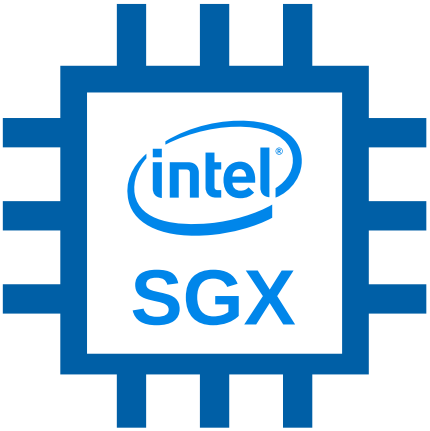
SGX - Secure Enclaves als Angriffsvektor

Daniel Gruss, Michael Schwarz, Moritz Lipp

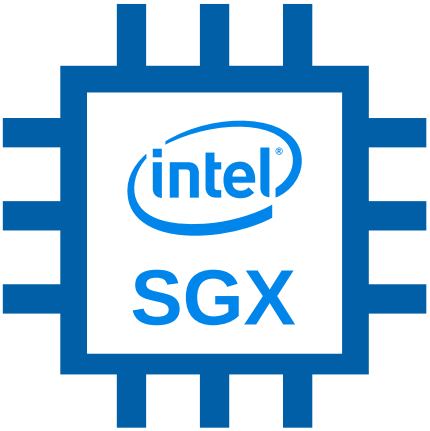
October 1, 2019 - IKT-Sicherheitskonferenz

