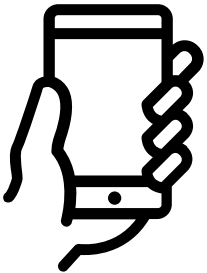
CISPA
HELMHOLTZ CENTER FOR INFORMATION SECURITY

# From Rowhammer to GhostWrite

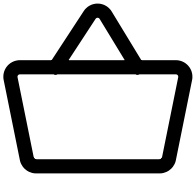
*Advanced Exploitation and Discovery of Hardware Bugs*

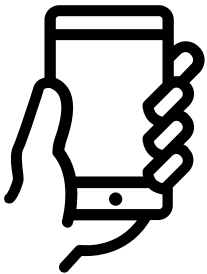
Fabian Thomas | HWIO Amsterdam 2024 | October 25, 2024

# Installing fishy Apps is fine, right?

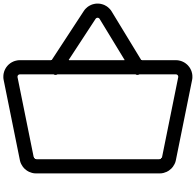


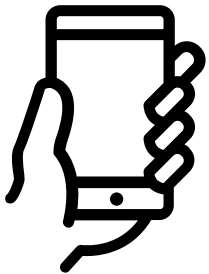
# Installing fishy Apps is fine, right?





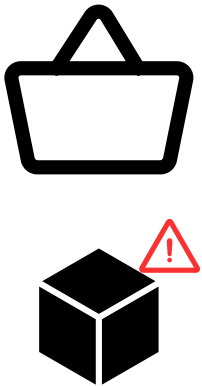
# Installing fishy Apps is fine, right?



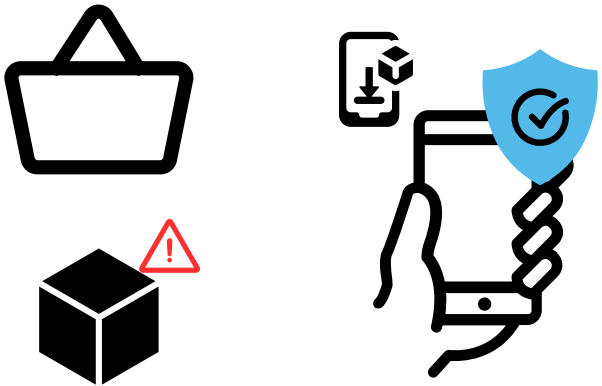


# Installing fishy Apps is fine, right?

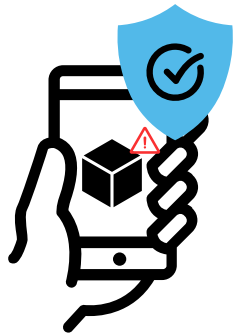
# Installing fishy Apps is fine, right?



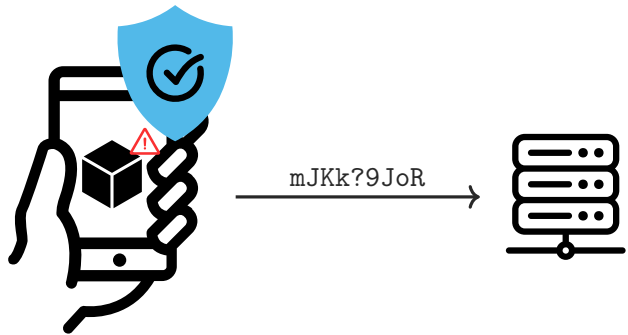
# Installing fishy Apps is fine, right?



# Installing fishy Apps is fine, right?



# Installing fishy Apps is fine, right?



# What happened here?

- No permission given to app

# What happened here?

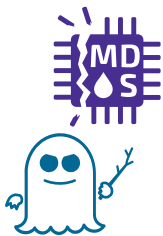
- No permission given to app
- Software security is easy:
  Memory-safe languages, control-flow integrity, memory tagging, . . .

# What happened here?

- No permission given to app
- Software security is easy:
  Memory-safe languages, control-flow integrity, memory tagging, . . .
- Is the hardware broken?

# Potential Attacks




# Potential Attacks

# Who are we?

**Fabian Thomas**

PhD student @CISPA (Germany)

**E-Mail** fabian.thomas@cispa.de

**Web** fabianthomas.de

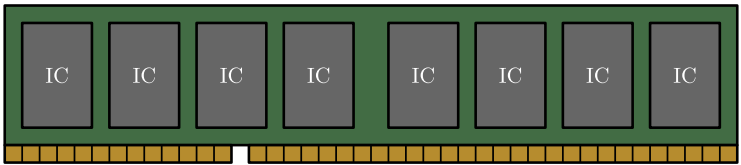
**Twitter** @fth0mas

# Who are we?

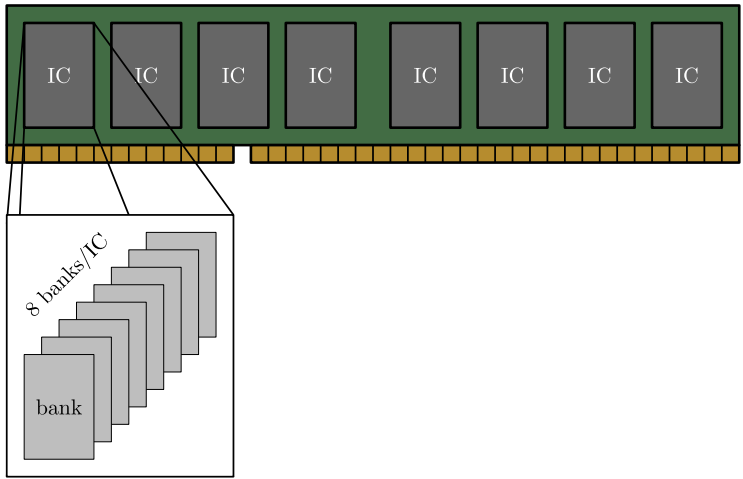
## Research Group Schwarz

- Research focus:
  - Hardware vulnerabilities
  - . . . from software
- Recent discoveries:

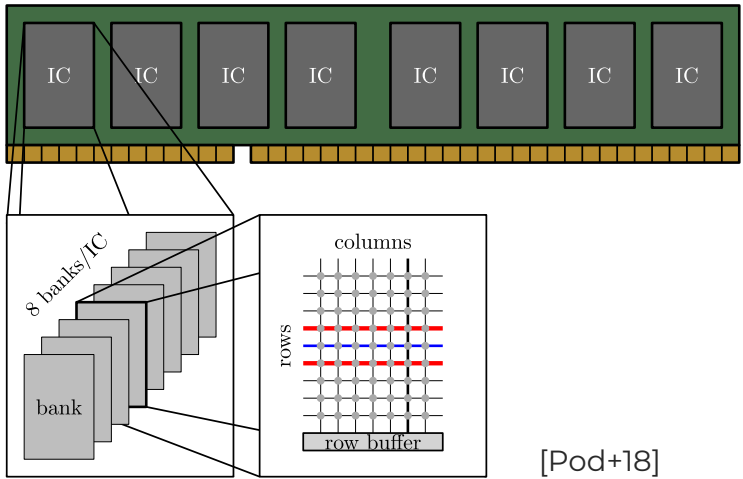

# The Classic: Rowhammer



# The Classic: Rowhammer



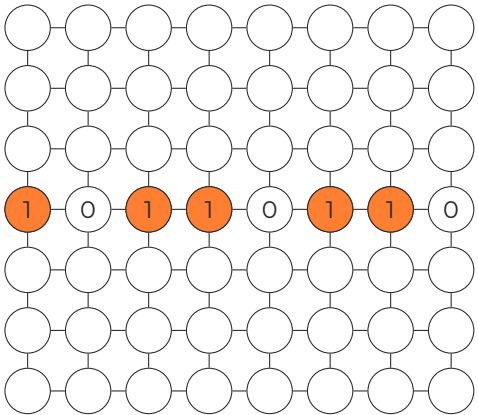
# The Classic: Rowhammer




[Pod+18]

