Side-Channel Security

Chapter 3: Trusted Execution Environments and Confidential Computing

Sudheendra Neela

March 13, 2025

Graz University of Technology

Motivation

- Systems run software from various sources

Motivation

- Systems run software from various sources
- Protect computation against compromised OS

Motivation

- Systems run software from various sources
- Protect computation against compromised OS
- Protect system against malicious software

Motivation

- Systems run software from various sources
- Protect computation against compromised OS
- Protect system against malicious software
- Cloud servers: must run any untrusted virtual machines

Motivation

- Systems run software from various sources
- Protect computation against compromised OS
- Protect system against malicious software
- Cloud servers: must run any untrusted virtual machines
- Cloud VMs: may run in compromised or hostile enviroments

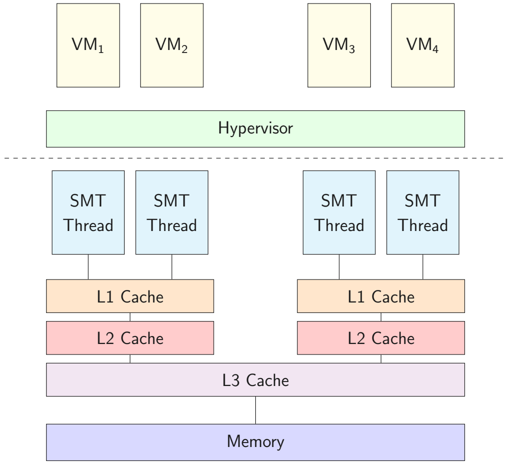
Motivation

- Systems run software from various sources
- Protect computation against compromised OS
- Protect system against malicious software
- Cloud servers: must run any untrusted virtual machines
- Cloud VMs: may run in compromised or hostile enviroments
- CPU providers: tamper-resistant mechanism

Motivation

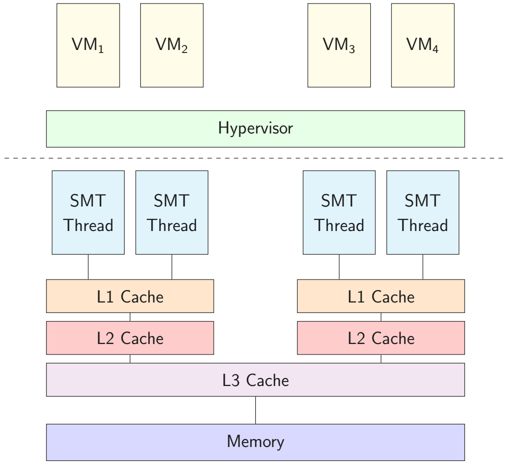
- Systems run software from various sources
- Protect computation against compromised OS
- Protect system against malicious software
- Cloud servers: must run any untrusted virtual machines
- Cloud VMs: may run in compromised or hostile enviroments
- CPU providers: tamper-resistant mechanism
- Key enabler of confidential computing

Virtualization



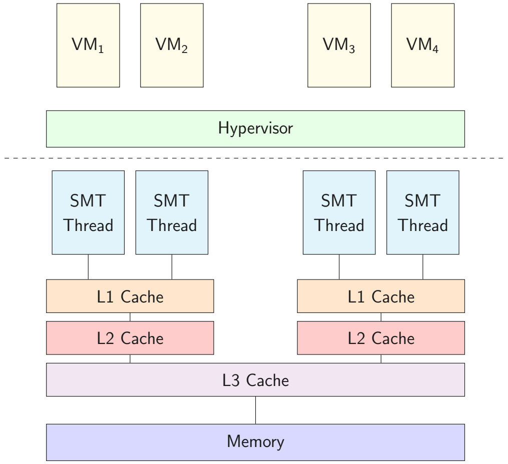
- Hypervisor: manages virtual machines (VMs)

Virtualization



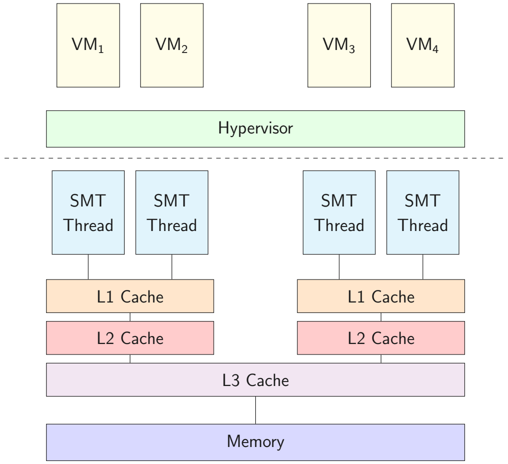
- Hypervisor: manages virtual machines (VMs)
- Traditional virtualization: Hypervisor has total control

Virtualization



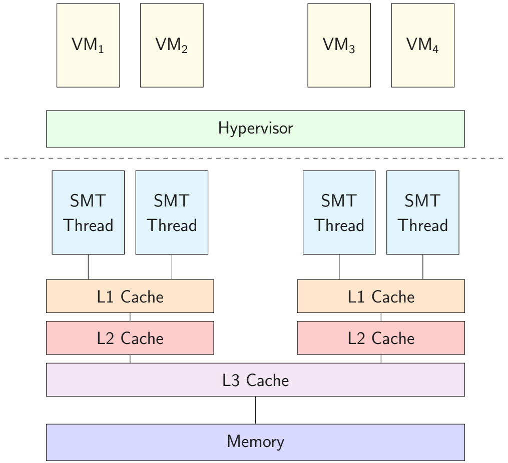
- Hypervisor: manages virtual machines (VMs)
- Traditional virtualization: Hypervisor has total control
- All VMs share components

Virtualization



- Hypervisor: manages virtual machines (VMs)
- Traditional virtualization: Hypervisor has total control
- All VMs share components
- Check out Cloud Operating Systems!

Virtualization



- Hypervisor: manages virtual machines (VMs)
- Traditional virtualization: Hypervisor has total control
- All VMs share components
- Check out Cloud Operating Systems!
- Confidential Computing (CoCo): hardware guarantees for secure VM operation

A Simple Correlation Attack [17] (2011)

- `VMEXIT`: VM hands control back to the hypervisor with a reason

A Simple Correlation Attack [17] (2011)

- `VMEXIT`: VM hands control back to the hypervisor with a reason:
    - `RDMSR`

A Simple Correlation Attack [17] (2011)

- `VMEXIT`: VM hands control back to the hypervisor with a reason:
    - `RDMSR`, `WRMSR`

A Simple Correlation Attack [17] (2011)

- `VMEXIT`: VM hands control back to the hypervisor with a reason:
  - `RDMSR`, `WRMSR`, `CPUID`

A Simple Correlation Attack [17] (2011)

- `VMEXIT`: VM hands control back to the hypervisor with a reason:
  - `RDMSR`, `WRMSR`, `CPUID`, Access Control Registers

A Simple Correlation Attack [17] (2011)

- `VMEXIT`: VM hands control back to the hypervisor with a reason:
    - `RDMSR`, `WRMSR`, `CPUID`, Access Control Registers, Debug Registers

A Simple Correlation Attack [17] (2011)

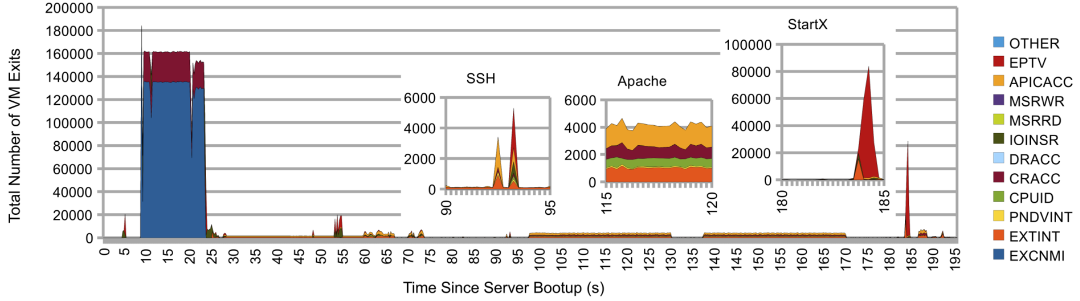
- `VMEXIT`: VM hands control back to the hypervisor with a reason:
  - `RDMSR`, `WRMSR`, `CPUID`, Access Control Registers, Debug Registers
- Number of `VMEXIT`s and reasons leak information

A Simple Correlation Attack [17] (2011)

- `VMEXIT`: VM hands control back to the hypervisor with a reason:
    - `RDMSR`, `WRMSR`, `CPUID`, Access Control Registers, Debug Registers
- Number of `VMEXIT`s and reasons leak information
- Infer what applications are running (Bootup, SSH, Apache, StartX)

A Simple Correlation Attack [17] (2011)

- `VMEXIT`: VM hands control back to the hypervisor with a reason:
  - `RDMSR`, `WRMSR`, `CPUID`, Access Control Registers, Debug Registers
- Number of `VMEXIT`s and reasons leak information
- Infer what applications are running (Bootup, SSH, Apache, StartX)



TEE & CoCo Throughout The Years

- ARM TrustZone — 2009 [2]

TEE & CoCo Throughout The Years

- ARM TrustZone — 2009 [2]
    - Samsung Knox

TEE & CoCo Throughout The Years

- ARM TrustZone — 2009 [2]
    - Samsung Knox
- Intel Software Guard Extensions (SGX) — 2015 [5]

TEE & CoCo Throughout The Years

- ARM TrustZone — 2009 [2]
    - Samsung Knox
- Intel Software Guard Extensions (SGX) — 2015 [5]
- AMD Secure Encrypted Virtualization (SEV) — 2016 [11]

TEE & CoCo Throughout The Years

- ARM TrustZone — 2009 [2]
    - Samsung Knox
- Intel Software Guard Extensions (SGX) — 2015 [5]
- AMD Secure Encrypted Virtualization (SEV) — 2016 [11]
- AMD SEV with Encrypted State (SEV-ES) — 2017 [10]

TEE & CoCo Throughout The Years

- ARM TrustZone — 2009 [2]
  - Samsung Knox
- Intel Software Guard Extensions (SGX) — 2015 [5]
- AMD Secure Encrypted Virtualization (SEV) — 2016 [11]
- AMD SEV with Encrypted State (SEV-ES) — 2017 [10]
- AMD SEV Secure Nested Paging (SEV-SNP) — 2020 [1]

TEE & CoCo Throughout The Years

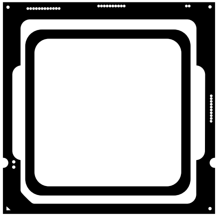
- ARM TrustZone — 2009 [2]
  - Samsung Knox
- Intel Software Guard Extensions (SGX) — 2015 [5]
- AMD Secure Encrypted Virtualization (SEV) — 2016 [11]
- AMD SEV with Encrypted State (SEV-ES) — 2017 [10]
- AMD SEV Secure Nested Paging (SEV-SNP) — 2020 [1]
- Intel Trusted Domain Extensions (TDX) — 2021 [8]

