Side-Channel Security

Chapter 3: Trusted Execution Environments

Daniel Gruss

March 14, 2024

Graz University of Technology

Trusted Execution Environments (TEEs)

Motivation

- Systems run software from various sources

Motivation

- Systems run software from various sources
- Providers: tamper-resistant way

Motivation

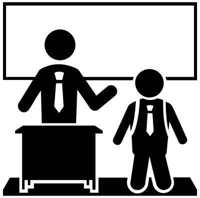
- Systems run software from various sources
- Providers: tamper-resistant way
- Protect computation against compromised OS

Motivation

- Systems run software from various sources
- Providers: tamper-resistant way
- Protect computation against compromised OS
- Key enabler of trusted cloud computing

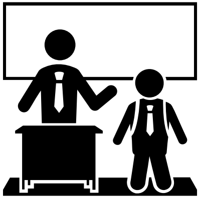
Intel Software Guard Extension (SGX)

SGX Introduction



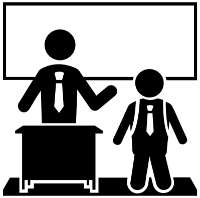
- x86 instruction-set extension

SGX Introduction



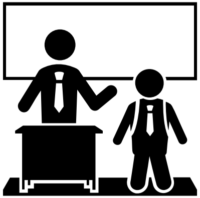
- x86 instruction-set extension
- Isolate trusted code from untrusted applications

SGX Introduction



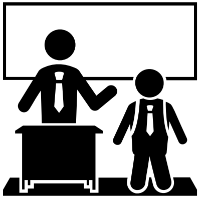
- x86 instruction-set extension
- Isolate trusted code from untrusted applications
- Neither app nor OS can access enclave memory

SGX Introduction



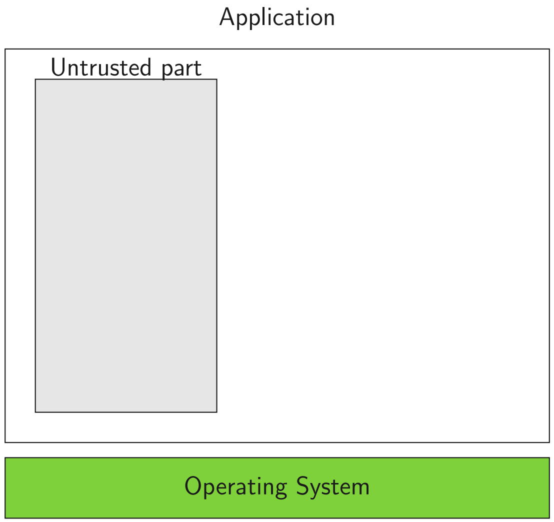
- x86 instruction-set extension
- Isolate trusted code from untrusted applications
- Neither app nor OS can access enclave memory
- Enclave memory is encrypted and integrity protected

SGX Introduction

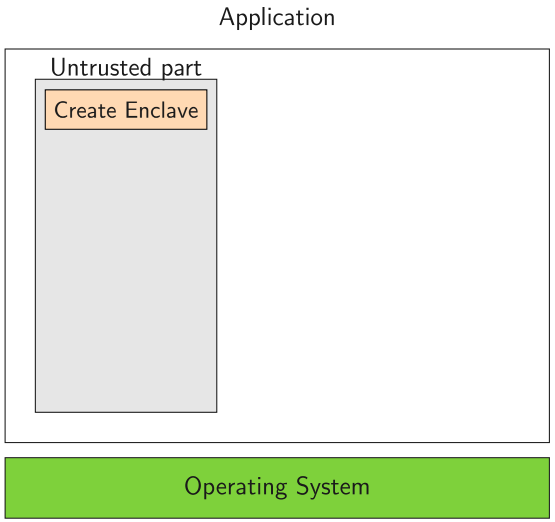


- x86 instruction-set extension
- Isolate trusted code from untrusted applications
- Neither app nor OS can access enclave memory
- Enclave memory is encrypted and integrity protected
- Enclave full access to virtual memory of host application

