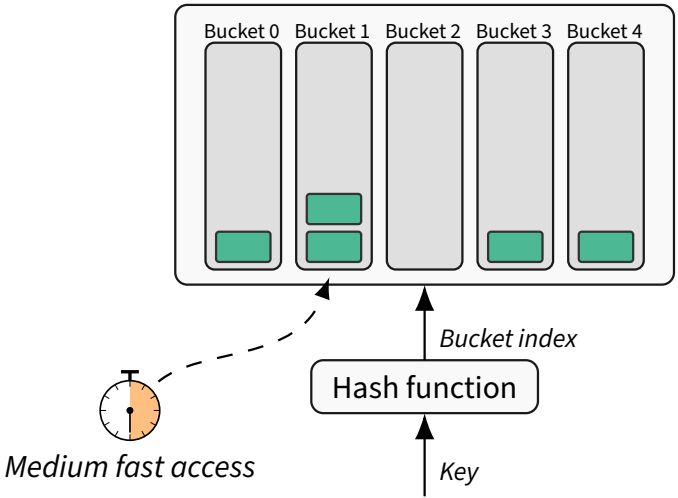
# Hash Tables



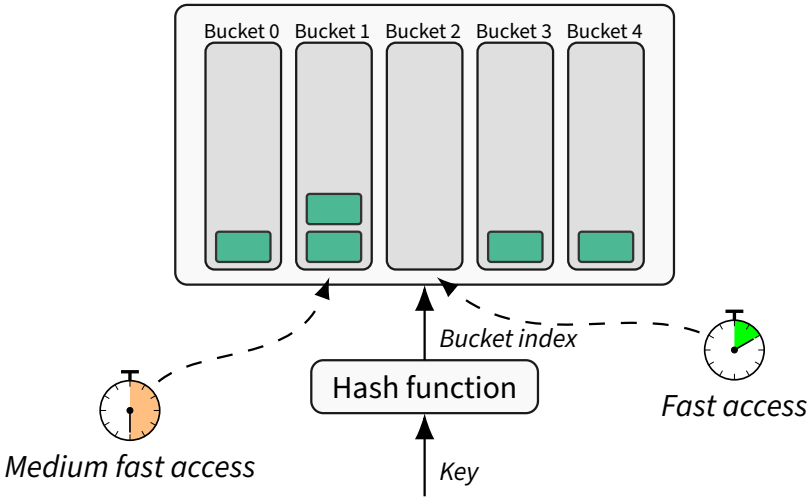
# Hash Tables

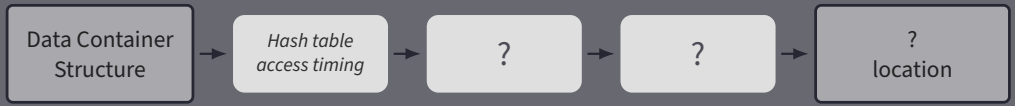

Bucket 0
Bucket 1
Bucket 2
Bucket 3
Bucket 4
Bucket index
Hash function
Key

# Hash Tables


Bucket 0
Bucket 1
Bucket 2
Bucket 3
Bucket 4
Bucket index
Hash function
Key
Medium fast access

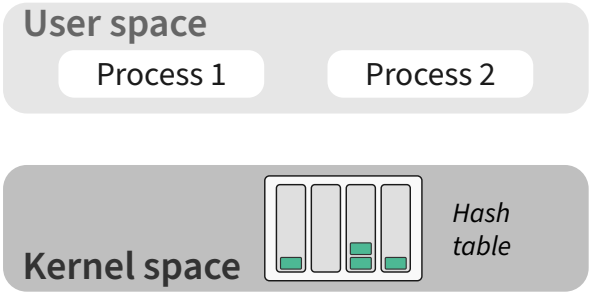
# Hash Tables


Bucket 0
Bucket 1
Bucket 2
Bucket 3
Bucket 4
Bucket index
Hash function
Key
Medium fast access
Fast access

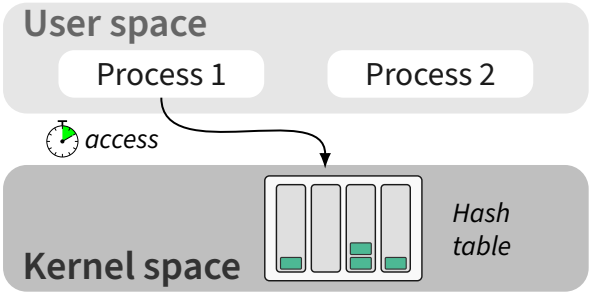

Data Container Structure
Hash table access timing
?
?
? location


Hardware Cache
?
?
?
? location

# Software-Induced Side Channel

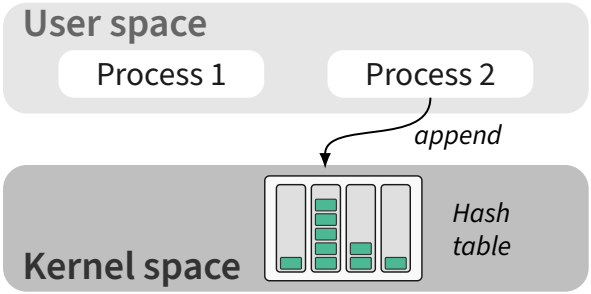


# Software-Induced Side Channel




Process 1 $\rightarrow$ syscall accesses the hash table

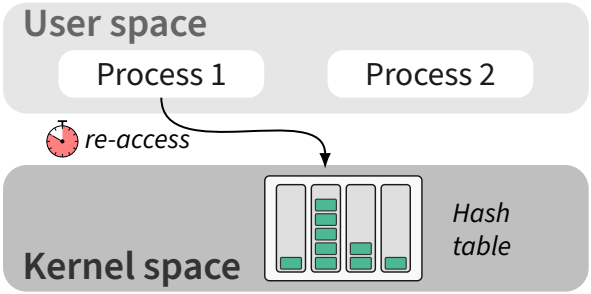
# Software-Induced Side Channel




Process 1 $\rightarrow$ syscall accesses the hash table

Process 2 $\rightarrow$ syscall appends data

# Software-Induced Side Channel

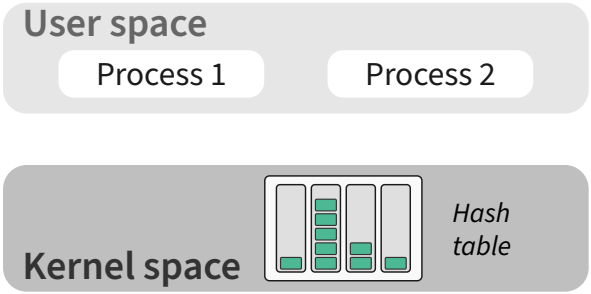



Process 1 $\rightarrow$ syscall accesses the hash table

Process 2 $\rightarrow$ syscall appends data

Process 1 $\rightarrow$ syscall re-accesses the hash table

# Software-Induced Side Channel




Process 1 $\rightarrow$ syscall accesses the hash table

Process 2 $\rightarrow$ syscall appends data

Process 1 $\rightarrow$ syscall re-accesses the hash table

**Deduce security-critical information**

# Primitives

# Primitives

## Access primitive

## Append/remove primitive

# Primitives

**Access primitive**

- Syscalls that access structures

**Append/remove primitive**

# Primitives

## Access primitive

- Syscalls that access structures
- Such as:

**Get POSIX time**

```
def sys_clock_gettime(id):
  sign = current.signal
  h = timer_hash(id, sign)
  hbucket =
    posix_timers_htable[h]
  for tim in hbucket:
    if tim.sign == sign and
       tim.id == id:
      return tim.get_time()
  return ERROR
```

## Append/remove primitive

# Primitives

## Access primitive

- Syscalls that access structures
- Such as:

**Get POSIX time**

```
1 def sys_clock_gettime(id):
2   sign = current.signal
3   h = timer_hash(id, sign)
4   hbucket =
      posix_timers_htable[h]
5   for tim in hbucket:
6     if tim.sign == sign and
7        tim.id == id:
8       return tim.get_time()
9   return ERROR
```

## Append/remove primitive

**Hash**

# Primitives

**Access primitive**

- Syscalls that access structures
- Such as:

Get POSIX time

```
1 def sys_clock_gettime(id):
2   sign = current.signal
3   h = timer_hash(id, sign)
4   hbucket =
      posix_timers_htable[h]
5   for tim in hbucket:
6     if tim.sign == sign and
7        tim.id == id:
8       return tim.get_time()
9   return ERROR
```

**Append/remove primitive**

Hash
Bucket

# Primitives

**Access primitive**

- Syscalls that access structures
- Such as:

Get POSIX time

```
1 def sys_clock_gettime(id):
2   sign = current.signal
3   h = timer_hash(id, sign)
4   hbucket =
      posix_timers_htable[h]
5   for tim in hbucket:
6     if tim.sign == sign and
7        tim.id == id:
8       return tim.get_time()
9   return ERROR
```

Hash

Bucket

Timing leak

**Append/remove primitive**

# Primitives

**Access primitive**

- Syscalls that access structures
- Such as:

Get POSIX time

```
1 def sys_clock_gettime(id):
2   sign = current.signal
3   h = timer_hash(id, sign)
4   hbucket =
      posix_timers_htable[h]
5   for tim in hbucket:
6     if tim.sign == sign and
7        tim.id == id:
8       return tim.get_time()
9   return ERROR
```

Hash

Bucket

Timing leak

**Append/remove primitive**

- Syscalls that modify structures

# Primitives

**Access primitive**

- Syscalls that access structures
- Such as:

Get POSIX time

```
1 def sys_clock_gettime(id):
2   sign = current.signal
3   h = timer_hash(id, sign)
4   hbucket =
      posix_timers_htable[h]
5   for tim in hbucket:
6     if tim.sign == sign and
7        tim.id == id:
8       return tim.get_time()
9   return ERROR
```

Hash
Bucket
Timing leak

**Append/remove primitive**

- Syscalls that modify structures
- Such as:

Create new POSIX timer

```
1 def sys_timer_create():
2   sign = current.signal
3   id = sign.next_id
4   h = timer_hash(id, sign)
5   hbucket =
      posix_timers_htable[h]
6   tim = k_itimer(sign, id)
7   hbucket.append(tim)
8   return id
```

# Primitives

**Access primitive**

- Syscalls that access structures
- Such as:

Get POSIX time

```
1 def sys_clock_gettime(id):
2   sign = current.signal
3   h = timer_hash(id, sign)
4   hbucket =
      posix_timers_htable[h]
5   for tim in hbucket:
6     if tim.sign == sign and
7        tim.id == id:
8       return tim.get_time()
9   return ERROR
```

Hash
Bucket
Timing leak

**Append/remove primitive**

- Syscalls that modify structures
- Such as:

Create new POSIX timer

```
1 def sys_timer_create():
2   sign = current.signal
3   id = sign.next_id
4   h = timer_hash(id, sign)
5   hbucket =
      posix_timers_htable[h]
6   tim = k_itimer(sign, id)
7   hbucket.append(tim)
8   return id
```