TEE & CoCo Throughout The Years

- ARM TrustZone — 2009 [2]
  - Samsung Knox
- Intel Software Guard Extensions (SGX) — 2015 [5]
- AMD Secure Encrypted Virtualization (SEV) — 2016 [11]
- AMD SEV with Encrypted State (SEV-ES) — 2017 [10]
- AMD SEV Secure Nested Paging (SEV-SNP) — 2020 [1]
- Intel Trusted Domain Extensions (TDX) — 2021 [8]
- ARM Confidential Computing Architecture (CCA) — 2021 [3]

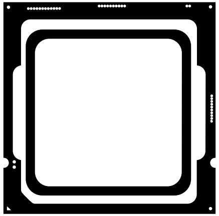
Intel Software Guard Extension (SGX)

Intel SGX Overview



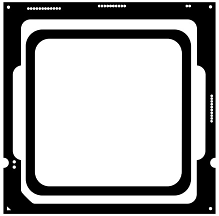
- x86 instruction-set extension

Intel SGX Overview



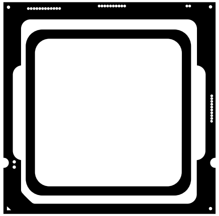
- x86 instruction-set extension
- Isolate trusted code from untrusted applications

Intel SGX Overview



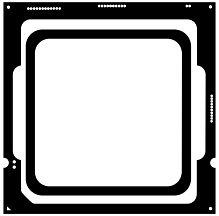
- x86 instruction-set extension
- Isolate trusted code from untrusted applications
- The OS cannot access enclave memory

Intel SGX Overview



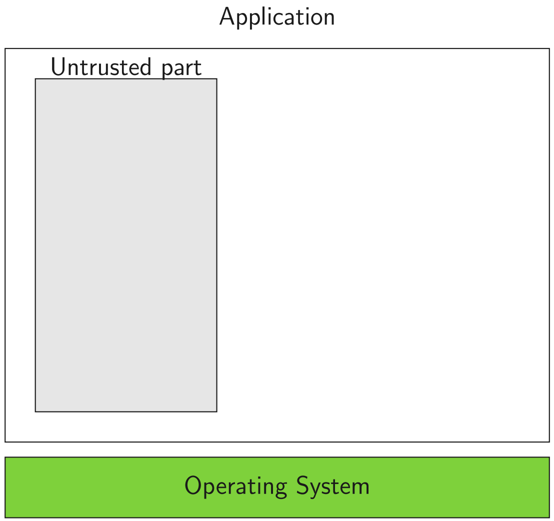
- x86 instruction-set extension
- Isolate trusted code from untrusted applications
- The OS cannot access enclave memory
- Enclave memory is encrypted and integrity protected

Intel SGX Overview

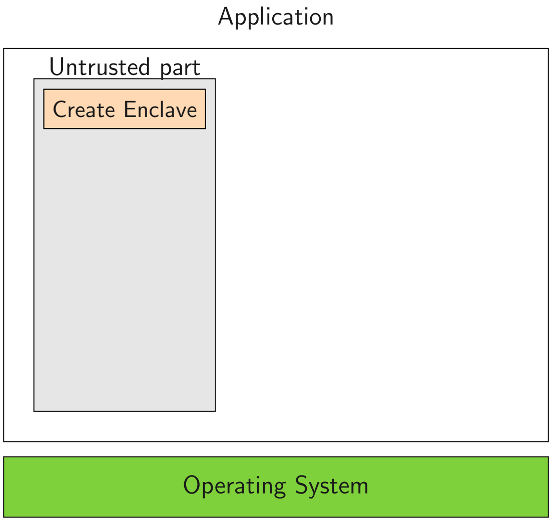


- x86 instruction-set extension
- Isolate trusted code from untrusted applications
- The OS cannot access enclave memory
- Enclave memory is encrypted and integrity protected
- Enclave has full access to virtual memory of host application

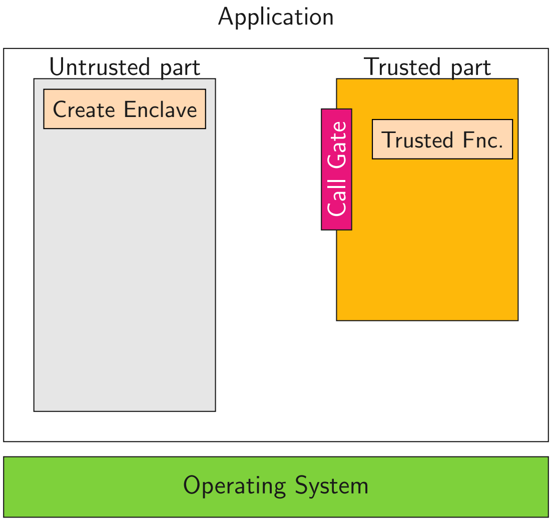
SGX Model



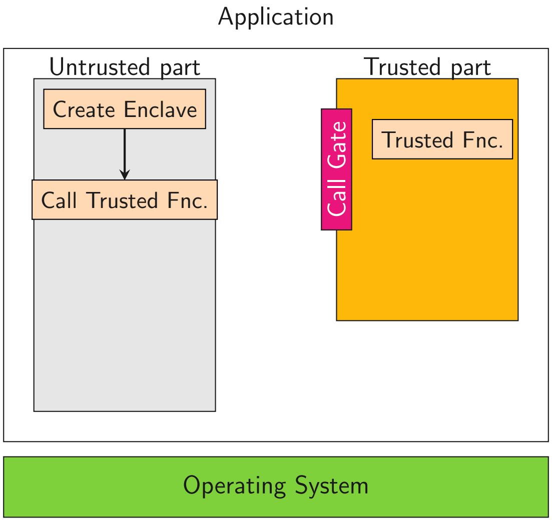
SGX Model


Application
Untrusted part
Create Enclave
Operating System

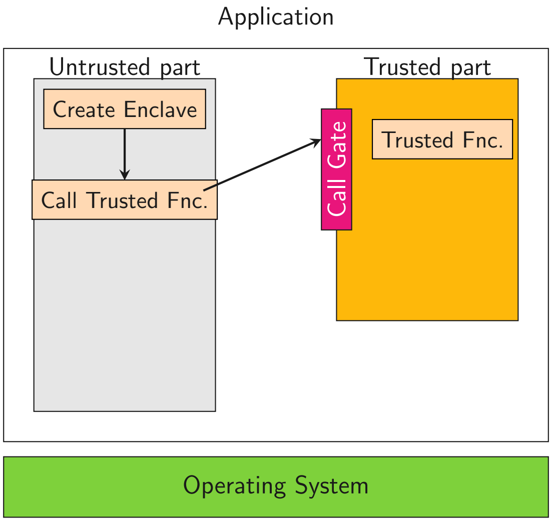
SGX Model


Application
Untrusted part
Create Enclave
Trusted part
Call Gate
Trusted Fnc.
Operating System

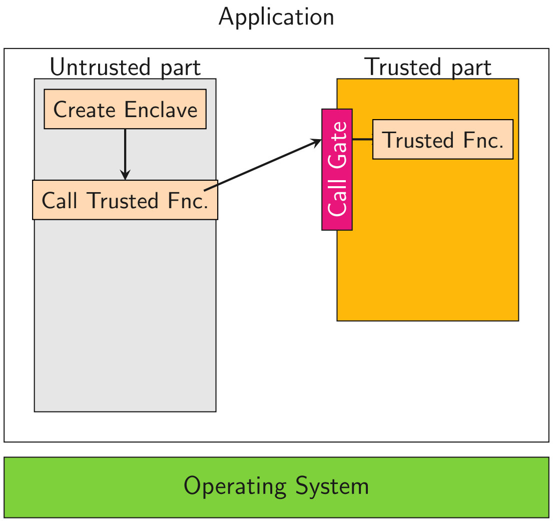
SGX Model


Application
Untrusted part
Create Enclave
Call Trusted Fnc.
Trusted part
Call Gate
Trusted Fnc.
Operating System

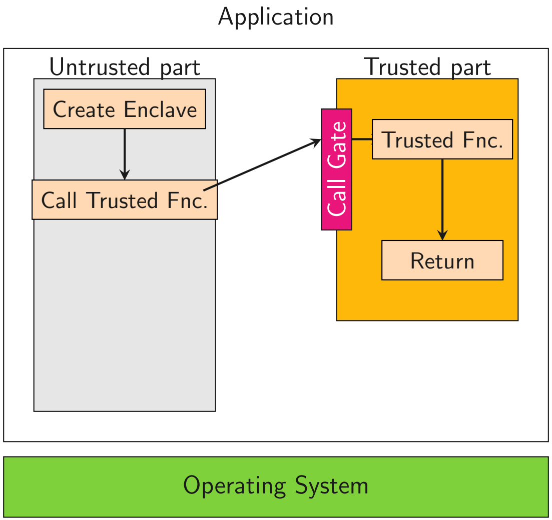
SGX Model


Application
Untrusted part
Create Enclave
Call Trusted Fnc.
Trusted part
Call Gate
Trusted Fnc.
Operating System

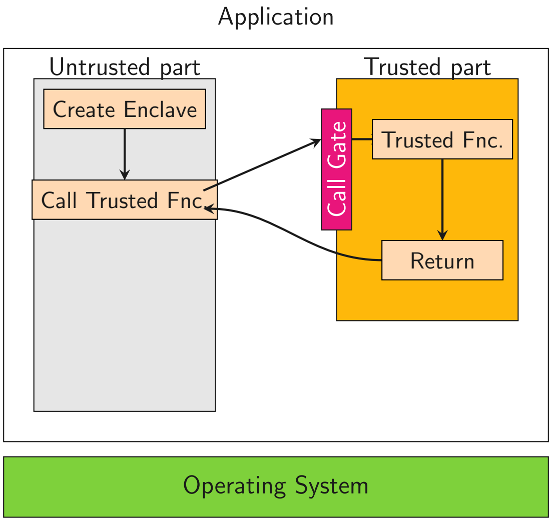
SGX Model


Application
Untrusted part
Trusted part
Create Enclave
Call Trusted Fnc.
Call Gate
Trusted Fnc.
Operating System