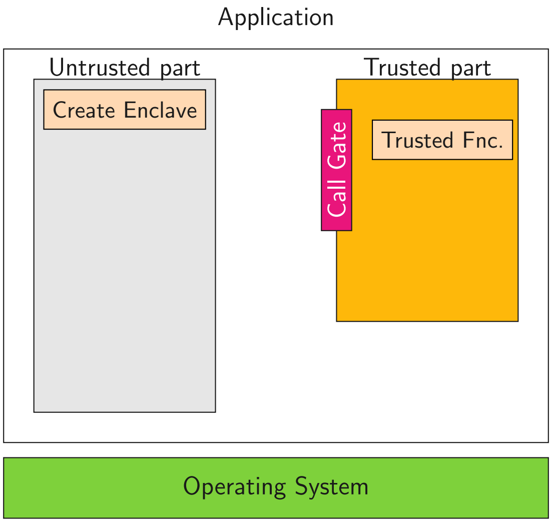
SGX Model



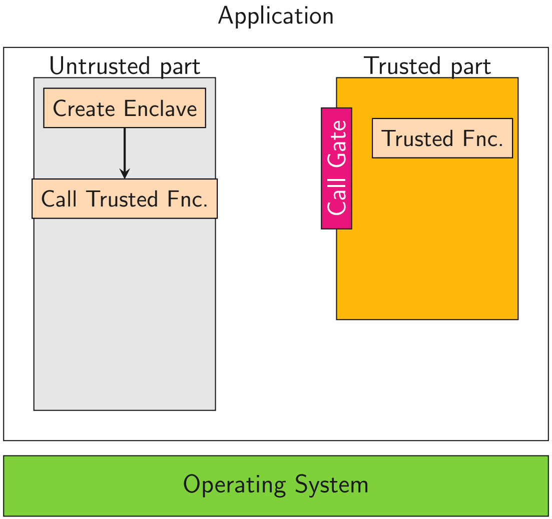
SGX Model


Application
Untrusted part
Create Enclave
Operating System

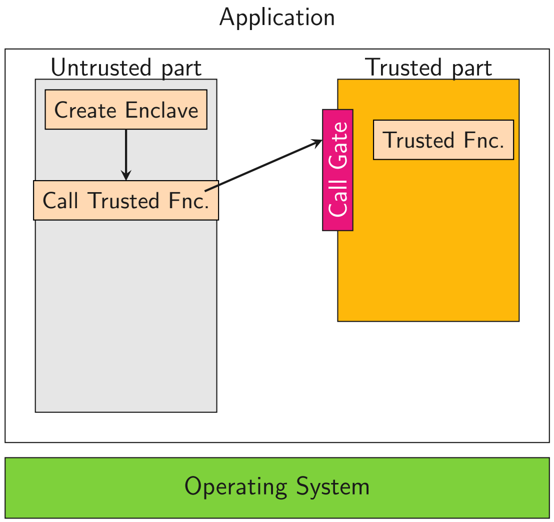
SGX Model


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

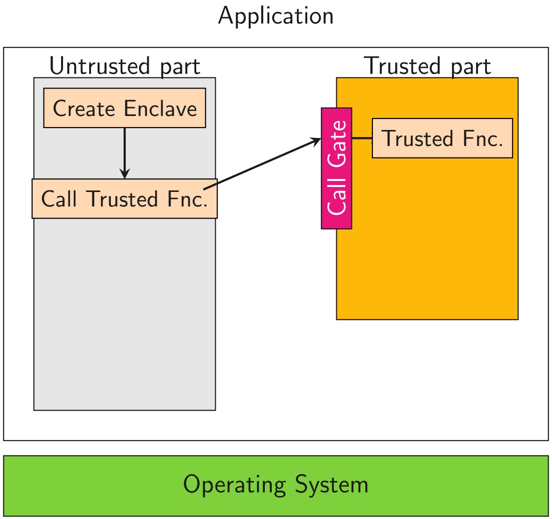
SGX Model


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

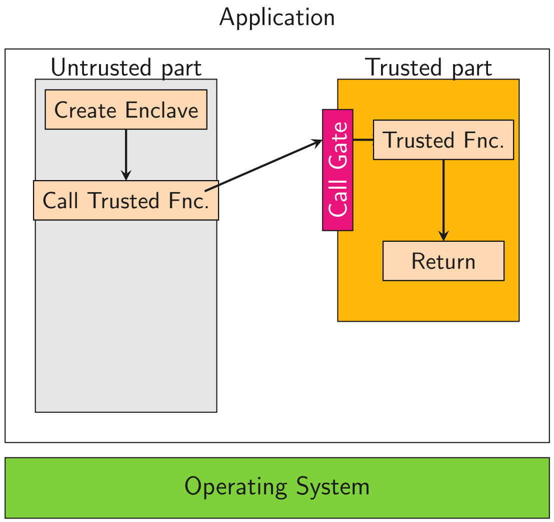
SGX Model


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

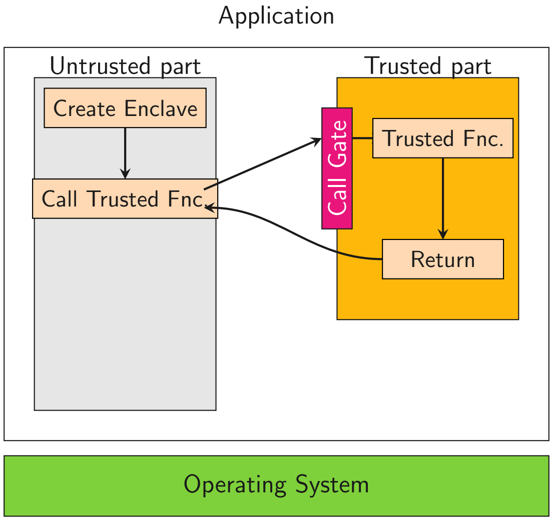
SGX Model


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

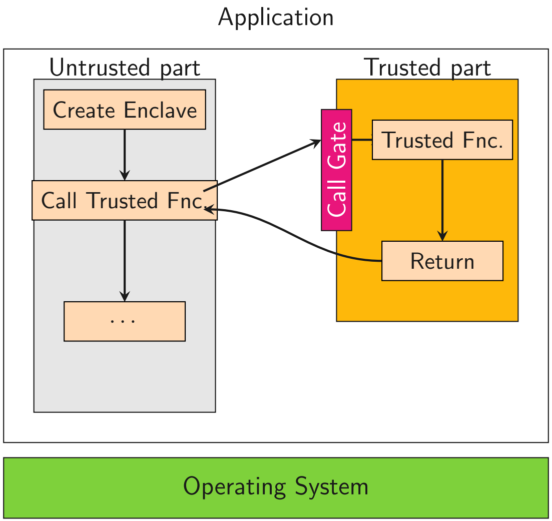
SGX Model


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

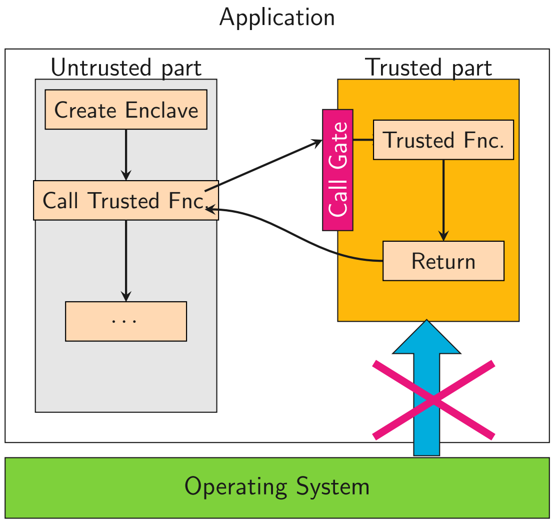
SGX Model


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

SGX Model

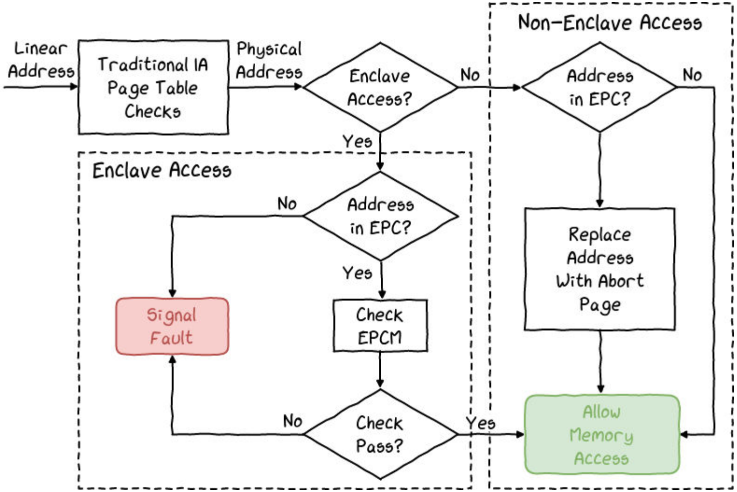

Application
Untrusted part
Trusted part
Create Enclave
Call Trusted Fnc.
. . .
Call Gate
Trusted Fnc.
Return
Operating System

SGX Model



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

Address Translation



Restrictions

- Cannot use I/O, including syscalls

Restrictions

- Cannot use I/O, including syscalls
- Certain instructions are forbidden (e.g., *rdtsc*)

Interrupt and Resume Enclave

1. Interrupt arrives

Interrupt and Resume Enclave

1. Interrupt arrives
2. Perform Asynchronous Enclave Exit (AEX)

Interrupt and Resume Enclave

1. Interrupt arrives
2. Perform Asynchronous Enclave Exit (AEX)
3. OS interrupt handler

Interrupt and Resume Enclave

1. Interrupt arrives
2. Perform Asynchronous Enclave Exit (AEX)
3. OS interrupt handler
4. Return to Asynchronous Exit Pointer (AEP) trampoline

Interrupt and Resume Enclave

1. Interrupt arrives
2. Perform Asynchronous Enclave Exit (AEX)
3. OS interrupt handler
4. Return to Asynchronous Exit Pointer (AEP) trampoline
5. *ERESUME*