Graz University of Technology

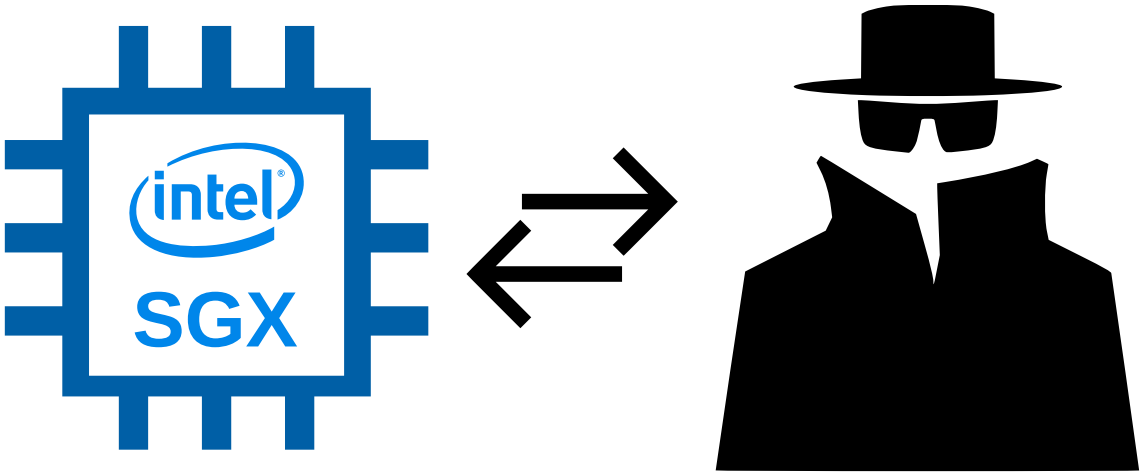
Outline

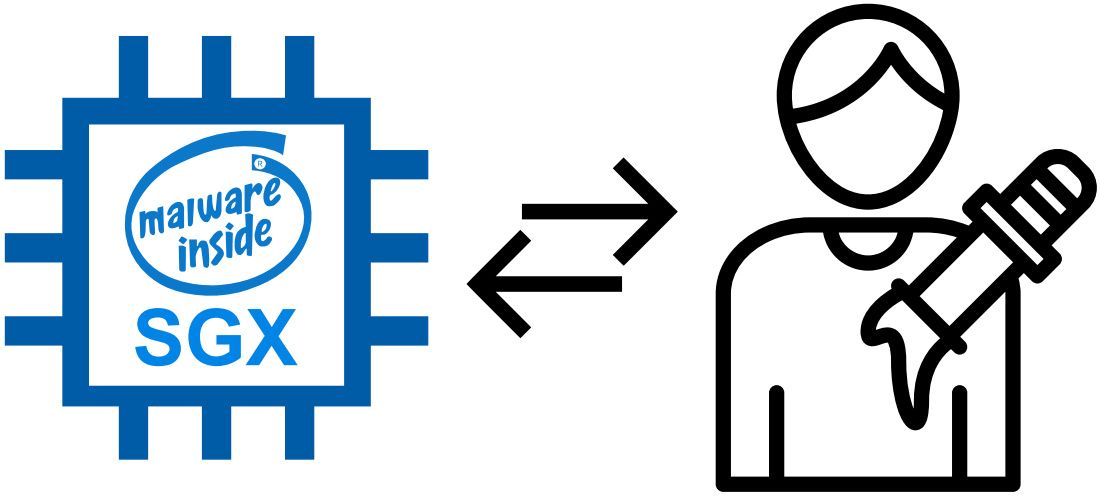


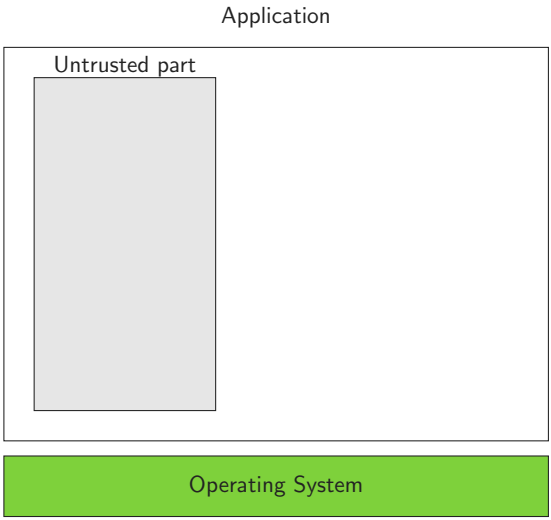
Outline

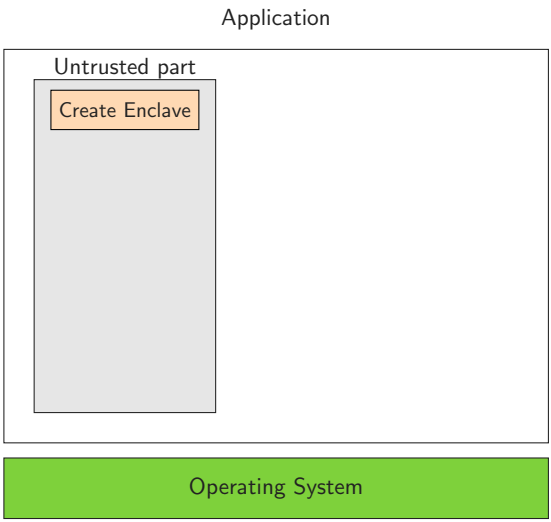


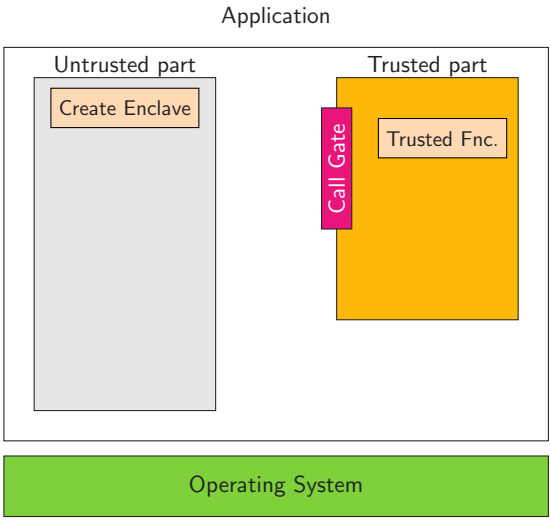
Outline

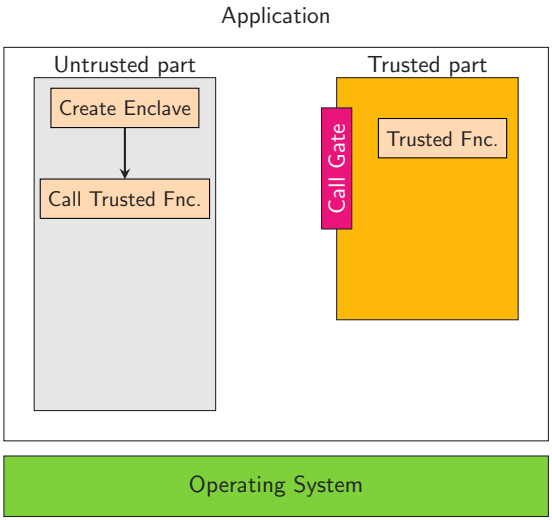


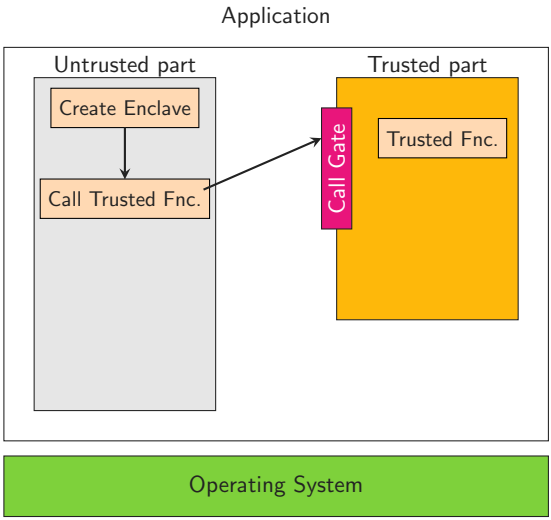

malware
inside
SGX

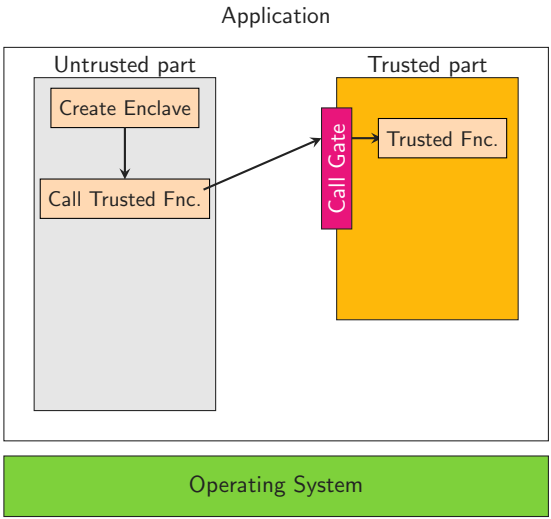

Application
Untrusted part
Operating System

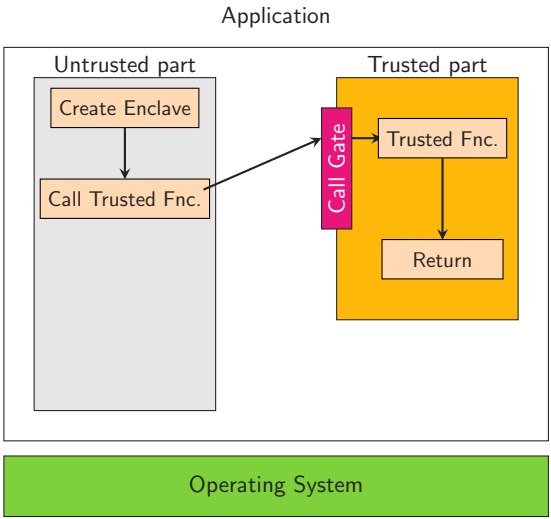

Application
Untrusted part
Create Enclave
Operating System

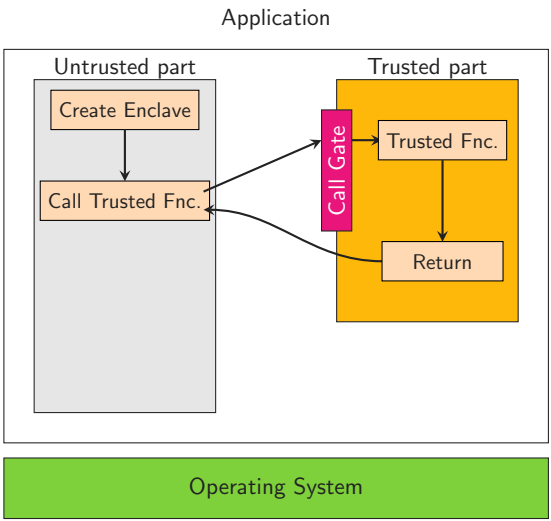

Application
Untrusted part
Create Enclave
Trusted part
Call Gate
Trusted Fnc.
Operating System

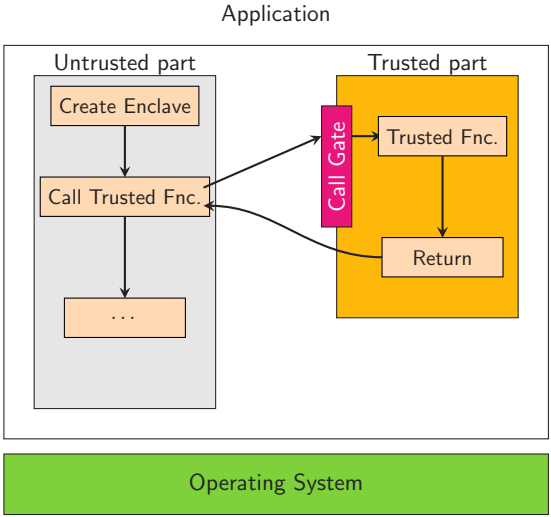

Application
Untrusted part
Create Enclave
Call Trusted Fnc.
Trusted part
Call Gate
Trusted Fnc.
Operating System

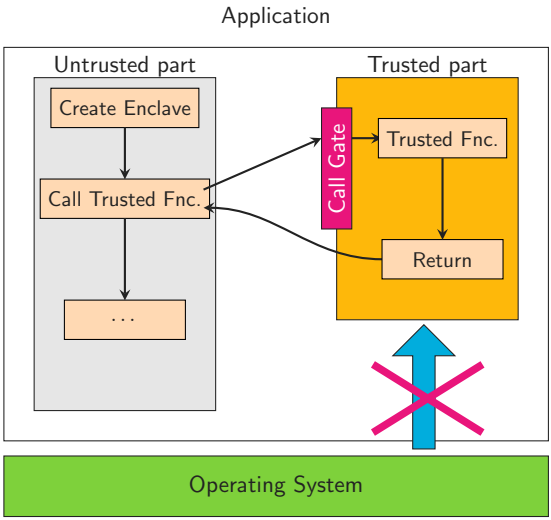

Application
Untrusted part
Trusted part
Create Enclave
Call Trusted Fnc.
Call Gate
Trusted Fnc.
Operating System


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


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


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


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


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

What if?