# Primitives

**Access primitive**

- Syscalls that access structures
- Such as:

Get POSIX time

```
1 def sys_clock_gettime(id):
2     sign = current.signal
3     h = timer_hash(id, sign)
4     hbucket =
        posix_timers_htable[h]
5     for tim in hbucket:
6       if tim.sign == sign and
7          tim.id == id:
8         return tim.get_time()
9     return ERROR
```

Hash
Bucket
Timing leak

**Append/remove primitive**

- Syscalls that modify structures
- Such as:

Create new POSIX timer

```
1 def sys_timer_create():
2     sign = current.signal
3     id = sign.next_id
4     h = timer_hash(id, sign)
5     hbucket =
        posix_timers_htable[h]
6     tim = k_itimer(sign, id)
7     hbucket.append(tim)
8     return id
```

# Primitives

**Access primitive**

- Syscalls that access structures
- Such as:

Get POSIX time

```
1 def sys_clock_gettime(id):
2     sign = current.signal
3     h = timer_hash(id, sign)
4     hbucket =
        posix_timers_htable[h]
5     for tim in hbucket:
6       if tim.sign == sign and
7          tim.id == id:
8         return tim.get_time()
9     return ERROR
```

Hash
Bucket
Timing leak
Append

**Append/remove primitive**

- Syscalls that modify structures
- Such as:

Create new POSIX timer

```
1 def sys_timer_create():
2     sign = current.signal
3     id = sign.next_id
4     h = timer_hash(id, sign)
5     hbucket =
        posix_timers_htable[h]
6     tim = k_itimer(sign, id)
7     hbucket.append(tim)
8     return id
```

# Timing Side Channel

Timing measurement of syscall

**Side-channel attack**

```
def side_channel_attack():
  times = []
  for id in ids:
    t0 = get_time()
    sys_clock_gettime(id)
    t1 = get_time()
    times.append(t1-t0)
```

# Timing Side Channel

Timing measurement of syscall

**Side-channel attack**

```
1 def side_channel_attack():
2   times = []
3   for id in ids:
4     t0 = get_time()
5     sys_clock_gettime(id)
6     t1 = get_time()
7     times.append(t1-t0)
```

**Invalid IDs**

# Timing Side Channel

Timing measurement of syscall

**Side-channel attack**

```
1 def side_channel_attack():
2   times = []
3   for id in ids:
4     t0 = get_time()
5     sys_clock_gettime(id)
6     t1 = get_time()
7     times.append(t1-t0)
```

**Invalid IDs**

**Leaks occupancy level**

Via syscall timing

# Timing Histogram

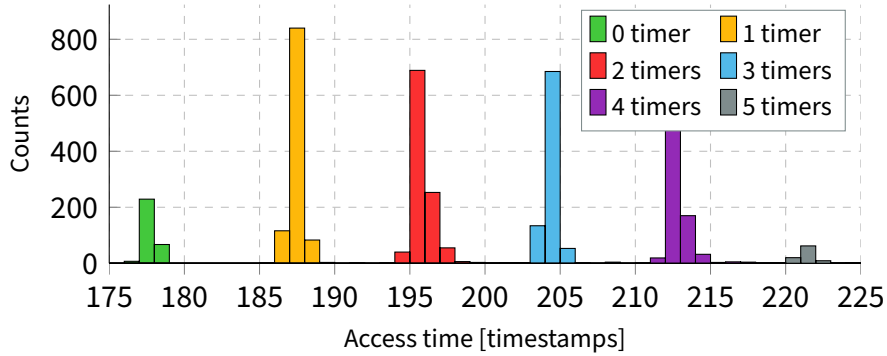
Perform $\sim$ 4000 bucket accesses

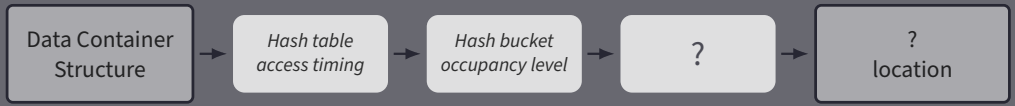
- Hash table storing POSIX timers
- Buckets between 0 and 5 timers

# Timing Histogram

Perform $\sim$ 4000 bucket accesses

- Hash table storing POSIX timers
- Buckets between 0 and 5 timers




Data Container Structure
Hash table access timing
Hash bucket occupancy level
?
? location


Hardware Cache
?
?
?
? location

# Hash Collision Identifier

**Access primitive**

```
index = hash_fn(kaddr, user_id)
bucket = hash_table[index]
for e in bucket
```

# Hash Collision Identifier

**Hash:**

- Known: `hash_fn`, `user_id`
- Unkown: `kaddr`

**Access primitive**

```
1 index = hash_fn(kaddr, user_id)
2 bucket = hash_table[index]
3 for e in bucket
```

# Hash Collision Identifier

**Access primitive**

```
1 index = hash_fn(kaddr, user_id)
2 bucket = hash_table[index]
3 for e in bucket
```

**Hash:**

- Known: `hash_fn`, `user_id`
- Unkown: `kaddr`

**Exploit strategy (online phase):**

- Same `kaddr`, but different `user_ids`
- Use our side channel to detect hash collisions

# Hash Collision Identifier

**Access primitive**

```
1 index = hash_fn(kaddr, user_id)
2 bucket = hash_table[index]
3 for e in bucket
```

**Hash:**

- Known: `hash_fn`, `user_id`
- Unkown: `kaddr`

**Exploit strategy (online phase):**

- Same `kaddr`, but different `user_ids`
- Use our side channel to detect hash collisions

**Bruteforce (offline phase):**

- Test all possible `kaddrs`
- Match hash collisions

# Attack Scenario

# Attack Scenario

- Target data structure: futex hash table
  - Stores kernel metadata for fast userspace mutexes
  - `user_id`: user address
  - `kaddr`: `mm_struct`

# Attack Scenario

- Target data structure: futex hash table
  - Stores kernel metadata for fast userspace mutexes
  - `user_id`: user address
  - `kaddr`: `mm_struct`
- Target system: x86_64
  - Similar applies to AArch64 and RISC-V

# Attack Scenario

- Target data structure: futex hash table
  - Stores kernel metadata for fast userspace mutexes
  - `user_id`: user address
  - `kaddr`: `mm_struct`
- Target system: x86_64
  - Similar applies to AArch64 and RISC-V
- Leak in less than 1 min

# Primitives

**Access primitive**

**Append/remove primitive**

# Primitives

## Access primitive

**Wake up thread**

```
1 def sys_futex_wake(uaddr):
2   mm = current.mm
3   h = futex_hash(uaddr, mm)
4   hbucket = futex_hash_tables[h]
5   for fqueue in hbucket:
6     if fqueue.mm == mm and
7        fqueue.uaddr == uaddr:
8       fqueue.wake()
```

## Append/remove primitive

# Primitives

## Access primitive

**Wake up thread**

```
def sys_futex_wake(uaddr):
  mm = current.mm
  h = futex_hash(uaddr, mm)
  hbucket = futex_hash_tables[h]
  for fqueue in hbucket:
    if fqueue.mm == mm and
       fqueue.uaddr == uaddr:
      fqueue.wake()
```

## Append/remove primitive

**Wait for thread**

```
def sys_futex_wait(uaddr):
  mm = current.mm
  h = futex_hash(uaddr, mm)
  hbucket = futex_hash_tables[h]
  fqueue = futex_q(uaddr, mm)
  hbucket.append(fqueue)
  fqueue.wait()
```

# Hash Collision Detection

Find 15 ids that cause a hash collision

```
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
```

# Hash Collision Detection

## Find 15 ids that cause a hash collision

```
1  futexes = new array(1<<30) // user identifiers for primitives
2  collisions = []            // found collisions
3  def find_collisions():
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
```

# Hash Collision Detection

## Find 15 ids that cause a hash collision

```
1  futexes = new array(1<<30) // user identifiers for primitives
2  collisions = []            // found collisions
3  def find_collisions():
4
5    flush_cpu_caches()
6    t0 = get_time()
7    sys_futex_wake(0xdeadbeef)
8    t1 = get_time()
9    threshold_time = 10*(t1-t0)
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
```

# Hash Collision Detection

## Find 15 ids that cause a hash collision

```
futexes = new array(1<<30) // user identifiers for primitives
collisions = []            // found collisions
def find_collisions():

  flush_cpu_caches()
  t0 = get_time()
  sys_futex_wake(0xdeadbeef)
  t1 = get_time()
  threshold_time = 10*(t1-t0)

  for i in range(4096):
    sys_futex_wait(futexes[0])
  collisions.append(futexes[0])
```

# Hash Collision Detection

## Find 15 ids that cause a hash collision

```
1  futexes = new array(1<<30) // user identifiers for primitives
2  collisions = []            // found collisions
3  def find_collisions():
4
5    flush_cpu_caches()
6    t0 = get_time()
7    sys_futex_wake(0xdeadbeef)
8    t1 = get_time()
9    threshold_time = 10*(t1-t0)
10
11   for i in range(4096):
12     sys_futex_wait(futexes[0])
13   collisions.append(futexes[0])
14
15   fid = 1
16   while sizeof(collisions) < 16:
17
18
19
20
21
22
23
24
25
26     fid++
```

# Hash Collision Detection

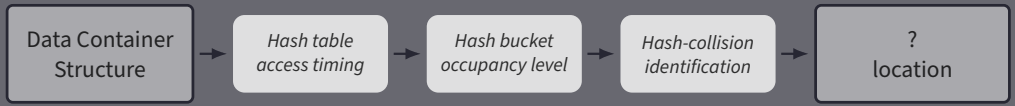
## Find 15 ids that cause a hash collision

```
1  futexes = new array(1<<30) // user identifiers for primitives
2  collisions = []            // found collisions
3  def find_collisions():
4
5    flush_cpu_caches()
6    t0 = get_time()
7    sys_futex_wake(0xdeadbeef)
8    t1 = get_time()
9    threshold_time = 10*(t1-t0)
10
11   for i in range(4096):
12     sys_futex_wait(futexes[0])
13   collisions.append(futexes[0])
14
15   fid = 1
16   while sizeof(collisions) < 16:
17
18     flush_cpu_caches()
19     t0 = get_time()
20     sys_futex_wake(futexes[fid])
21     t1 = get_time()
22
23
24
25
26     fid++
```

# Hash Collision Detection

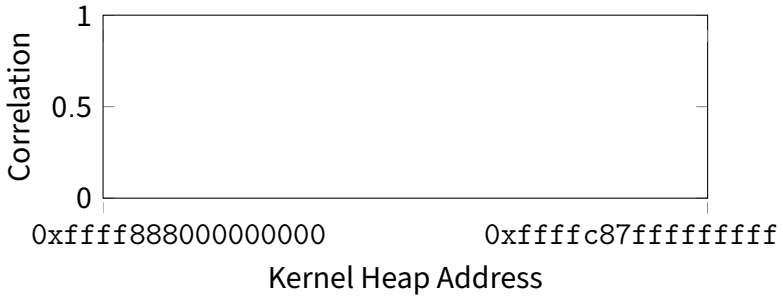
## Find 15 ids that cause a hash collision

```
1  futexes = new array(1<<30) // user identifiers for primitives
2  collisions = []            // found collisions
3  def find_collisions():
4
5    flush_cpu_caches()
6    t0 = get_time()
7    sys_futex_wake(0xdeadbeef)
8    t1 = get_time()
9    threshold_time = 10*(t1-t0)
10
11   for i in range(4096):
12     sys_futex_wait(futexes[0])
13   collisions.append(futexes[0])
14
15   fid = 1
16   while sizeof(collisions) < 16:
17
18     flush_cpu_caches()
19     t0 = get_time()
20     sys_futex_wake(futexes[fid])
21     t1 = get_time()
22
23     if (t1-t0) > threshold_time:
24       collisions.append(futexes[fid])
25
26     fid++
```


