CISPA
HELMHOLTZ CENTER FOR INFORMATION SECURITY

# Leaking Kernel Secrets using Transient Execution

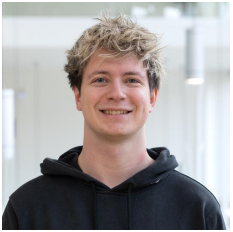
*Hands-On Session*

**Tristan Hornetz, Daniel Weber**, Michael Schwarz

MICSEC 2025

# Who Are We?

Tristan Hornetz
PhD Student at CISPA
tristan.hornetz@cispa.de



Daniel Weber
PhD Student at CISPA
daniel.weber@cispa.de

# Agenda

CPU Caches:
Flush+Reload

# Agenda

CPU Caches:
Flush+Reload

Meltdown:
Breaking KASLR

# Agenda

CPU Caches:
Flush+Reload

Meltdown:
Breaking KASLR

Meltdown:
Leaking Kernel Data

# Setup

- We provide access to **lab machines via SSH**

# Setup

- We provide access to **lab machines via SSH**
- These machines are **intentionally vulnerable**

# Setup

- We provide access to **lab machines via SSH**
- These machines are **intentionally vulnerable**
- Solving the last task gives you **root** access

# Setup

- We provide access to **lab machines via SSH**
- These machines are **intentionally vulnerable**
- Solving the last task gives you **root** access
- Please **act responsibly** with this access!

# Meltdown: Mounting a Transient Execution Attack

- Leaks **kernel memory** from user space

# Meltdown: Mounting a Transient Execution Attack

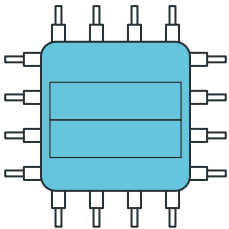
- Leaks **kernel memory** from user space
- Based on **Transient Execution**:
  - Leakage **not architecuturally** visible

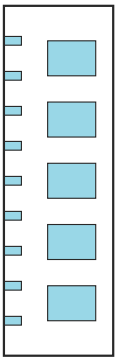
# Meltdown: Mounting a Transient Execution Attack

- Leaks **kernel memory** from user space
- Based on **Transient Execution**:
  - Leakage **not architecuturally** visible
  - Requires **covert channel** to leak data

# CPU Optimization: Cache

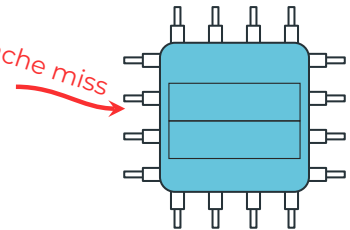
```
printf("%d", i);

printf("%d", i);
```

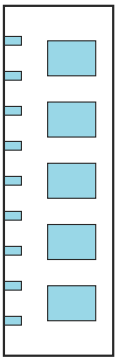




# CPU Optimization: Cache

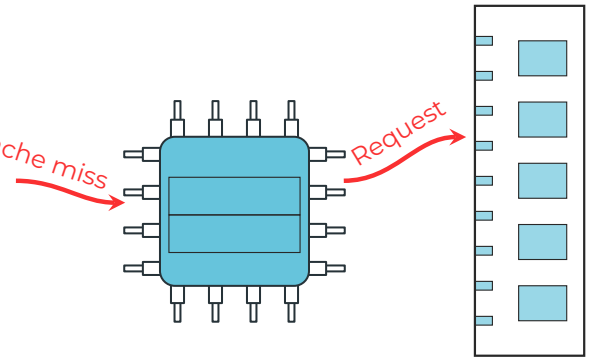
```
printf("%d", i);

printf("%d", i);
```



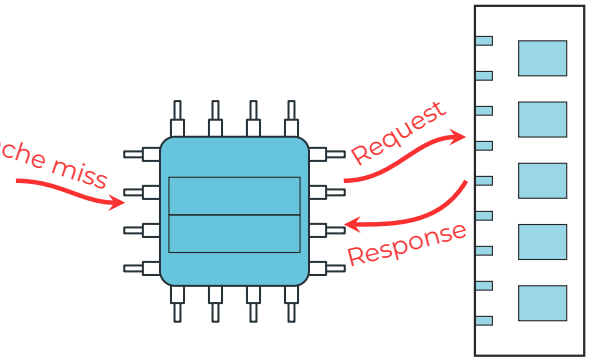
# CPU Optimization: Cache

```
printf("%d", i);

printf("%d", i);
```