- Enclaves are black boxes

What if?

- Enclaves are black boxes
- Protected from all applications and OS

What if?

- Enclaves are black boxes
- Protected from all applications and OS
- What if they contain malicious code?

What if?

- Enclaves are black boxes
- Protected from all applications and OS
- What if they contain malicious code?
- Can we hide zero days?

The Invisible Things Lab's blog

Kernel, Hypervisor, Virtualization, Trusted Computing and other system-level security stuff

Monday, September 23, 2013

Thoughts on Intel's upcoming Software Guard Extensions (Part 2)

In the first part of this article published a few weeks ago, I have discussed the basics of Intel SGX technology, and also discussed challenges with using SGX for securing desktop systems, specifically focusing on the problem of trusted input and output. In this part we will look at some other aspects of Intel SGX, and we will start with a discussion of how it could be used to create a truly irreversible software.

SGX Blackboxing – Apps and malware that cannot be reverse engineered?

About Me

Joanna Rutkowska

Founder of Invisible Things Lab, Qubes OS project lead.

View my complete profile

Links

twitter: @rootkovska

Qubes Project

http://theinvisiblethings.blogspot.com/2013/09/thoughts-on-intels-upcoming-software.html

Intel's Statement

[...] Intel is aware of this research which is based upon assumptions that are outside the threat model for Intel SGX. The value of Intel SGX is to execute code in a protected enclave; however, Intel SGX does not guarantee that the code executed in the enclave is from a trusted source [...]

Classical exploits cannot be mounted within SGX:

SGX Limitations

Classical exploits cannot be mounted within SGX:

- No syscalls

Classical exploits cannot be mounted within SGX:

- No syscalls
- No shared memory/libraries

Classical exploits cannot be mounted within SGX:

- No syscalls
- No shared memory/libraries
- No interprocess communication

Classical exploits cannot be mounted within SGX:

- No syscalls
- No shared memory/libraries
- No interprocess communication
- Blocked instructions

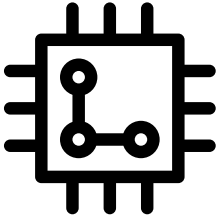
State-of-the-art Malicious Enclaves

- Side-channel attacks from SGX [Sch+17]

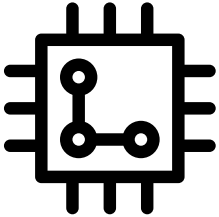
- Side-channel attacks from SGX [Sch+17]
- Fault attacks from SGX [Gru+18; Jan+17]

State-of-the-art Malicious Enclaves

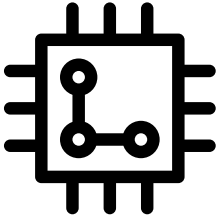
- Side-channel attacks from SGX [Sch+17]
- Fault attacks from SGX [Gru+18; Jan+17]
- No real exploits from SGX so far



- SGX enclaves run on CPU

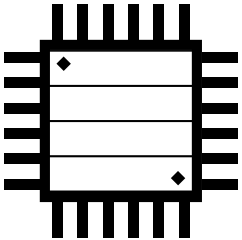


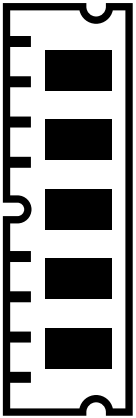
- SGX enclaves run on CPU
- Shared hardware resources



- SGX enclaves run on CPU
- Shared hardware resources
- Spy on the system (resource utilization)

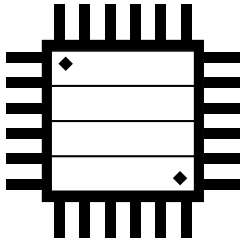
```
printf("%d", i);
printf("%d", i);
```

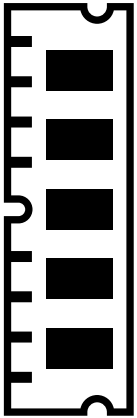


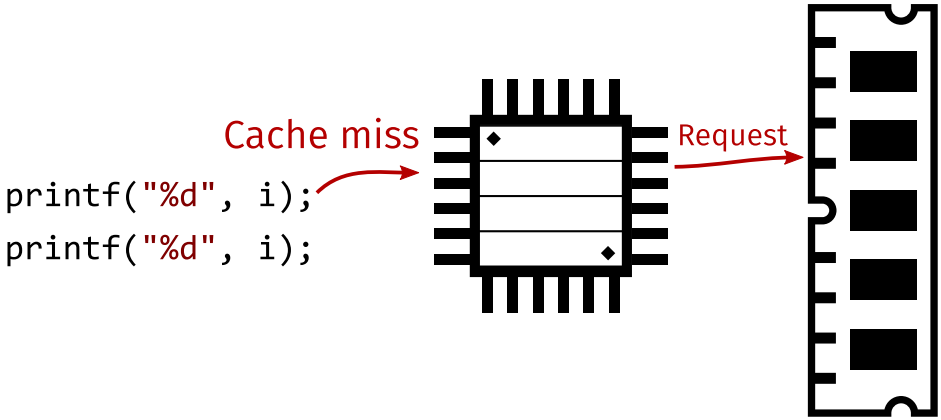


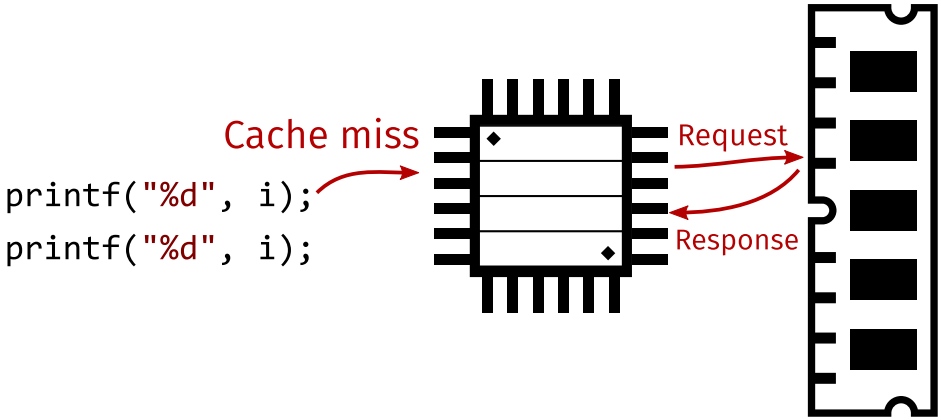
Cache miss

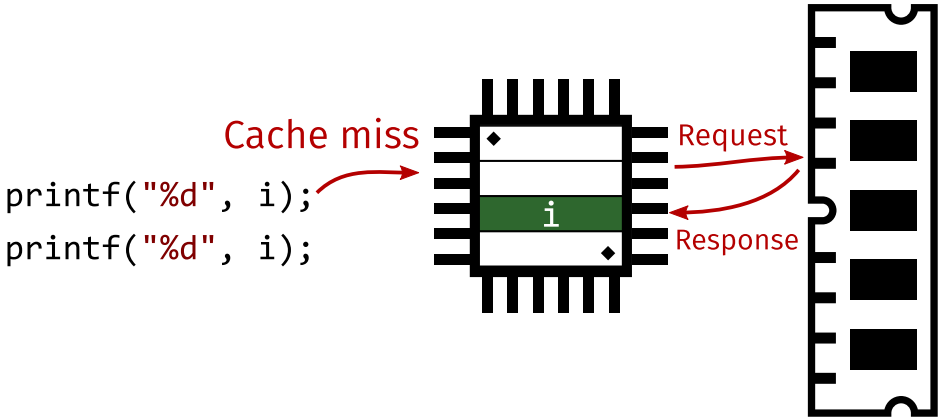
```
printf("%d", i);
printf("%d", i);
```