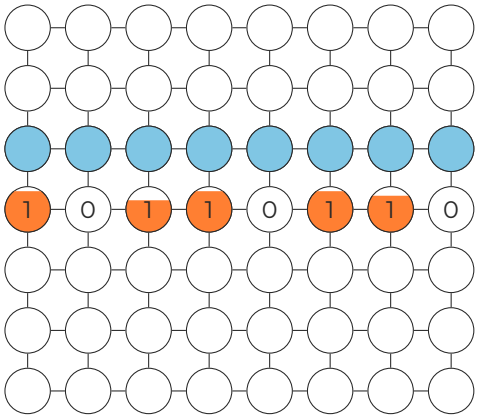
# Rowhammer: Triggering Bit Flips

- Row activations drain neighbor cells
- Too frequent ⇝ bit flips



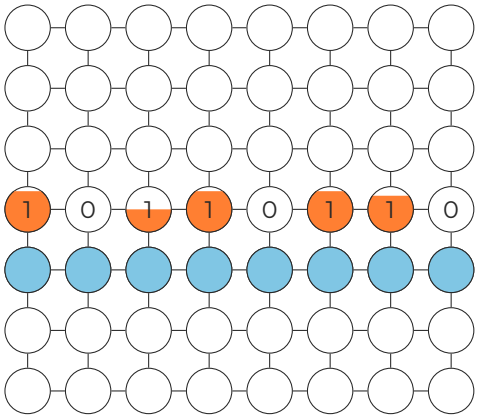
# Rowhammer: Triggering Bit Flips

- Row activations drain neighbor cells
- Too frequent ⇝ bit flips



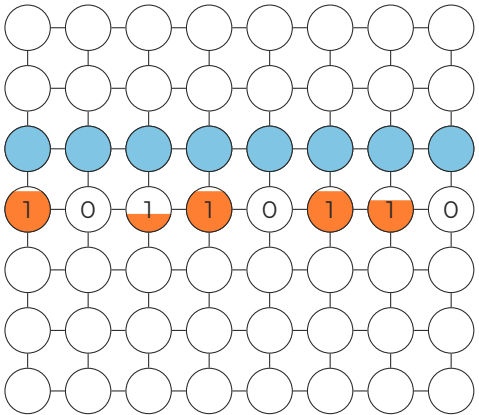
# Rowhammer: Triggering Bit Flips

- Row activations drain neighbor cells
- Too frequent ⇝ bit flips



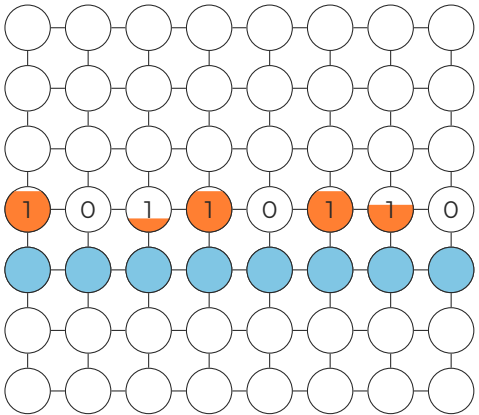
# Rowhammer: Triggering Bit Flips

- Row activations drain neighbor cells
- Too frequent ⇝ bit flips



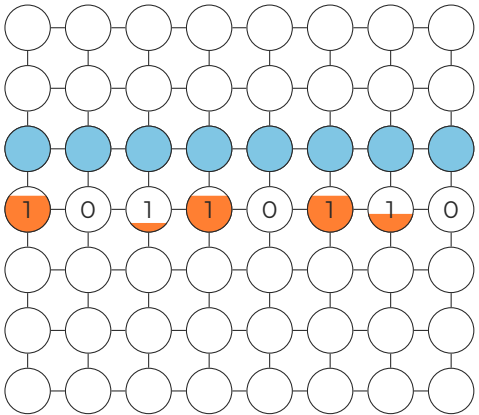
# Rowhammer: Triggering Bit Flips

- Row activations drain neighbor cells
- Too frequent ⇝ bit flips



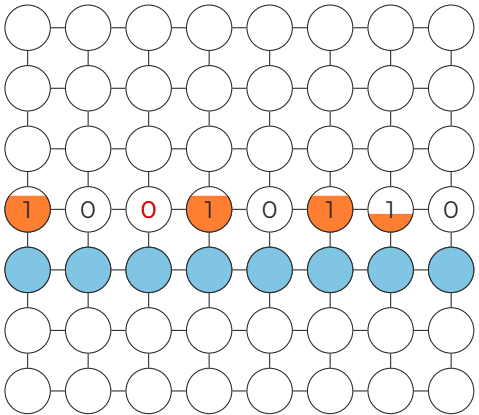
# Rowhammer: Triggering Bit Flips

- Row activations drain neighbor cells
- Too frequent ⇝ bit flips



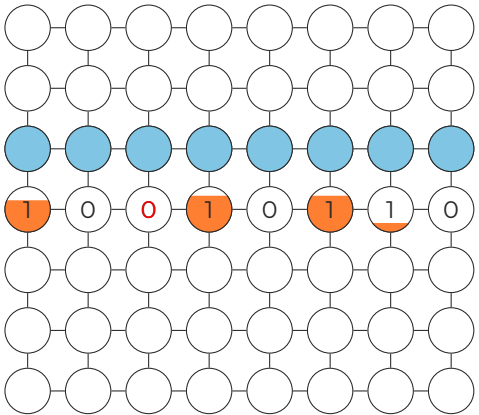
# Rowhammer: Triggering Bit Flips

- Row activations drain neighbor cells
- Too frequent ⇝ bit flips



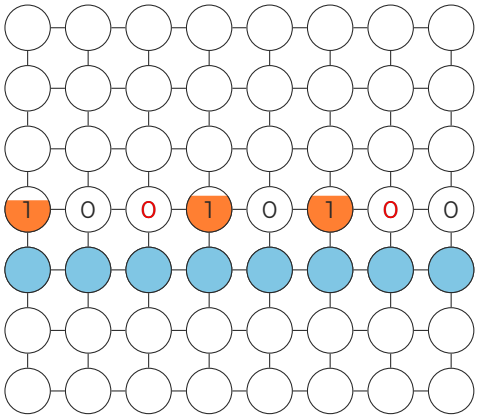
# Rowhammer: Triggering Bit Flips

- Row activations drain neighbor cells
- Too frequent ⇝ bit flips



# Rowhammer: Triggering Bit Flips

- Row activations drain neighbor cells
- Too frequent ⇝ bit flips



# Rowhammer: Exploitation

Data Structures

# Rowhammer: Exploitation

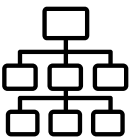

Data Structures


Opcodes

# Rowhammer: Exploitation


Data Structures


Opcodes

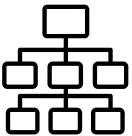


Page Tables

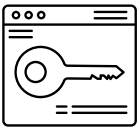
# Rowhammer: Exploitation


Data Structures


Opcodes



Page Tables

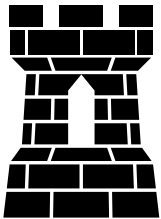


Crypto

# Rowhammer: Summary

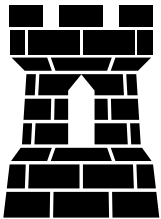
| | Rowhammer |
|---|---|
| **Restrictions** | *bit flips* |
| **Speed** | 😭 |
| **Practicality** | 😕 |

# CacheWarp: AMD SEV



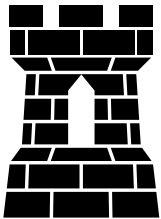
- Secure Encrypted Virtualization

# CacheWarp: AMD SEV



- Secure Encrypted Virtualization
- Encrypts entire VMs

# CacheWarp: AMD SEV



- Secure Encrypted Virtualization
- Encrypts entire VMs
- Similar threat model as SGX

# CacheWarp: INVD Instruction

- Flushes internal CPU caches

# CacheWarp: INVD Instruction

- Flushes internal CPU caches
- Does not write modifications to memory

# CacheWarp: INVD Instruction

- Flushes internal CPU caches
- Does not write modifications to memory

**Intel**

Use this instruction with care. Data cached internally and not written back to main memory will be lost. [...] software should instead use the WBINVD instruction.

# CacheWarp: Basic Primitive

**Hypervisor**

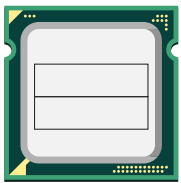
invd

**Victim VM**

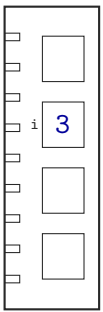
```
printf("%d", i);
i = 4;
printf("%d", i);
```

Output

**CPU Cache**



**DRAM**



# CacheWarp: Basic Primitive

**Hypervisor**

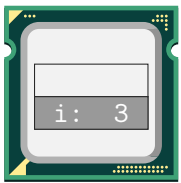
```
invd
```

**Victim VM**

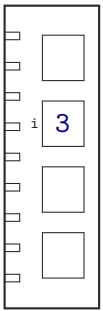
```
printf("%d", i);
i = 4;
printf("%d", i);
```

Output

**CPU Cache**




**DRAM**



# CacheWarp: Basic Primitive

**Hypervisor**

```
invd
```

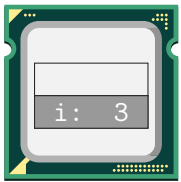
**Victim VM**

```
printf("%d", i);
i = 4;
printf("%d", i);
```

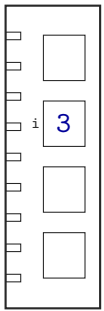
Output

```
3
```

**CPU Cache**




**DRAM**



# CacheWarp: Basic Primitive

**Hypervisor**

```
invd
```

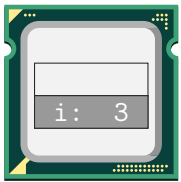
**Victim VM**

```
printf("%d", i);
i = 4;
printf("%d", i);
```

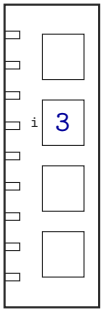
Output

```
3
```

**CPU Cache**




**DRAM**



# CacheWarp: Basic Primitive

**Hypervisor**

```
invd
```

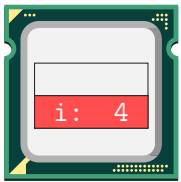
**Victim VM**

```
printf("%d", i);
i = 4;
printf("%d", i);
```

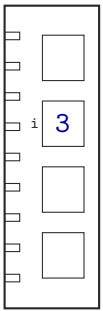
Output

```
3
```

**CPU Cache**




**DRAM**



# CacheWarp: Basic Primitive

**Hypervisor**

- `invd`

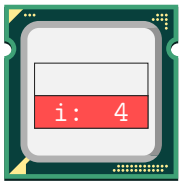
**Victim VM**

```
printf("%d", i);
i = 4;
printf("%d", i);
```

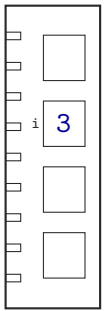
Output

```
3
```

**CPU Cache**




**DRAM**



# CacheWarp: Basic Primitive

**Hypervisor**

- `invd`

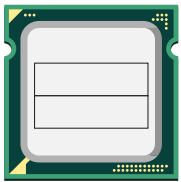
**Victim VM**

```
printf("%d", i);
i = 4;
printf("%d", i);
```

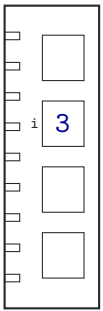
Output

```
3
```

**CPU Cache**



**DRAM**



# CacheWarp: Basic Primitive

**Hypervisor**

```
invd
```

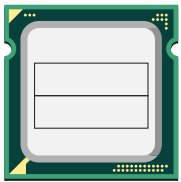
**Victim VM**

```
printf("%d", i);
i = 4;
▶ printf("%d", i);
```

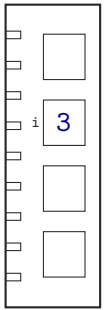
Output

```
3
```

**CPU Cache**



**DRAM**



# CacheWarp: Basic Primitive

**Hypervisor**

```
invd
```

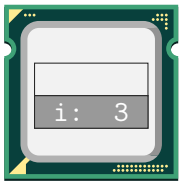
**Victim VM**

```
printf("%d", i);
i = 4;
▶ printf("%d", i);
```

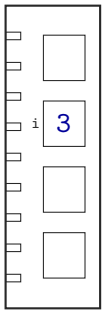
Output

```
3
```

**CPU Cache**




**DRAM**



# CacheWarp: Basic Primitive

**Hypervisor**

```
invd
```

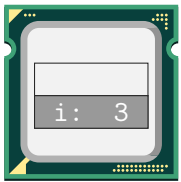
**Victim VM**

```
printf("%d", i);
i = 4;
▶ printf("%d", i);
```

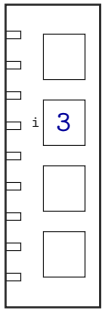
Output

```
3
3
```

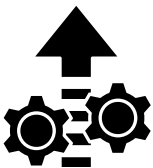
**CPU Cache**




**DRAM**

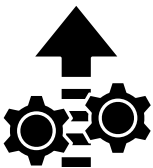


# CacheWarp: sudo Exploit



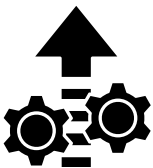
- CacheWarp on `sudo` to gain root privileges

# CacheWarp: sudo Exploit



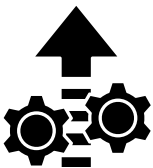
- CacheWarp on `sudo` to gain root privileges
- User ID is zero initialized

# CacheWarp: sudo Exploit



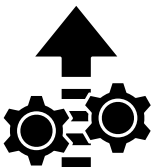
- CacheWarp on `sudo` to gain root privileges
- User ID is zero initialized
- Zero = root user

# CacheWarp: sudo Exploit



- CacheWarp on `sudo` to gain root privileges
- User ID is zero initialized
- Zero = root user
- `sudo` queries real user ID → dropped with CacheWarp

# CacheWarp: sudo Exploit



- CacheWarp on `sudo` to gain root privileges
- User ID is zero initialized
- Zero = root user
- `sudo` queries real user ID → dropped with CacheWarp

→ `sudo` continues as root without further checks

# Timewarp: Attacking Return Addresses

- Return addresses: implicit stores by the CPU

# Timewarp: Attacking Return Addresses

- Return addresses: implicit stores by the CPU
- Can be reverted as well

# Timewarp: Attacking Return Addresses

- Return addresses: implicit stores by the CPU
- Can be reverted as well
- Return uses stale value