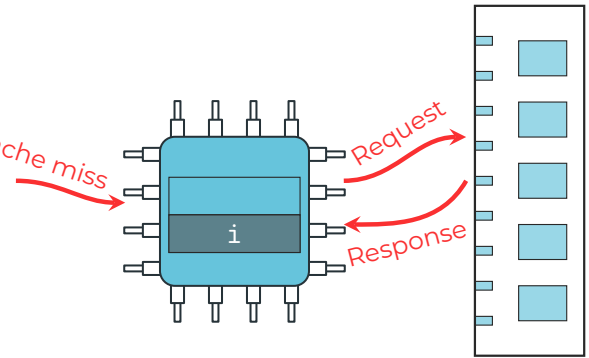
# CPU Optimization: Cache

```
printf("%d", i);

printf("%d", i);
```

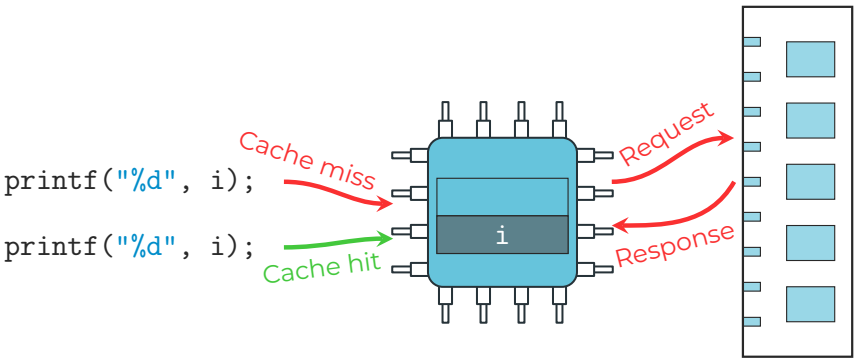


# CPU Optimization: Cache

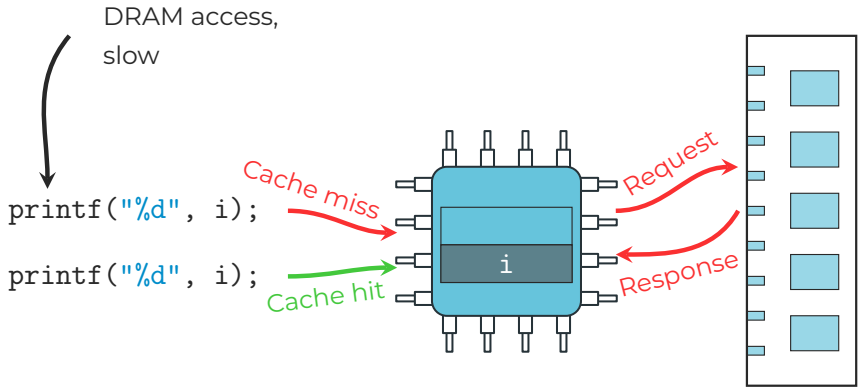
```
printf("%d", i);

printf("%d", i);
```