Data Container Structure
Hash table access timing
Hash bucket occupancy level
Hash-collision identification
? location


Hardware Cache
?
?
?
? location

# Bruteforce Kernel Address

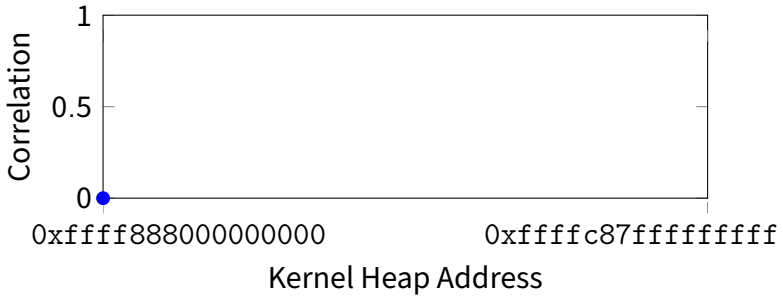



| uaddr | coll. |
|---|---|
| 0x501008 | - |
| 0x503010 | - |
| 0x5041f0 | - |
| 0x507b10 | - |
| 0x5090a8 | - |
| ... | ... |

**Bruteforce attack**

```
1 for uaddr in uaddrs:
2   found &= (index == futex_hash(kaddr, uaddr))
```

# Bruteforce Kernel Address

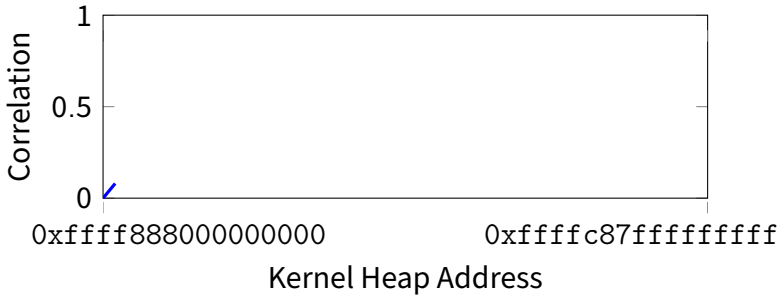



| uaddr | coll. |
|---|---|
| 0x501008 | ✗ |
| 0x503010 | ✗ |
| 0x5041f0 | ✗ |
| 0x507b10 | ✗ |
| 0x5090a8 | ✗ |
| ... | ... |

**Bruteforce attack**

```
1 for uaddr in uaddrs:
2   found &= (index == futex_hash(0xffff888000000000, uaddr))
```

# Bruteforce Kernel Address

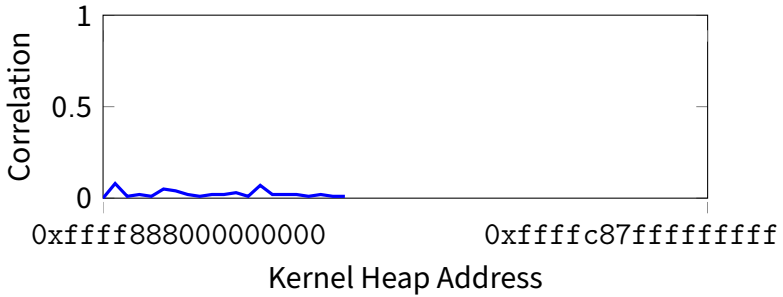



| uaddr | coll. |
|---|---|
| 0x501008 | ✓ |
| 0x503010 | ✓ |
| 0x5041f0 | ✗ |
| 0x507b10 | ✗ |
| 0x5090a8 | ✗ |
| ... | ... |

**Bruteforce attack**

```
1 for uaddr in uaddrs:
2   found &= (index == futex_hash(0xffff888000000580, uaddr))
```

# Bruteforce Kernel Address

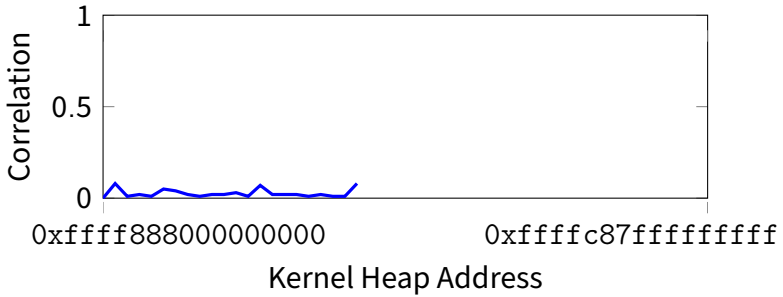



| uaddr | coll. |
|---|---|
| 0x501008 | ✗ |
| 0x503010 | ✗ |
| 0x5041f0 | ✗ |
| 0x507b10 | ✗ |
| 0x5090a8 | ✗ |
| ... | ... |

**Bruteforce attack**

```
1 for uaddr in uaddrs:
2   found &= (index == futex_hash(0xffff8880017cf080, uaddr))
```

# Bruteforce Kernel Address

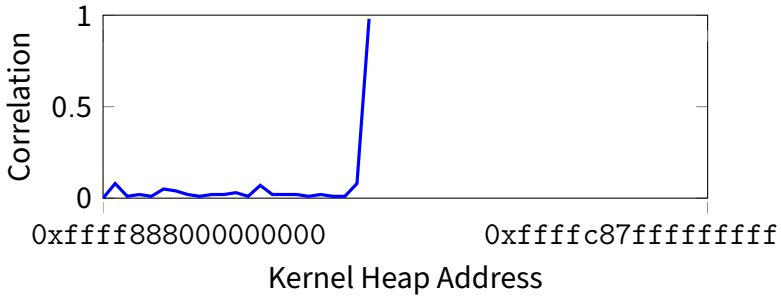



| uaddr | coll. |
|---|---|
| 0x501008 | ✗ |
| 0x503010 | ✓ |
| 0x5041f0 | ✗ |
| 0x507b10 | ✗ |
| 0x5090a8 | ✓ |
| ... | ... |

**Bruteforce attack**

```
1 for uaddr in uaddrs:
2   found &= (index == futex_hash(0xffff8880017cf600, uaddr))
```

# Bruteforce Kernel Address

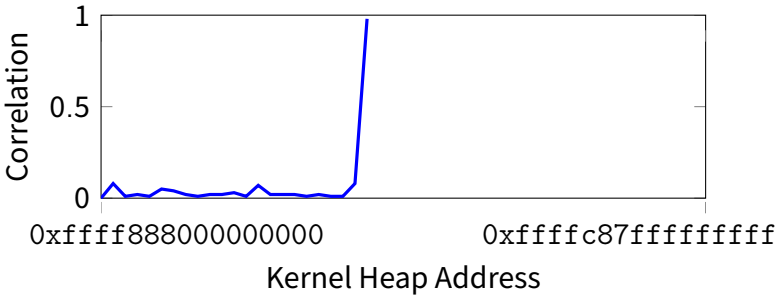



| uaddr | coll. |
|---|---|
| 0x501008 | ✓ |
| 0x503010 | ✓ |
| 0x5041f0 | ✓ |
| 0x507b10 | ✓ |
| 0x5090a8 | ✓ |
| ... | ... |