# Timewarp: Attacking Return Addresses

- Return addresses: implicit stores by the CPU
- Can be reverted as well
- Return uses stale value
  - → Can be previous return address

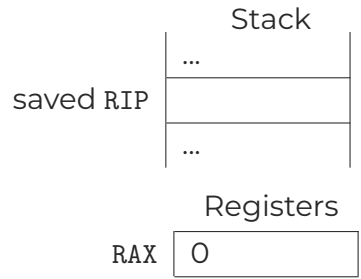
# Timewarp: Attacking Return Addresses

- Return addresses: implicit stores by the CPU
- Can be reverted as well
- Return uses stale value
  - → Can be previous return address

→ Jump “back in time”

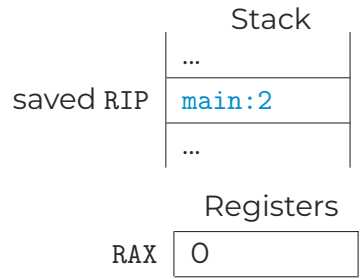
# Timewarp

```
1  call   ret1
2  cmp    rax, 0
3  je     win
4  call   ret0
5  jmp    fail
6
7  ret0:
8    mov  rax, 0
9    ret
10
11 ret1:
12   mov  rax, 1
13   ret
```



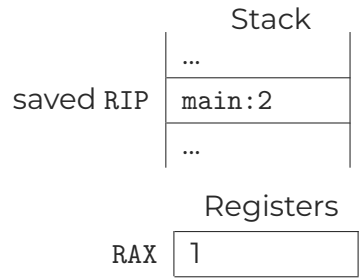
# Timewarp

```
1  call   ret1
2  cmp    rax, 0
3  je     win
4  call   ret0
5  jmp    fail
6
7  ret0:
8    mov  rax, 0
9    ret
10
11 ret1:
12   mov  rax, 1
13   ret
```



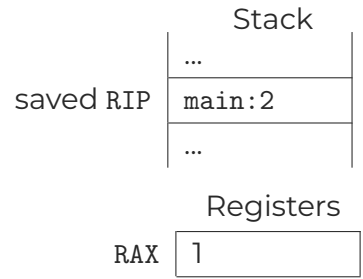
# Timewarp

```
1  call   ret1
2  cmp    rax, 0
3  je     win
4  call   ret0
5  jmp    fail
6
7  ret0:
8    mov  rax, 0
9    ret
10
11 ret1:
12   mov  rax, 1
13   ret
```



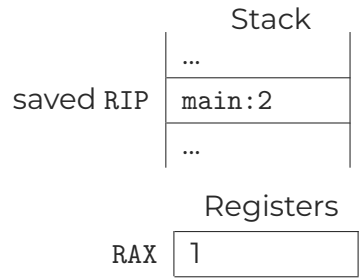
# Timewarp

```
1  call   ret1
2  cmp    rax, 0
3  je     win
4  call   ret0
5  jmp    fail
6
7  ret0:
8    mov  rax, 0
9    ret
10
11 ret1:
12   mov  rax, 1
13   ret
```



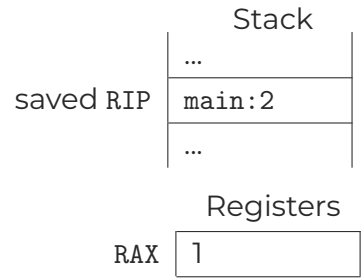
# Timewarp

```
1  call   ret1
2  cmp    rax, 0
3  je     win
4  call   ret0
5  jmp    fail
6
7  ret0:
8    mov  rax, 0
9    ret
10
11 ret1:
12   mov  rax, 1
13   ret
```



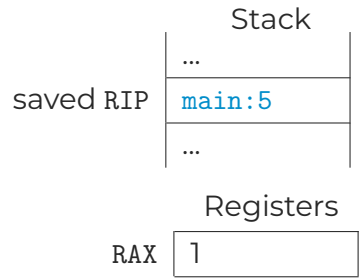
# Timewarp

```
1  call   ret1
2  cmp    rax, 0
3  je     win
4  call   ret0
5  jmp    fail
6
7  ret0:
8    mov  rax, 0
9    ret
10
11 ret1:
12   mov  rax, 1
13   ret
```



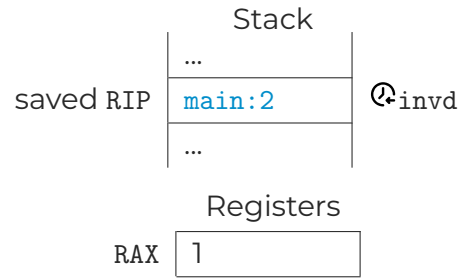
# Timewarp

```
1  call   ret1
2  cmp    rax, 0
3  je     win
4  call   ret0
5  jmp    fail
6
7  ret0:
8    mov  rax, 0
9    ret
10
11 ret1:
12   mov  rax, 1
13   ret
```



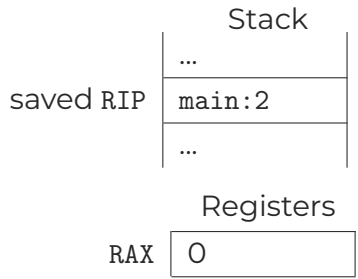
# Timewarp

```
1  call   ret1
2  cmp    rax, 0
3  je     win
4  call   ret0
5  jmp    fail
6
7  ret0:
8    mov  rax, 0
9    ret
10
11 ret1:
12   mov  rax, 1
13   ret
```



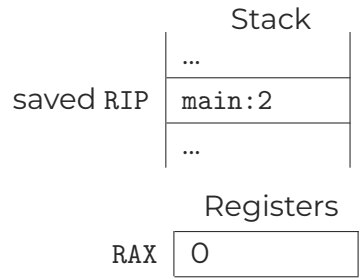
# Timewarp

```
1  call   ret1
2  cmp    rax, 0
3  je     win
4  call   ret0
5  jmp    fail
6
7  ret0:
8    mov  rax, 0
9    ret
10
11 ret1:
12   mov  rax, 1
13   ret
```



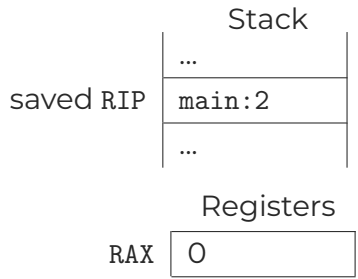
# Timewarp

```
1  call   ret1
2  cmp    rax, 0
3  je     win
4  call   ret0
5  jmp    fail
6
7  ret0:
8    mov  rax, 0
9    ret
10
11 ret1:
12   mov  rax, 1
13   ret
```



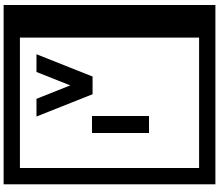
# Timewarp

```
1  call   ret1
2  cmp    rax, 0
3  je     win
4  call   ret0
5  jmp    fail
6
7  ret0:
8    mov  rax, 0
9    ret
10
11 ret1:
12   mov  rax, 1
13   ret
```



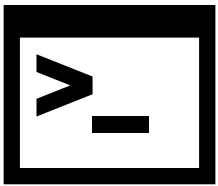

1

# CacheWarp: OpenSSH Exploit



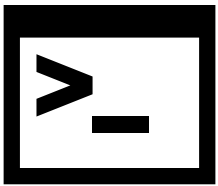
- Timewarp on OpenSSH

# CacheWarp: OpenSSH Exploit



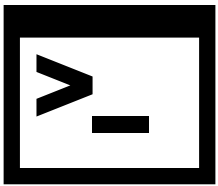
- Timewarp on OpenSSH
- Login using wrong password

# CacheWarp: OpenSSH Exploit



- Timewarp on OpenSSH
- Login using wrong password
- Timewarp to compare password hash with itself

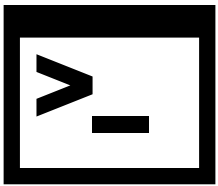
# CacheWarp: OpenSSH Exploit



- Timewarp on OpenSSH
- Login using wrong password
- Timewarp to compare password hash with itself

→ Entered password is not relevant

# CacheWarp: OpenSSH Exploit

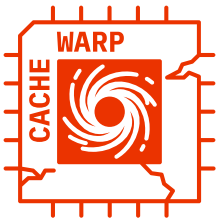


- Timewarp on OpenSSH
- Login using wrong password
- Timewarp to compare password hash with itself

→ Entered password is not relevant

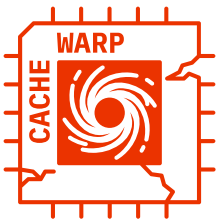
- Attacker is logged in via SSH

# CacheWarp: Mitigation



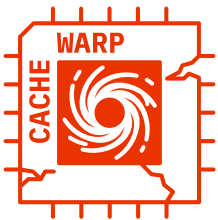

- Microcode update for SEV-SNP

# CacheWarp: Mitigation




- Microcode update for SEV-SNP
- Converts `invd` to `wbinvd`

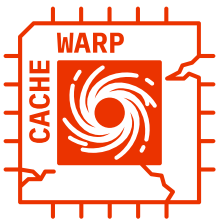
# CacheWarp: Mitigation




- Microcode update for SEV-SNP
- Converts `invd` to `wbinvd`

→ Modifications are written back, not dropped

# CacheWarp: Mitigation




- Microcode update for SEV-SNP
- Converts `invd` to `wbinvd`

→ Modifications are written back, not dropped

- Hardware fixes on Zen 4

# CacheWarp: Comparison

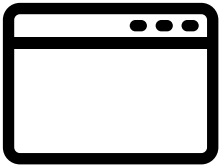
| | Rowhammer |
|---|---|
| **Restrictions** | *bit flips* |
| **Speed** | 😭 |
| **Practicality** | 😕 |

# CacheWarp: Comparison

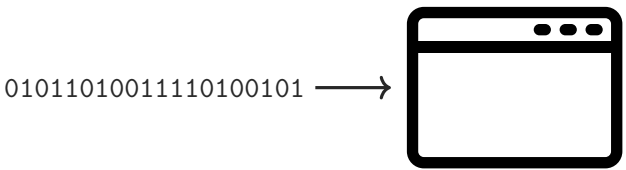
| | Rowhammer | CacheWarp |
|---|---|---|
| **Restrictions** | *bit flips* | *old state* |
| **Speed** | 😭 | 😃 |
| **Practicality** | 😕 | 😐 |