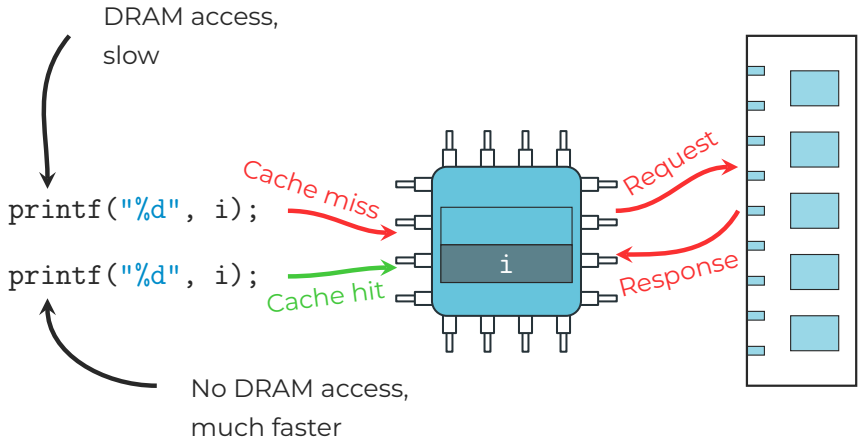
# CPU Optimization: Cache



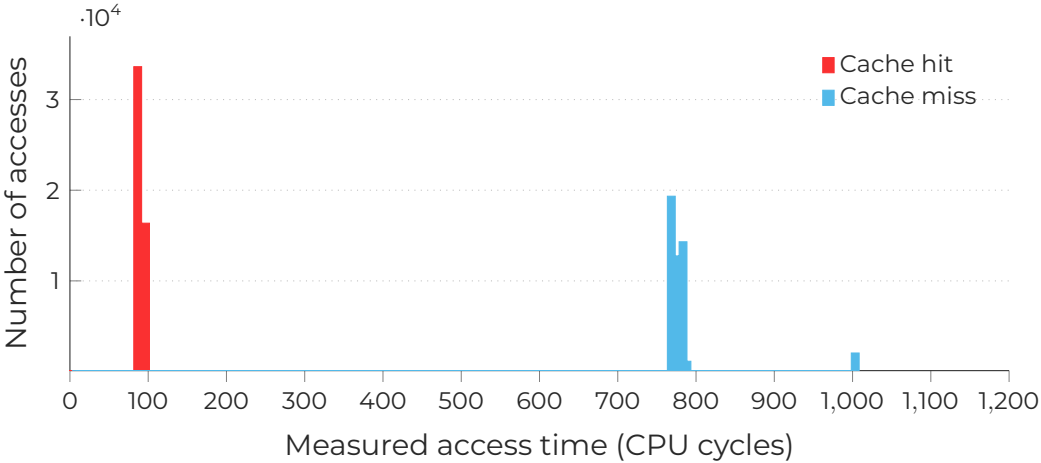
# CPU Optimization: Cache



# CPU Optimization: Cache

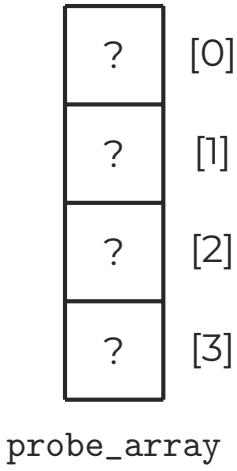


# Caching speeds up Memory Accesses

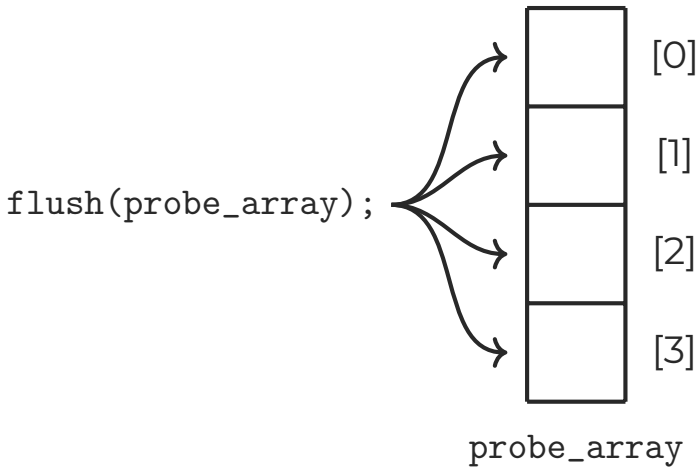


We can *transmit* data with that!

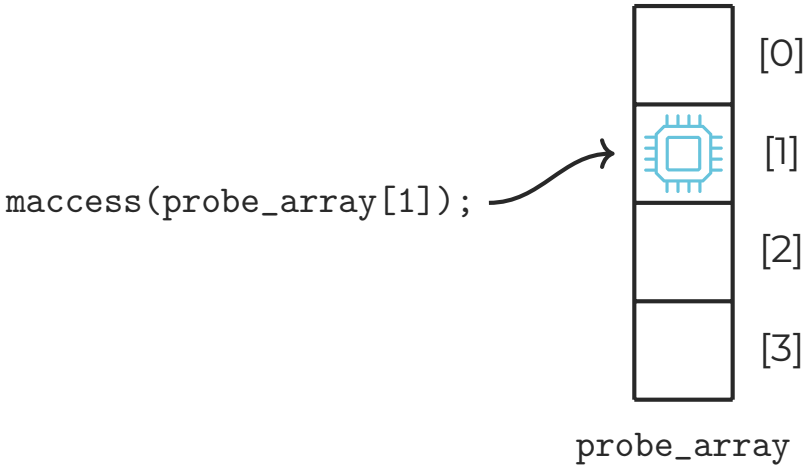
# Flush+Reload – Covert Channel Communication