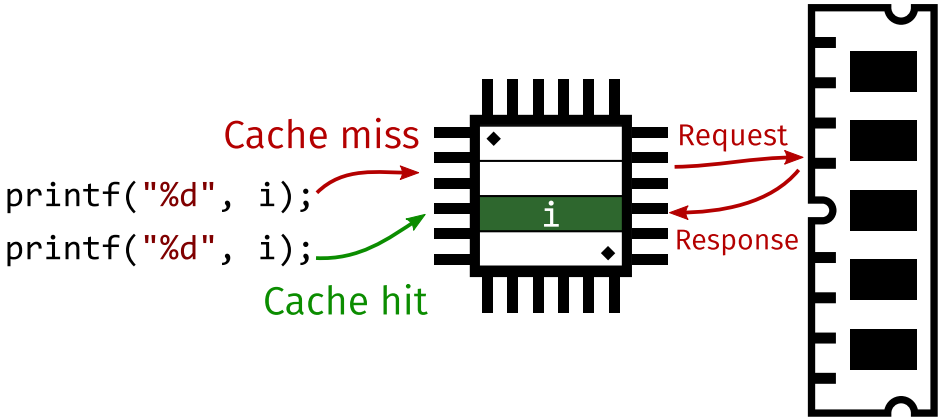


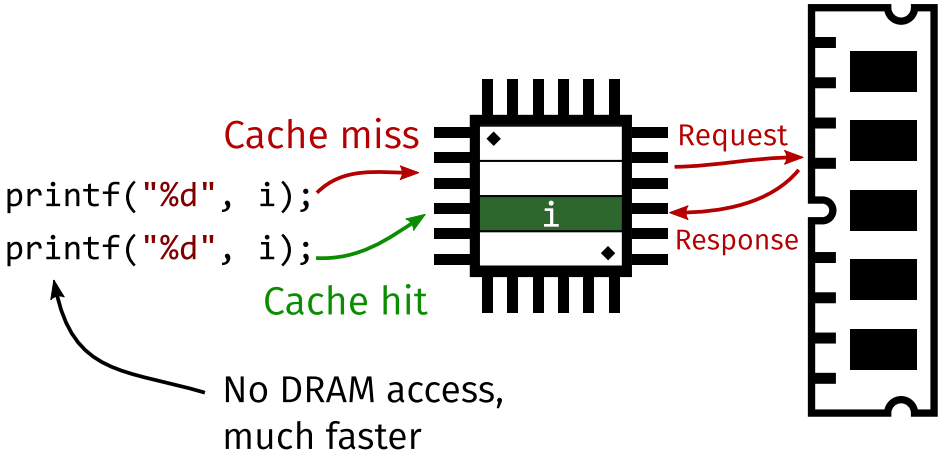


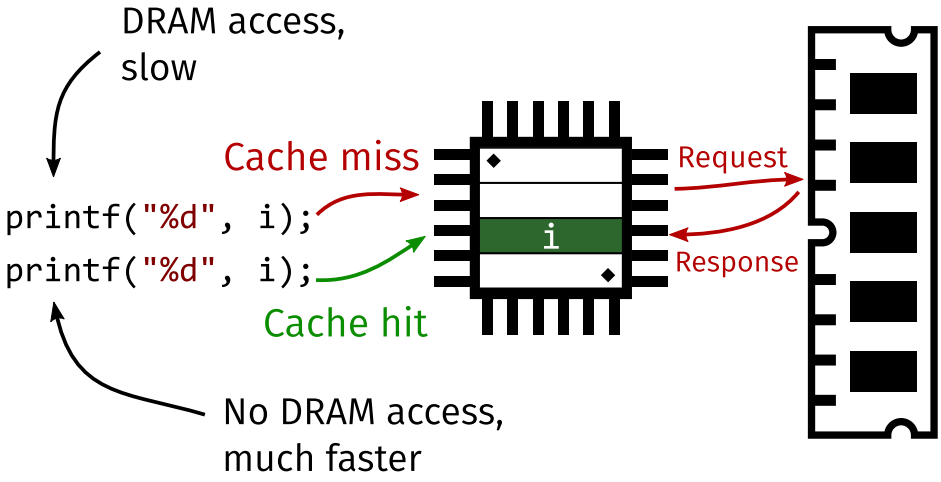

Cache miss
Request
printf("%d", i);
printf("%d", i);

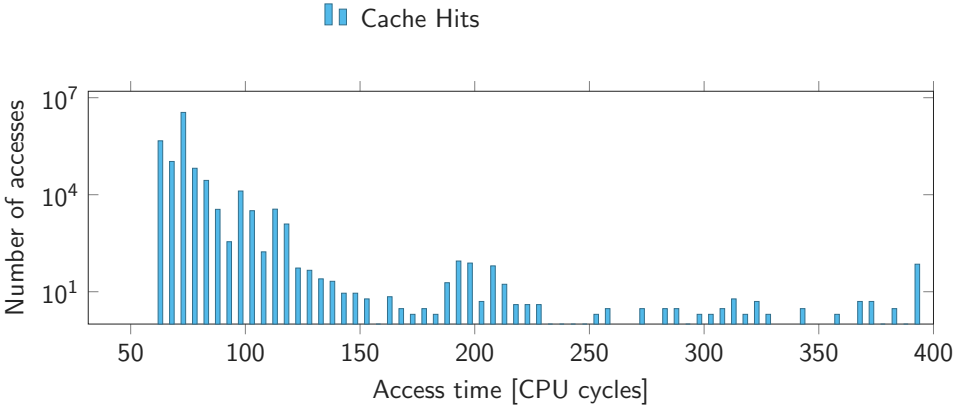

Cache miss
printf("%d", i);
printf("%d", i);
Request
Response

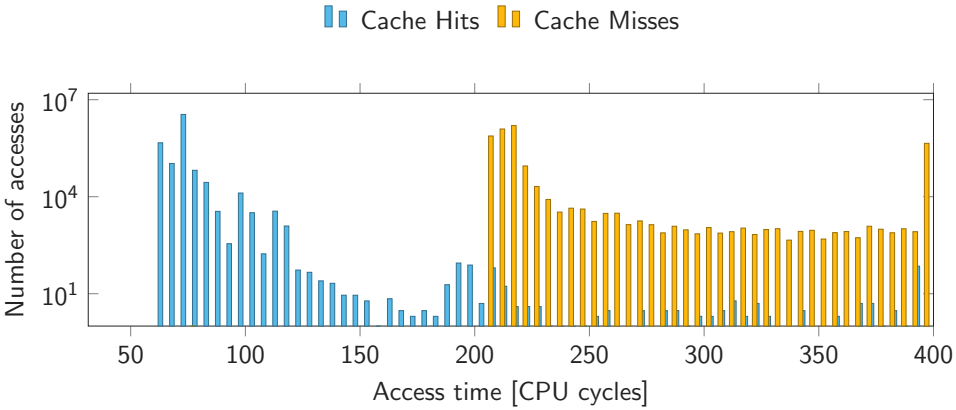

Cache miss
printf("%d", i);
printf("%d", i);
i
Request
Response


Cache miss
printf("%d", i);
printf("%d", i);
Cache hit
i
Request
Response


Cache miss
printf("%d", i);
printf("%d", i);
Cache hit
i
Request
Response
No DRAM access,
much faster