**Bruteforce attack**

```
1 for uaddr in uaddrs:
2   found &= (index == futex_hash(0xffff8880017cfb80, uaddr))
```

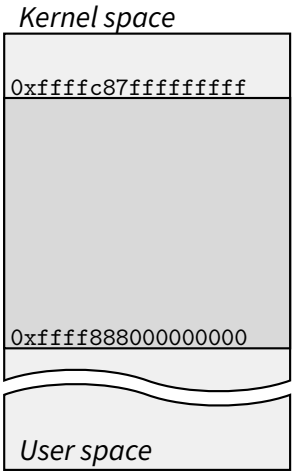
# Bruteforce Kernel Address




**Bruteforce attack**

```
1 for uaddr in uaddrs:
2   found &= (index == futex_hash(0xffff8880017cfb80, uaddr))
```

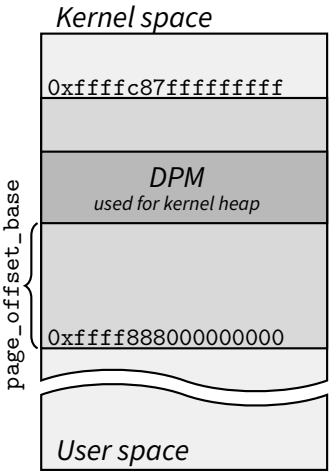
# Search Space Reduction




*Entropy:*

$2^{46}$

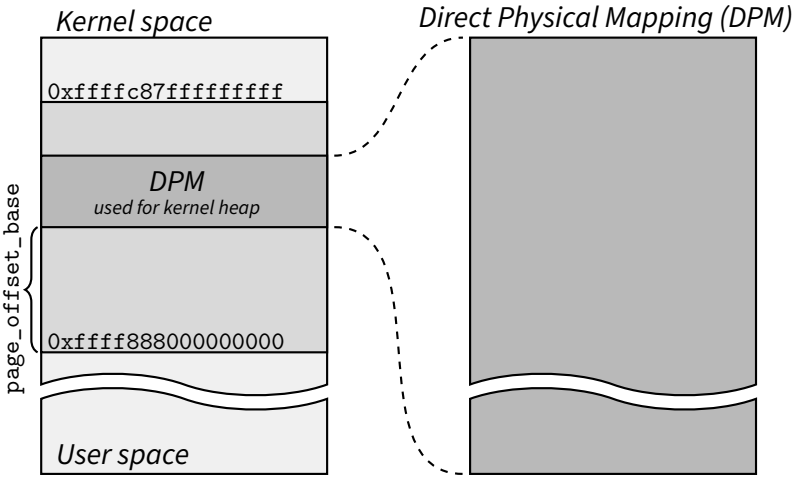
# Search Space Reduction




*Entropy:*

$2^{46}$

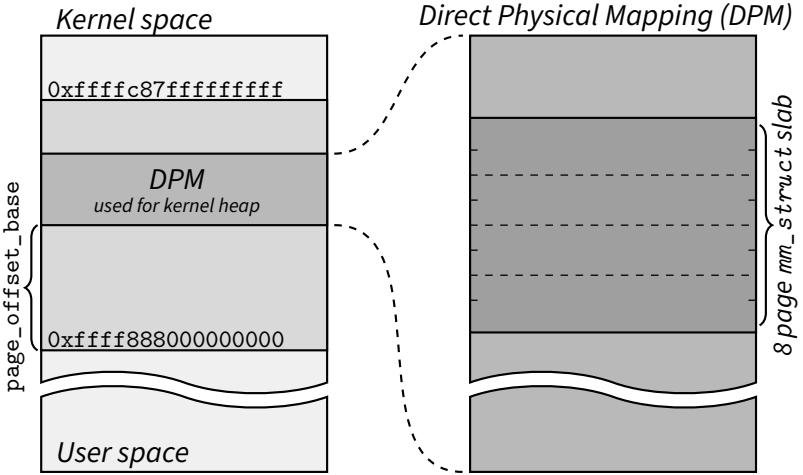
# Search Space Reduction




*Entropy:*

$2^{46}$

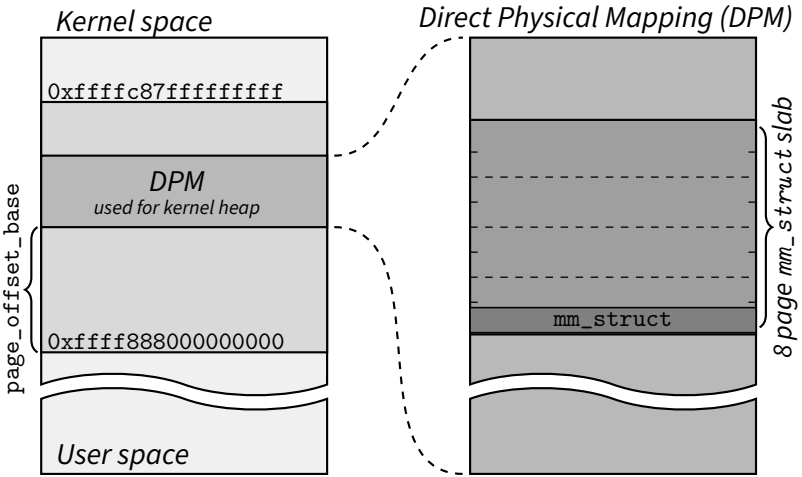
# Search Space Reduction




*Entropy:*

$$2^{46-3-12} \cdot x$$

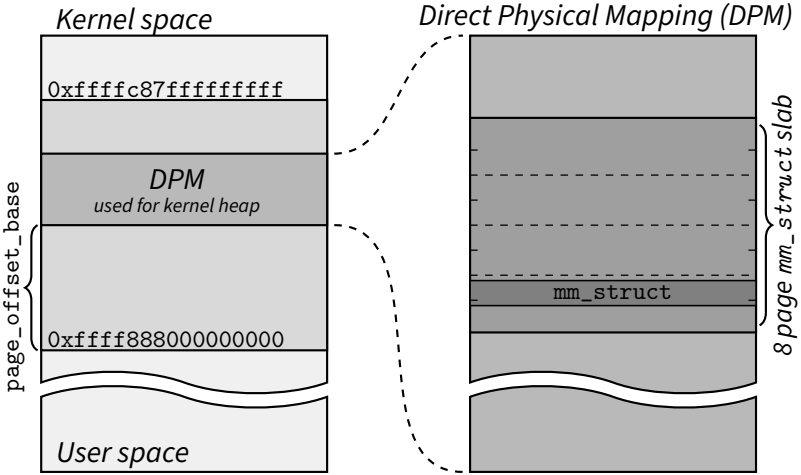
# Search Space Reduction




*Entropy:*

$$2^{46-3-12} \cdot x$$

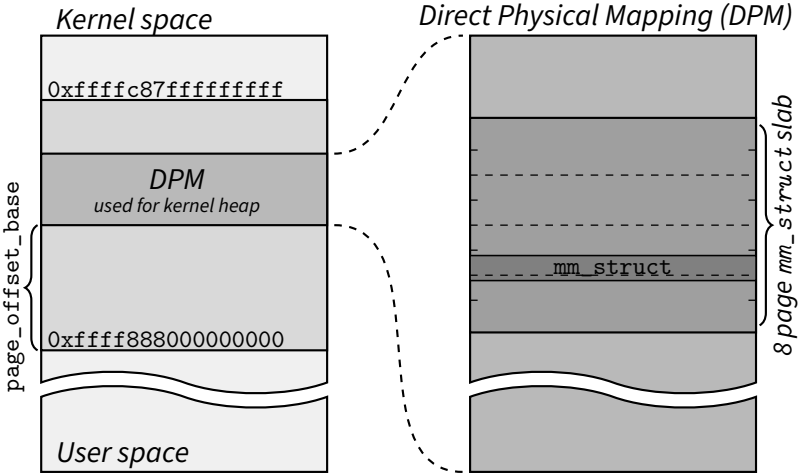
# Search Space Reduction




*Entropy:*

$$2^{46-3-12} \cdot x$$

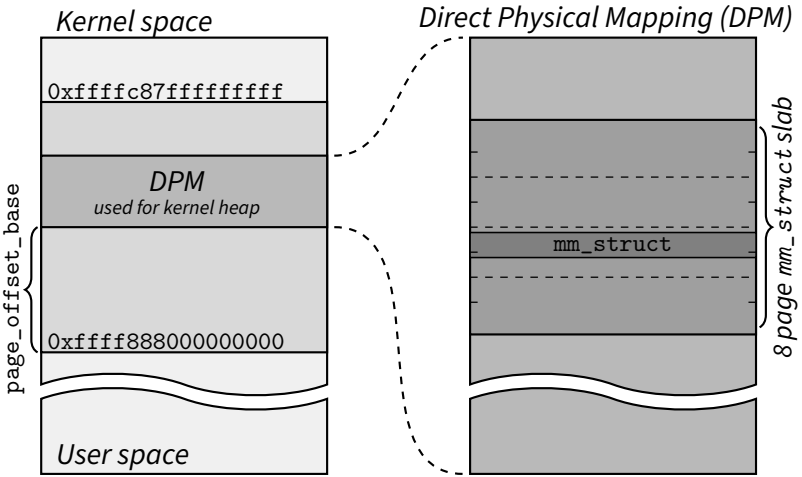
# Search Space Reduction




*Entropy:*

$$2^{46-3-12} \cdot x$$

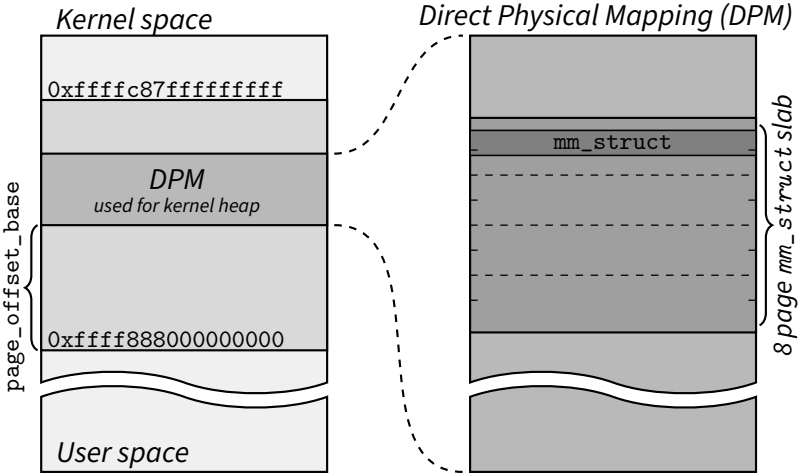
# Search Space Reduction




*Entropy:*

$$2^{46-3-12} \cdot x$$

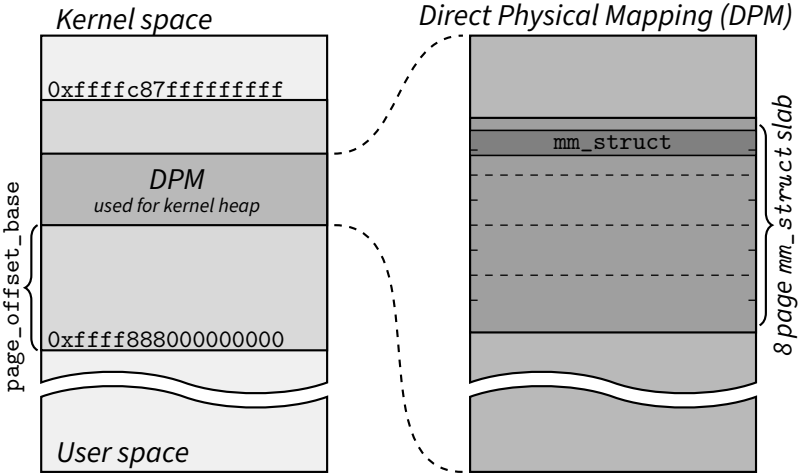
# Search Space Reduction




*Entropy:*

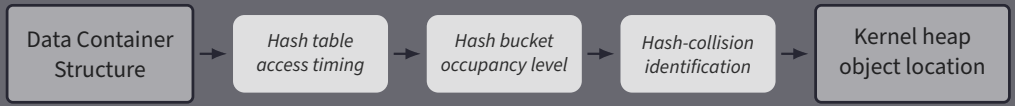
$2^{46-3-12} \cdot 23$

# Search Space Reduction




*Entropy:*

$2^{35.5}$


Data Container Structure
Hash table access timing
Hash bucket occupancy level
Hash-collision identification
Kernel heap object location


Hardware Cache
?
?
?
? location

# System Dependencies

# System Dependencies

- Target: Ubuntu kernel `6.8.0-52-generic`

# System Dependencies

- Target: Ubuntu kernel ~~`6.8.0-52-generic`~~ `6.8.0-79-generic`

# System Dependencies

- Target: Ubuntu kernel ~~`6.8.0-52-generic`~~ `6.8.0-79-generic`
- **Slab page order size?**
- `mm_struct` **size?**

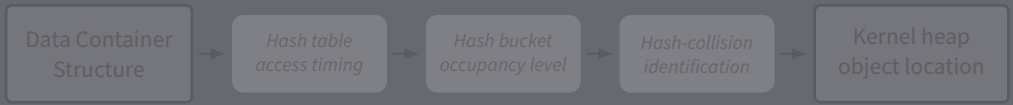
# System Dependencies

- Target: Ubuntu kernel ~~`6.8.0-52-generic`~~ `6.8.0-79-generic`
- **Slab page order size?**
- `mm_struct` **size?**
- `/sys/kernel/slab/mm_struct> grep .* *`

**Output**

```
...
objs_per_slab:23 # objects per slab
order:3          # page order
...
slab_size:1408   # size of mm_struct
...
```

# Let’s leak some kernel addresses

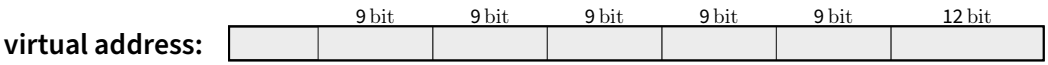

Data Container Structure
Hash table access timing
Hash bucket occupancy level
Hash-collision identification
Kernel heap object location