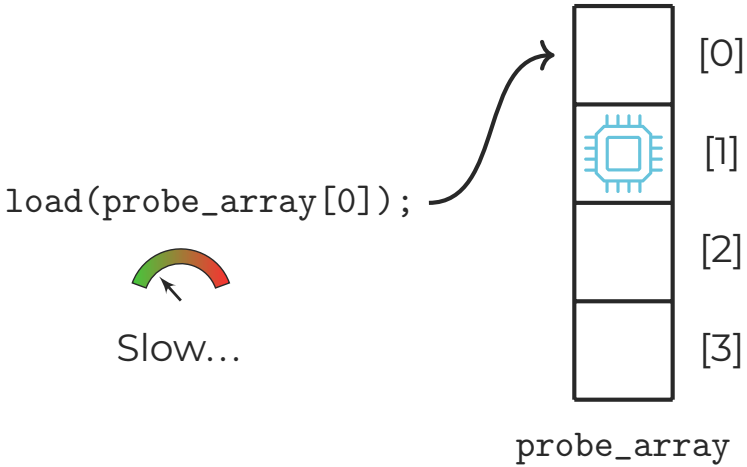
# Flush+Reload – Covert Channel Communication



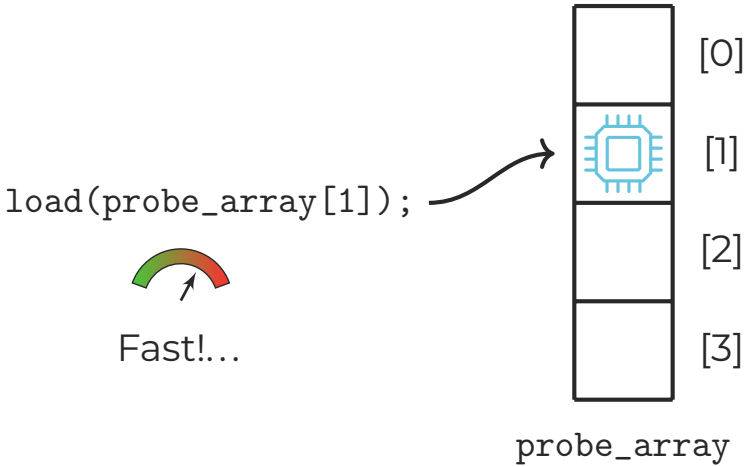
# Flush+Reload – Covert Channel Communication



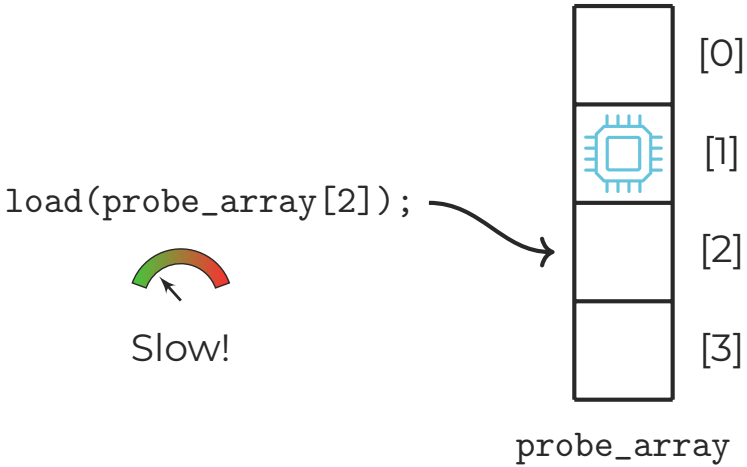
# Flush+Reload – Covert Channel Communication



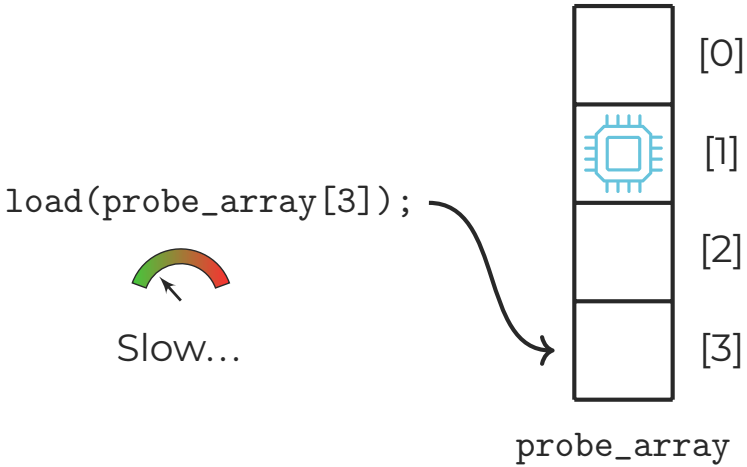
# Flush+Reload – Covert Channel Communication



# Flush+Reload – Covert Channel Communication



# Flush+Reload – Covert Channel Communication



Let’s get our hands dirty!