DRAM access,
slow
Cache miss
printf("%d", i);
printf("%d", i);
Cache hit
No DRAM access,
much faster
i
Request
Response

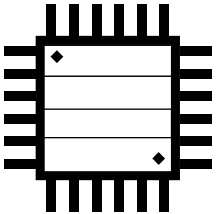

Cache Hits
Number of accesses
10^7
10^4
10^1
50
100
150
200
250
300
350
400
Access time [CPU cycles]


Cache Hits
Cache Misses
Number of accesses
10^7
10^4
10^1
50
100
150
200
250
300
350
400
Access time [CPU cycles]

ATTACKER

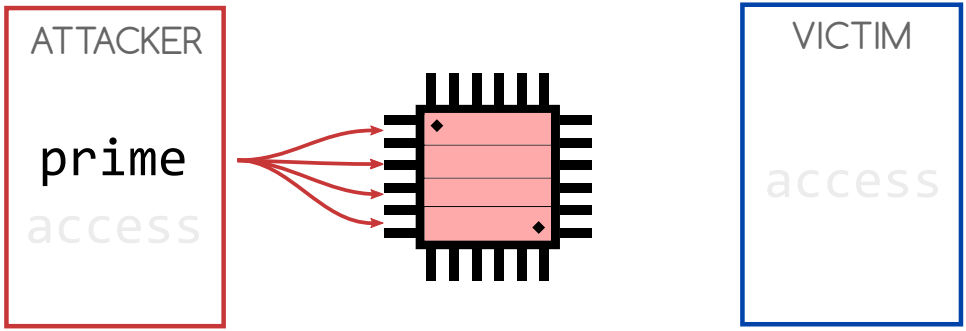
prime

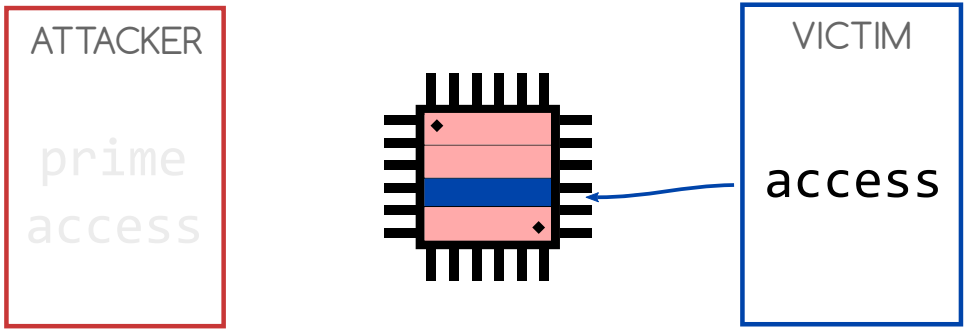
access

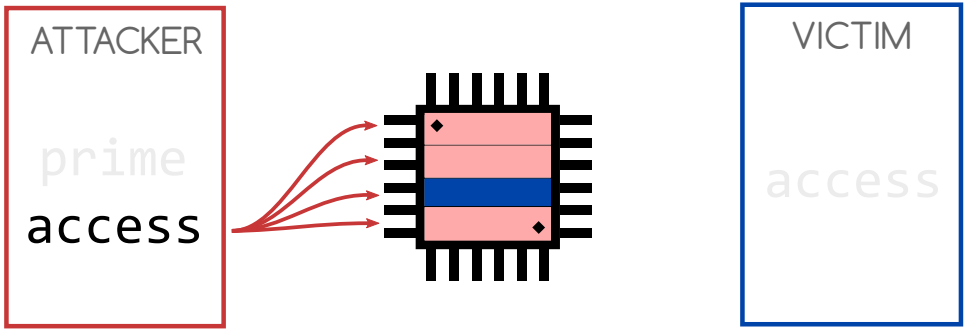


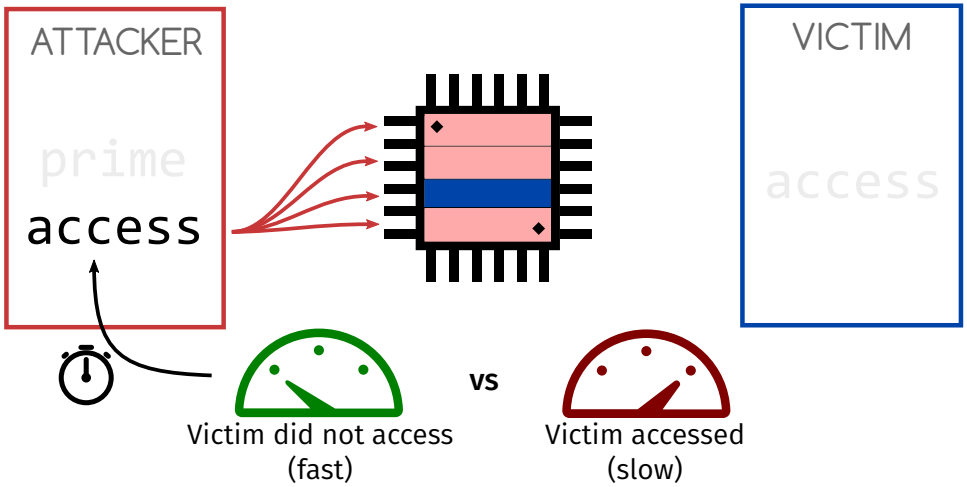
VICTIM

access

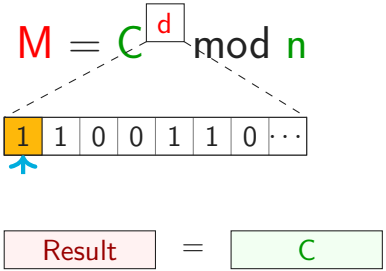

ATTACKER
prime
access
VICTIM
access

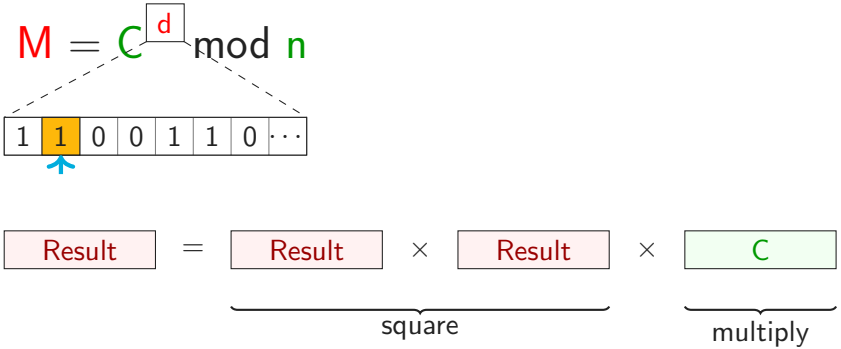

ATTACKER
prime
access
VICTIM
access

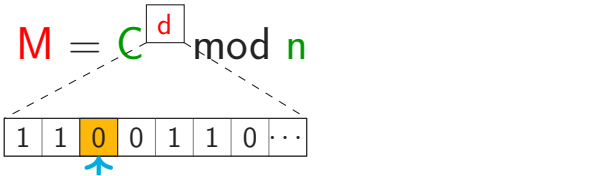

ATTACKER
prime
access
VICTIM
access

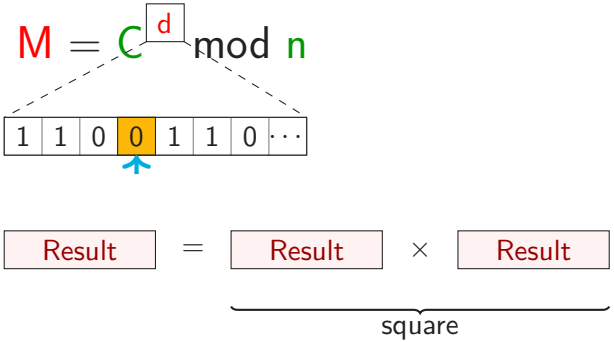

ATTACKER
prime
access
VICTIM
access
vs
Victim did not access
(fast)
Victim accessed
(slow)