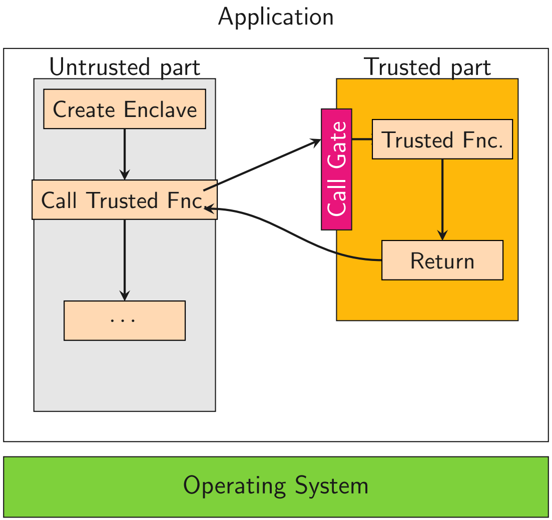
SGX Model


Application
Untrusted part
Create Enclave
Call Trusted Fnc.
Trusted part
Call Gate
Trusted Fnc.
Return
Operating System

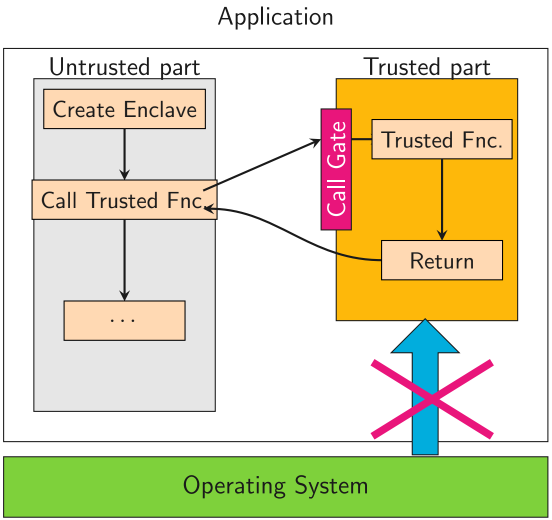
SGX Model


Application
Untrusted part
Trusted part
Create Enclave
Call Trusted Fnc.
Call Gate
Trusted Fnc.
Return
Operating System

SGX Model


Application
Untrusted part
Trusted part
Create Enclave
Call Trusted Fnc.
Call Gate
Trusted Fnc.
Return
. . .
Operating System

SGX Model


Application
Untrusted part
Trusted part
Create Enclave
Call Trusted Fnc.
. . .
Call Gate
Trusted Fnc.
Return
Operating System

Threat Model

- Attacking the enclave: malicious OS

Threat Model

- Attacking the enclave: malicious OS
- Attacking the OS: malicious enclave

Threat Model

- Attacking the enclave: malicious OS
- Attacking the OS: malicious enclave
- Side-Channel Attacks are out of scope

Threat Model

- Attacking the enclave: malicious OS
- Attacking the OS: malicious enclave
- Side-Channel Attacks are out of scope
- Only CPU is trusted

Attack Targets

What are some components of a system?

Attack Targets

What are some components of a system?

Cache

Attack Targets

What are some components of a system?

Cache

Page Table

Attack Targets

What are some components of a system?

Cache

Page Table

DRAM

Attack Targets

What are some components of a system?

Cache

Page Table

DRAM

Network

Attack Targets

What are some components of a system?

Cache

Page Table

DRAM

Network

Predictors

Attack Targets

What are some components of a system?

Cache Page Table DRAM Network Predictors Interrupt

Attack Targets

What are some components of a system?

Cache

Page Table

DRAM

Network

Predictors

Interrupt

CPU Ports

Attack Targets

What are some components of a system?

Cache

Page Table

DRAM

Network

Predictors

Interrupt

CPU Ports

Power

Attack Targets

What are some components of a system?

Cache

Page Table

DRAM

Network

Predictors

Interrupt

CPU Ports

Power

Counters

Attack Targets

What are some components of a system?

Cache

Page Table

DRAM

Network

Predictors

Interrupt

CPU Ports

Power

Counters

Fault Attacks (Lecture 4)

Attack Targets

What are some components of a system?

Cache

Page Table

DRAM

Network

Predictors

Interrupt

CPU Ports

Power

Counters

Fault Attacks (Lecture 4)

Transient Execution (Lecture 3)

Attack Targets

What are some components of a system?

Cache

Page Table

DRAM

Network

Predictors

Interrupt

CPU Ports

Power

Counters

Fault Attacks (Lecture 4)

Transient Execution (Lecture 3)

Read “SoK: SGX.Fail: How Stuff Gets eXposed” [20]

Side-Channel Attacks on Intel SGX

Controlled-Channel Attacks [24]

- Target mechanism which translates virtual to physical addresses

Controlled-Channel Attacks [24]

- Target mechanism which translates virtual to physical addresses
- Enclave memory is set up by OS

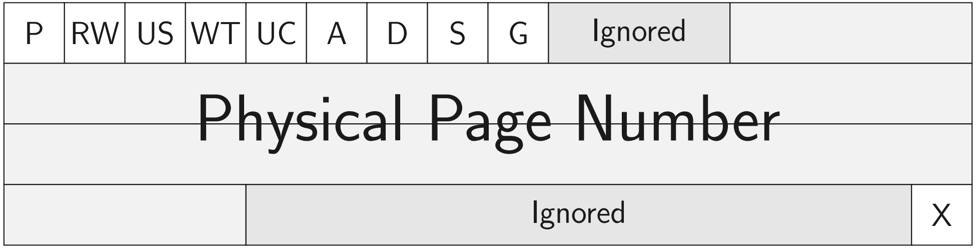
Controlled-Channel Attacks [24]

- Target mechanism which translates virtual to physical addresses
- Enclave memory is set up by OS
- Consequence: OS can unmap page, observe page fault

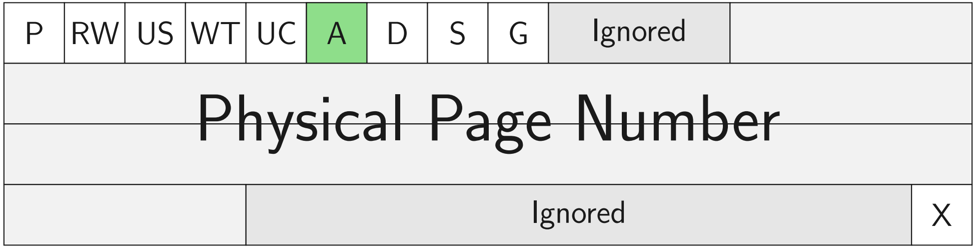
Controlled-Channel Attacks [24]

- Target mechanism which translates virtual to physical addresses
- Enclave memory is set up by OS
- Consequence: OS can unmap page, observe page fault
- Granularity: 1 page (4kB)

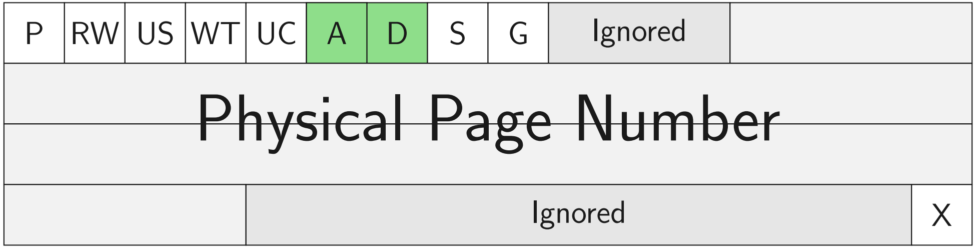
Stealthier Controlled-Channel Attacks [19, 21]



Stealthier Controlled-Channel Attacks [19, 21]



Stealthier Controlled-Channel Attacks [19, 21]



DRAM Attacks [21]

- Enclaves share same physical range of memory

DRAM Attacks [21]

- Enclaves share same physical range of memory
- DRAM contains row buffers

DRAM Attacks [21]

- Enclaves share same physical range of memory
- DRAM contains row buffers
- Use row conflicts to spy on victim

DRAM Attacks [21]

- Enclaves share same physical range of memory
- DRAM contains row buffers
- Use row conflicts to spy on victim
- Granularity: 512B to 8KB

Cache Attacks

- Flush+Reload not possible

Cache Attacks

- Flush+Reload not possible, Prime+Probe is possible

Cache Attacks

- Flush+Reload not possible, Prime+Probe is possible
- Physical address determines cache set

Cache Attacks

- Flush+Reload not possible, Prime+Probe is possible
- Physical address determines cache set
- Easy to prime cache set as OS

Cache Attacks

- Flush+Reload not possible, Prime+Probe is possible
- Physical address determines cache set
- Easy to prime cache set as OS
- Examples: [15], [21], [4], [14]

Malicious Enclave

- SGX Bomb [9]: Rowhammer within enclave

Malicious Enclave

- SGX Bomb [9]: Rowhammer within enclave, cause bit flips

Malicious Enclave

- SGX Bomb [9]: Rowhammer within enclave, cause bit flips, integrity check fail

Malicious Enclave

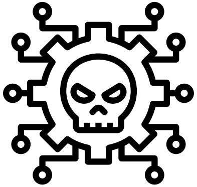
- SGX Bomb [9]: Rowhammer within enclave, cause bit flips, integrity check fail, system lock

Malicious Enclave

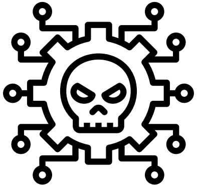
- SGX Bomb [9]: Rowhammer within enclave, cause bit flips, integrity check fail, system lock ⇒ Denial of Service

Malicious Enclave

- SGX Bomb [9]: Rowhammer within enclave, cause bit flips, integrity check fail, system lock $\Rightarrow$ Denial of Service
- Another Flip in the Wall of Rowhammer Defenses [7]:

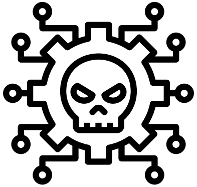


Malicious Enclave



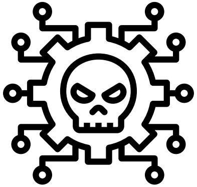
- SGX Bomb [9]: Rowhammer within enclave, cause bit flips, integrity check fail, system lock $\Rightarrow$ Denial of Service
- Another Flip in the Wall of Rowhammer Defenses [7]:
    - A new hammering technique bypasses all rowhammer defenses

Malicious Enclave



- SGX Bomb [9]: Rowhammer within enclave, cause bit flips, integrity check fail, system lock $\Rightarrow$ Denial of Service
- Another Flip in the Wall of Rowhammer Defenses [7]:
  - A new hammering technique bypasses all rowhammer defenses
  - Leverages SGX to be stealthy

Malicious Enclave



- SGX Bomb [9]: Rowhammer within enclave, cause bit flips, integrity check fail, system lock ⇒ Denial of Service
- Another Flip in the Wall of Rowhammer Defenses [7]:
  - A new hammering technique bypasses all rowhammer defenses
  - Leverages SGX to be stealthy
  - SGX prevents inspection of enclave memory

Malicious Enclave

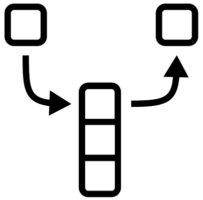
- SGX Bomb [9]: Rowhammer within enclave, cause bit flips, integrity check fail, system lock $\Rightarrow$ Denial of Service
- Another Flip in the Wall of Rowhammer Defenses [7]:
  - A new hammering technique bypasses all rowhammer defenses
  - Leverages SGX to be stealthy
  - SGX prevents inspection of enclave memory
  - SGX makes it hard for host OS to detect enclave's behavior by excluding CPU perfomance counter tracking

Malicious Enclave

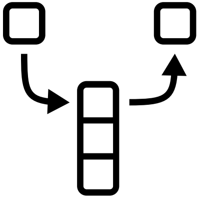
- SGX Bomb [9]: Rowhammer within enclave, cause bit flips, integrity check fail, system lock $\Rightarrow$ Denial of Service
- Another Flip in the Wall of Rowhammer Defenses [7]:
  - A new hammering technique bypasses all rowhammer defenses
  - Leverages SGX to be stealthy
  - SGX prevents inspection of enclave memory
  - SGX makes it hard for host OS to detect enclave's behavior by excluding CPU perfomance counter tracking $\Rightarrow$ perfect way to be stealthy
- ...bizarre threat model: Why would an enclave be malicious?