# Exercise 1: Flush+Reload


**The Task:**

Encode the Fibonacci sequence:

→ Flush the “probe array”

→ Access the n-th entry (already implemented)

→ Find the cached entry

# Exercise 1: Flush+Reload

**The Task:**

Encode the Fibonacci sequence:

→ Flush the “probe array”

→ Access the n-th entry (already implemented)

→ Find the cached entry

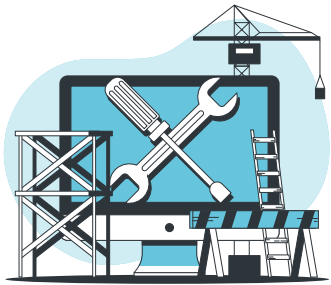
**How to start**

- Open `task-1-flush-reload.c`
- Implement `flush_entire_buffer`
- Implement `recover_encoded_number`


## Exercise 1:

# Flush+Reload

https://d-we.me/ms/ex1-fr.pdf



We have our main building block!

# The Meltdown Attack

```
kaddr = 0xffffffff81000000;
x = load(kaddr);
load(probe_array[x]);
```

# The Meltdown Attack

```
kaddr = 0xffffffff81000000;
x = load(kaddr);
load(probe_array[x]);
```

# The Meltdown Attack


```
kaddr = 0xffffffff81000000;
x = load(kaddr);
load(probe_array[x]);
```

# The Meltdown Attack

**#GP Fault**

```
kaddr = 0xffffffff81000000;
x = load(kaddr);
```

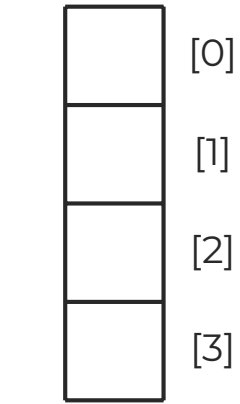
# The Meltdown Attack

```
kaddr = 0xffffffff81000000;
x = load(kaddr);
```

```
[1] 624141 segmentation fault (core dumped) ./program
```

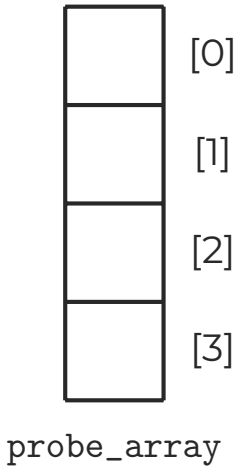
# The Meltdown Attack

```
kaddr = 0xffffffff81000000;
x = load(kaddr);
load(probe_array[x]);
```



# The Meltdown Attack

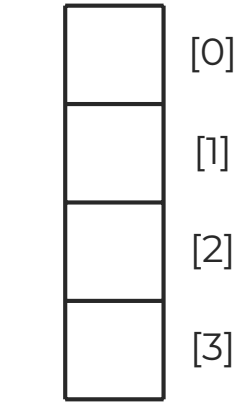
```
kaddr = 0xffffffff81000000;
x = load(kaddr);
load(probe_array[x]);
```



# The Meltdown Attack

Loading Complete!

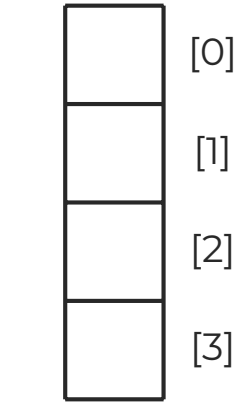
```
kaddr = 0xffffffff81000000;
2 = load(kaddr);
load(probe_array[x]);
```



# The Meltdown Attack

Loading Complete!