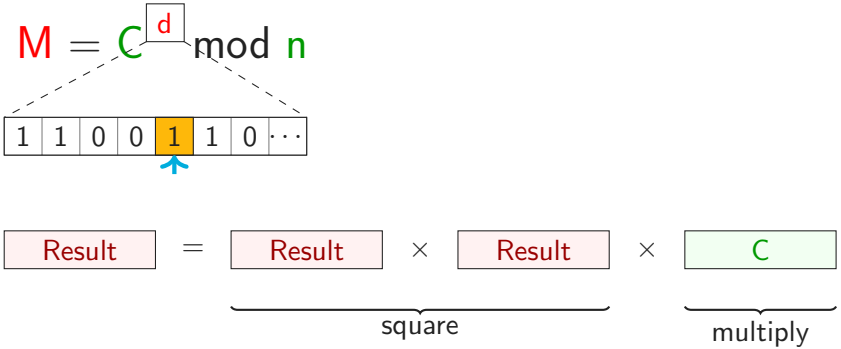
$$M = C^{d} \bmod n$$

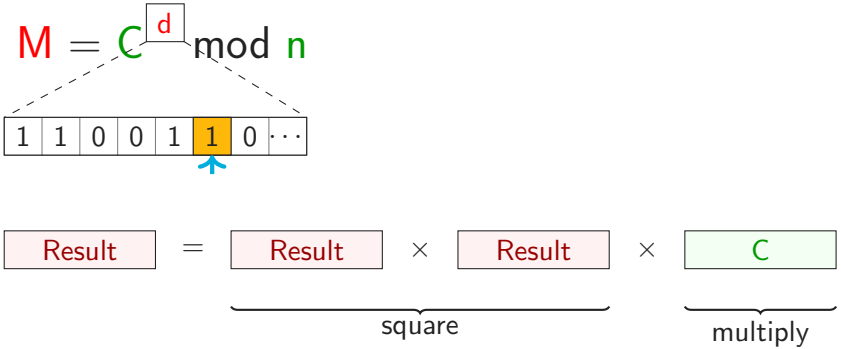

M = C d mod n
1 1 0 0 1 1 0 ···
Result = C

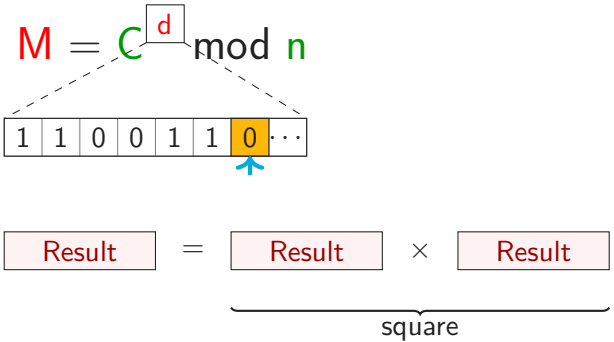

M = C^d mod n
1 1 0 0 1 1 0 ···
Result = Result × Result × C
square
multiply


M = C^d mod n
1 1 0 0 1 1 0 ···
Result = Result × Result
square


M = C^d mod n
1 1 0 0 1 1 0 ···
Result = Result × Result
square

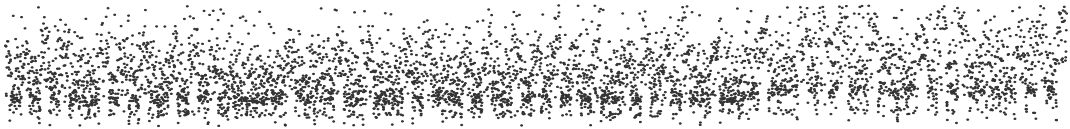

M = C^d mod n
1 1 0 0 1 1 0 ···
Result = Result × Result × C
square
multiply


M = C^d mod n
1 1 0 0 1 1 0 ···
Result = Result × Result × C
square
multiply


M = C^d mod n
1 1 0 0 1 1 0 ···
Result = Result × Result
square

Measured Trace

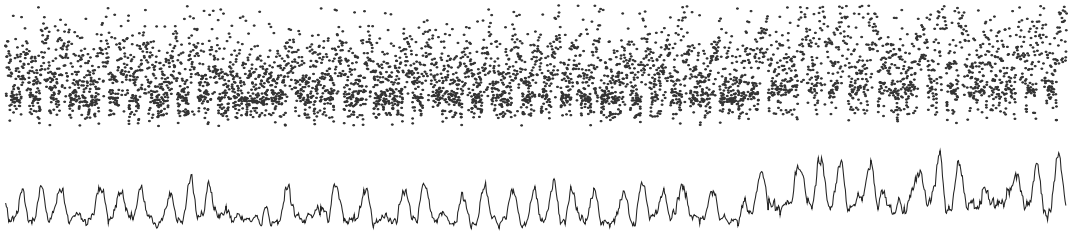
Raw Prime+Probe trace...



Malware Guard Extension: Using SGX to Conceal Cache Attacks.
Michael Schwarz, Samuel Weiser, Daniel Gruss, Clémentine Maurice, Stefan Mangard. DIMVA'17

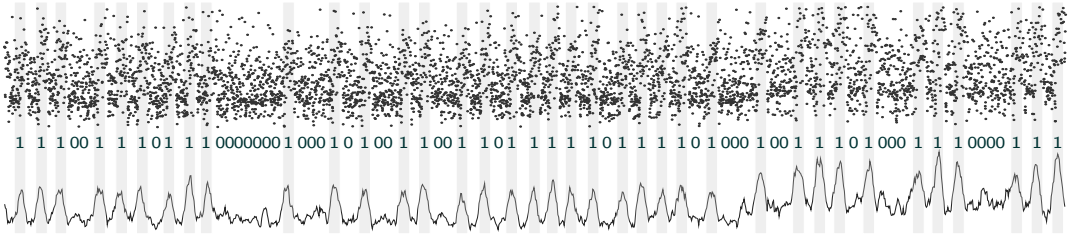
Measured Trace

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



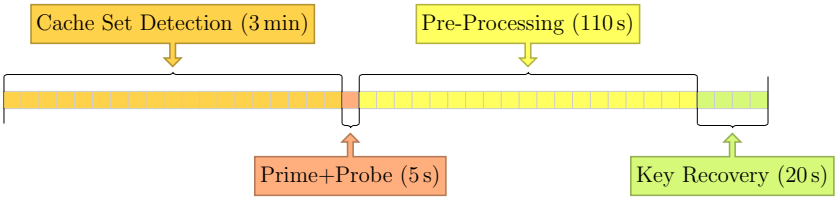
Malware Guard Extension: Using SGX to Conceal Cache Attacks.
Michael Schwarz, Samuel Weiser, Daniel Gruss, Clémentine Maurice, Stefan Mangard. DIMVA'17