Threat Model

- Enclaves isolated from all software

Threat Model

- Enclaves isolated from all software
- Allows for malicious OS/hypervisor

Threat Model

- Enclaves isolated from all software
- Allows for malicious OS/hypervisor
- Allows physical attacker

Threat Model

- Enclaves isolated from all software
- Allows for malicious OS/hypervisor
- Allows physical attacker
- Allows root attacker

Threat Model

- Enclaves isolated from all software
- Allows for malicious OS/hypervisor
- Allows physical attacker
- Allows root attacker
- Side-Channel Attacks are out of scope

Threat Model

- Enclaves isolated from all software
- Allows for malicious OS/hypervisor
- Allows physical attacker
- Allows root attacker
- Side-Channel Attacks are out of scope
- Only CPU is trusted

Attack Targets

Attack Targets

Side-Channel Attacks:

Page Table

Attack Targets

Side-Channel Attacks:

Page Table

DRAM

Attack Targets

Side-Channel Attacks:

Page Table

DRAM

Cache

Attack Targets

Side-Channel Attacks:

Page Table

DRAM

Cache

Predictors

Attack Targets

Side-Channel Attacks:

Page Table

DRAM

Cache

Predictors

Interrupt

Attack Targets

Side-Channel Attacks:

Page Table

DRAM

Cache

Predictors

Interrupt

Transient-Execution Attacks (Lecture 3):

Attack Targets

Side-Channel Attacks:

Page Table

DRAM

Cache

Predictors

Interrupt

Transient-Execution Attacks (Lecture 3):

Meltdown

Attack Targets

Side-Channel Attacks:

Page Table

DRAM

Cache

Predictors

Interrupt

Transient-Execution Attacks (Lecture 3):

Meltdown

Spectre

Attack Targets

Side-Channel Attacks:

Page Table

DRAM

Cache

Predictors

Interrupt

Transient-Execution Attacks (Lecture 3):

Meltdown

Spectre

ZombieLoad

Attack Targets

Side-Channel Attacks:

Page Table

DRAM

Cache

Predictors

Interrupt

Transient-Execution Attacks (Lecture 3):

Meltdown

Spectre

ZombieLoad

Fault Attacks (Lecture 5):

Attack Targets

Side-Channel Attacks:

Page Table

DRAM

Cache

Predictors

Interrupt

Transient-Execution Attacks (Lecture 3):

Meltdown

Spectre

ZombieLoad

Fault Attacks (Lecture 5):

Attack Targets

Side-Channel Attacks:

Page Table

DRAM

Cache

Predictors

Interrupt

Transient-Execution Attacks (Lecture 3):

Meltdown

Spectre

ZombieLoad

Fault Attacks (Lecture 5):

Controlled-Channel Attacks [8]

- Target mechanism which translates virtual to physical addresses

Controlled-Channel Attacks [8]

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

Controlled-Channel Attacks [8]

- Target mechanism which translates virtual to physical addresses
- Enclave memory is set up by OS
- Hardware only ensures one page is mapped by 1 enclave

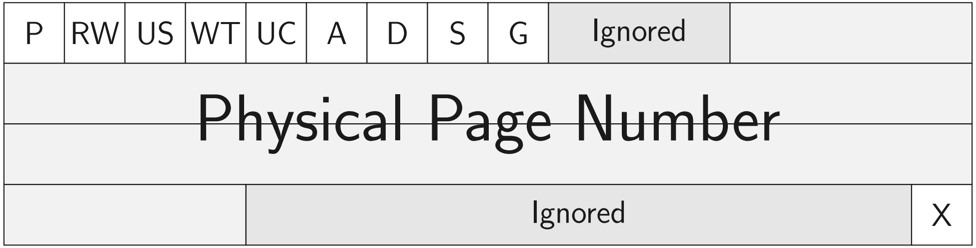
Controlled-Channel Attacks [8]

- Target mechanism which translates virtual to physical addresses
- Enclave memory is set up by OS
- Hardware only ensures one page is mapped by 1 enclave
- Consequence: OS can unmap page, observe page fault

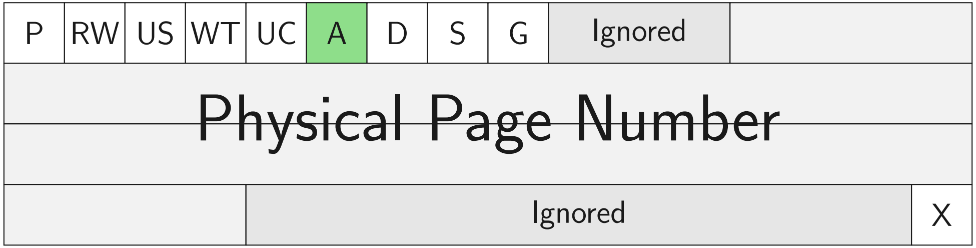
Controlled-Channel Attacks [8]

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

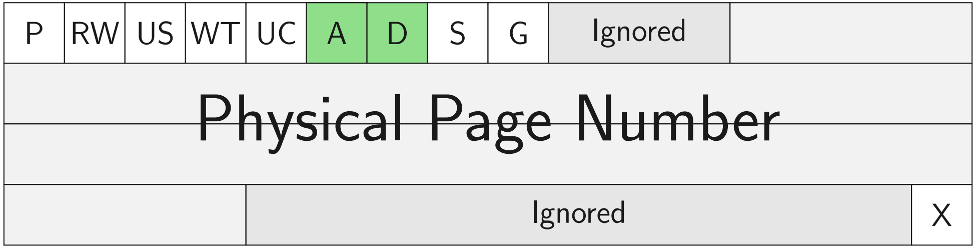
Stealthier Controlled-Channel Attacks [6, 7]



Stealthier Controlled-Channel Attacks [6, 7]



Stealthier Controlled-Channel Attacks [6, 7]



DRAM Attacks [7]

- Enclaves share same physical range of memory

DRAM Attacks [7]

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

DRAM Attacks [7]

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

DRAM Attacks [7]

- 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, Prime+Probe is

Cache Attacks

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

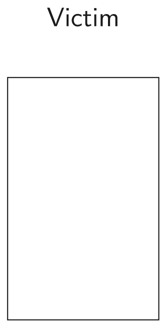
Cache Attacks

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

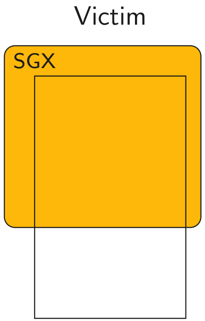
Cache Attacks

- Flush+Reload not possible, Prime+Probe is
- Physical address determines cache set
- Easy to prime cache set as OS
- Examples: [5], [7], [1], [3]

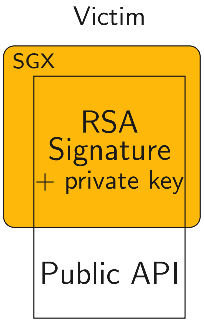
Malware Guard Extension [5]