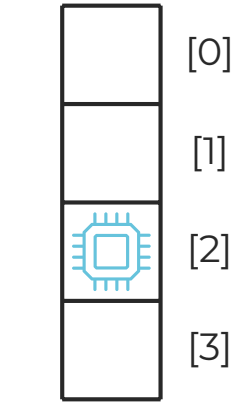
```
kaddr = 0xffffffff81000000;
2 = load(kaddr);
load(probe_array[2]);
```



# The Meltdown Attack

Loading Complete!

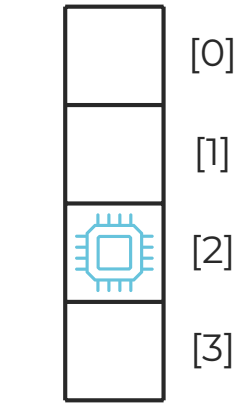
```
kaddr = 0xffffffff81000000;
2 = load(kaddr);
load(probe_array[2]);
```



# The Meltdown Attack

**#GP Fault**

```
kaddr = 0xffffffff81000000;
x = load(kaddr);
load(probe_array[2]);
```

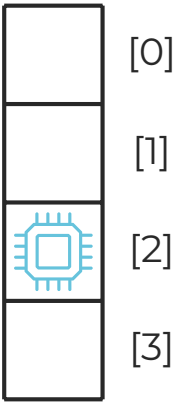


# The Meltdown Attack

**#GP Fault**

```
kaddr = 0xffffffff81000000;
x = load(kaddr);
```

Rollback

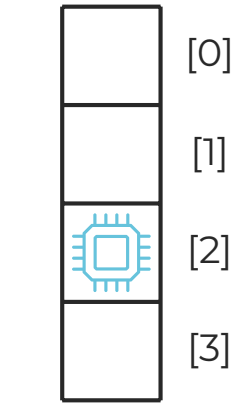



probe_array

# The Meltdown Attack

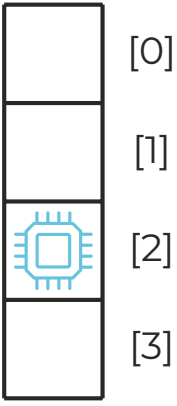
**#GP Fault**

```
kaddr = 0xffffffff81000000;
x = load(kaddr);
```