# How to find such bugs?

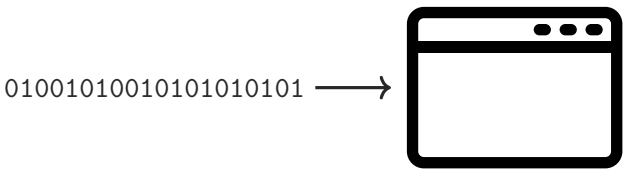
# Software Fuzzing



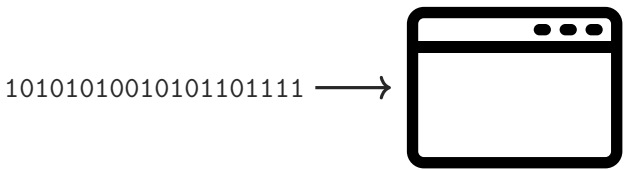
# Software Fuzzing



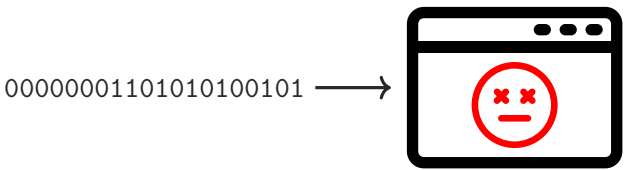
# Software Fuzzing



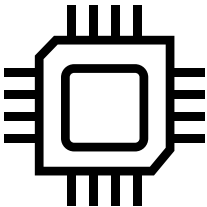
# Software Fuzzing



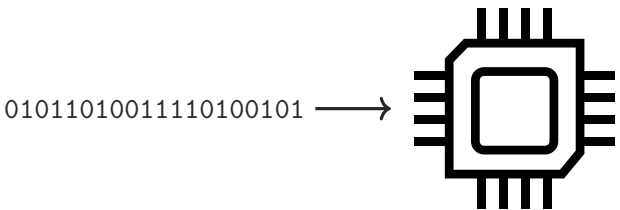
# Software Fuzzing



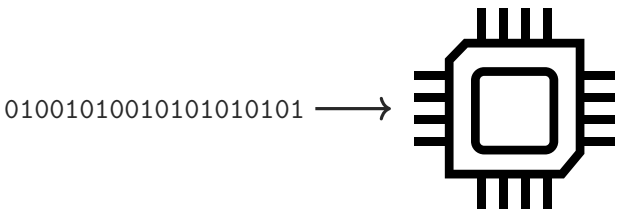
# Hardware Fuzzing



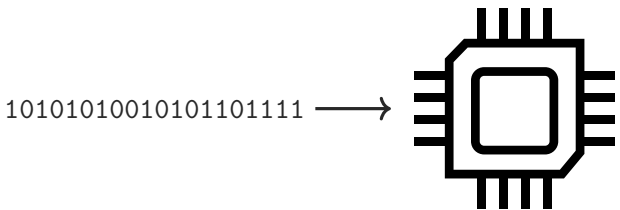
# Hardware Fuzzing



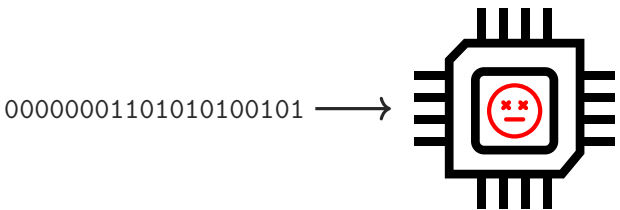
# Hardware Fuzzing



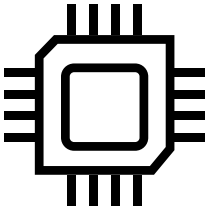
# Hardware Fuzzing

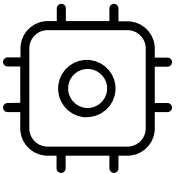


# Hardware Fuzzing

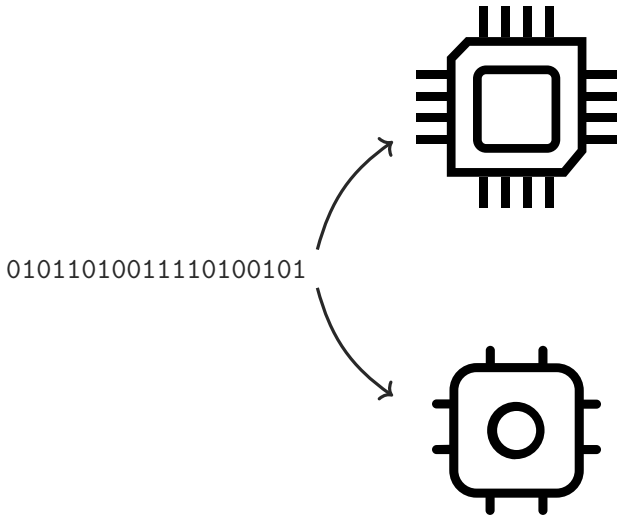


# Differential Fuzzing

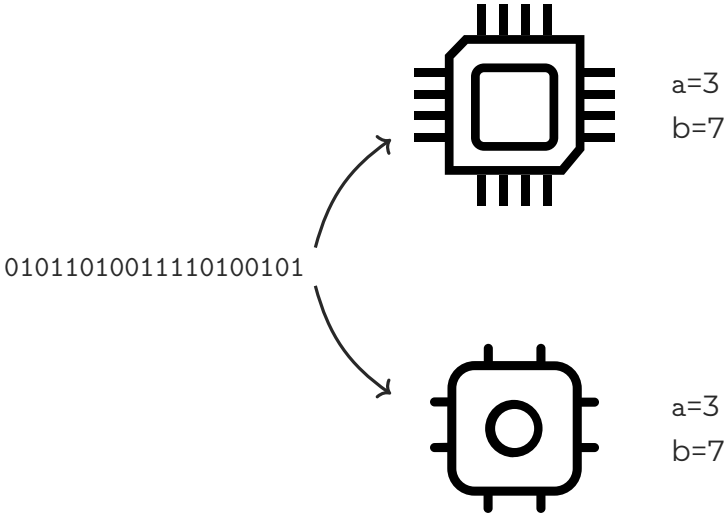




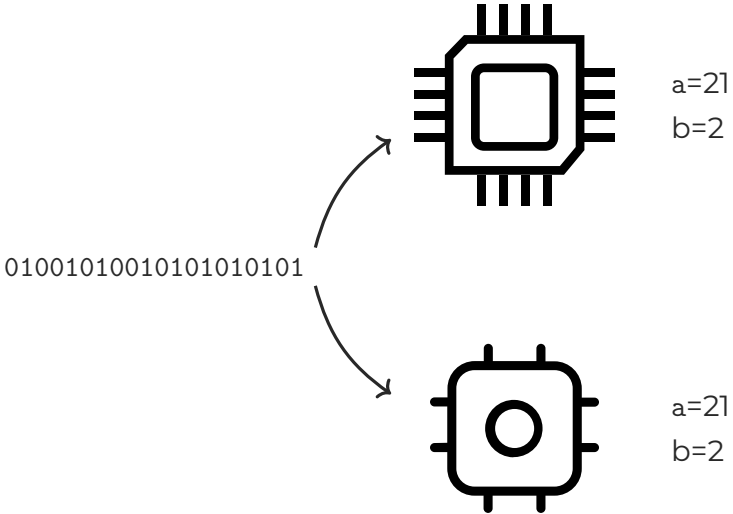
# Differential Fuzzing



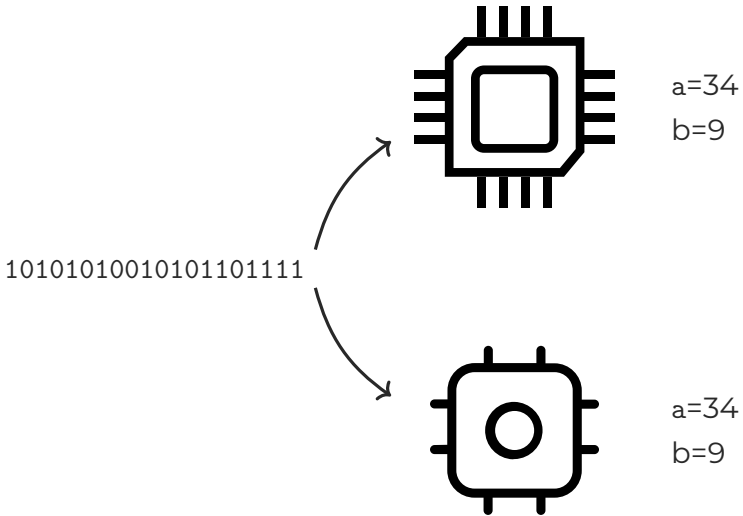
# Differential Fuzzing



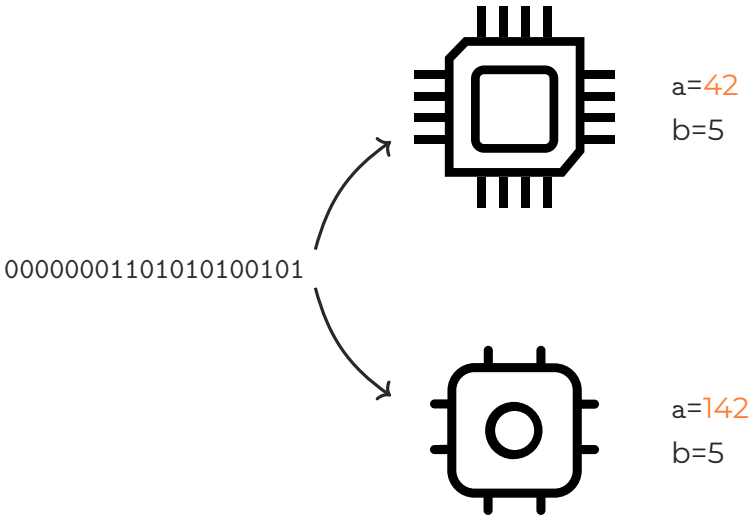
# Differential Fuzzing



# Differential Fuzzing



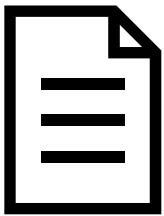
# Differential Fuzzing



# Instruction Set Architecture (ISA)

Specifies behavior

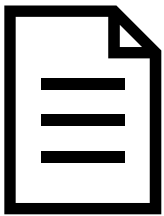
# Instruction Set Architecture (ISA)



Specifies behavior

Defines legal programs

# Instruction Set Architecture (ISA)

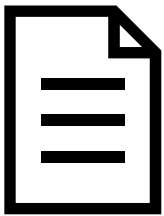


Specifies behavior

Defines legal programs

Licensing fees

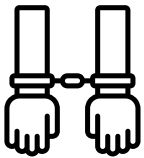
# Instruction Set Architecture (ISA)