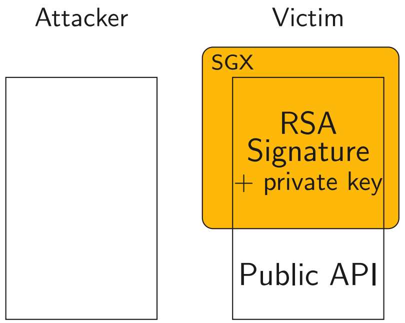
Malware Guard Extension [5]



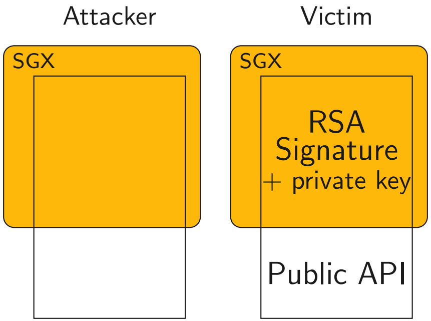
Malware Guard Extension [5]



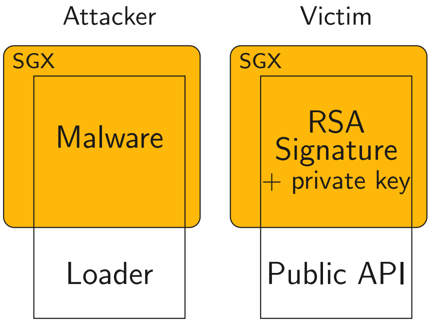
Malware Guard Extension [5]



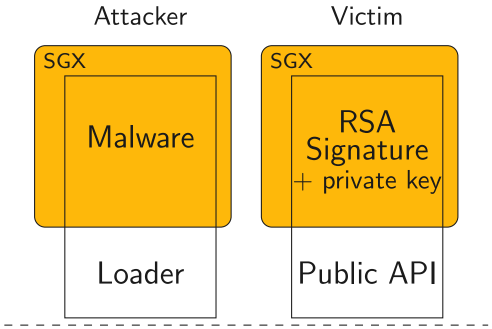
Malware Guard Extension [5]



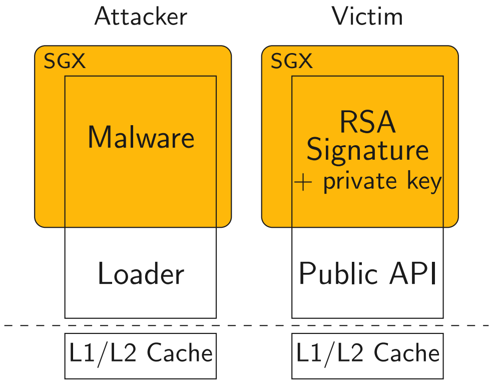
Malware Guard Extension [5]



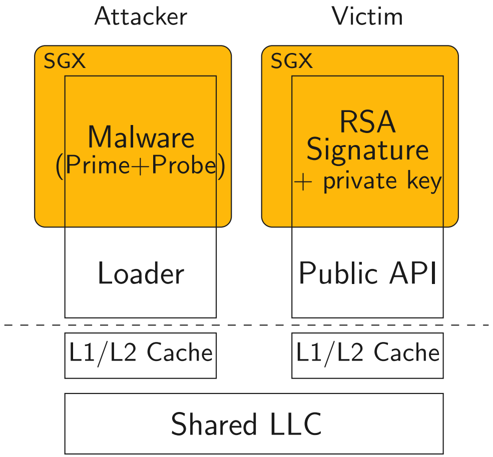
Malware Guard Extension [5]



Malware Guard Extension [5]



Malware Guard Extension [5]



SGX Limitations

- No access to high-precision timer (`rdtsc`)

SGX Limitations

- No access to high-precision timer (`rdtsc`)
- No syscalls

SGX Limitations

- No access to high-precision timer (`rdtsc`)
- No syscalls
- No shared memory

SGX Limitations

- No access to high-precision timer (`rdtsc`)
- No syscalls
- No shared memory
- No physical addresses

SGX Limitations

- No access to high-precision timer (`rdtsc`)
- No syscalls
- No shared memory
- No physical addresses
- No 2 MB large pages

Timer

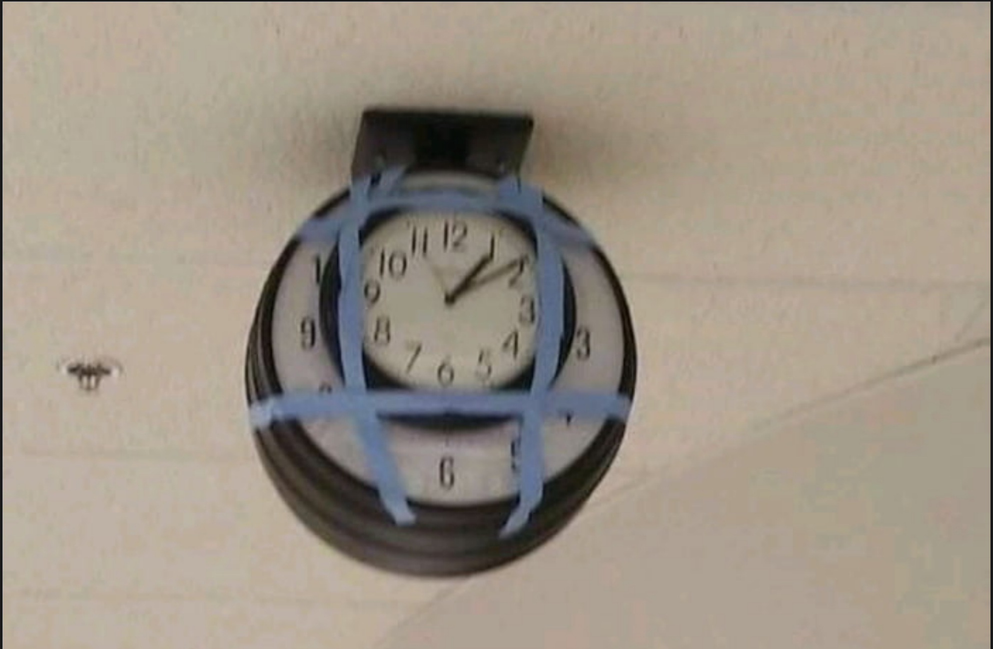
- We can build our own timer [2, 5]

Timer

- We can build our own timer [2, 5]
- Start a thread that continuously increments a global variable

Timer

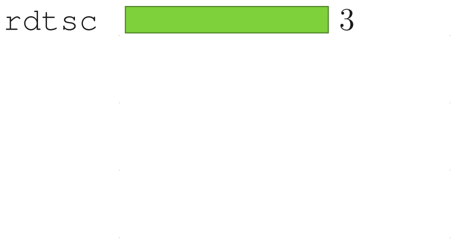
- We can build our own timer [2, 5]
- Start a thread that continuously increments a global variable
- The global variable is our timestamp



ARE YOU REALLY EXPECTING TO
OUTPERFORM THE HARDWARE COUNTER?