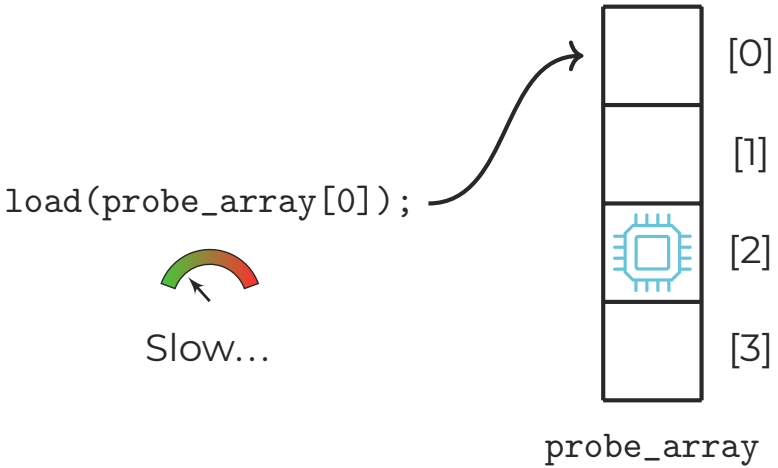


# The Meltdown Attack

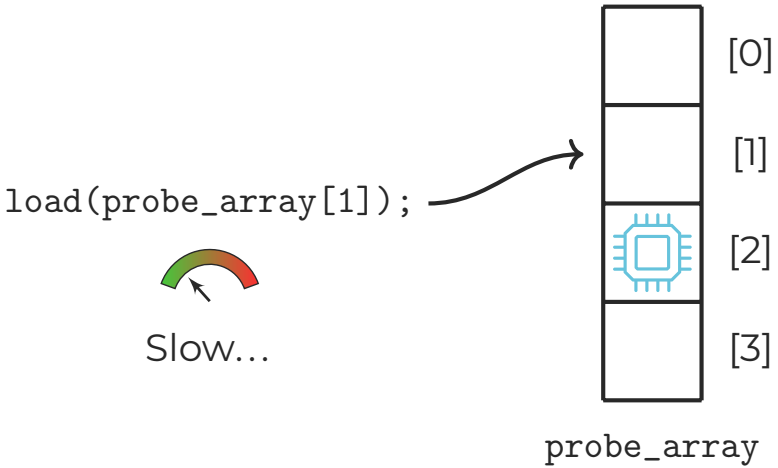
**Handle Fault**



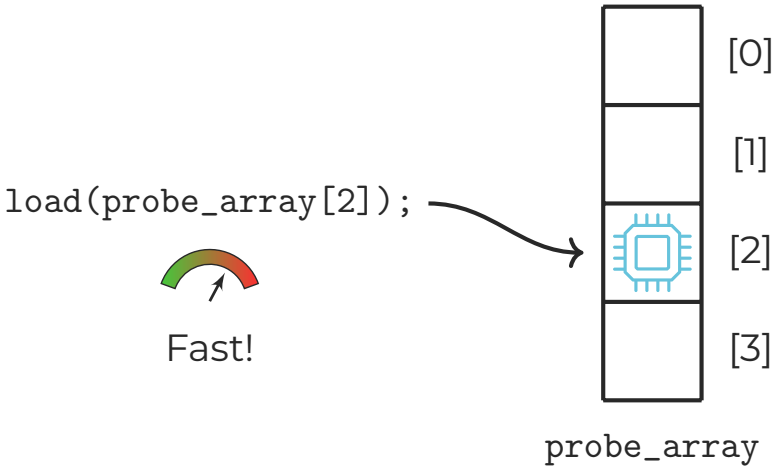
# The Meltdown Attack



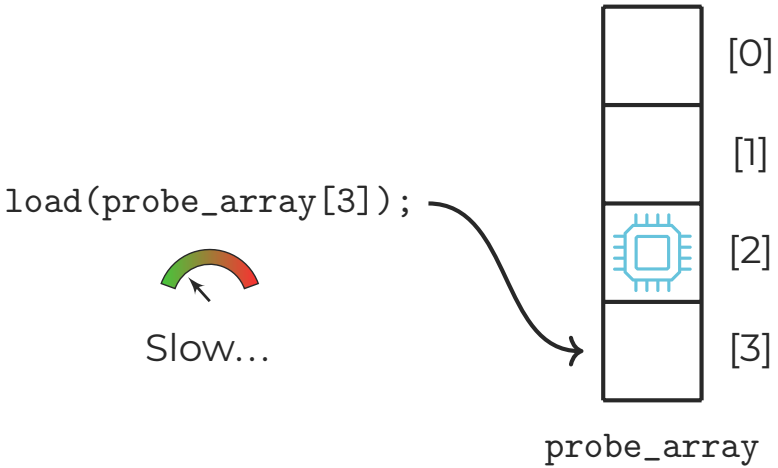
# The Meltdown Attack



# The Meltdown Attack



# The Meltdown Attack



Let’s implement Meltdown!

# Exercise 2: Leaking Data with Meltdown


**The Task:**

Implement Meltdown:

→ Flush the CPU cache

→ Cache the kernel address

→ Access and encode the address

→ Handle the fault and decode

# Exercise 2: Leaking Data with Meltdown


**The Task:**

Implement Meltdown:

→ Flush the CPU cache

→ Cache the kernel address

→ Access and encode the address

→ Handle the fault and decode

**How to start**

- Open `task-2-meltdown.c`
- Implement `recover_byte`


## Exercise 2:

# Leaking Memory using Meltdown

https://d-we.me/ms/ex2-md.pdf

# What Now?

- We can now **leak arbitrary memory** from the kernel!

# What Now?

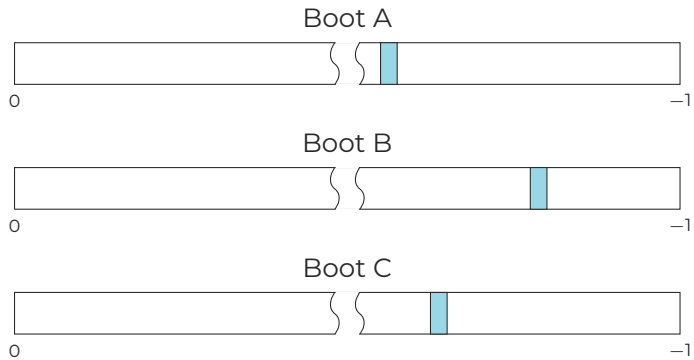
- We can now **leak arbitrary memory** from the kernel!
- But **which** memory should we leak?

# What Now?

- We can now **leak arbitrary memory** from the kernel!
- But **which** memory should we leak?
- We need to **find interesting addresses** first!