Specifies behavior

Defines legal programs

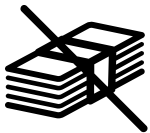
Licensing fees



Limited customization

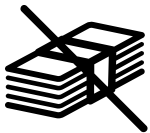

open, community-driven


open, community-driven



no licensing fees

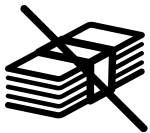

open, community-driven



no licensing fees

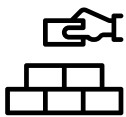
well designed


open, community-driven



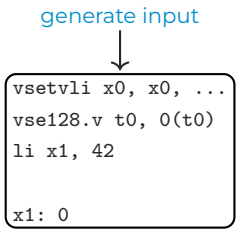
no licensing fees

well designed

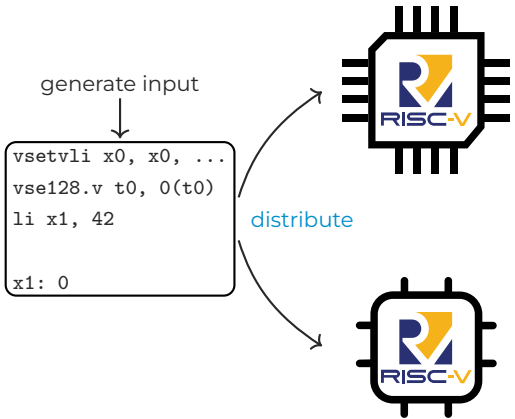


extensible

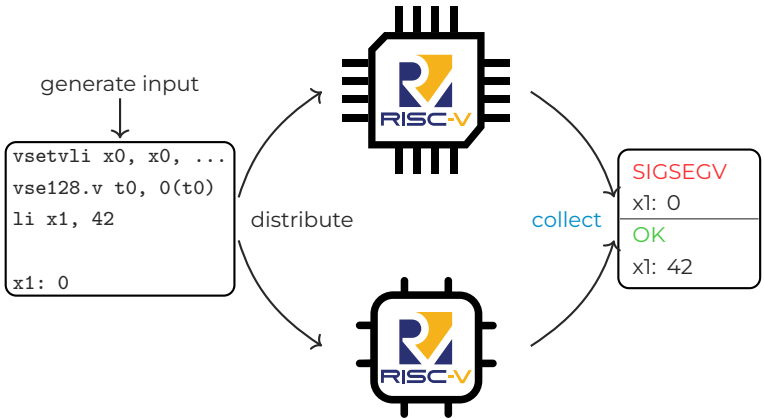
# Differential CPU Fuzzing



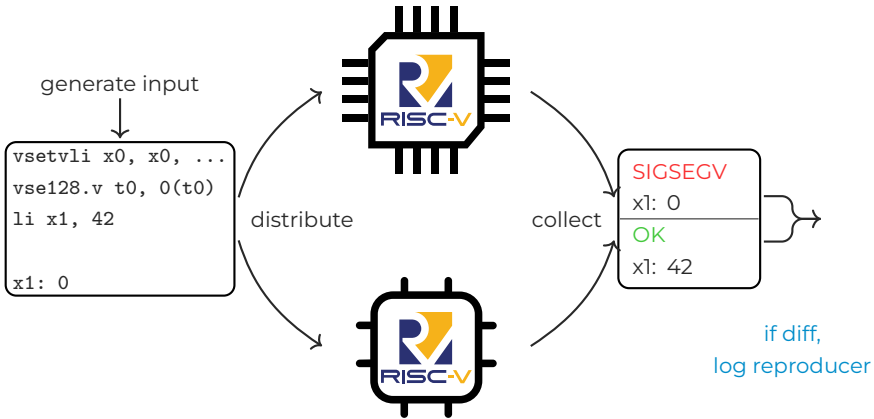
# Differential CPU Fuzzing



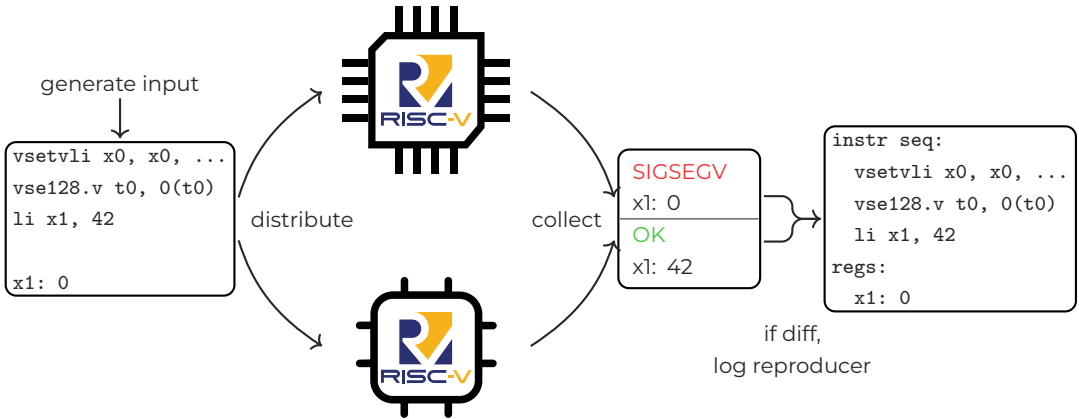
# Differential CPU Fuzzing



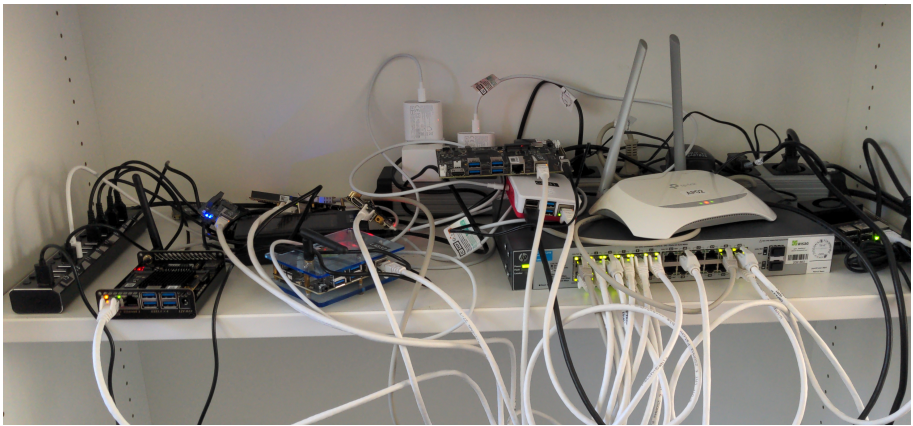
# Differential CPU Fuzzing



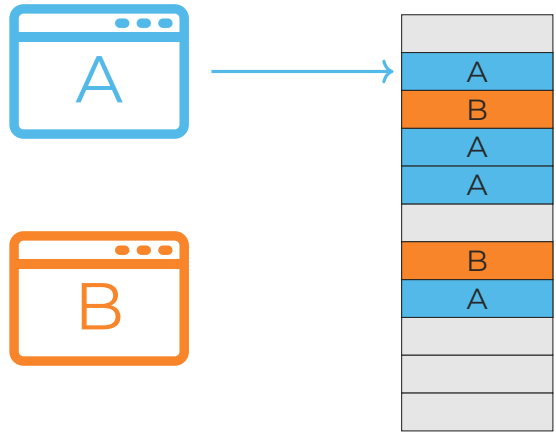
# Differential CPU Fuzzing



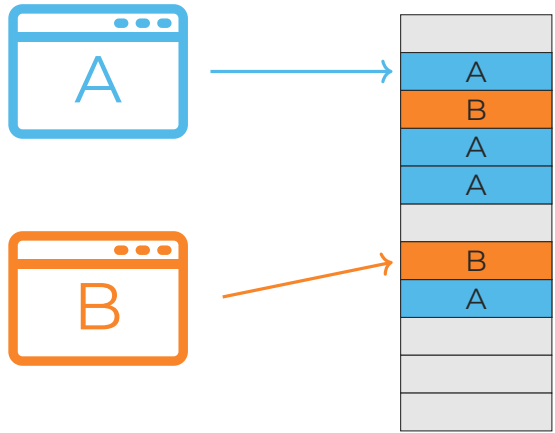
# Let’s start fuzzing!



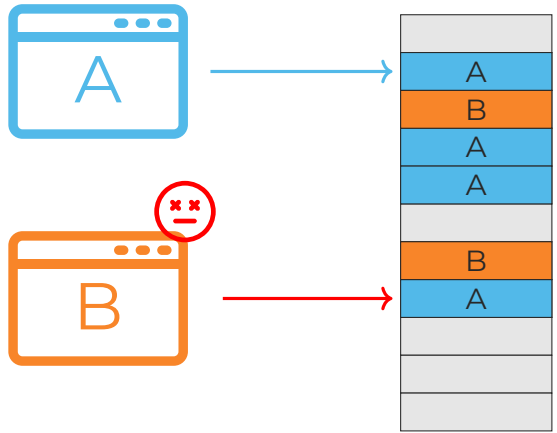
# Memory Isolation



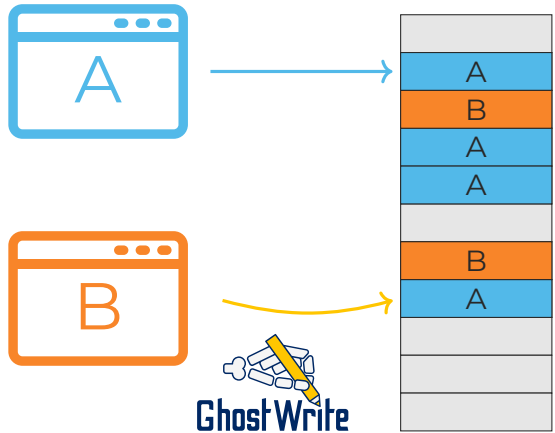
# Memory Isolation



# Memory Isolation



# ~~Memory Isolation~~: GhostWrite



# ~~Full Isolation~~: GhostWrite

Software World

# ~~Full Isolation~~: GhostWrite

Software World


Hardware World

# The Exploit

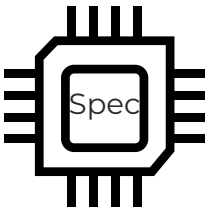
```
mv t0, addr
vmv.v.x v0, value
vsetvli zero, zero, e8, m1
vse128.v v0, 0(t0)
```

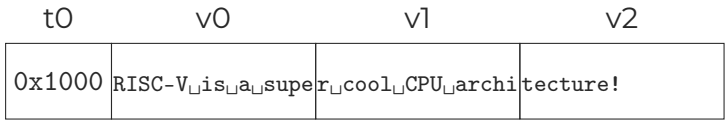
# The Exploit

```
mv t0, addr
vmv.v.x v0, value
vsetvli zero, zero, e8, m1
vse128.v v0, 0(t0)
```

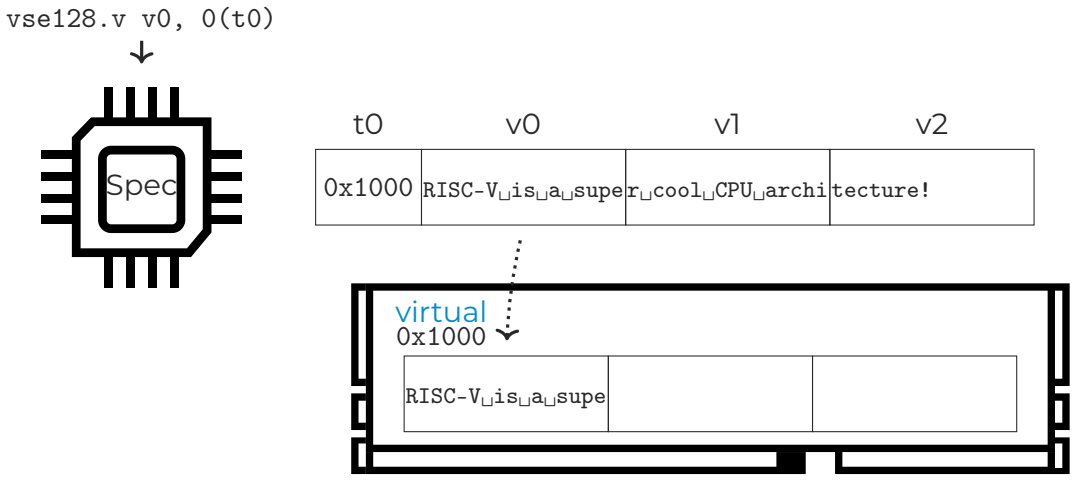
# GhostWrite: Investigation

```
vse128.v v0, 0(t0)
```