SGX ROP [16]: A Malicious Enclave

- From enclave: host application's memory is accessible, but only if mapped

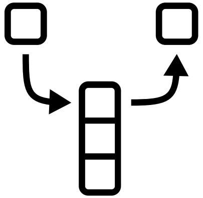


SGX ROP [16]: A Malicious Enclave



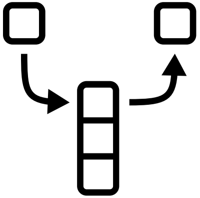
- From enclave: host application's memory is accessible, but only if mapped
- If enclave reads unmapped host memory

SGX ROP [16]: A Malicious Enclave



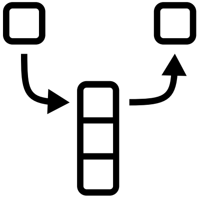
- From enclave: host application's memory is accessible, but only if mapped
- If enclave reads unmapped host memory ⇒ terminated

SGX ROP [16]: A Malicious Enclave



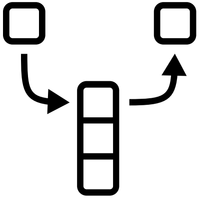
- From enclave: host application's memory is accessible, but only if mapped
- If enclave reads unmapped host memory ⇒ terminated
- Transactional Synchronization Extensions (TSX): hardware support for transactional memory

SGX ROP [16]: A Malicious Enclave



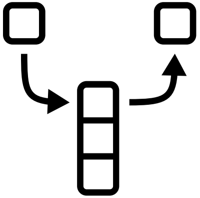
- From enclave: host application's memory is accessible, but only if mapped
- If enclave reads unmapped host memory ⇒ terminated
- Transactional Synchronization Extensions (TSX): hardware support for transactional memory
- Enclave wraps memory access inside a TSX transaction

SGX ROP [16]: A Malicious Enclave



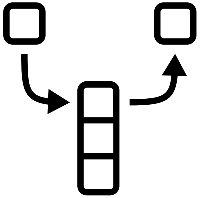
- From enclave: host application's memory is accessible, but only if mapped
- If enclave reads unmapped host memory ⇒ terminated
- Transactional Synchronization Extensions (TSX): hardware support for transactional memory
- Enclave wraps memory access inside a TSX transaction
- If accessible: transaction completes successfully

SGX ROP [16]: A Malicious Enclave



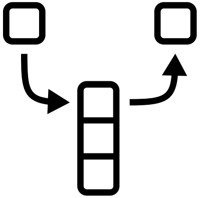
- From enclave: host application's memory is accessible, but only if mapped
- If enclave reads unmapped host memory $\Rightarrow$ terminated
- Transactional Synchronization Extensions (TSX): hardware support for transactional memory
- Enclave wraps memory access inside a TSX transaction
- If accessible: transaction completes successfully
- If inaccessible: TSX aborts the transaction and supresses enclave termination

SGX ROP [16]: A Malicious Enclave



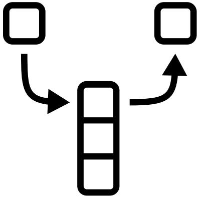
- From enclave: host application's memory is accessible, but only if mapped
- If enclave reads unmapped host memory ⇒ terminated
- Transactional Synchronization Extensions (TSX): hardware support for transactional memory
- Enclave wraps memory access inside a TSX transaction
- If accessible: transaction completes successfully
- If inaccessible: TSX aborts the transaction and supresses enclave termination
- Search for ROP Gadgets

SGX ROP [16]: A Malicious Enclave



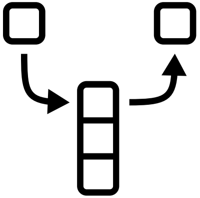
- From enclave: host application's memory is accessible, but only if mapped
- If enclave reads unmapped host memory ⇒ terminated
- Transactional Synchronization Extensions (TSX): hardware support for transactional memory
- Enclave wraps memory access inside a TSX transaction
- If accessible: transaction completes successfully
- If inaccessible: TSX aborts the transaction and supresses enclave termination
- Search for ROP Gadgets, manipulate the stack

SGX ROP [16]: A Malicious Enclave



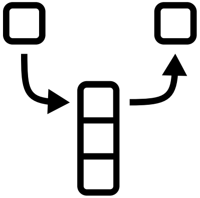
- From enclave: host application's memory is accessible, but only if mapped
- If enclave reads unmapped host memory ⇒ terminated
- Transactional Synchronization Extensions (TSX): hardware support for transactional memory
- Enclave wraps memory access inside a TSX transaction
- If accessible: transaction completes successfully
- If inaccessible: TSX aborts the transaction and supresses enclave termination
- Search for ROP Gadgets, manipulate the stack, execute the attack

SGX ROP [16]: A Malicious Enclave



- From enclave: host application's memory is accessible, but only if mapped
- If enclave reads unmapped host memory ⇒ terminated
- Transactional Synchronization Extensions (TSX): hardware support for transactional memory
- Enclave wraps memory access inside a TSX transaction
- If accessible: transaction completes successfully
- If inaccessible: TSX aborts the transaction and supresses enclave termination
- Search for ROP Gadgets, manipulate the stack, execute the attack, come back to the enclave!

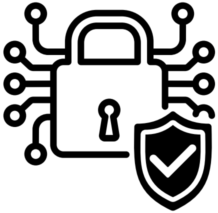
SGX ROP [16]: A Malicious Enclave



- From enclave: host application's memory is accessible, but only if mapped
- If enclave reads unmapped host memory ⇒ terminated
- Transactional Synchronization Extensions (TSX): hardware support for transactional memory
- Enclave wraps memory access inside a TSX transaction
- If accessible: transaction completes successfully
- If inaccessible: TSX aborts the transaction and supresses enclave termination
- Search for ROP Gadgets, manipulate the stack, execute the attack, come back to the enclave! Read the paper!

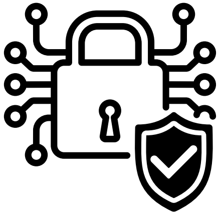
Confidential Computing (CoCo)

Confidential Computing (CoCo) Overview



- x86 instruction-set extension

Confidential Computing (CoCo) Overview

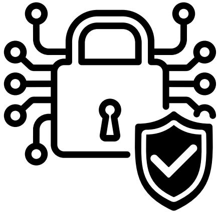


- x86 instruction-set extension
- Little reliance on the hypervisor: emulation, timekeeping, interrupts, faults, privileged operations

Confidential Computing (CoCo) Overview

- x86 instruction-set extension
- Little reliance on the hypervisor: emulation, timekeeping, interrupts, faults, privileged operations
- Confidential VMs (CVMs) and hypervisor are isolated

Confidential Computing (CoCo) Overview

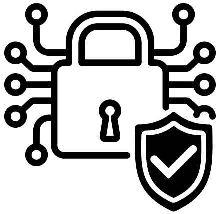


- x86 instruction-set extension
- Little reliance on the hypervisor: emulation, timekeeping, interrupts, faults, privileged operations
- Confidential VMs (CVMs) and hypervisor are isolated
- CVM memory is encrypted, possibly integrity protected

Confidential Computing (CoCo) Overview

- x86 instruction-set extension
- Little reliance on the hypervisor: emulation, timekeeping, interrupts, faults, privileged operations
- Confidential VMs (CVMs) and hypervisor are isolated
- CVM memory is encrypted, possibly integrity protected
- CVMs cannot access hypervisor memory (unlike SGX)

Confidential Computing (CoCo) Overview



- x86 instruction-set extension
- Little reliance on the hypervisor: emulation, timekeeping, interrupts, faults, privileged operations
- Confidential VMs (CVMs) and hypervisor are isolated
- CVM memory is encrypted, possibly integrity protected
- CVMs cannot access hypervisor memory (unlike SGX)
- Available in server CPUs (Intel Xeon, AMD EPYC)

Attack Targets

Once again, what are some components of a system?

Attack Targets

Once again, what are some components of a system?

Cache

Page Table

DRAM

Network

Predictors

Interrupt

CPU Ports

Power

Counters

Fault Attacks (Lecture 4)

Transient Execution (Lecture 3)

Threat Model

- Attacking the CVM: malicious hypervisor

Threat Model

- Attacking the CVM: malicious hypervisor
- Attacking the hypervisor: malicious CVM

Threat Model

- Attacking the CVM: malicious hypervisor
- Attacking the hypervisor: malicious CVM
- Malicious CVM attacks another CVM

Threat Model

- Attacking the CVM: malicious hypervisor
- Attacking the hypervisor: malicious CVM
- Malicious CVM attacks another CVM
- Side-Channel Attacks are out of scope

Threat Model

- Attacking the CVM: malicious hypervisor
- Attacking the hypervisor: malicious CVM
- Malicious CVM attacks another CVM
- Side-Channel Attacks are out of scope
- Only CPU is trusted

Side-Channel Attacks on CoCo

Register Inference Attacks [22]

- Recall: `VMEXIT` is an event where VM hands control back to the hypervisor

Register Inference Attacks [22]

- Recall: `VMEXIT` is an event where VM hands control back to the hypervisor
- AMD SEV left CVM's registers exposed after switching to hypervisor

Register Inference Attacks [22]

- Recall: `VMEXIT` is an event where VM hands control back to the hypervisor
- AMD SEV left CVM's registers exposed after switching to hypervisor
- Hypervisor can infer the CVM's computation just by inspecting the registers

Register Inference Attacks [22]

- Recall: `VMEXIT` is an event where VM hands control back to the hypervisor
- AMD SEV left CVM's registers exposed after switching to hypervisor
- Hypervisor can infer the CVM's computation just by inspecting the registers
- With AMD SEV-ES: registers are encrypted and integrity protected

Register Inference Attacks [22]

- Recall: `VMEXIT` is an event where VM hands control back to the hypervisor
- AMD SEV left CVM's registers exposed after switching to hypervisor
- Hypervisor can infer the CVM's computation just by inspecting the registers
- With AMD SEV-ES: registers are encrypted and integrity protected
- :)

Ciphertext Inference Attacks [12]

VM Save Area

Offset	**Size**	**Content**
0x150	16 bytes	CR3 & CR0
0x170	16 bytes	RFLAGS & RIP
0x1D8	8 bytes	RSP
0x1F8	8 bytes	RAX
0x240	8 bytes	CR2
0x308	8 bytes	RCX
0x310	16 bytes	RDX & RBX
...	...	...