Can’t we just **search** for them in the **kernel image**?

# Kernel Address Space Layout Randomization




Kernel is loaded to a **different** offset on every boot

# KASLR Weakness

- It’s enough to leak a **single address** inside the kernel

# KASLR Weakness

- It’s enough to leak a **single address** inside the kernel
- Everything else is **relative** to that single address

# KASLR Weakness

- It's enough to leak a **single address** inside the kernel
- Everything else is **relative** to that single address

→ **Guess** a kernel address

# KASLR Weakness

- It's enough to leak a **single address** inside the kernel
- Everything else is **relative** to that single address

→ **Guess** a kernel address

→ **Check** if guess is correct

Sound's easy? Let's try it!

# Exercise 3: Breaking KASLR with Meltdown


**The Task:**

Iterate over the possible kernel locations.

For each location:

→ Compute the address of (part of) the kernel banner

→ Try to leak the kernel banner

→ Check if you leak “Linux”

# Exercise 3: Breaking KASLR with Meltdown

**The Task:**

Iterate over the possible kernel locations.

For each location:

→ Compute the address of (part of) the kernel banner

→ Try to leak the kernel banner

→ Check if you leak “Linux”

**How to start**

- Open `task-3-kaslr.c`
- Implement `is_correct_base_address`


## Exercise 3:

# Breaking KASLR using Meltdown

https://d-we.me/ms/ex3-kaslr.pdf

# Exploiting a Kernel Module

- VMs contain the `ms_admin` kernel module

# Exploiting a Kernel Module

- VMs contain the `ms_admin` kernel module
- Password-protected root shell

# Exploiting a Kernel Module

- VMs contain the `ms_admin` kernel module
- Password-protected root shell
- Login information is checked against `struct admin_creds`

# Exploiting a Kernel Module

- VMs contain the `ms_admin` kernel module
- Password-protected root shell
- Login information is checked against `struct admin_creds`

→ **Leak** the struct using Meltdown

One more to go!

Let’s become root!

# Exercise 4: Putting it all together


**The Task:**

Leak the admin password and get a root shell

→ Leak the content of `struct admin_creds`

→ We provide you the offset of `admin_creds`

→ Use the leaked password to login

# Exercise 4: Putting it all together

**The Task:**

Leak the admin password and get a root shell

→ Leak the content of `struct admin_creds`

→ We provide you the offset of `admin_creds`

→ Use the leaked password to login

**How to start**

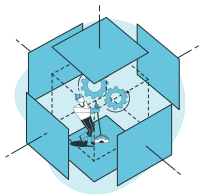
- Open `task-4-root-shell.c`
- Implement `read_credentials`


## Exercise 4:

# Putting it all together

https://d-we.me/ms/ex4-exp.pdf

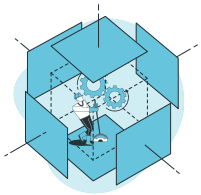
# Recap



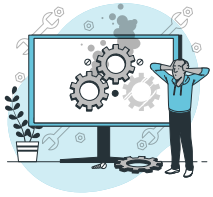
**CPU Cache:**

Reasoning about transient state

# Recap

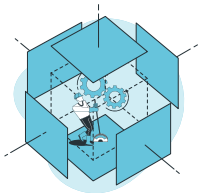


**CPU Cache:**
Reasoning about transient state



**KASLR:**
Breaks with a single leak

# Recap

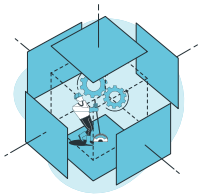


**CPU Cache:**
Reasoning about transient state

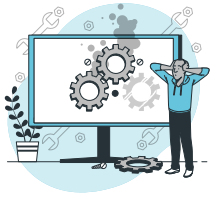
**KASLR:**
Breaks with a single leak

**Meltdown:**
Leaking arbitrary kernel memory

# Recap



**CPU Cache:**
Reasoning about transient state



**KASLR:**
Breaks with a single leak

**Meltdown:**
Leaking arbitrary kernel memory

Many **CPU vulnerabilities** can be exploited in a **similar way**!

CISPA
HELMHOLTZ CENTER FOR INFORMATION SECURITY

# Leaking Kernel Secrets using Transient Execution

*Hands-On Session*

**Tristan Hornetz, Daniel Weber**, Michael Schwarz

MICSEC 2025

# References

- Icons and Images from
  - storyset.com
  - thenounproject.com