Hardware Cache
?
?
?
? location

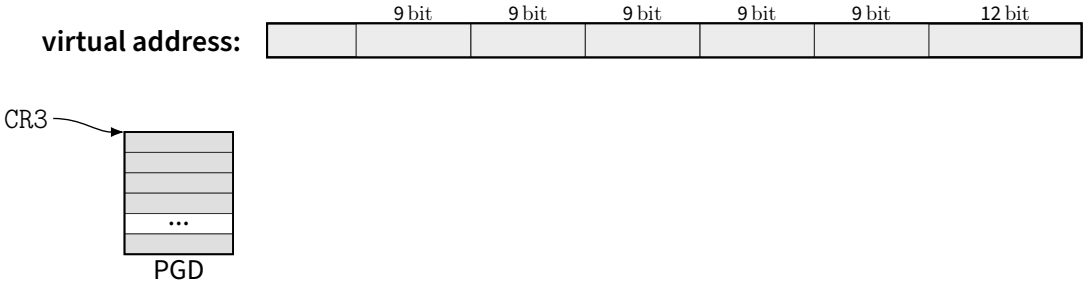
# Address Translation

**virtual address:** 

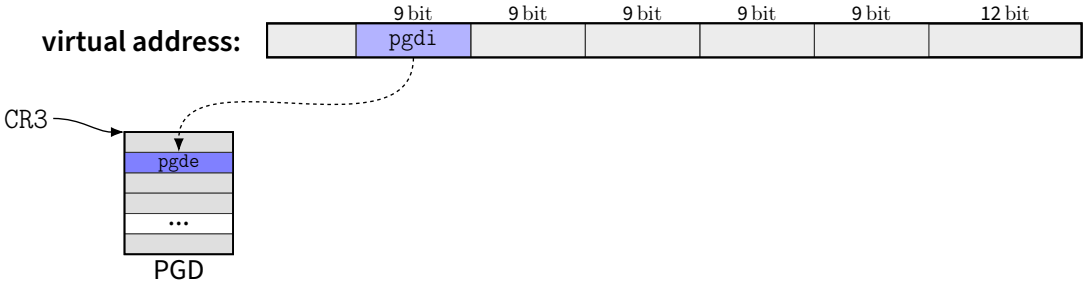
# Address Translation



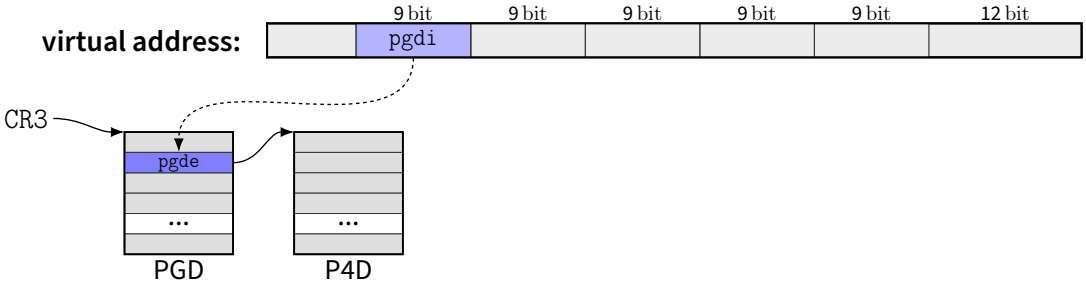
# Address Translation



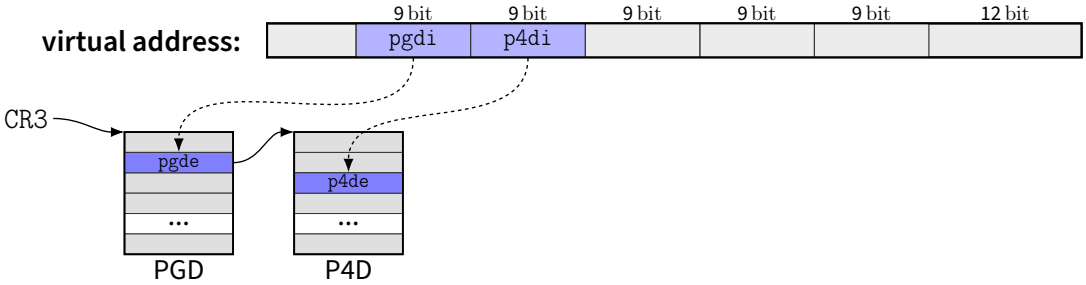
# Address Translation



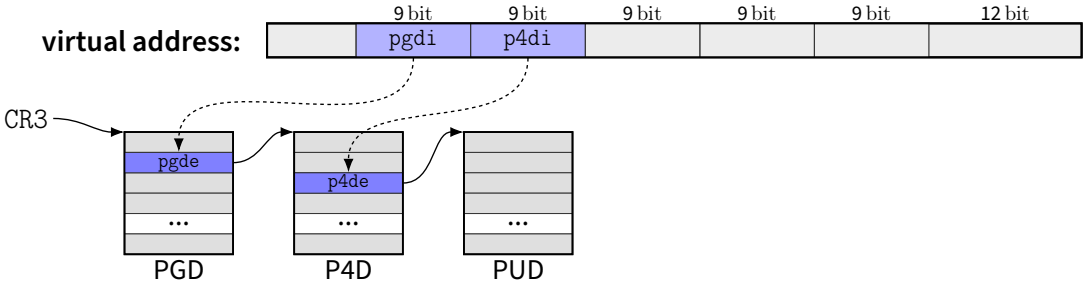
# Address Translation



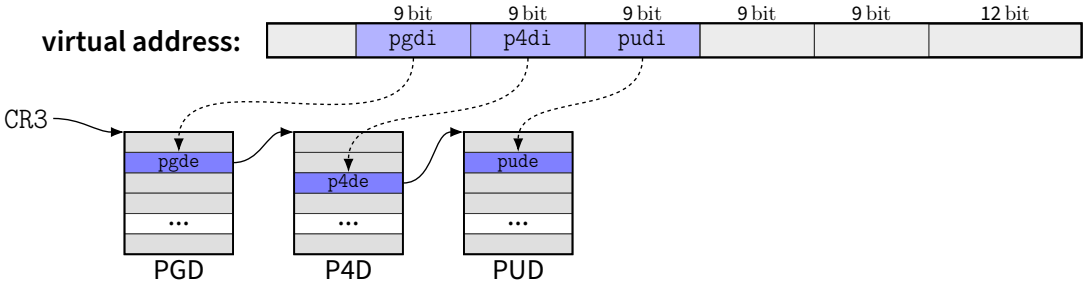
# Address Translation



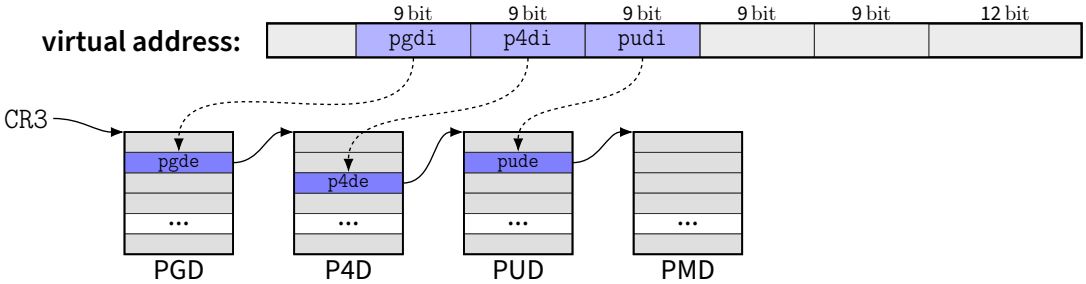
# Address Translation



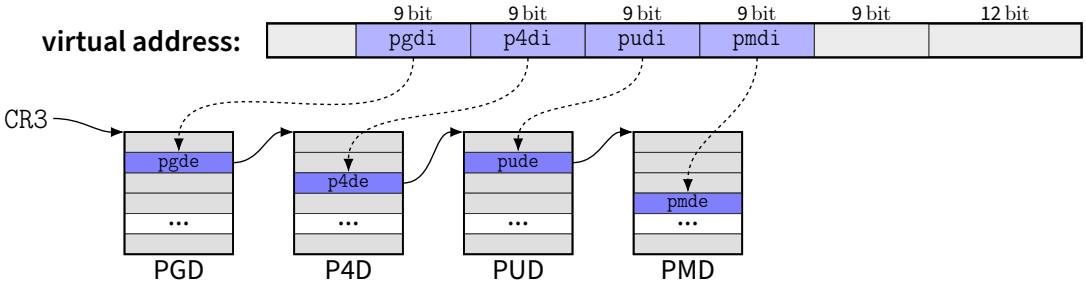
# Address Translation



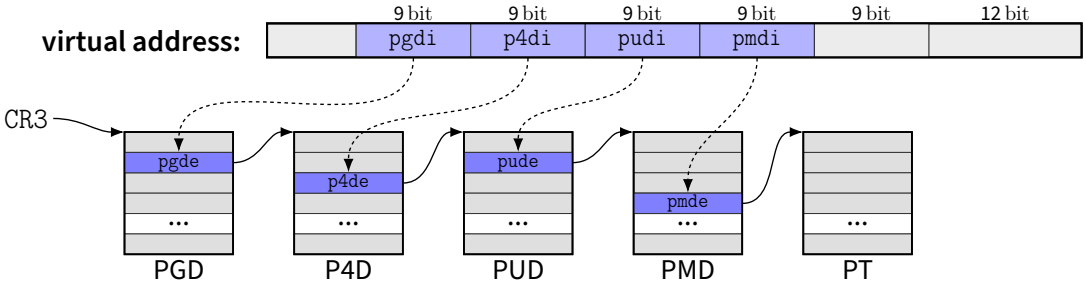
# Address Translation



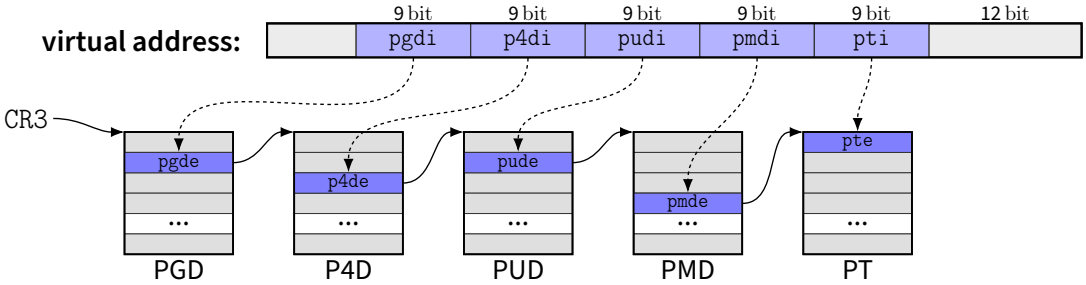
# Address Translation



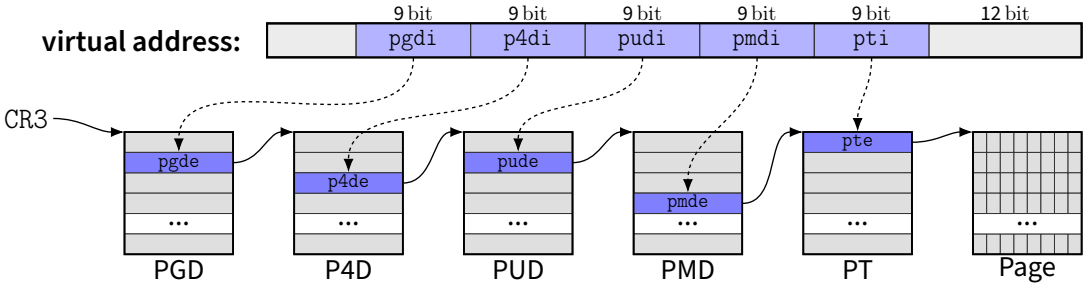
# Address Translation



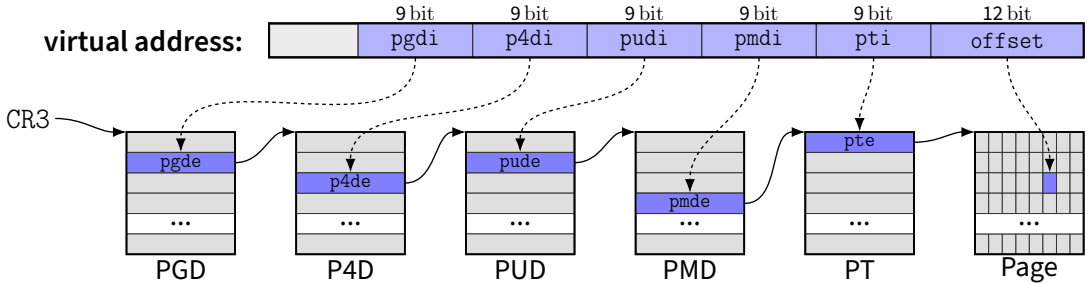
# Address Translation



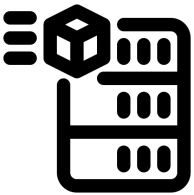
# Address Translation



# Address Translation

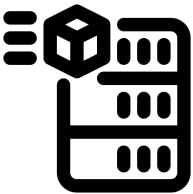


# Translation Lookaside Buffer



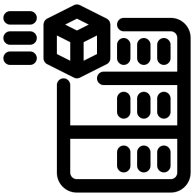
- Each page-table access $\rightarrow$ memory fetch
  - Slow!

# Translation Lookaside Buffer

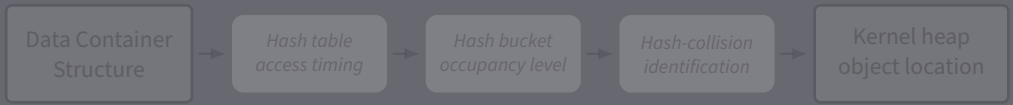


- Each page-table access $\rightarrow$ memory fetch
  - Slow!
- TLB
  - CPU cache
  - Stores page-table entries

# Translation Lookaside Buffer

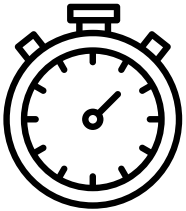


- Each page-table access $\rightarrow$ memory fetch
  - Slow!
- TLB
  - CPU cache
  - Stores page-table entries
- Faster!


Data Container Structure
Hash table access timing