Self-built Timer

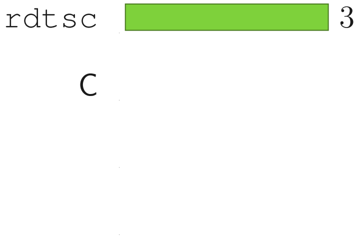
CPU cycles one increment takes



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

Self-built Timer

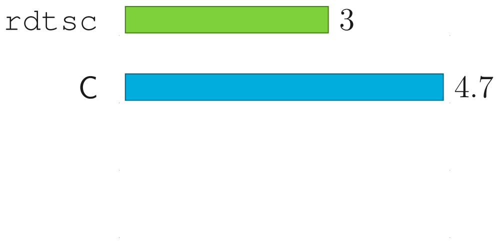
CPU cycles one increment takes



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

Self-built Timer

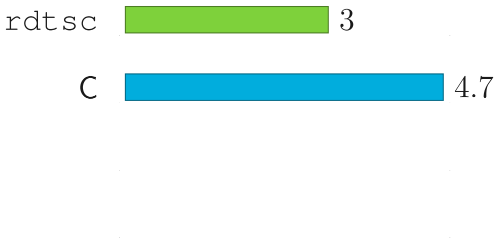
CPU cycles one increment takes



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

Self-built Timer

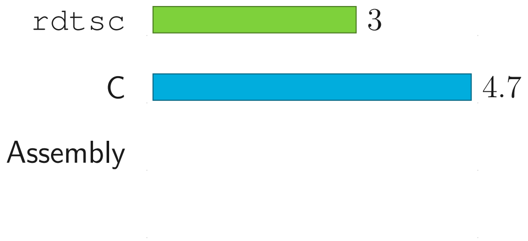
CPU cycles one increment takes



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

Self-built Timer

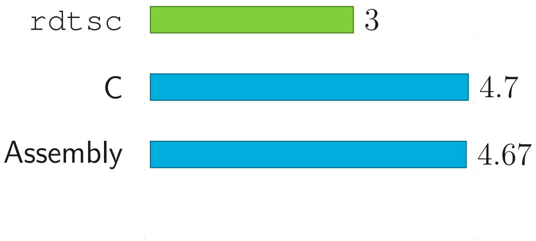
CPU cycles one increment takes



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

Self-built Timer

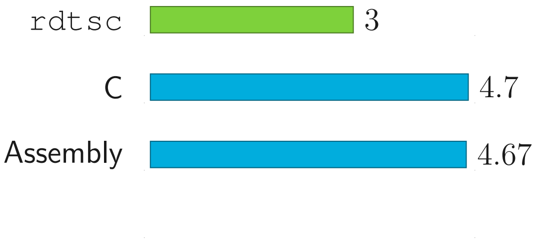
CPU cycles one increment takes



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

Self-built Timer

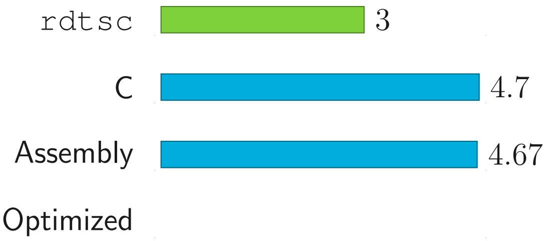
CPU cycles one increment takes



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

Self-built Timer

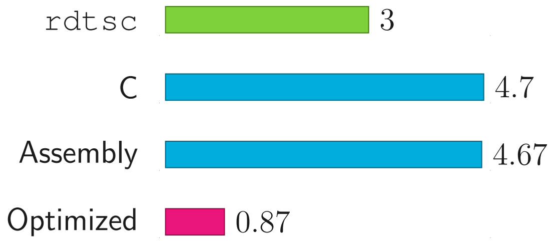
CPU cycles one increment takes



```
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

1. Use the counting primitive to measure DRAM accesses

Combining Everything

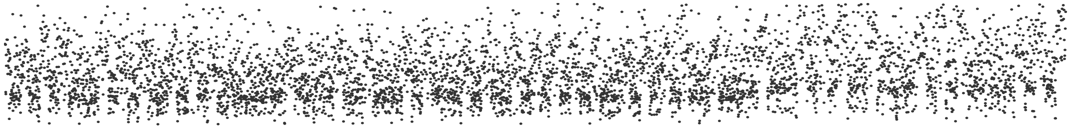
1. Use the counting primitive to measure DRAM accesses
2. Use DRAM side-channel to build eviction set

Combining Everything

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

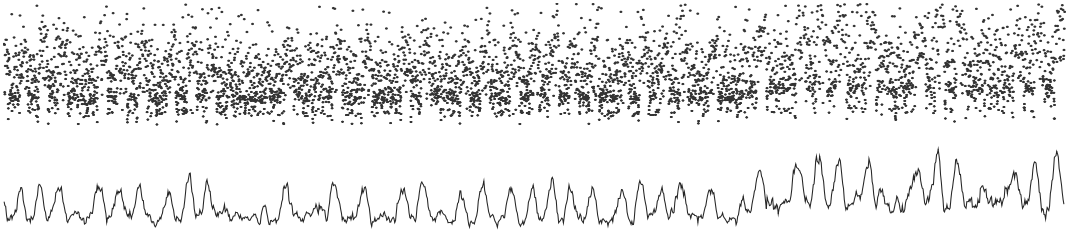
Measured Trace

Raw Prime+Probe trace...



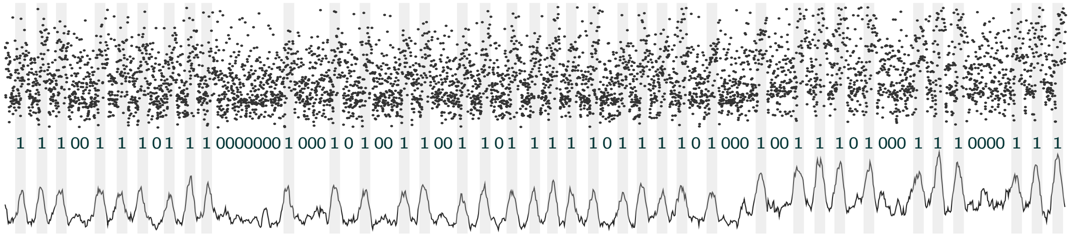
Measured Trace

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



Measured Trace

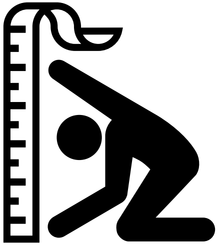
...allows to clearly see the bits of the exponent



Single-Stepping

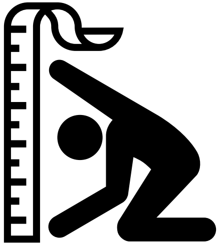
- local Advanced Programmable Interrupt Controller (APIC)

Single-Stepping



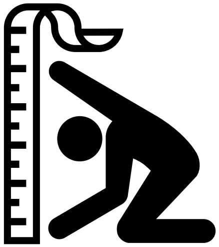
- local Advanced Programmable Interrupt Controller (APIC)
- Timer: 3 modes

Single-Stepping



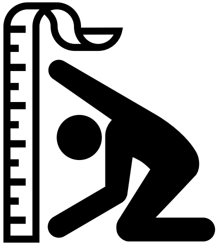
- local Advanced Programmable Interrupt Controller (APIC)
- Timer: 3 modes
  - One-shot

Single-Stepping



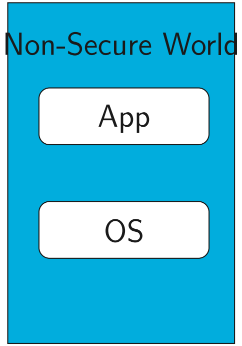
- local Advanced Programmable Interrupt Controller (APIC)
- Timer: 3 modes
  - One-shot
  - Periodic

Single-Stepping

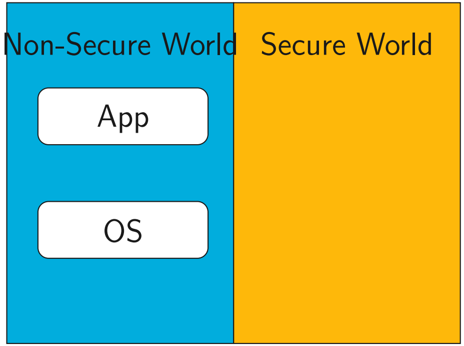