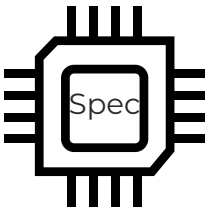



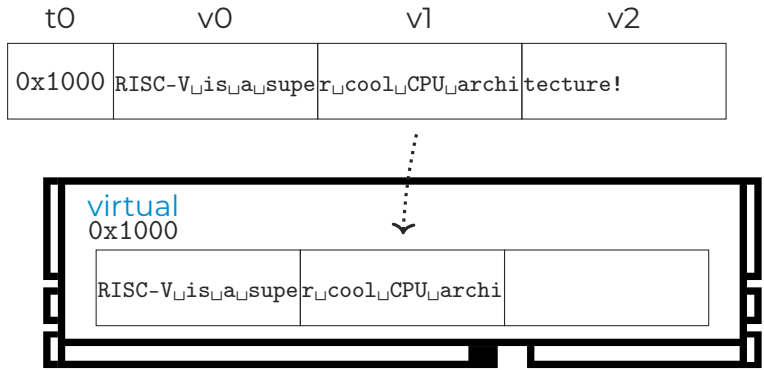

# GhostWrite: Investigation



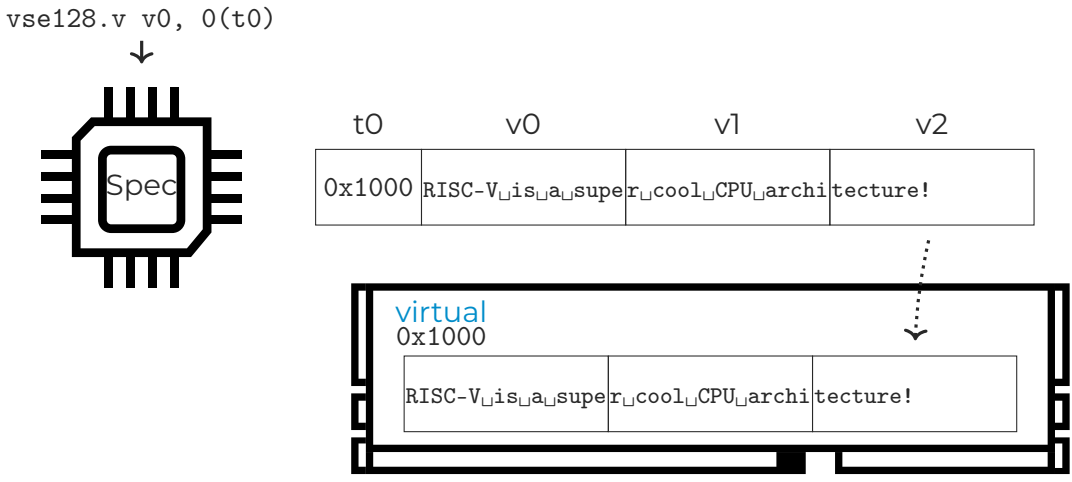
# GhostWrite: Investigation

```
vse128.v v0, 0(t0)
```

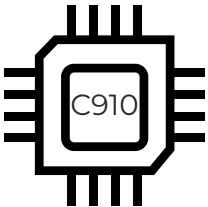


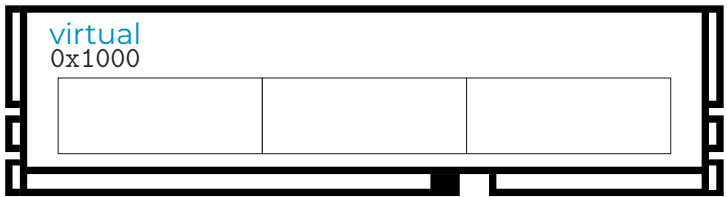
# GhostWrite: Investigation



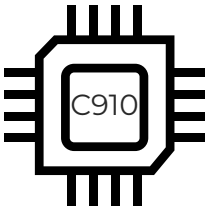
# GhostWrite: Investigation



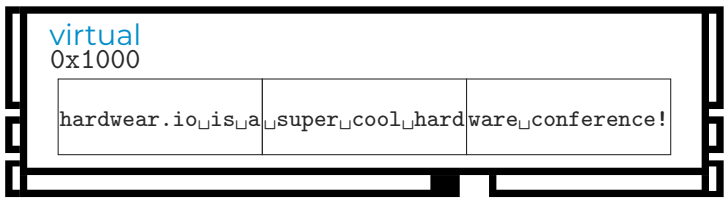

| t0 | v0 | v1 | v2 |
|---|---|---|---|
| 0x1000 | RISC-V␣is␣a␣super | ␣cool␣CPU␣archi | tecture! |



# GhostWrite: Investigation

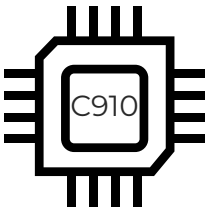


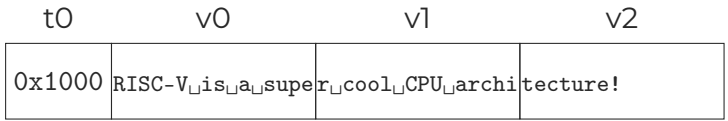

| t0 | v0 | v1 | v2 |
|---|---|---|---|
| 0x1000 | RISC-V␣is␣a␣super | r␣cool␣CPU␣archi | tecture! |