Hash bucket occupancy level
Hash-collision identification
Kernel heap object location

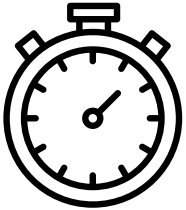

Hardware Cache
TLB access timings
?
?
? location

# TLB Timing Side Channel



- Measure an access:

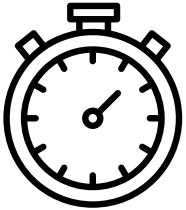
# TLB Timing Side Channel



- Measure an access:

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

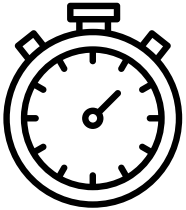
# TLB Timing Side Channel



- Measure an access:

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

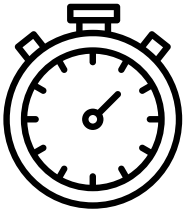
# TLB Timing Side Channel



- Measure an access:

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

# TLB Timing Side Channel

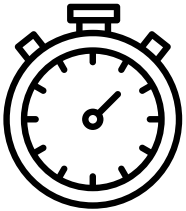


- Measure an access:

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

- How to measure kernel pages?

# TLB Timing Side Channel



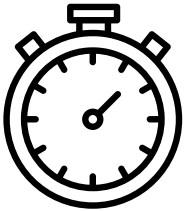
- Measure an access:

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

- How to measure kernel pages?

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

# TLB Timing Side Channel



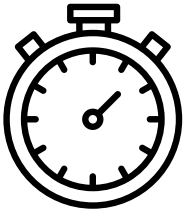
- Measure an access:

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

- How to measure kernel pages?

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

# TLB Timing Side Channel



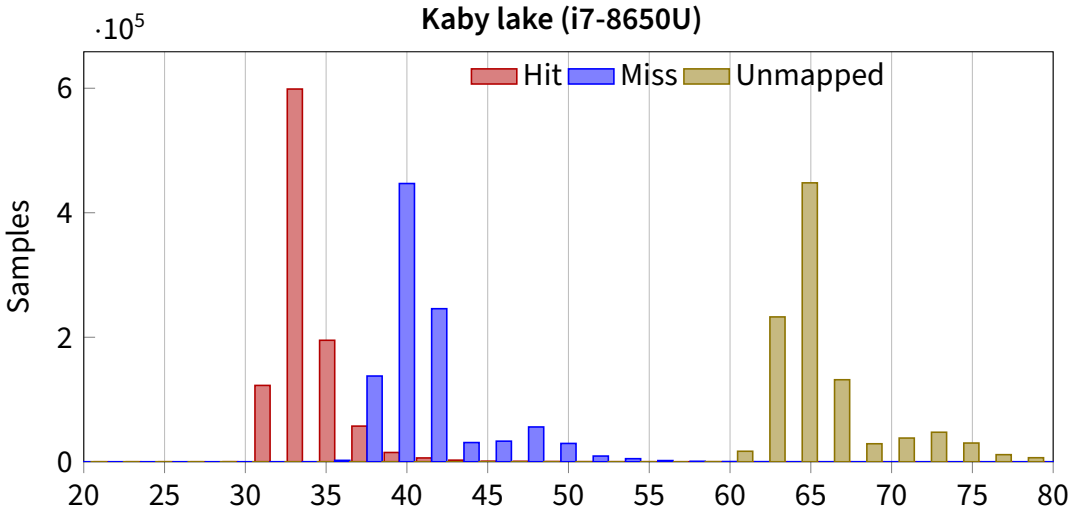
- Measure an access:

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

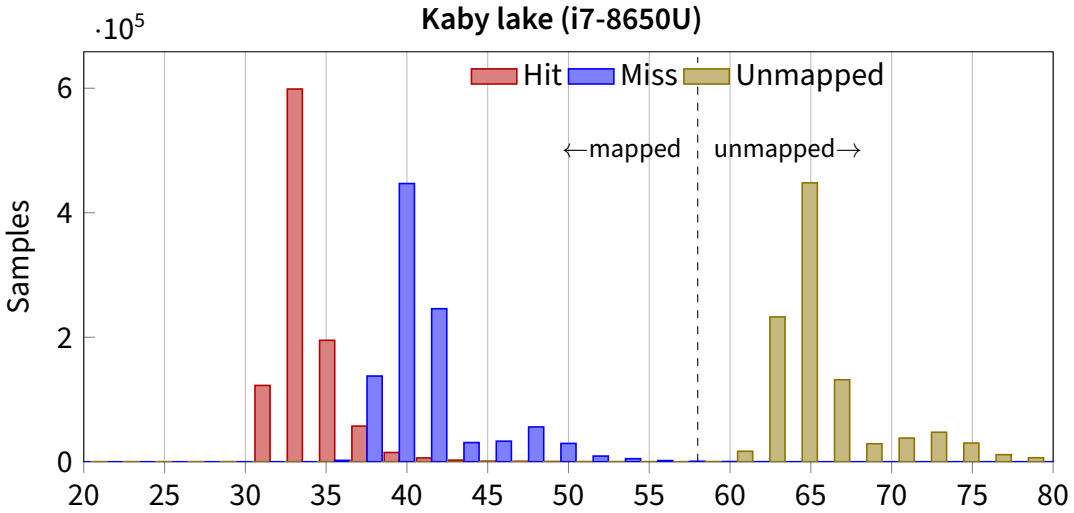
- How to measure kernel pages?