- local Advanced Programmable Interrupt Controller (APIC)
- Timer: 3 modes
    - One-shot
    - Periodic
    - TSC-deadline

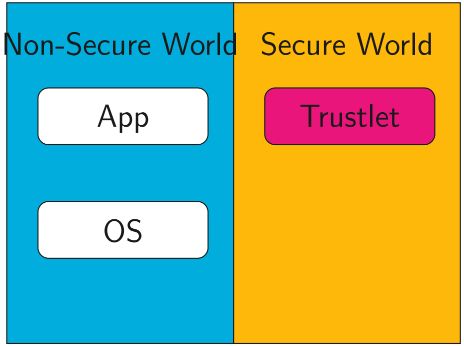
ARM TrustZone



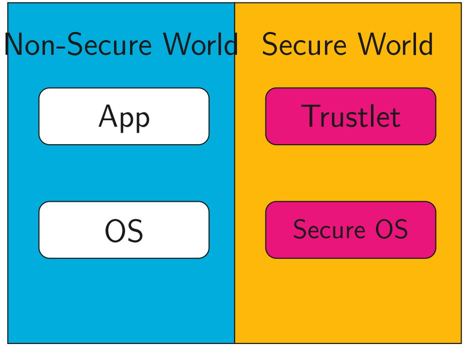
ARM TrustZone



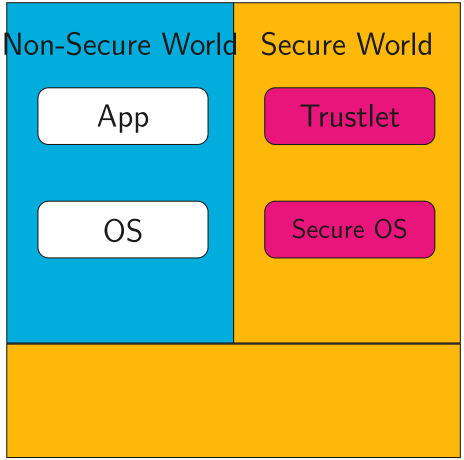
ARM TrustZone



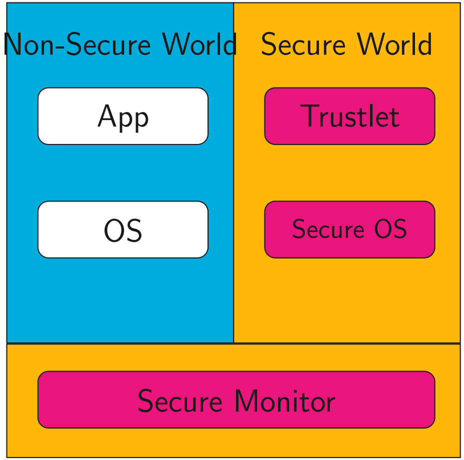
ARM TrustZone



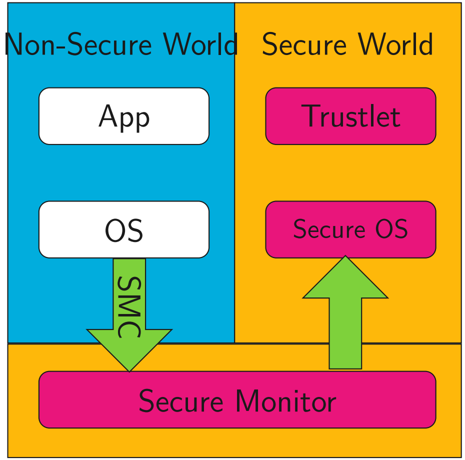
ARM TrustZone



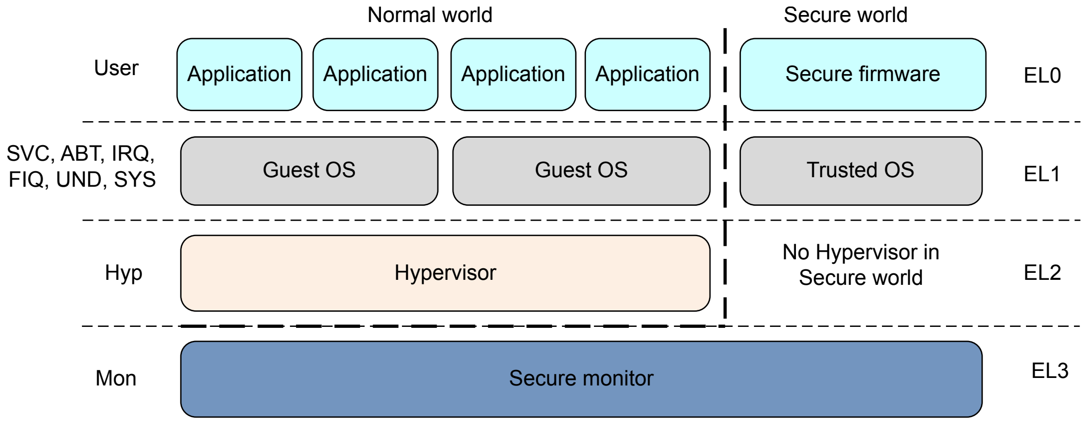
ARM TrustZone



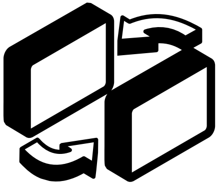
ARM TrustZone



Exception Levels

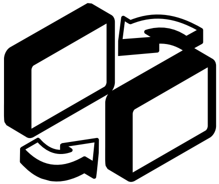


Switching Worlds



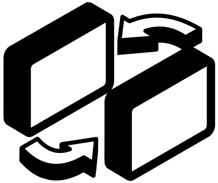
- 2 Virtual Processors: time-sliced fashion operation

Switching Worlds



- 2 Virtual Processors: time-sliced fashion operation
- Normal$\rightarrow$Secure: exceptions to monitor mode

Switching Worlds



- 2 Virtual Processors: time-sliced fashion operation
- Normal→Secure: exceptions to monitor mode

→ SMC or hardware exception mechanism

Switching Worlds

- 2 Virtual Processors: time-sliced fashion operation
- Normal$\rightarrow$Secure: exceptions to monitor mode

$\rightarrow$ SMC or hardware exception mechanism

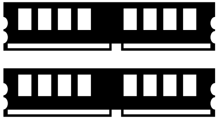
What can be protected?

- Main system bus: read and write channels (AXI)

What can be protected?

- Main system bus: read and write channels (AXI)
- Peripherals: interrupt controllers, timers, user I/O devices (APB)

Attack 1: Double Fetches [4]



- A time-of-check-to-time-of-use (TOCTTOU) bug

Attack 1: Double Fetches [4]

- A time-of-check-to-time-of-use (TOCTTOU) bug
- Shared memory might change after sanity check

Attack 1: Double Fetches [4]

- A time-of-check-to-time-of-use (TOCTTOU) bug
- Shared memory might change after sanity check
- Adversary can abuse this to provide invalid data to application

Attack 1: Double Fetches [4]

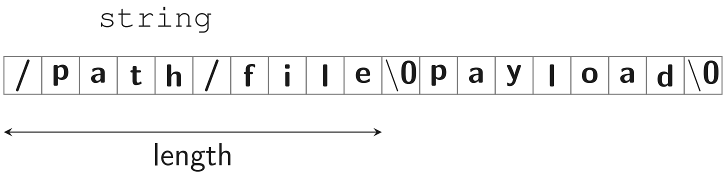
- A time-of-check-to-time-of-use (TOCTTOU) bug
- Shared memory might change after sanity check
- Adversary can abuse this to provide invalid data to application
- Caused by accessing the shared memory twice

Attack 1: Double Fetches [4]

- A time-of-check-to-time-of-use (TOCTTOU) bug
- Shared memory might change after sanity check
- Adversary can abuse this to provide invalid data to application
- Caused by accessing the shared memory twice
- Also called double-fetch bugs

A Double Fetch

A Double Fetch

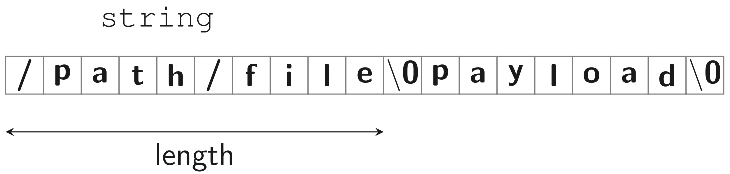


Thread 1

Thread 2