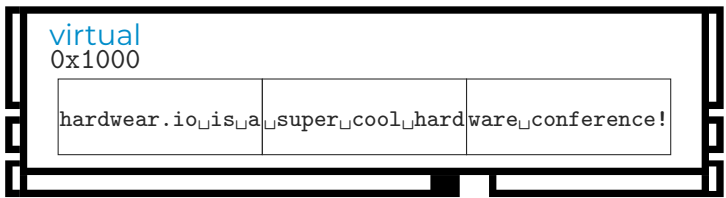


# GhostWrite: Investigation

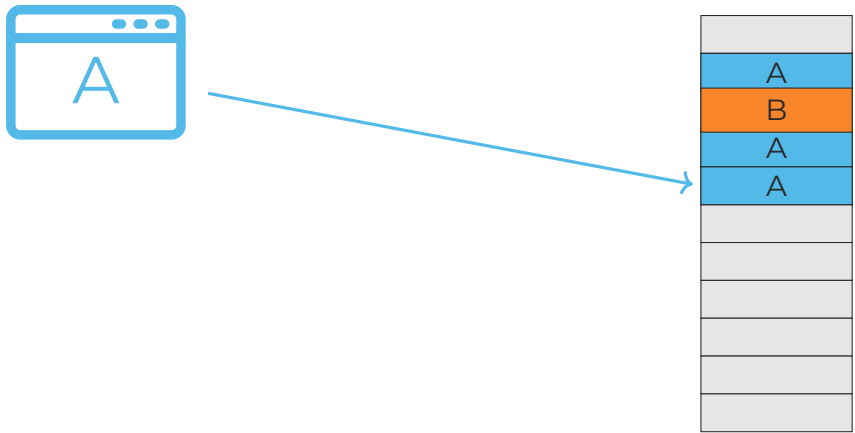
```
vse128.v v0, 0(t0)
```



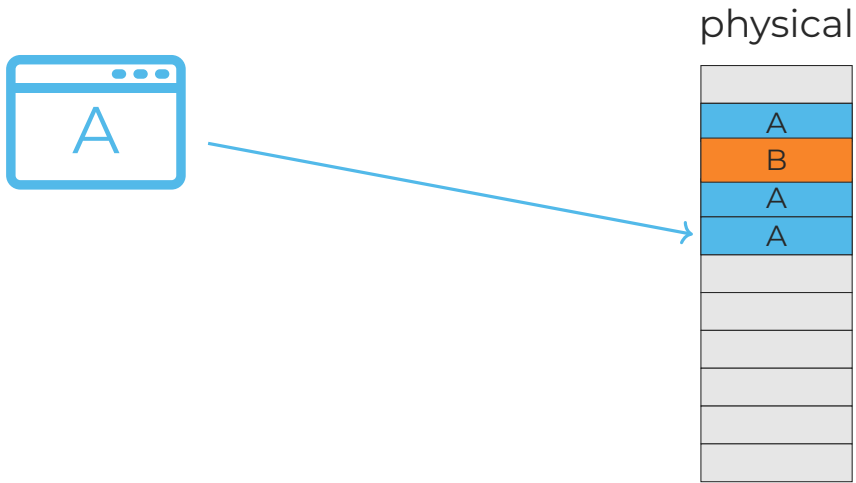



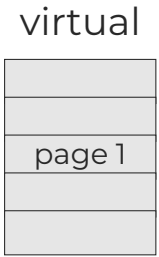
# GhostWrite: Virtual Memory

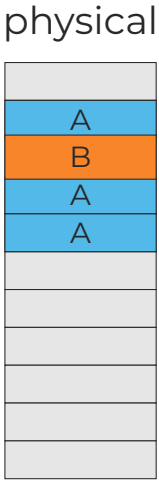


# GhostWrite: Virtual Memory

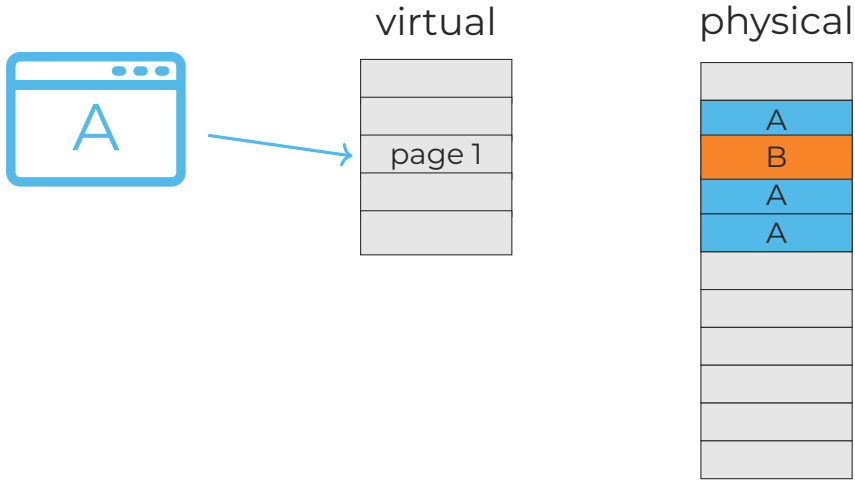


# GhostWrite: Virtual Memory

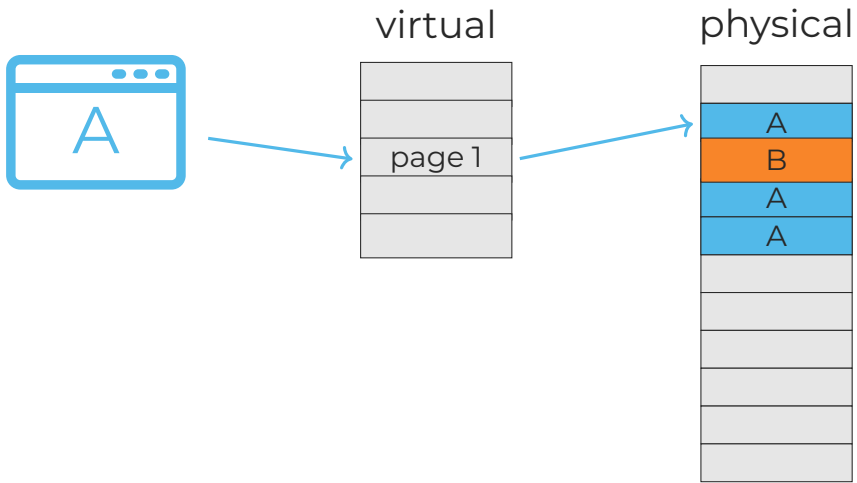



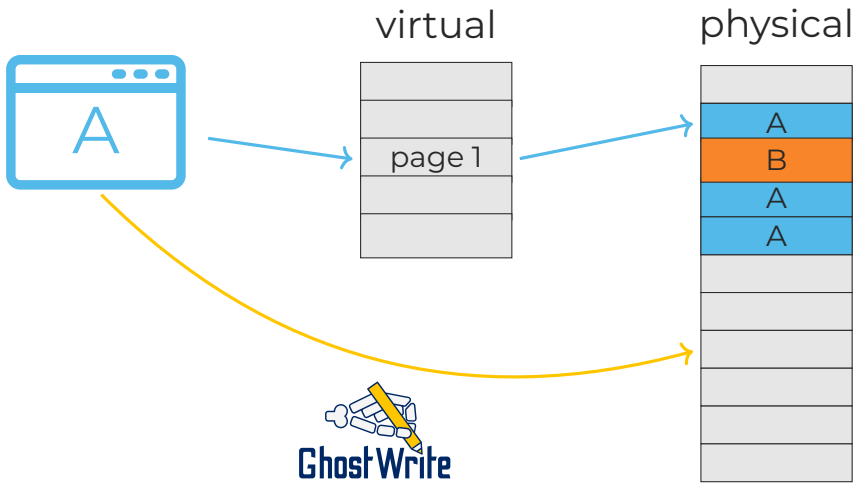
# GhostWrite: Virtual Memory



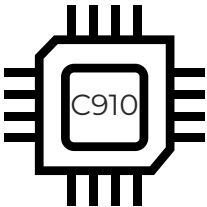
# GhostWrite: Virtual Memory



# GhostWrite: Virtual Memory



# GhostWrite: Investigation




| t0 | v0 | v1 | v2 |
|---|---|---|---|
| 0x1000 | RISC-V␣is␣a␣super | r␣cool␣CPU␣archi | tecture! |

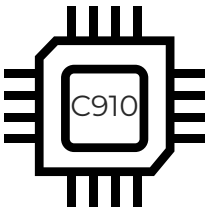
physical

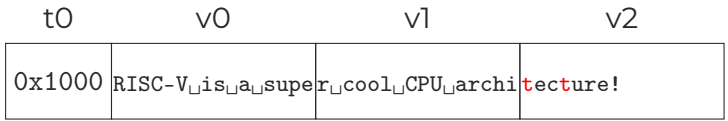
0x1000

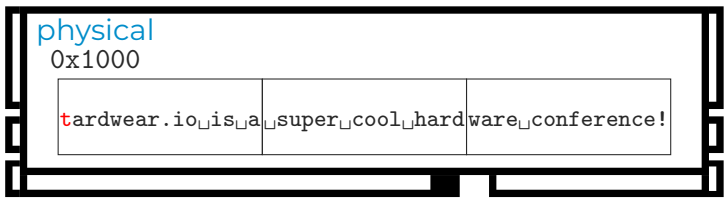
| | | |
|---|---|---|
| hardwear.io␣is␣a | ␣super␣cool␣hard | ware␣conference! |

# GhostWrite: Investigation

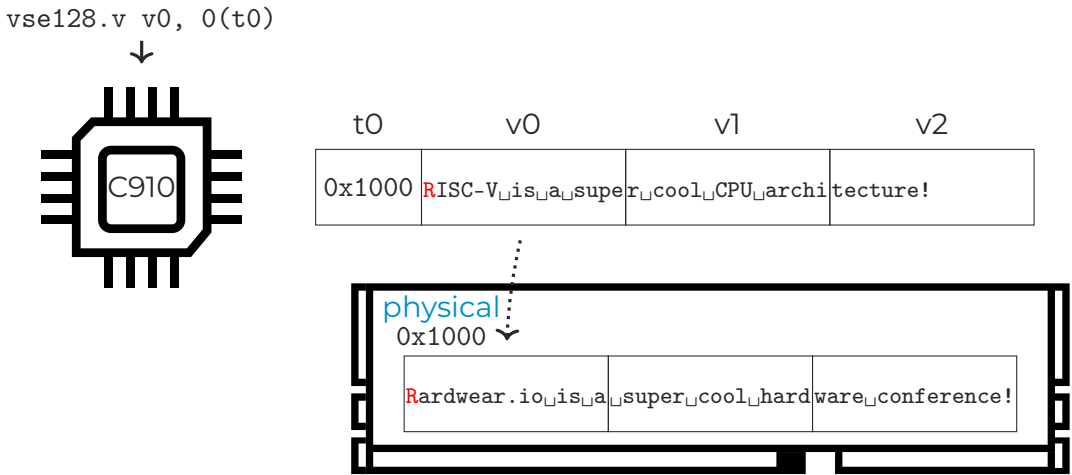
```
vse128.v v0, 0(t0)
```



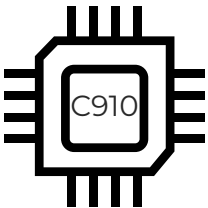



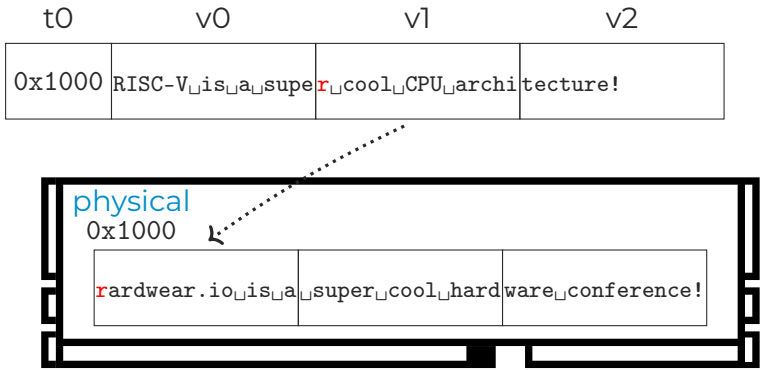
# GhostWrite: Investigation



# GhostWrite: Investigation

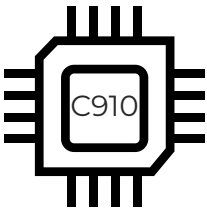
```
vse128.v v0, 0(t0)
```

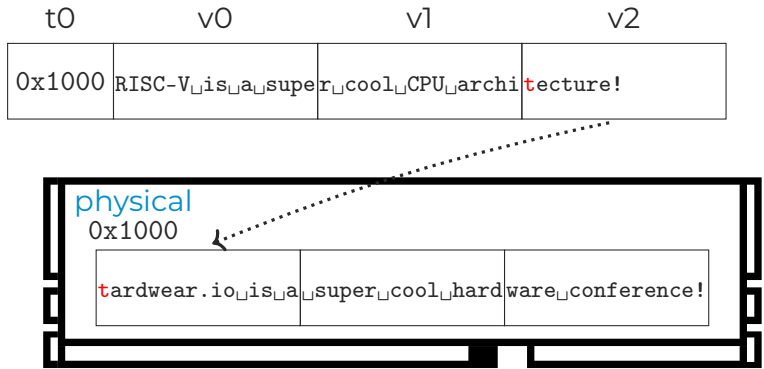


# GhostWrite: Investigation

```
vse128.v v0, 0(t0)
```



# T-Head XuanTie C910


- One of the fastest RISC-V CPUs

# T-Head XuanTie C910


- One of the fastest RISC-V CPUs
- 4 cores, 2GHz, vector extension

# T-Head XuanTie C910


- One of the fastest RISC-V CPUs
- 4 cores, 2GHz, vector extension
- Available in the cloud

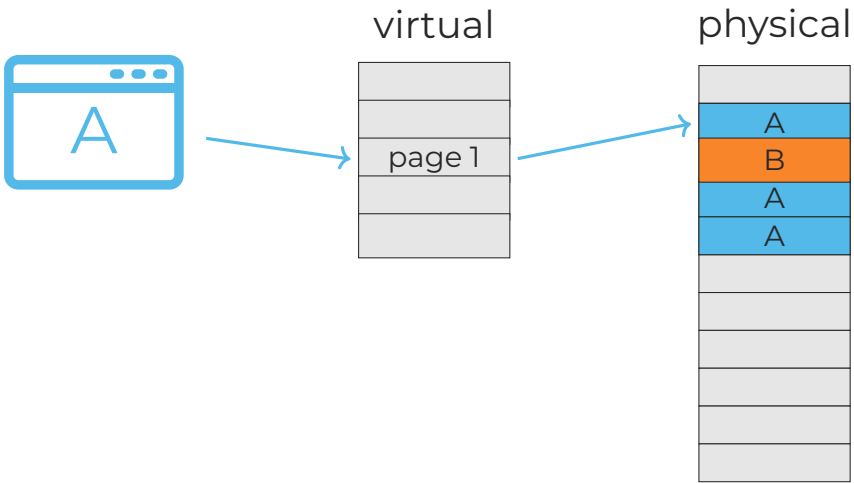
Scaleway

# T-Head XuanTie C910

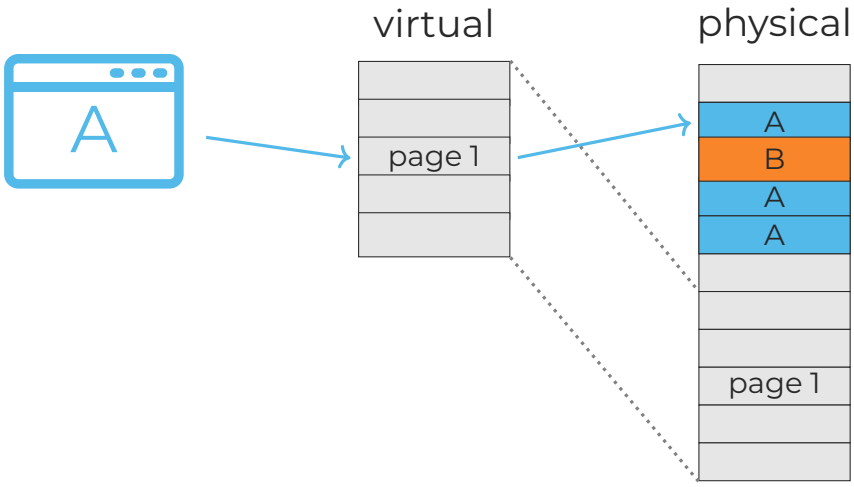

- One of the fastest RISC-V CPUs
- 4 cores, 2GHz, vector extension
- Available in the cloud
- . . . and laptops