- With AMD SEV-ES: registers are encrypted and integrity protected

Ciphertext Inference Attacks [12]

VM Save Area

Offset	**Size**	**Content**
0x150	16 bytes	CR3 & CR0
0x170	16 bytes	RFLAGS & RIP
0x1D8	8 bytes	RSP
0x1F8	8 bytes	RAX
0x240	8 bytes	CR2
0x308	8 bytes	RCX
0x310	16 bytes	RDX & RBX
...	...	...

- With AMD SEV-ES: registers are encrypted and integrity protected
- 16-byte blocks are encrypted independently using AES XEX (XOR-Encrypt-XOR)

Ciphertext Inference Attacks [12]

VM Save Area

Offset	Size	Content
0x150	16 bytes	CR3 & CR0
0x170	16 bytes	RFLAGS & RIP
0x1D8	8 bytes	RSP
0x1F8	8 bytes	RAX
0x240	8 bytes	CR2
0x308	8 bytes	RCX
0x310	16 bytes	RDX & RBX
...	...	...

- With AMD SEV-ES: registers are encrypted and integrity protected
- 16-byte blocks are encrypted independently using AES XEX (XOR-Encrypt-XOR)
- The same plaintext always has the same ciphertext

Ciphertext Inference Attacks [12]

VM Save Area

Offset	Size	Content
0x150	16 bytes	CR3 & CR0
0x170	16 bytes	RFLAGS & RIP
0x1D8	8 bytes	RSP
0x1F8	8 bytes	RAX
0x240	8 bytes	CR2
0x308	8 bytes	RCX
0x310	16 bytes	RDX & RBX
...	...	...

- With AMD SEV-ES: registers are encrypted and integrity protected
- 16-byte blocks are encrypted independently using AES XEX (XOR-Encrypt-XOR)
- The same plaintext always has the same ciphertext
- Change in the CVM's ciphertext: malicious hypervisor can infer the changes of the corresponding plaintext

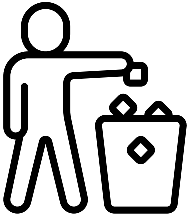
Ciphertext Inference Attacks [12]

VM Save Area

Offset	**Size**	**Content**
0x150	16 bytes	CR3 & CR0
0x170	16 bytes	RFLAGS & RIP
0x1D8	8 bytes	RSP
0x1F8	8 bytes	RAX
0x240	8 bytes	CR2
0x308	8 bytes	RCX
0x310	16 bytes	RDX & RBX
...	...	...

- With AMD SEV-ES: registers are encrypted and integrity protected
- 16-byte blocks are encrypted independently using AES XEX (XOR-Encrypt-XOR)
- The same plaintext always has the same ciphertext
- Change in the CVM's ciphertext: malicious hypervisor can infer the changes of the corresponding plaintext
- Build a dictionary of plaintext-ciphertext pairs for targeted registers

CacheWarp [25]



- `INVD`

CacheWarp [25]

- `INVD`: Invalidates all levels of cache
- `INVD`

CacheWarp [25]

- `INVD`: Invalidates all levels of cache
- `INVD`: No data is written back to main memory

CacheWarp [25]

- `INVD`: Invalidates all levels of cache
- `INVD`: No data is written back to main memory
- `WBINVD`

CacheWarp [25]

- `INVD`: Invalidates all levels of cache
- `INVD`: No data is written back to main memory
- `WBINVD`: data is written back to main memory and invalidates cache

CacheWarp [25]

- `INVD`: Invalidates all levels of cache
- `INVD`: No data is written back to main memory
- `WBINVD`: data is written back to main memory and invalidates cache
- Intel SGX & TDX: disable `INVD`

CacheWarp [25]

- `INVD`: Invalidates all levels of cache
- `INVD`: No data is written back to main memory
- `WBINVD`: data is written back to main memory and invalidates cache
- Intel SGX & TDX: disable `INVD`
- AMD SEV, SEV-ES, SEV-SNP: `INVD` works

CacheWarp [25]

- `INVD`: Invalidates all levels of cache
- `INVD`: No data is written back to main memory
- `WBINVD`: data is written back to main memory and invalidates cache
- Intel SGX & TDX: disable `INVD`
- AMD SEV, SEV-ES, SEV-SNP: `INVD` works
- How can a malicious hypervisor exploit this?

CacheWarp [25]

```c
int ret1() {
  return 1;
}
int ret0() {
  return 0;
}
int main() {
  while(1){
    if (ret1() == 0){
      printf("Win!");
    }
    ret0();
  }
}
```

CacheWarp [25]

```c
int ret1() {
  return 1;
}
int ret0() {
  return 0;
}
int main() {
  while(1){
    if (ret1() == 0){
      printf("Win!");
    }
    ret0();
  }
}
```

```
main:
  push %rbp
  mov %rsp,%rbp
  mov $0x0,%eax
  call <ret1>
  test %eax,%eax
  jne 118c <main+0x25>
  lea 0xe80(%rip),%rax
  mov %rax,%rdi
  call <printf@plt>
  mov $0x0,%eax
  call 1158 <ret0>
  jmp 116f <main+0x8>
```

CacheWarp [25]

```c
int ret1() {
  return 1;
}
int ret0() {
  return 0;
}
int main() {
  while(1){
    if (ret1() == 0){
      printf("Win!");
    }
    ret0();
  }
}
```

```
1  main:
2    push %rbp
3    mov %rsp,%rbp
4    mov $0x0,%eax
5    call <ret1>
6    test %eax,%eax
7    jne 118c <main+0x25>
8    lea 0xe80(%rip),%rax
9    mov %rax,%rdi
10   call <printf@plt>
11   mov $0x0,%eax
12   call 1158 <ret0>
13   jmp 116f <main+0x8>
```

Cache:

$\boxed{\text{main:6}}$

CacheWarp [25]

```c
1  int ret1() {
2    return 1;
3  }
4  int ret0() {
5    return 0;
6  }
7  int main() {
8    while(1){
9      if (ret1() == 0){
10       printf("Win!");
11     }
12     ret0();
13   }
14 }
```

```asm
1  main:
2    push %rbp
3    mov %rsp,%rbp
4    mov $0x0,%eax
5    call <ret1>
6    test %eax,%eax
7    jne 118c <main+0x25>
8    lea 0xe80(%rip),%rax
9    mov %rax,%rdi
10   call <printf@plt>
11   mov $0x0,%eax
12   call 1158 <ret0>
13   jmp 116f <main+0x8>
```

WBINVD

Cache:

main:6

CacheWarp [25]

```
int ret1() {
  return 1;
}
int ret0() {
  return 0;
}
int main() {
  while(1){
    if (ret1() == 0){
      printf("Win!");
    }
    ret0();
  }
}
```

```
main:
  push %rbp
  mov %rsp,%rbp
  mov $0x0,%eax
  call <ret1>
  test %eax,%eax
  jne 118c <main+0x25>
  lea 0xe80(%rip),%rax
  mov %rax,%rdi
  call <printf@plt>
  mov $0x0,%eax
  call 1158 <ret0>
  jmp 116f <main+0x8>
```

WBINVD

Cache:

main:6

Memory:

main:6

CacheWarp [25]

```c
int ret1() {
  return 1;
}
int ret0() {
  return 0;
}
int main() {
  while(1){
    if (ret1() == 0){
      printf("Win!");
    }
    ret0();
  }
}
```

```
main:
  push %rbp
  mov %rsp,%rbp
  mov $0x0,%eax
  call <ret1>
  test %eax,%eax
  jne 118c <main+0x25>
  lea 0xe80(%rip),%rax
  mov %rax,%rdi
  call <printf@plt>
  mov $0x0,%eax
  call 1158 <ret0>
  jmp 116f <main+0x8>
```

Cache:

main:6

Memory:

main:6

CacheWarp [25]

```c
int ret1() {
  return 1;
}
int ret0() {
  return 0;
}
int main() {
  while(1){
    if (ret1() == 0){
      printf("Win!");
    }
    ret0();
  }
}
```

```
main:
  push %rbp
  mov %rsp,%rbp
  mov $0x0,%eax
  call <ret1>
  test %eax,%eax
  jne 118c <main+0x25>
  lea 0xe80(%rip),%rax
  mov %rax,%rdi
  call <printf@plt>
  mov $0x0,%eax
  call 1158 <ret0>
  jmp 116f <main+0x8>
```

Cache:

main:6

Memory:

main:6

Registers:

EAX: 1

CacheWarp [25]

```c
int ret1() {
  return 1;
}
int ret0() {
  return 0;
}
int main() {
  while(1){
    if (ret1() == 0){
      printf("Win!");
    }
    ret0();
  }
}
```

```
 1 main:
 2   push %rbp
 3   mov %rsp,%rbp
 4   mov $0x0,%eax
 5   call <ret1>
 6   test %eax,%eax
 7   jne 118c <main+0x25>
 8   lea 0xe80(%rip),%rax
 9   mov %rax,%rdi
10   call <printf@plt>
11   mov $0x0,%eax
12   call 1158 <ret0>
13   jmp 116f <main+0x8>
```

Cache:

Memory:

main:6

Registers:

EAX: 1

CacheWarp [25]

```
1  int ret1() {
2    return 1;
3  }
4  int ret0() {
5    return 0;
6  }
7  int main() {
8    while(1){
9      if (ret1() == 0){
10       printf("Win!");
11     }
12     ret0();
13   }
14 }
```

```
1  main:
2    push %rbp
3    mov %rsp,%rbp
4    mov $0x0,%eax
5    call <ret1>
6    test %eax,%eax
7    jne 118c <main+0x25>
8    lea 0xe80(%rip),%rax
9    mov %rax,%rdi
10   call <printf@plt>
11   mov $0x0,%eax
12   call 1158 <ret0>
13   jmp 116f <main+0x8>
```

Cache:

Memory:

main:6

Registers:

EAX: 1

CacheWarp [25]

```c
int ret1() {
  return 1;
}
int ret0() {
  return 0;
}
int main() {
  while(1){
    if (ret1() == 0){
      printf("Win!");
    }
    ret0();
  }
}
```

```
main:
  push %rbp
  mov %rsp,%rbp
  mov $0x0,%eax
  call <ret1>
  test %eax,%eax
  jne 118c <main+0x25>
  lea 0xe80(%rip),%rax
  mov %rax,%rdi
  call <printf@plt>
  mov $0x0,%eax
  call 1158 <ret0>
  jmp 116f <main+0x8>
```

Cache:

main:13

Memory:

main:6

Registers:

EAX: 1

CacheWarp [25]

```c
int ret1() {
  return 1;
}
int ret0() {
  return 0;
}
int main() {
  while(1){
    if (ret1() == 0){
      printf("Win!");
    }
    ret0();
  }
}
```

```
main:
  push %rbp
  mov %rsp,%rbp
  mov $0x0,%eax
  call <ret1>
  test %eax,%eax
  jne 118c <main+0x25>
  lea 0xe80(%rip),%rax
  mov %rax,%rdi
  call <printf@plt>
  mov $0x0,%eax
  call 1158 <ret0>
  jmp 116f <main+0x8>
```

Cache:

main:13

Memory:

main:6

Registers:

EAX: 1

CacheWarp [25]

```c
int ret1() {
  return 1;
}
int ret0() {
  return 0;
}
int main() {
  while(1){
    if (ret1() == 0){
      printf("Win!");
    }
    ret0();
  }
}
```

```
main:
  push %rbp
  mov %rsp,%rbp
  mov $0x0,%eax
  call <ret1>
  test %eax,%eax
  jne 118c <main+0x25>
  lea 0xe80(%rip),%rax
  mov %rax,%rdi
  call <printf@plt>
  mov $0x0,%eax
  call 1158 <ret0>
  jmp 116f <main+0x8>
```

INVD

Cache:

main:13

Memory:

main:6

Registers:

EAX: 1

CacheWarp [25]

```c
int ret1() {
  return 1;
}
int ret0() {
  return 0;
}
int main() {
  while(1){
    if (ret1() == 0){
      printf("Win!");
    }
    ret0();
  }
}
```

```
main:
  push %rbp
  mov %rsp,%rbp
  mov $0x0,%eax
  call <ret1>
  test %eax,%eax
  jne 118c <main+0x25>
  lea 0xe80(%rip),%rax
  mov %rax,%rdi
  call <printf@plt>
  mov $0x0,%eax
  call 1158 <ret0>
  jmp 116f <main+0x8>
```

INVD

Cache:

Memory:

main:6

Registers:

EAX: 1

CacheWarp [25]

```c
int ret1() {
  return 1;
}
int ret0() {
  return 0;
}
int main() {
  while(1){
    if (ret1() == 0){
      printf("Win!");
    }
    ret0();
  }
}
```

```
main:
  push %rbp
  mov %rsp,%rbp
  mov $0x0,%eax
  call <ret1>
  test %eax,%eax
  jne 118c <main+0x25>
  lea 0xe80(%rip),%rax
  mov %rax,%rdi
  call <printf@plt>
  mov $0x0,%eax
  call 1158 <ret0>
  jmp 116f <main+0x8>
```

Cache:

Memory:

main:6

Registers:

EAX: 0

CacheWarp [25]

```c
int ret1() {
  return 1;
}
int ret0() {
  return 0;
}
int main() {
  while(1){
    if (ret1() == 0){
      printf("Win!");
    }
    ret0();
  }
}
```

```asm
main:
  push %rbp
  mov %rsp,%rbp
  mov $0x0,%eax
  call <ret1>
  test %eax,%eax
  jne 118c <main+0x25>
  lea 0xe80(%rip),%rax
  mov %rax,%rdi
  call <printf@plt>
  mov $0x0,%eax
  call 1158 <ret0>
  jmp 116f <main+0x8>
```

Cache:

main:6

Memory:

main:6

Registers:

EAX: 0

CacheWarp [25]

```
 1 int ret1() {
 2   return 1;
 3 }
 4 int ret0() {
 5   return 0;
 6 }
 7 int main() {
 8   while(1){
 9     if (ret1() == 0){
10       printf("Win!");
11     }
12     ret0();
13   }
14 }
```

```
 1 main:
 2   push %rbp
 3   mov %rsp,%rbp
 4   mov $0x0,%eax
 5   call <ret1>
 6   test %eax,%eax
 7   jne 118c <main+0x25>
 8   lea 0xe80(%rip),%rax
 9   mov %rax,%rdi
10   call <printf@plt>
11   mov $0x0,%eax
12   call 1158 <ret0>
13   jmp 116f <main+0x8>
```

Cache:

main:6

Memory:

main:6

Registers:

EAX: 0

CacheWarp [25]

```c
int ret1() {
  return 1;
}
int ret0() {
  return 0;
}
int main() {
  while(1){
    if (ret1() == 0){
      printf("Win!");
    }
    ret0();
  }
}
```

```
main:
  push %rbp
  mov %rsp,%rbp
  mov $0x0,%eax
  call <ret1>
  test %eax,%eax
  jne 118c <main+0x25>
  lea 0xe80(%rip),%rax
  mov %rax,%rdi
  call <printf@plt>
  mov $0x0,%eax
  call 1158 <ret0>
  jmp 116f <main+0x8>
```

Cache:

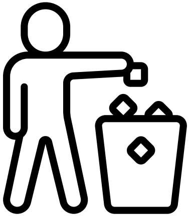
Memory:

$\boxed{\text{main:6}}$

Registers:

EAX: 0

CacheWarp [25]



- Bypass OpenSSH authentication: `sys_auth_passwd`

CacheWarp [25]

- Bypass OpenSSH authentication: `sys_auth_passwd`
- Break RSA-CRT: Drop write using `INVD`, generate faulty signature

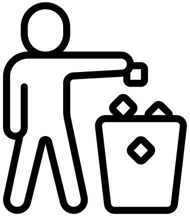
CacheWarp [25]

- Bypass OpenSSH authentication: `sys_auth_passwd`
- Break RSA-CRT: Drop write using `INVD`, generate faulty signature
- Bypass sudo authentication:

CacheWarp [25]

- Bypass OpenSSH authentication: `sys_auth_passwd`
- Break RSA-CRT: Drop write using `INVD`, generate faulty signature
- Bypass sudo authentication:
    - Normal user: UID >0

CacheWarp [25]

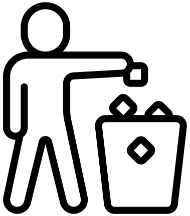


- Bypass OpenSSH authentication: `sys_auth_passwd`
- Break RSA-CRT: Drop write using `INVD`, generate faulty signature
- Bypass sudo authentication:
  - Normal user: UID $>0 \Rightarrow$ sudo fails

CacheWarp [25]

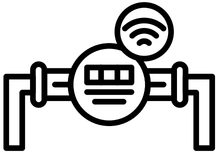
- Bypass OpenSSH authentication: `sys_auth_passwd`
- Break RSA-CRT: Drop write using `INVD`, generate faulty signature
- Bypass sudo authentication:
  - Normal user: UID $>0 \Rightarrow$ sudo fails
  - Drop write when sudo checks UID (GUID, RUID, EUID)

CacheWarp [25]



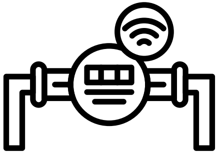
- Bypass OpenSSH authentication: `sys_auth_passwd`
- Break RSA-CRT: Drop write using `INVD`, generate faulty signature
- Bypass sudo authentication:
  - Normal user: UID $>0 \Rightarrow$ sudo fails
  - Drop write when sudo checks UID (GUID, RUID, EUID)
  - UID 0: root

CounterSEVeillance [6]



- CPU provides hardware performance counters:

CounterSEVeillance [6]



- CPU provides hardware performance counters:
  - Retired Instructions

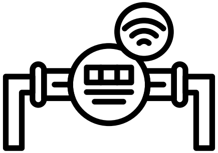
CounterSEVeillance [6]

- CPU provides hardware performance counters:
    - Retired Instructions
    - Retired Branch Instructions

CounterSEVeillance [6]

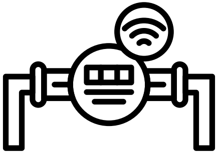
- CPU provides hardware performance counters:
    - Retired Instructions
    - Retired Branch Instructions
    - Retired Taken Branch Instructions

CounterSEVeillance [6]



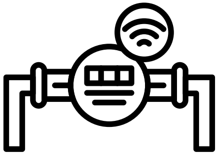
- CPU provides hardware performance counters:
    - Retired Instructions
    - Retired Branch Instructions
    - Retired Taken Branch Instructions
- AMD: Report accurate values when SEV, SEV-ES, SEV-SNP CVMs run

CounterSEVeillance [6]



- CPU provides hardware performance counters:
  - Retired Instructions
  - Retired Branch Instructions
  - Retired Taken Branch Instructions
- AMD: Report accurate values when SEV, SEV-ES, SEV-SNP CVMs run
- Intel: Disabled hardware performance counters when SGX enclaves, TDX CVMs run

CounterSEVeillance [6]



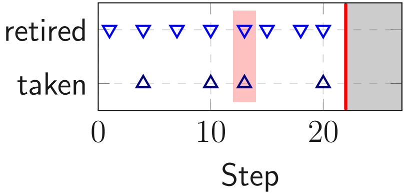
- CPU provides hardware performance counters:
    - Retired Instructions
    - Retired Branch Instructions
    - Retired Taken Branch Instructions
- AMD: Report accurate values when SEV, SEV-ES, SEV-SNP CVMs run
- Intel: Disabled hardware performance counters when SGX enclaves, TDX CVMs run
- Leak whether branches (`if`) were taken

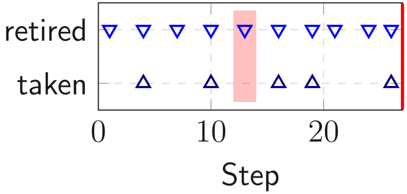
CounterSEVeillance [6]

- CPU provides hardware performance counters:
  - Retired Instructions
  - Retired Branch Instructions
  - Retired Taken Branch Instructions
- AMD: Report accurate values when SEV, SEV-ES, SEV-SNP CVMs run
- Intel: Disabled hardware performance counters when SGX enclaves, TDX CVMs run
- Leak whether branches (`if`) were taken

CounterSEVeillance [6]

```
char time_str[data->digits+1];
memset(time_str, 0, data->digits+1);
for (size_t i=0; i<data->digits; i++) {
  if (key[i] != time_str[i])
    return OTP_ERROR;
}
return OTP_OK;
```