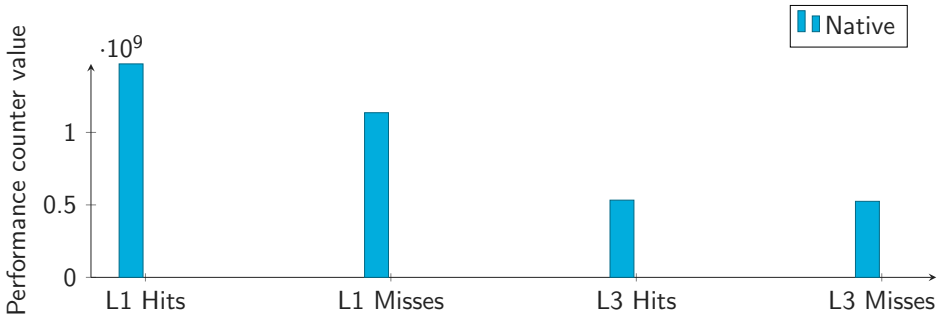
...allows to clearly see the bits of the exponent

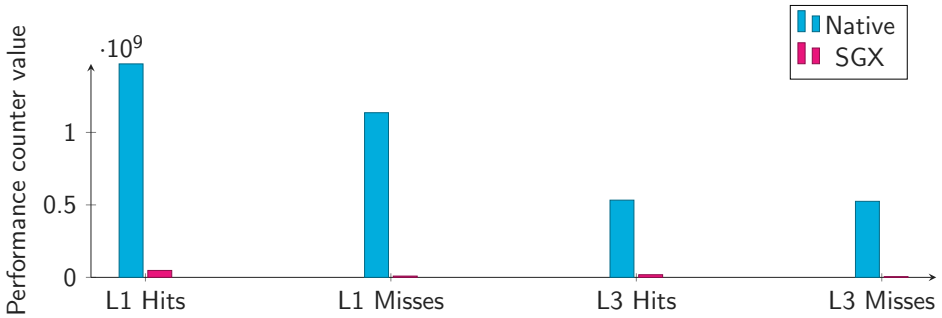


Malware Guard Extension: Using SGX to Conceal Cache Attacks.
Michael Schwarz, Samuel Weiser, Daniel Gruss, Clémentine Maurice, Stefan Mangard. DIMVA'17

Full recovery of a 4096-bit RSA key in approximately 5 minutes




Native
Performance counter value
$\cdot 10^9$
1
0.5
0
L1 Hits
L1 Misses
L3 Hits
L3 Misses


Native
SGX
Performance counter value
·10^9
1
0.5
0
L1 Hits
L1 Misses
L3 Hits
L3 Misses

- Side-channel attacks from SGX possible

- Side-channel attacks from SGX possible
- Allow attacker to spy on meta data

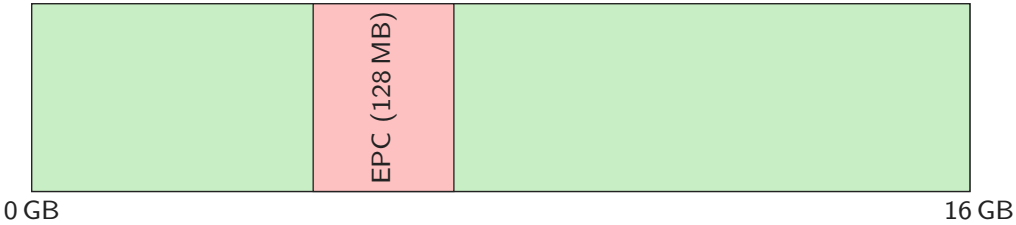
- Side-channel attacks from SGX possible
- Allow attacker to spy on meta data
- Completely hide an attack

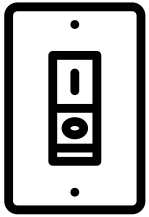
- Cache attacks preventable on source level

- Cache attacks preventable on source level
- Side-channel resistant crypto

- Cache attacks preventable on source level
- Side-channel resistant crypto
- Default in most crypto libraries

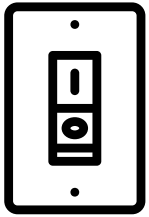

0 GB
16 GB


EPC (128 MB)
0 GB
16 GB



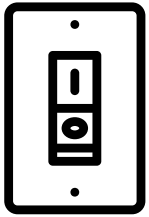
- What happens if a bit flips in the EPC?

Bit Flips in the EPC



- What happens if a bit flips in the EPC?
- Integrity check will fail!

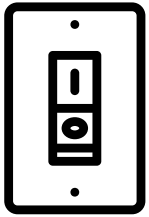
Bit Flips in the EPC



- What happens if a bit flips in the EPC?
- Integrity check will fail!

→ Locks up the memory controller

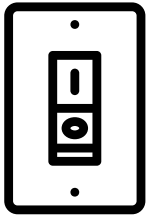
Bit Flips in the EPC



- What happens if a bit flips in the EPC?
- Integrity check will fail!

→ Locks up the memory controller

→ Not a single further memory access!



- What happens if a bit flips in the EPC?
- Integrity check will fail!

→ Locks up the memory controller

→ Not a single further memory access!

→ System halts immediately

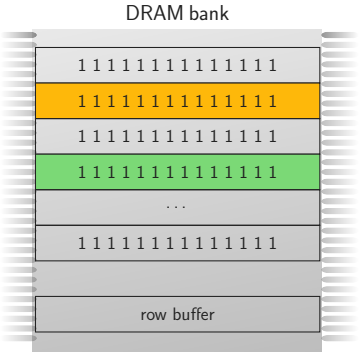
- If a malicious enclave induces a bit flip, . . .

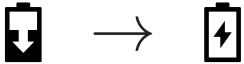
Bit Flips in the EPC

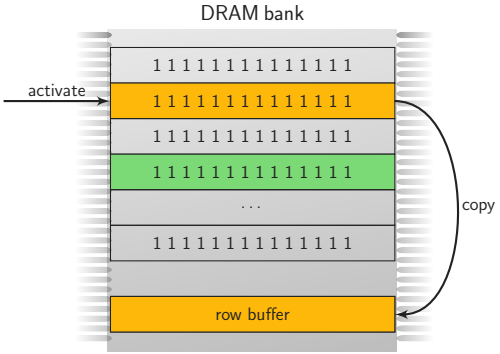
- If a malicious enclave induces a bit flip, ...
- ...the entire machine halts

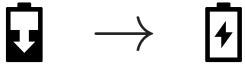
- If a malicious enclave induces a bit flip, ...
- ... the entire machine halts
- ... including co-located tenants

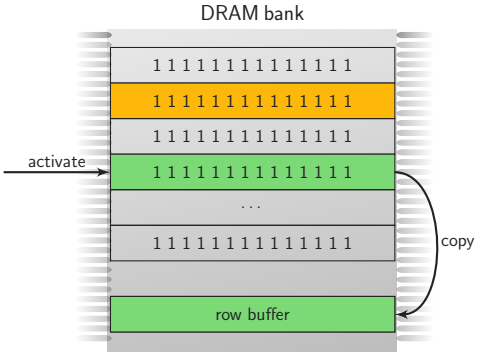
- If a malicious enclave induces a bit flip, ...
- ... the entire machine halts
- ... including co-located tenants
- Denial-of-Service Attacks in the Cloud [Gru+18; Jan+17]