```
strcpy(string, "/path/file\0
    payload");
open(string, O_CREAT);
```

A Double Fetch

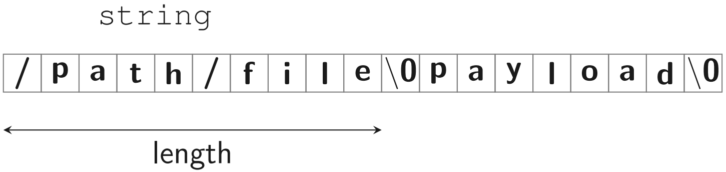


Thread 1

Thread 2

```
strcpy(string, "/path/file\0
    payload");
open(string, O_CREAT);

// <switch to kernel>
```

A Double Fetch

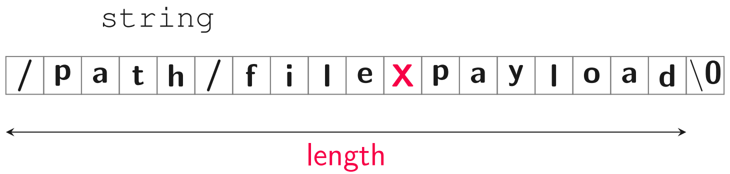


Thread 1

```
strcpy(string, "/path/file\0
    payload");
open(string, O_CREAT);

// <switch to kernel>

int len = strlen(string);
char* local = malloc(len + 1);
```

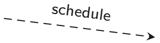
Thread 2

A Double Fetch



Thread 1

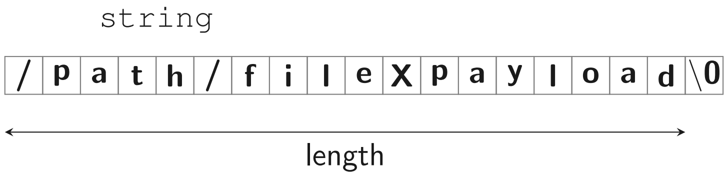
```
strcpy(string, "/path/file\0
    payload");
open(string, O_CREAT);

// <switch to kernel>

int len = strlen(string);
char* local = malloc(len + 1);
```



Thread 2

```
string[10] = 'X';
```

A Double Fetch



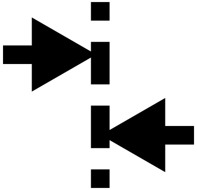
Thread 1

```
strcpy(string, "/path/file\0
    payload");
open(string, O_CREAT);

// <switch to kernel>

int len = strlen(string);
char* local = malloc(len + 1);