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

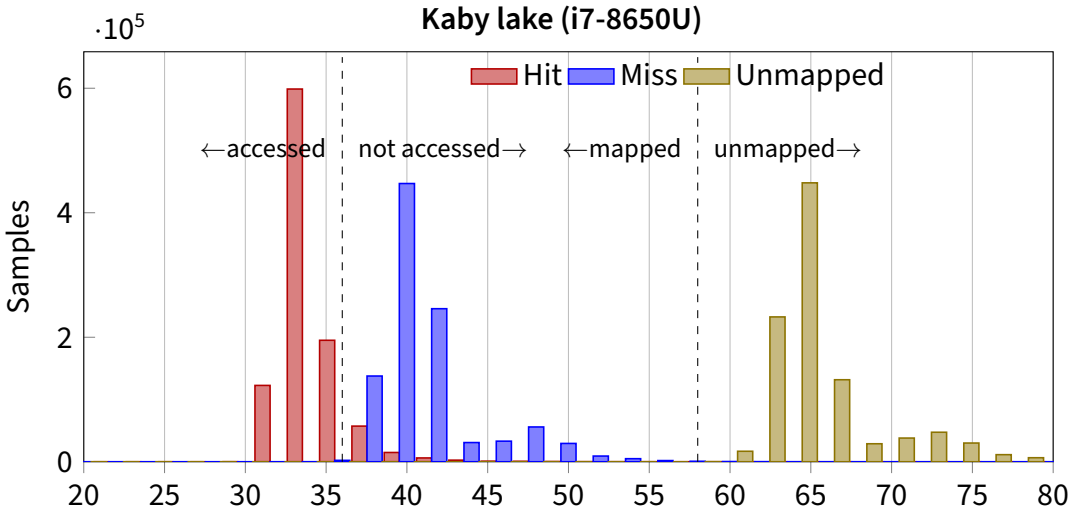
# TLB Timing Histogram



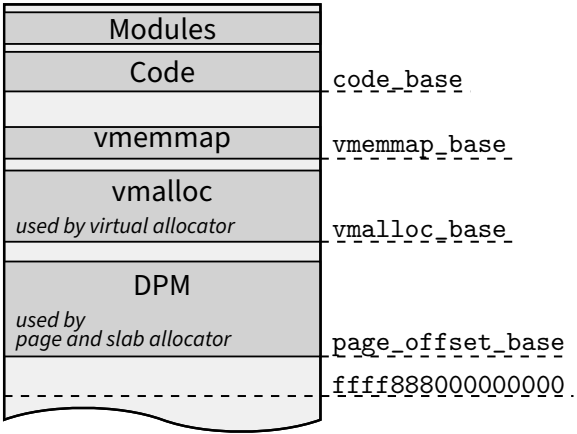
# TLB Timing Histogram



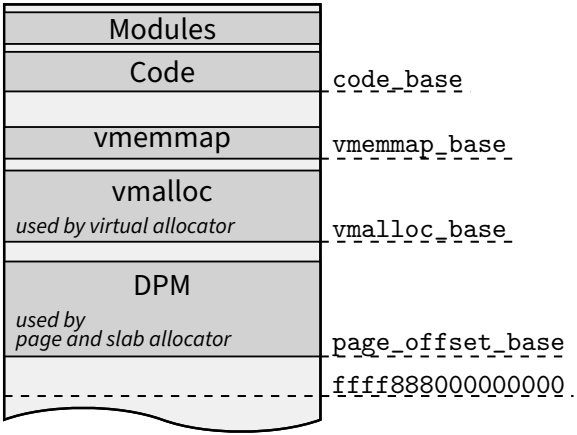
# TLB Timing Histogram



# TLB Memory Mapping Leakage

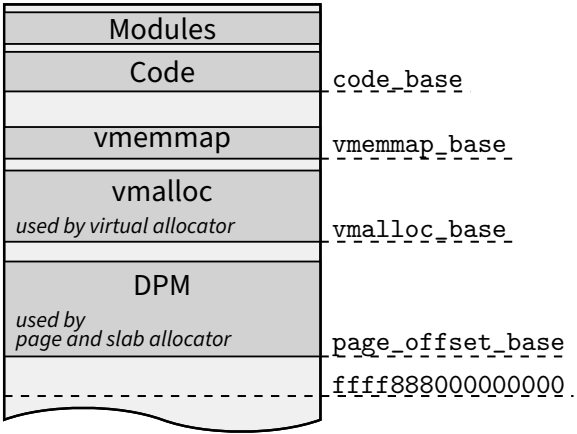


# TLB Memory Mapping Leakage



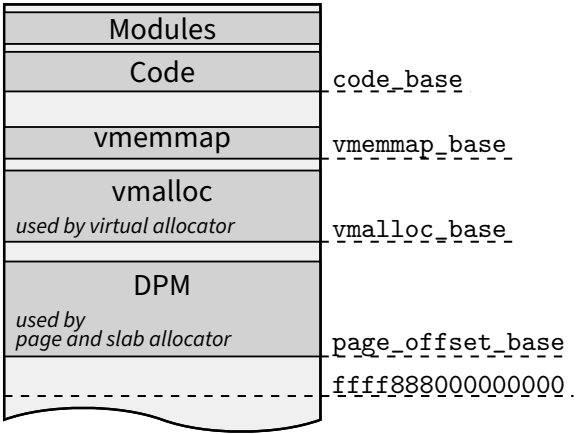

- Leak region bases

# TLB Memory Mapping Leakage

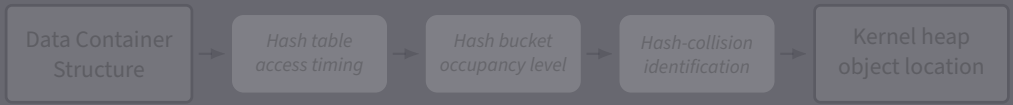



- Leak region bases
- Leak access location

# TLB Memory Mapping Leakage

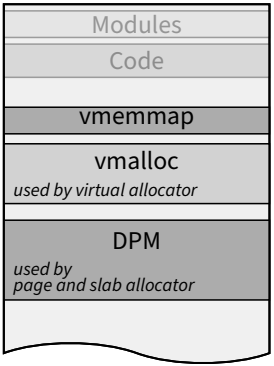



- Leak region bases
- Leak access location
- Page granularity, 2 MB?


Data Container Structure
Hash table access timing
Hash bucket occupancy level
Hash-collision identification
Kernel heap object location

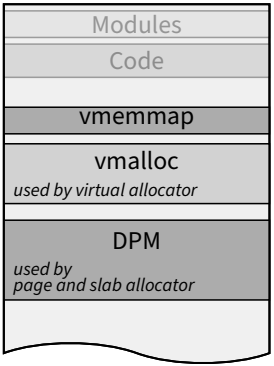

Hardware Cache
TLB access timings
2 MB-aligned memory sections
?
? location

# Enforcing 4 kB Memory Mappings




No Mapping
4kB Mapping
2MB Mapping

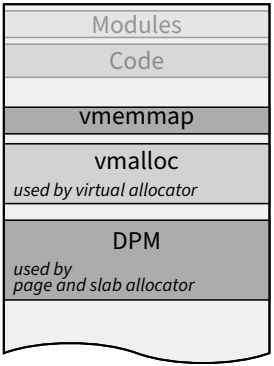
# Enforcing 4 kB Memory Mappings




No Mapping
4kB Mapping
2MB Mapping

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

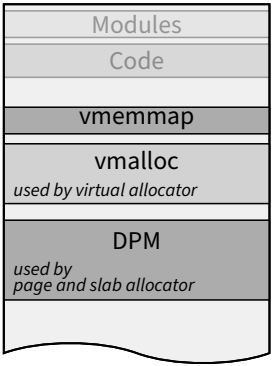
# Enforcing 4 kB Memory Mappings




No Mapping
4kB Mapping
2MB Mapping

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

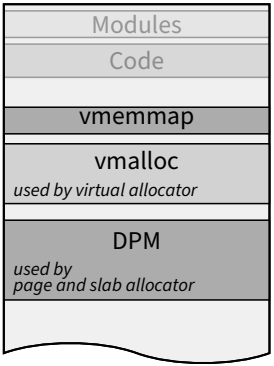
# Enforcing 4 kB Memory Mappings




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`.

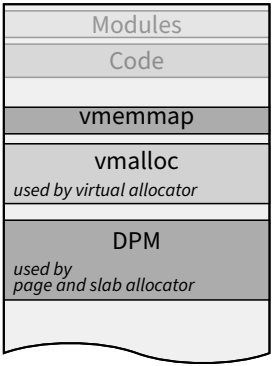
# Enforcing 4 kB Memory Mappings




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`.
  - `CONFIG_SLAB_VIRTUAL`:
    Virtualize heap on 4 kB mappings.

# 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`.
  - `CONFIG_SLAB_VIRTUAL`:
    Virtualize heap on 4 kB mappings.
  - `CONFIG_STRICT_MODULE_RWX`:
    Split DPM to 4 kB mappings.

# 4 kB Access Primitive

# 4 kB Access Primitive

- Syscalls to load 4 kB-aligned kernel address:

# 4 kB Access Primitive

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

# 4 kB Access Primitive

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

```
syscall(-1)
```

# 4 kB Access Primitive

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

# 4 kB Access Primitive

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

# 4 kB Access Primitive

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

# 4 kB Access Primitive

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

# 4 kB Access Primitive

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

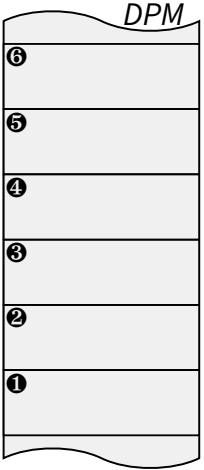
# 4 kB Access Primitive

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

# 4 kB Access Primitive

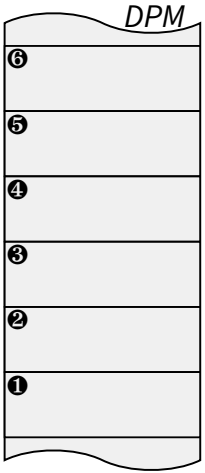
- Syscalls to load 4 kB-aligned kernel address:
  - Kernel stack:
    `syscall(-1)`
  - `msg_msg`:
    `sys_msgrcv`
  - `pipe_buffer`:
    `sys_read`
  - Page tables:
    `sys_mprotect`
  - ...
- Multiple addresses are loaded to the TLB ☹

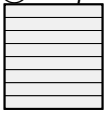
# Leaking 4 kB-Aligned Address




```
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

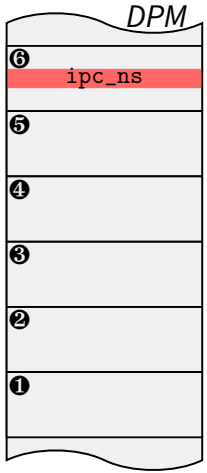


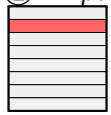




```
sys_msgrcv(id, mtext, mtype):
  queue = ipc_ns.root_rt[id]
  msg = find_msg(queue, mtype)
  copy_to_user(mtext, msg.mtext)
```

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

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

# Leaking 4 kB-Aligned Address

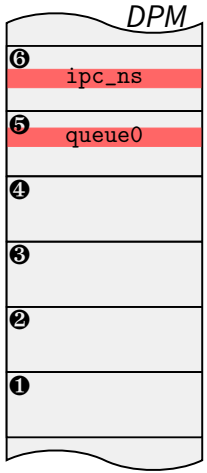


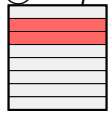




```
sys_msgrcv(id, mtext, mtype):
    queue = ipc_ns.root_rt[id]
    msg = find_msg(queue, mtype)
    copy_to_user(mtext, msg.mtext)
```

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

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

# Leaking 4 kB-Aligned Address




```
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
```

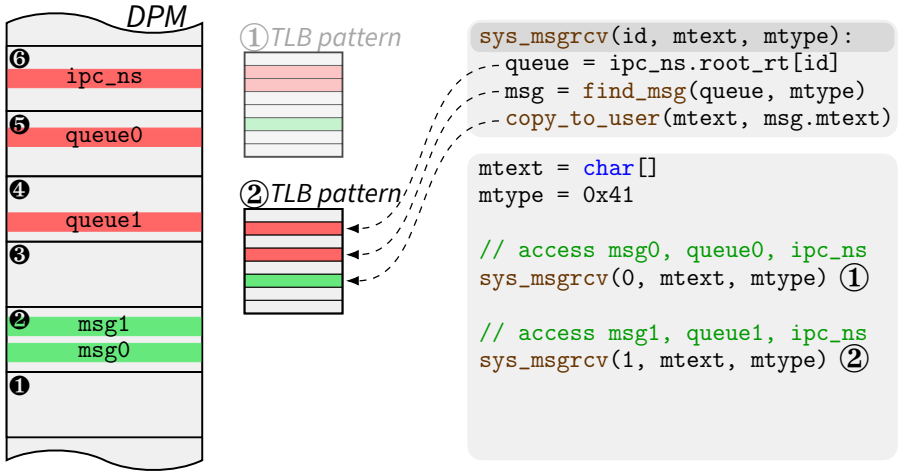
# Leaking 4 kB-Aligned Address


①TLB pattern

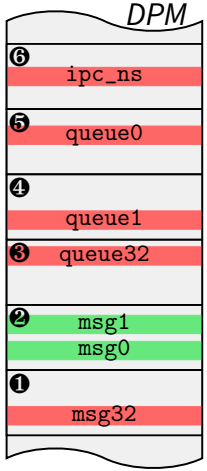
```
sys_msgrcv(id, mtext, mtype):
    queue = ipc_ns.root_rt[id]
    msg = find_msg(queue, mtype)
    copy_to_user(mtext, msg.mtext)
```

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

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

# Leaking 4 kB-Aligned Address



# Leaking 4 kB-Aligned Address

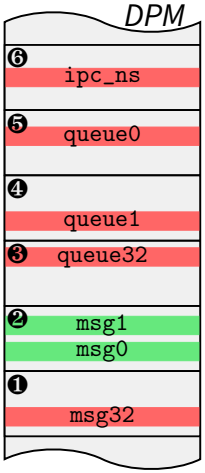


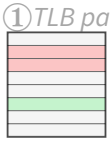