Limitations & Solutions

Some limitations

- No shared memory

Some limitations

- No shared memory
- No physical addresses

Some limitations

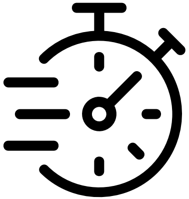
- No shared memory
- No physical addresses
- No access to high-precision timer: `rdtsc`[a]

Some limitations

- No shared memory
- No physical addresses
- No access to high-precision timer: `rdtsc`[a]
- No syscalls (SGX)

[a] AMD SEV-SNP and Intel TDX now have secure timers

Timer



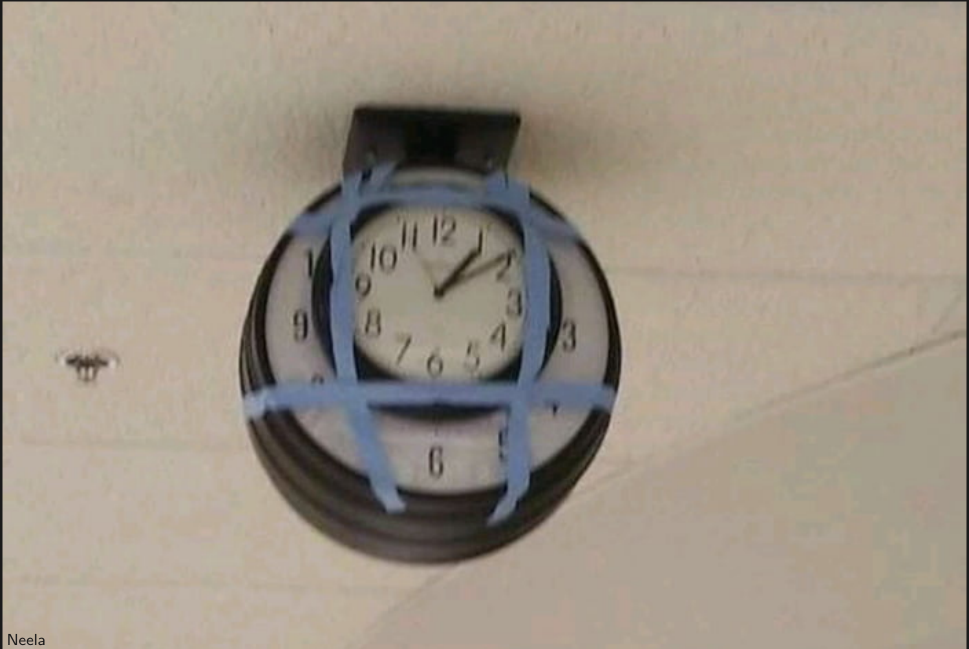
- We can build our own timer [13, 15]

Timer

- We can build our own timer [13, 15]
- Start a thread that continuously increments a global variable

Timer

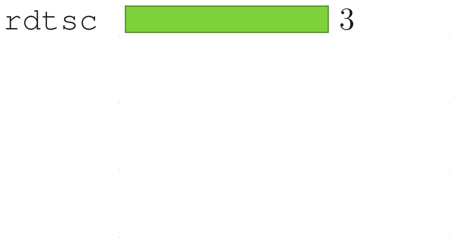
- We can build our own timer [13, 15]
- Start a thread that continuously increments a global variable
- The global variable is our timestamp


Neela

ARE YOU REALLY EXPECTING TO
OUTPERFORM THE HARDWARE COUNTER?
imgflip.com

Self-built Timer

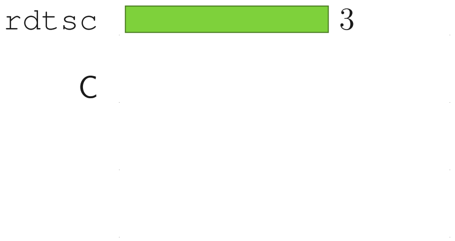
CPU cycles one increment takes



```
timestamp = rdtsc();
```

Self-built Timer

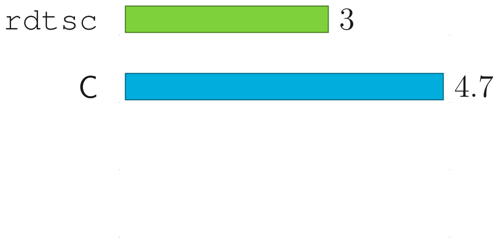
CPU cycles one increment takes



```
while(1) {
  timestamp++;
}
```

Self-built Timer

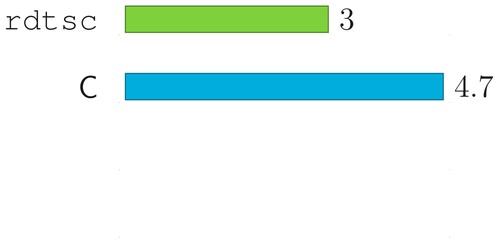
CPU cycles one increment takes



```
while(1) {
  timestamp++;
}
```

Self-built Timer

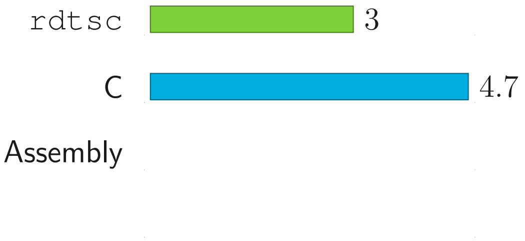
CPU cycles one increment takes



```
while(1) {
  timestamp++;
}
```

Self-built Timer

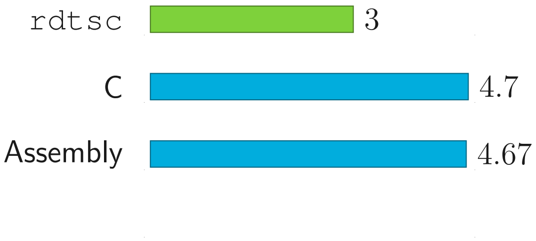
CPU cycles one increment takes



```
mov &timestamp, %rcx
  1: incl (%rcx)
  jmp 1b
```

Self-built Timer

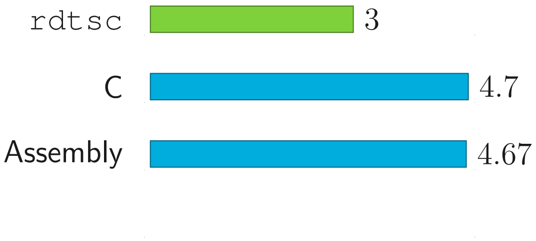
CPU cycles one increment takes



```
mov &timestamp, %rcx
  1: incl (%rcx)
  jmp 1b
```

Self-built Timer

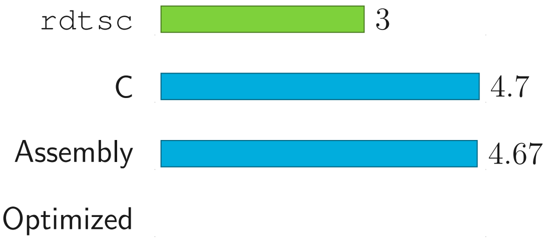
CPU cycles one increment takes



```
mov &timestamp, %rcx
  1: incl (%rcx)
  jmp 1b
```

Self-built Timer

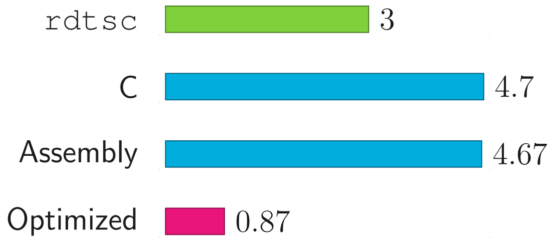
CPU cycles one increment takes



```
mov &timestamp, %rcx
1: inc %rax
mov %rax, (%rcx)
jmp 1b
```

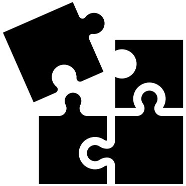
Self-built Timer

CPU cycles one increment takes



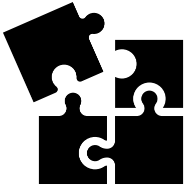
```
mov &timestamp, %rcx
1: inc %rax
mov %rax, (%rcx)
jmp 1b
```

Combining Everything: Malware Guard Extensions (on SGX) [15]



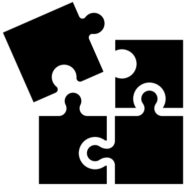
1. Use the counting primitive to measure DRAM accesses

Combining Everything: Malware Guard Extensions (on SGX) [15]



1. Use the counting primitive to measure DRAM accesses
2. Use DRAM side-channel to build eviction set

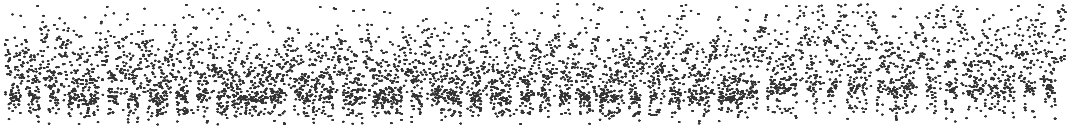
Combining Everything: Malware Guard Extensions (on SGX) [15]



1. Use the counting primitive to measure DRAM accesses
2. Use DRAM side-channel to build eviction set
3. Mount Prime+Probe on the buffer containing the multiplier

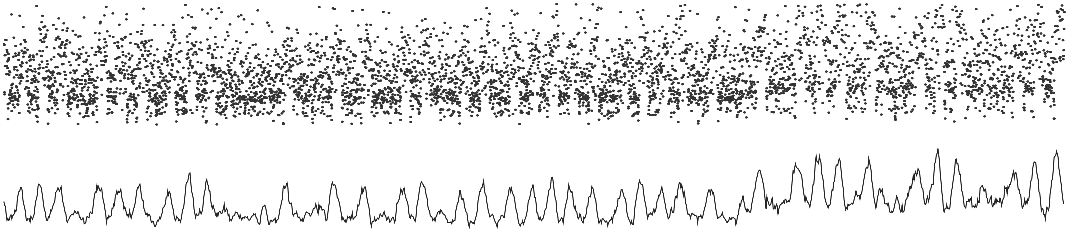
Measured Trace

Raw Prime+Probe trace...



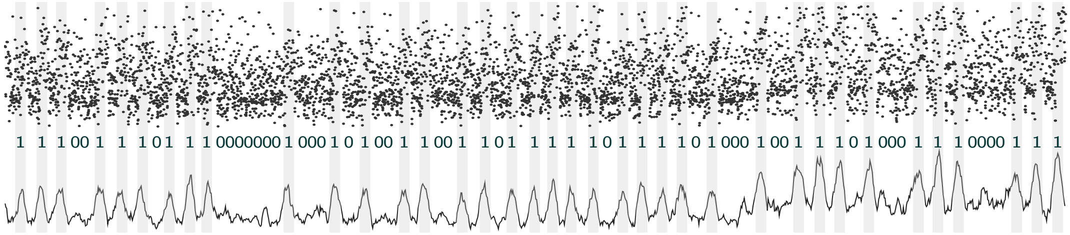
Measured Trace

...processed with a simple moving average...