Scaleway

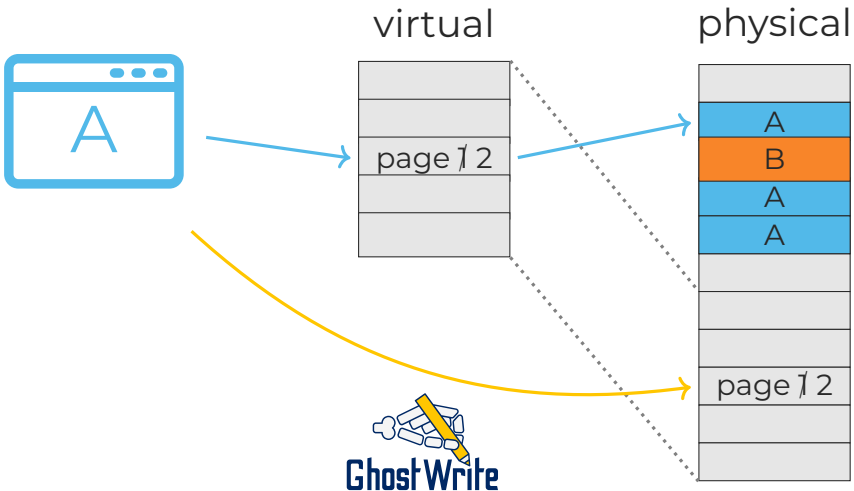
# Reading Arbitrary Memory



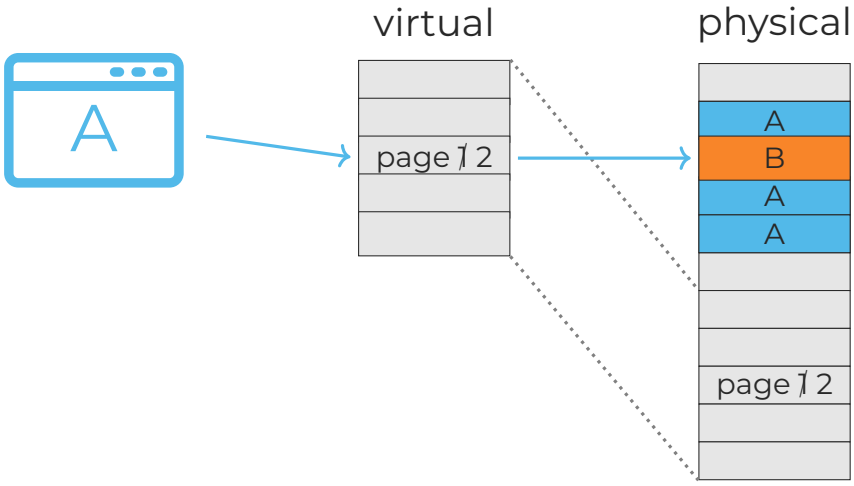
# Reading Arbitrary Memory



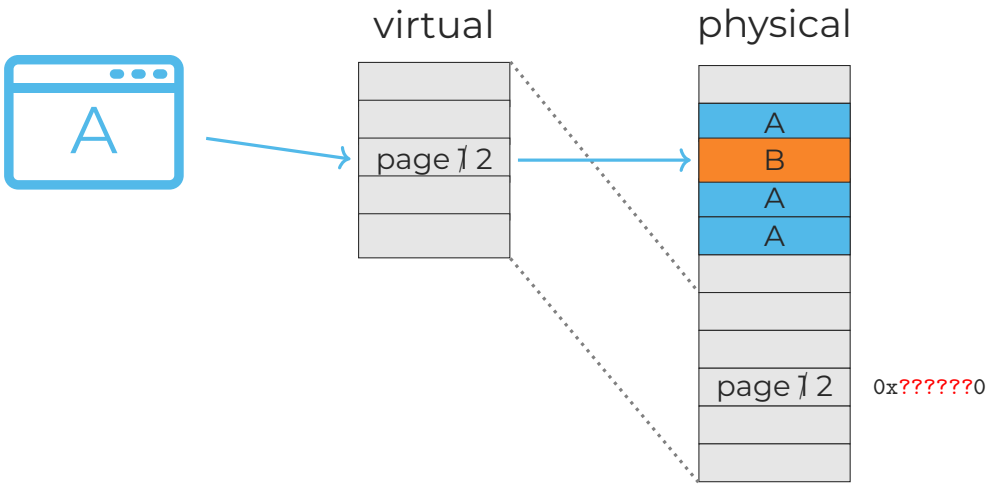
# Reading Arbitrary Memory



# Reading Arbitrary Memory



# Reading Arbitrary Memory



# Getting root

```
if kernel_get_user() is not root then
  require_authentication()
start_root_shell()
```


```
kernel_get_user:
process = get_current_process()
user = user_for_process(process)
return user
```

OS

# Getting root

```
if kernel_get_user() is not root then
  require_authentication()
start_root_shell()
```


syscall

```
kernel_get_user:
process = get_current_process()
user = user_for_process(process)
return user
```

OS

# Getting root

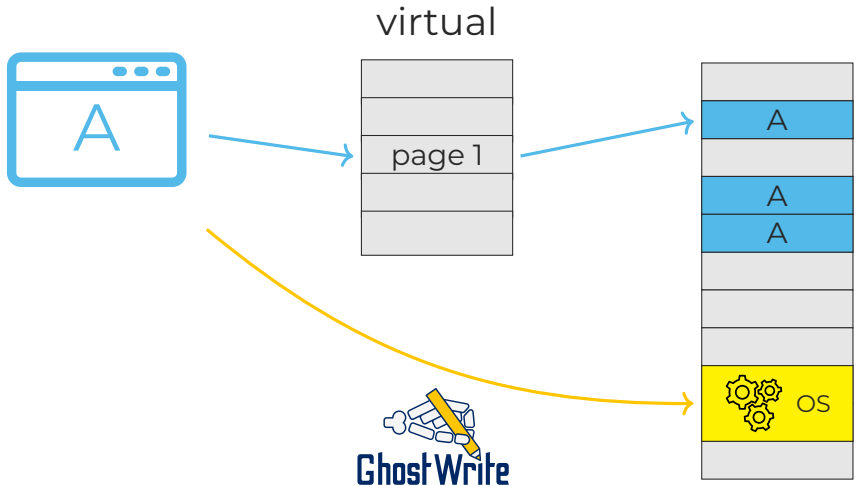
```
if kernel_get_user() is not root then
  require_authentication()
start_root_shell()
```


syscall

```
kernel_get_user:
process = get_current_process()
user = user_for_process(process)
return root
```

OS

# Getting root: Patching the Kernel



# Demo: Getting root

# GhostWrite: Mitigation


- C910 has no microcode

# GhostWrite: Mitigation


- C910 has no microcode
- OS mitigation: disable vector extension

# GhostWrite: Mitigation


- C910 has no microcode
- OS mitigation: disable vector extension
- Up to 33% overhead

# GhostWrite: Mitigation


- C910 has no microcode
- OS mitigation: disable vector extension
- Up to 33% overhead
- Lose $\sim$ 50% instructions

# GhostWrite: Comparison


| | Rowhammer | CacheWarp |
|---|---|---|
| **Restrictions** | *bit flips* | *old state* |
| **Speed** | 😭 | 😃 |
| **Practicality** | 😕 | 😐 |

# GhostWrite: Comparison

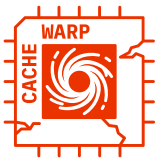
| |  Rowhammer |   CacheWarp |  GhostWrite |
|---|---|---|---|
| **Restrictions** | *bit flips* | *old state* | – |
| **Speed** | 😭 | 😀 | 😀 |
| **Practicality** | 😕 | 😐 | 😀 |

# Summary

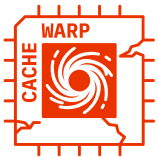
Lots of hardware bugs

# Summary

Lots of hardware bugs




# Summary

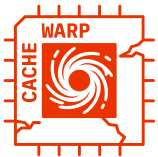




Lots of hardware bugs

We can't trust system if hardware broken

# Summary



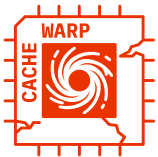


Lots of hardware bugs

We can't trust system if hardware broken

Quality control important

# Summary

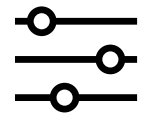




Lots of hardware bugs

We can't trust system if hardware broken

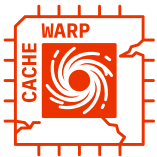
Quality control important



Remedy: Configurable hardware

# Summary

@fth0mas
fabianthomas.de

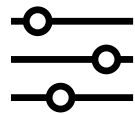




Lots of hardware bugs

We can't trust system if hardware broken

Quality control important



Remedy: Configurable hardware

# References

[Pod+18] D. Poddebniak, J. Somorovsky, S. Schinzel, M. Lochter, and P. Rösler. **Attacking deterministic signature schemes using fault attacks.** In: EuroS&P). 2018.

# C908 and C906

# Demo: Freezing the C906

# How to fix Hardware Bugs?


OS: disable extension

# How to fix Hardware Bugs?


OS: disable extension

up to 33% overhead

lose ~ 50% instructions

# How to fix Hardware Bugs?



OS: disable extension

up to 33% overhead

lose ~ 50% instructions

# How to fix Hardware Bugs?


OS: disable extension

up to 33% overhead

lose ~ 50% instructions


OS: disable extension

# How to fix Hardware Bugs?


OS: disable extension

up to 33% overhead

lose ~ 50% instructions


OS: disable extension

up to 77% overhead

lose ~ 50% instructions

# How to fix Hardware Bugs?


OS: disable extension

up to 33% overhead

lose ~ 50% instructions


OS: disable extension

up to 77% overhead

lose ~ 50% instructions

# How to fix Hardware Bugs?


OS: disable extension

up to 33% overhead

lose ~ 50% instructions

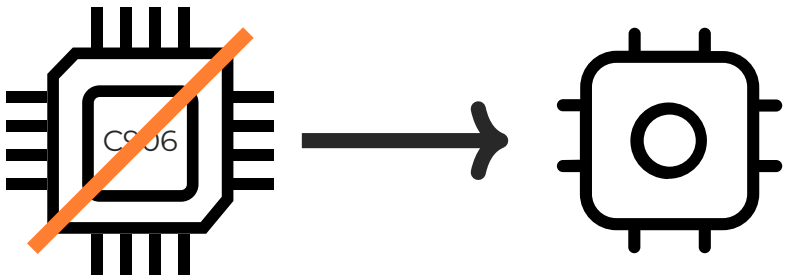

OS: disable extension

up to 77% overhead

lose ~ 50% instructions


no mitigation

# How to the fix C906?



# Demo: Reading Arbitrary Memory