```
sys_msgrcv(id, mtext, mtype):
  queue = ipc_ns.root_rt[id]
  msg = find_msg(queue, mtype)
  copy_to_user(mtext, msg.mtext)
```

```
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





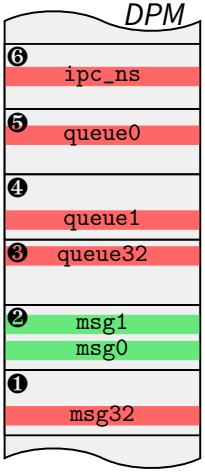

```
sys_msgrcv(id, mtext, mtype):
  queue = ipc_ns.root_rt[id]
  msg = find_msg(queue, mtype)
  copy_to_user(mtext, msg.mtext)
```

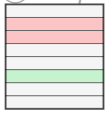
```
mtext = char[]
```

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

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

# Leaking 4 kB-Aligned Address





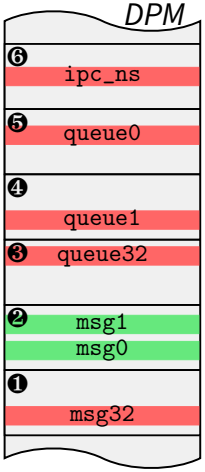

```
sys_msgrcv(id, mtext, mtype):
  queue = ipc_ns.root_rt[id]
  msg = find_msg(queue, mtype)
  copy_to_user(mtext, msg.mtext)
```

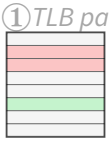
```
mtext = char[]
```

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

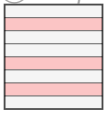
```
sys_msgrcv(32, mtext, mtype) ③
```

# Leaking 4 kB-Aligned Address







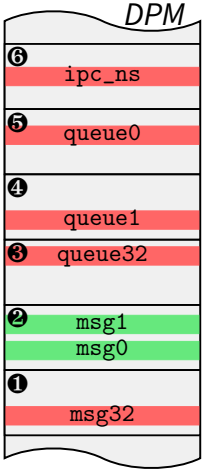

```
sys_msgrcv(id, mtext, mtype):
  queue = ipc_ns.root_rt[id]
  msg = find_msg(queue, mtype)
  copy_to_user(mtext, msg.mtext)
```

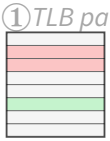
```
mtext = char[]
```

$$[❷, ❻] \setminus [❶, ❸, ❻]$$

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

# Leaking 4 kB-Aligned Address

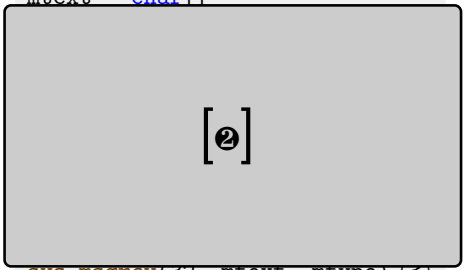


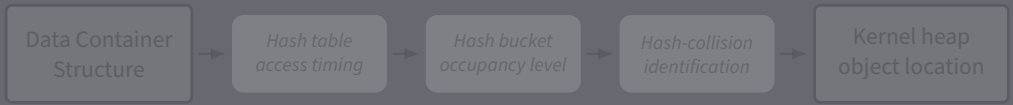



```
sys_msgrcv(id, mtext, mtype):
  queue = ipc_ns.root_rt[id]
  msg = find_msg(queue, mtype)
  copy_to_user(mtext, msg.mtext)
```

```
mtext = char[]
```




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


Data Container Structure
Hash table access timing
Hash bucket occupancy level
Hash-collision identification
Kernel heap object location


Hardware Cache
TLB access timings
2 MB-aligned memory sections
4 kB-aligned slab pages
? location

# Massaging

# Massaging

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

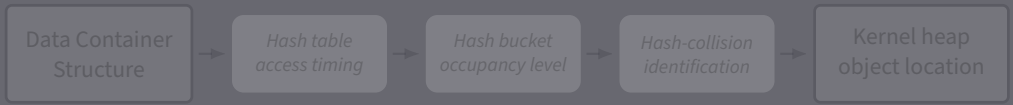
# Massaging

- **Ideal page:**
  - Contains only attacker-controlled objects
- **How?**
  - Use slab side channel

# Massaging

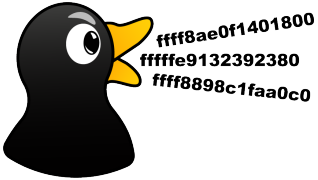
- **Ideal page:**
  - Contains only attacker-controlled objects
- **How?**
  - Use slab side channel

- **Leaked all object locations**
  - Known offsets within slab page


Data Container Structure
Hash table access timing
Hash bucket occupancy level
Hash-collision identification
Kernel heap object location


Hardware Cache
TLB access timings
2 MB-aligned memory sections
4 kB-aligned slab pages
Kernel heap object, stack, and pt locations

# Location Disclosure Attacks

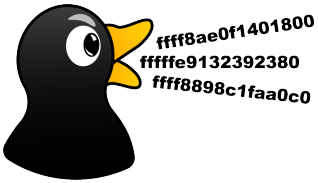


# Location Disclosure Attacks

**Evaluated Linux kernel:**

v5.15, v6.5, and v6.8

# Location Disclosure Attacks

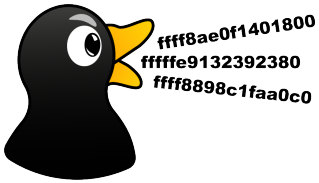



- **Evaluated Linux kernel**:
  v5.15, v6.5, and v6.8
- **CPUs**:
  Intel Kaby, Coffee, Alder, Raptor, and Meteor Lake *evaluated*
  AMD and some ARM *affected*

# Location Disclosure Attacks


- **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**:

# Location Disclosure Attacks




- **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

# Location Disclosure Attacks


- **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`

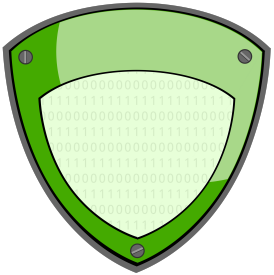
# Location Disclosure Attacks


- **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

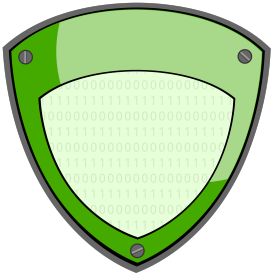
# Let’s leak some more kernel addresses

# Discussion & Takeaways

# Mitigations

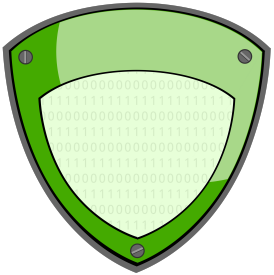


# Mitigations



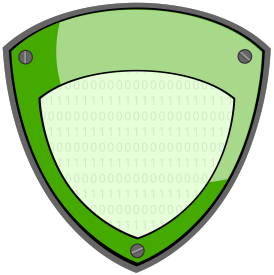
- Software-induced side channel

# Mitigations



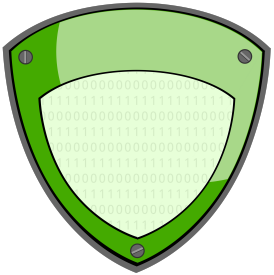
- **Software-induced side channel**
  - **Linus Torvalds** rejected patch
    - "What voodoo programming is this?"

# Mitigations



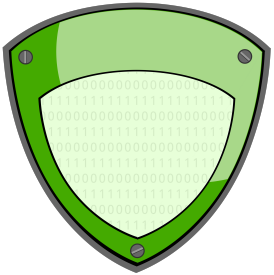
- **Software-induced side channel**
  - **Linus Torvalds** rejected patch
    - "What voodoo programming is this?"
  - Still exploitable!

# Mitigations



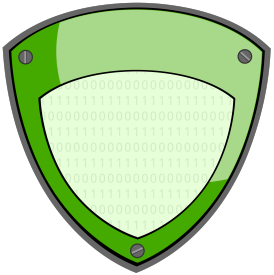
- **Software-induced side channel**
  - **Linus Torvalds** rejected patch
    - "What voodoo programming is this?"
  - Still exploitable!
- **Hardware-induced side channel**

# Mitigations



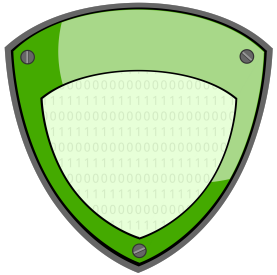
- **Software-induced side channel**
  - **Linus Torvalds** rejected patch
    - "What voodoo programming is this?"
  - Still exploitable!
- **Hardware-induced side channel**
  - **Linus Torvalds** forwarded to vendors
    - "It is their damn problem!"

# Mitigations



- **Software-induced side channel**
  - **Linus Torvalds** rejected patch
    - "What voodoo programming is this?"
  - Still exploitable!
- **Hardware-induced side channel**
  - **Linus Torvalds** forwarded to vendors
    - "It is their damn problem!"
  - **Intel** Linear Address Space Separation (LASS)

# Mitigations



- **Software-induced side channel**
  - **Linus Torvalds** rejected patch
    - "What voodoo programming is this?"
  - Still exploitable!
- **Hardware-induced side channel**
  - **Linus Torvalds** forwarded to vendors
    - "It is their damn problem!"
  - **Intel** Linear Address Space Separation (LASS)
  - Exploitable on:
    - most Intel
    - AMD
    - potentially ARM

# Security Implications

# Security Implications


**Side Effects:**

Defenses, software implementations, and allocators allow/amplify side channels

# Security Implications


- **Side Effects:**
  Defenses, software implementations, and allocators allow/amplify side channels
- **Leakage:**
  Side channels provide strong disclosure primitives

# Security Implications


- **Side Effects:**
  Defenses, software implementations, and allocators allow/amplify side channels
- **Leakage:**
  Side channels provide strong disclosure primitives
- **Exploitation:**
  Leakages enhance kernel exploit reliability

# 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.


TU Graz

SCIENCE
PASSION
TECHNOLOGY

# Derandomizing Kernel Object Locations with Software- and Hardware-Induced Side Channels

**Lukas Maar**

September 4-5, 2025

Nullcon Berlin

› isec.tugraz.at