strcpy(local, string);
```

Thread 2

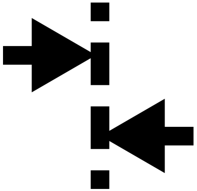
```
string[10] = 'X';
```

Double Fetches



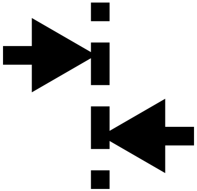
- Not all double fetches are exploitable

Double Fetches



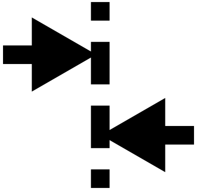
- Not all double fetches are exploitable
- Changing data after sanity check allows exploitation

Double Fetches



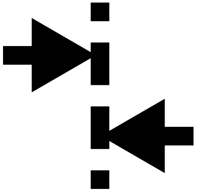
- Not all double fetches are exploitable
- Changing data after sanity check allows exploitation
- Critical if privilege boundaries are crossed

Double Fetches



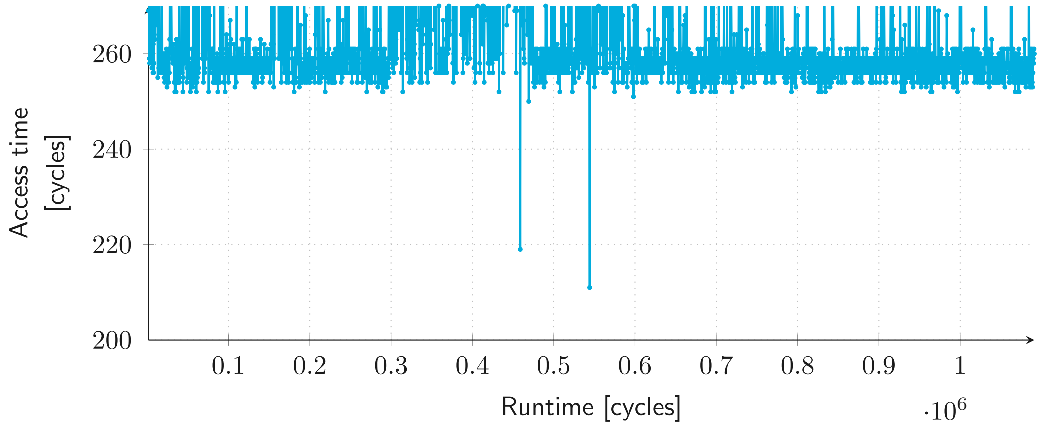
- Not all double fetches are exploitable
- Changing data after sanity check allows exploitation
- Critical if privilege boundaries are crossed
  - User space $\leftrightarrow$ Kernel space
  - Untrusted code $\leftrightarrow$ Trusted code

Double Fetches

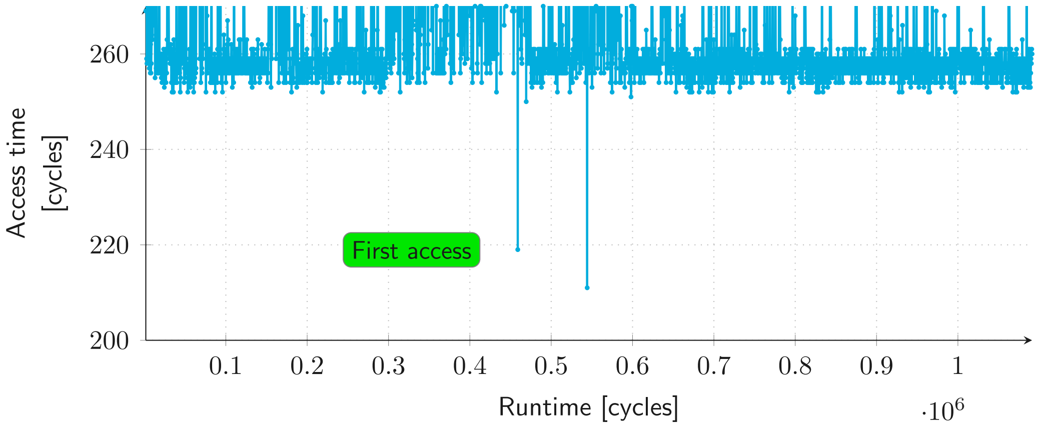


- Not all double fetches are exploitable
- Changing data after sanity check allows exploitation
- Critical if privilege boundaries are crossed
  - User space ↔ Kernel space
  - Untrusted code ↔ Trusted code
- Common to share data across these domains

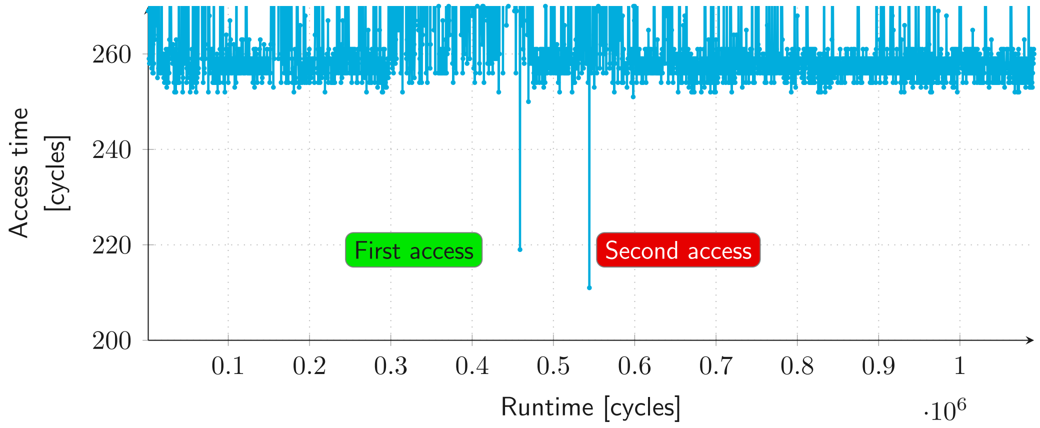
Double-fetch Detection



Double-fetch Detection



Double-fetch Detection



Attack 2: Armageddon [2]

- Different trustlets running in secure world

Attack 2: Armageddon [2]

- Different trustlets running in secure world
    - Credential-store

Attack 2: Armageddon [2]

- Different trustlets running in secure world
  - Credential-store
  - Secure element for payments

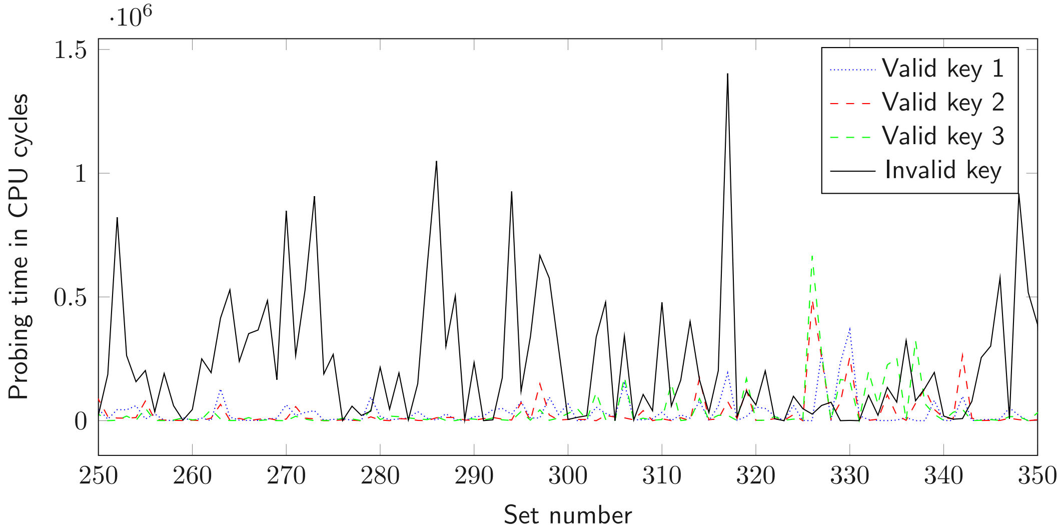
Attack 2: Armageddon [2]

- Different trustlets running in secure world
    - Credential-store
    - Secure element for payments
    - DRM

Attack 2: Armageddon [2]

- Different trustlets running in secure world
  - Credential-store
  - Secure element for payments
  - DRM
- TrustZone leaks through the cache

Leakage from ARM TrustZone (RSA signatures)



Conclusion

- TEEs developed to protect sensitiv information/critical code execution

Conclusion

- TEEs developed to protect sensitiv information/critical code execution
- Allow for a powerful threat model

Conclusion

- TEEs developed to protect sensitiv information/critical code execution
- Allow for a powerful threat model
- SCAs often not “out of scope”

Side-Channel Security

Chapter 3: Trusted Execution Environments

Daniel Gruss

March 14, 2024

Graz University of Technology

References

[1] 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*.

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

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

[4] Schwarz, M., Gruss, D., Lipp, M., Maurice, C., Schuster, T., Fogh, A., and Mangard, S. (2018). Automated Detection, Exploitation, and Elimination of Double-Fetch Bugs using Modern CPU Features. In *AsiaCCS*.

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

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

[7] 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*.

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