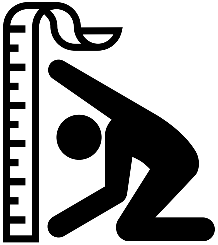


Measured Trace

...allows to clearly see the bits of the exponent

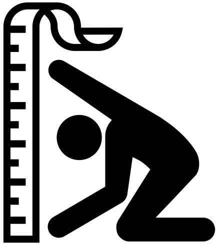


Single-Stepping [18, 23]



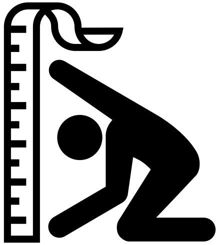
- CVM / Enclave: executes many instructions until support from the hypervisor / host is required

Single-Stepping [18, 23]



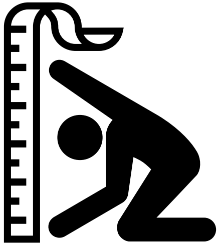
- CVM / Enclave: executes many instructions until support from the hypervisor / host is required
- Single Step: CVM / Enclave executes only one instruction at a time

Single-Stepping [18, 23]



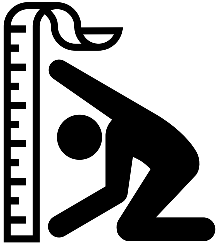
- CVM / Enclave: executes many instructions until support from the hypervisor / host is required
- Single Step: CVM / Enclave executes only one instruction at a time
- local Advanced Programmable Interrupt Controller (APIC)

Single-Stepping [18, 23]



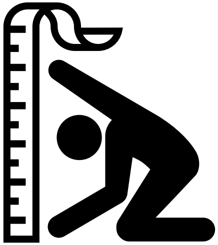
- CVM / Enclave: executes many instructions until support from the hypervisor / host is required
- Single Step: CVM / Enclave executes only one instruction at a time
- local Advanced Programmable Interrupt Controller (APIC)
- Timer: 3 modes

Single-Stepping [18, 23]



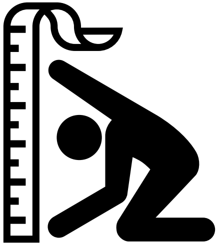
- CVM / Enclave: executes many instructions until support from the hypervisor / host is required
- Single Step: CVM / Enclave executes only one instruction at a time
- local Advanced Programmable Interrupt Controller (APIC)
- Timer: 3 modes
  - One-shot

Single-Stepping [18, 23]



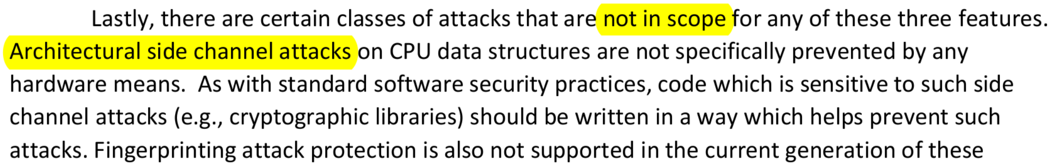
- CVM / Enclave: executes many instructions until support from the hypervisor / host is required
- Single Step: CVM / Enclave executes only one instruction at a time
- local Advanced Programmable Interrupt Controller (APIC)
- Timer: 3 modes
    - One-shot
    - Periodic

Single-Stepping [18, 23]



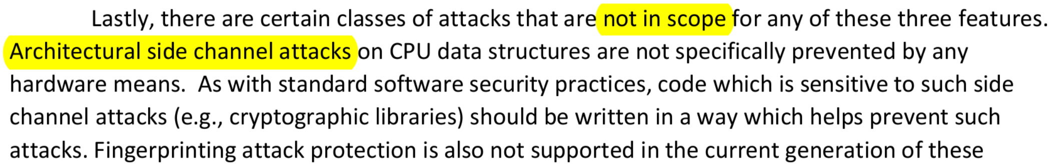
- CVM / Enclave: executes many instructions until support from the hypervisor / host is required
- Single Step: CVM / Enclave executes only one instruction at a time
- local Advanced Programmable Interrupt Controller (APIC)
- Timer: 3 modes
  - One-shot
  - Periodic
  - TSC-deadline

Conclusion


Lastly, there are certain classes of attacks that are not in scope for any of these three features. Architectural side channel attacks on CPU data structures are not specifically prevented by any hardware means. As with standard software security practices, code which is sensitive to such side channel attacks (e.g., cryptographic libraries) should be written in a way which helps prevent such attacks. Fingerprinting attack protection is also not supported in the current generation of these

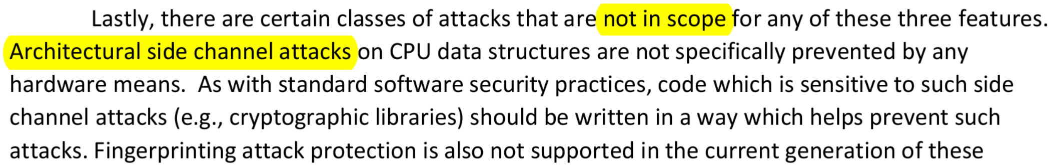
- TEEs / CVMs developed to protect sensitive information/critical code execution

Conclusion


Lastly, there are certain classes of attacks that are not in scope for any of these three features. Architectural side channel attacks on CPU data structures are not specifically prevented by any hardware means. As with standard software security practices, code which is sensitive to such side channel attacks (e.g., cryptographic libraries) should be written in a way which helps prevent such attacks. Fingerprinting attack protection is also not supported in the current generation of these

- TEEs / CVMs developed to protect sensitive information/critical code execution
- Allow for a very powerful threat model: Malicious hypervisor

Conclusion


Lastly, there are certain classes of attacks that are not in scope for any of these three features. Architectural side channel attacks on CPU data structures are not specifically prevented by any hardware means. As with standard software security practices, code which is sensitive to such side channel attacks (e.g., cryptographic libraries) should be written in a way which helps prevent such attacks. Fingerprinting attack protection is also not supported in the current generation of these

- TEEs / CVMs developed to protect sensitive information/critical code execution
- Allow for a very powerful threat model: Malicious hypervisor
- SCAs often not "out of scope"

Side-Channel Security

Chapter 3: Trusted Execution Environments and Confidential Computing

Sudheendra Neela

March 13, 2025

Graz University of Technology

References

[1] AMD (2020). AMD SEV-SNP: Strengthening VM Isolation with Integrity Protection and More.

[2] ARM (2009). Building a Secure System using TrustZone Technology.

[3] ARM (2021). Arm CCA Security Model 1.0.

[4] Brasser, F., Müller, U., Dmitrienko, A., Kostiainen, K., Capkun, S., and Sadeghi, A.-R. (2017). Software Grand Exposure: SGX Cache Attacks Are Practical. In *WOOT*.

[5] Costan, V. and Devadas, S. (2016). Intel SGX Explained.

[6] Gast, S., Weissteiner, H., Schröder, R. L., and Gruss, D. (2025). CounterSEVeillance: Performance-Counter Attacks on AMD SEV-SNP. In *NDSS*.

[7] Gruss, D., Lipp, M., Schwarz, M., Genkin, D., Juffinger, J., O'Connell, S., Schoechl, W., and Yarom, Y. (2018). Another Flip in the Wall of Rowhammer Defenses. In *S&P*.

[8] Intel (2024). Intel Trust Domain Extensions Module Base Architecture Specification.

[9] Jang, Y., Lee, J., Lee, S., and Kim, T. (2017). SGX-Bomb: Locking Down the Processor via Rowhammer Attack. In *SysTEX*.

[10] Kaplan, D. (2017). Protecting VM register state with SEV-ES.

[11] Kaplan, D., Powell, J., and Woller, T. (2016). AMD Memory Encryption.

[12] Li, M., Zhang, Y., Wang, H., Li, K., and Cheng, Y. (2021). CIPHERLEAKS: Breaking Constant-time Cryptography on AMD SEV via the Ciphertext Side Channel. In *USENIX Security*.

[13] Lipp, M., Gruss, D., Spreitzer, R., Maurice, C., and Mangard, S. (2016). ARMageddon: Cache Attacks on Mobile Devices. In *USENIX Security Symposium*.

[14] Moghimi, A., Irazoqui, G., and Eisenbarth, T. (2017). CacheZoom: How SGX amplifies the power of cache attacks. In *CHES*.

[15] Schwarz, M., Gruss, D., Weiser, S., Maurice, C., and Mangard, S. (2017). Malware Guard Extension: Using SGX to Conceal Cache Attacks. In *DIMVA*.

[16] Schwarz, M., Weiser, S., and Gruss, D. (2019). Practical Enclave Malware with Intel SGX. In *DIMVA*.

[17] Szefer, J., Keller, E., Lee, R. B., and Rexford, J. (2011). Eliminating the hypervisor attack surface for a more secure cloud. In *CCS*.

[18] Van Bulck, J., Piessens, F., and Strackx, R. (2017a). SGX-Step: A Practical Attack Framework for Precise Enclave Execution Control. In *SysTEX*.

[19] Van Bulck, J., Weichbrodt, N., Kapitza, R., Piessens, F., and Strackx, R. (2017b). Telling Your Secrets Without Page Faults: Stealthy Page Table-Based Attacks on Enclaved Execution. In *USENIX Security*.

[20] Van Schaik, S., Seto, A., Yurek, T., Batori, A., AlBassam, B., Genkin, D., Miller, A., Ronen, E., Yarom, Y., and Garman, C. (2024). SoK: SGX.Fail: How Stuff Gets eXposed. In *S&P*.

[21] Wang, W., Chen, G., Pan, X., Zhang, Y., Wang, X., Bindschaedler, V., Tang, H., and Gunter, C. A. (2017). Leaky Cauldron on the Dark Land: Understanding Memory Side-Channel Hazards in SGX. In *CCS*.

[22] Werner, J., Mason, J., Antonakakis, M., Polychronakis, M., and Monrose, F. (2019). The SEVerESt Of Them All: Inference Attacks Against Secure Virtual Enclaves. In *AsiaCCS*.

[23] Wilke, L., Wichelmann, J., Rabich, A., and Eisenbarth, T. (2023). SEV-Step A Single-Stepping Framework for AMD-SEV. *CHES*.

[24] Xu, Y., Cui, W., and Peinado, M. (2015). Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems. In *S&P*.

[25] Zhang, R., Center, C. H., Gerlach, L., Weber, D., Hetterich, L., Lü, Y., Kogler, A., and Schwarz, M. (2024). CacheWarp: Software-based Fault Injection using Selective State Reset. In *USENIX Security*.