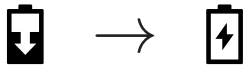

DRAM bank
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
. . .
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
row buffer



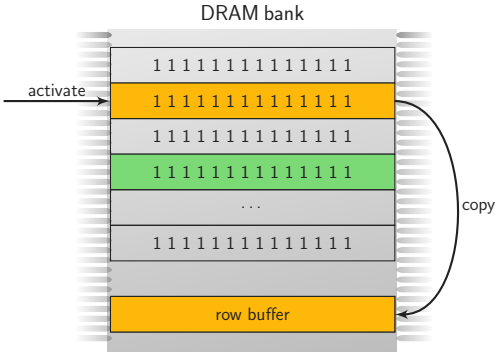

DRAM bank
activate
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
. . .
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
copy
row buffer

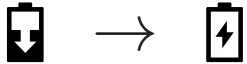




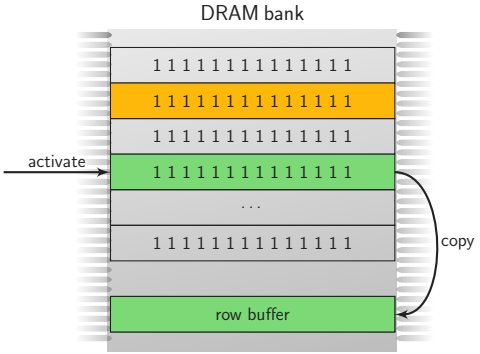


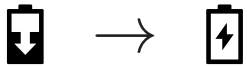
Cells leak faster upon proximate accesses $\rightarrow$ Rowhammer



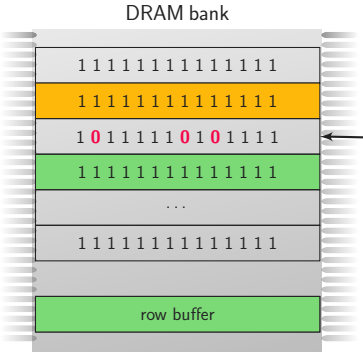


Cells leak faster upon proximate accesses $\rightarrow$ Rowhammer

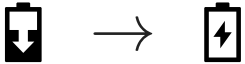




Cells leak faster upon proximate accesses $\rightarrow$ Rowhammer



bit flips in row 2!



Cells leak faster upon proximate accesses → Rowhammer

How widespread is the issue?

- 85% affected (estimation 2014)
- 52% affected (estimation 2015)

How widespread is the issue?

- 85% affected (estimation 2014)
- 52% affected (estimation 2015)

- First believed to be safe
- We showed bit flips in 2016
- 67% affected (estimation 2016)

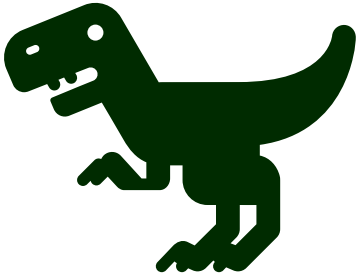
- Dangerous attacks but difficult in practice

- Dangerous attacks but difficult in practice
- More relevant: zero days in enclaves

Zero Days?

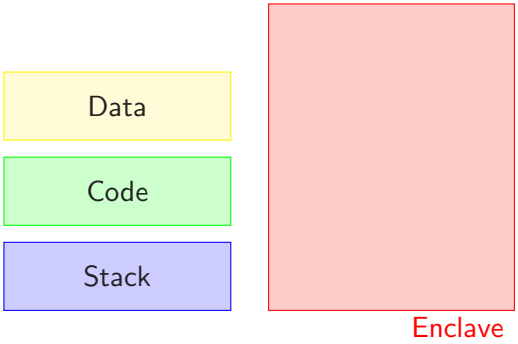
- Dangerous attacks but difficult in practice
- More relevant: zero days in enclaves

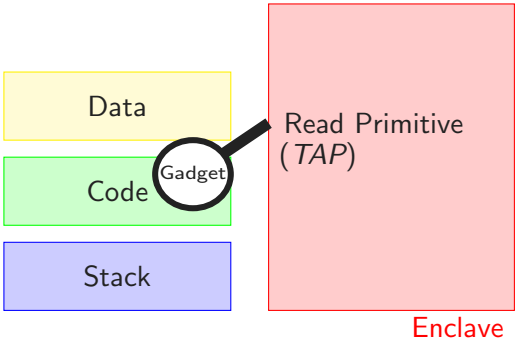
→ Super malware

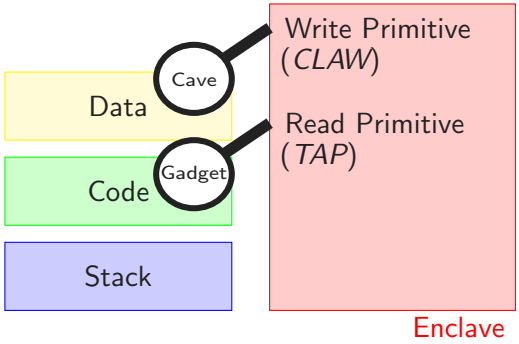


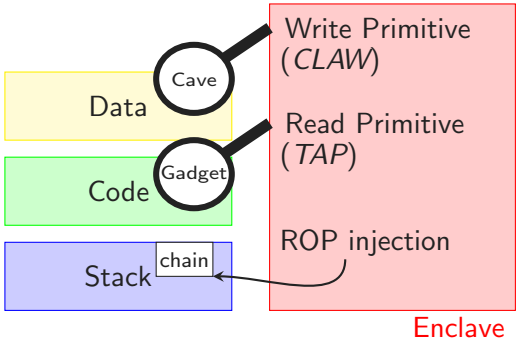
TEE-REX

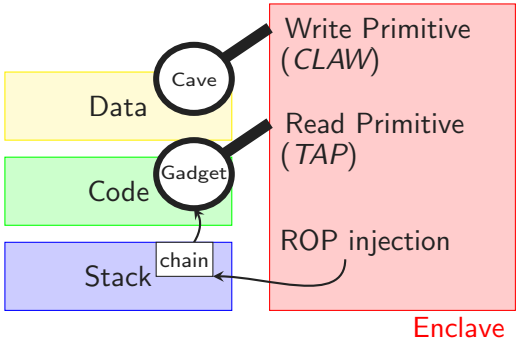
Trusted **E**xecution **E**nvironment **R**eturn-oriented-programming **EX**ploit

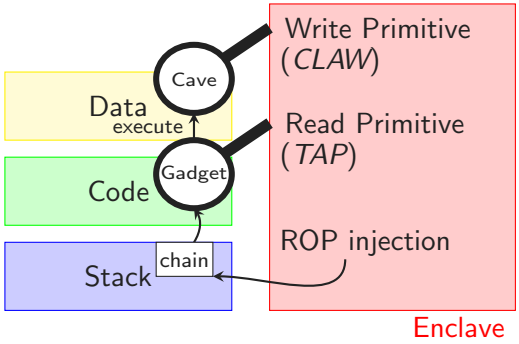

Data
Code
Stack
Enclave

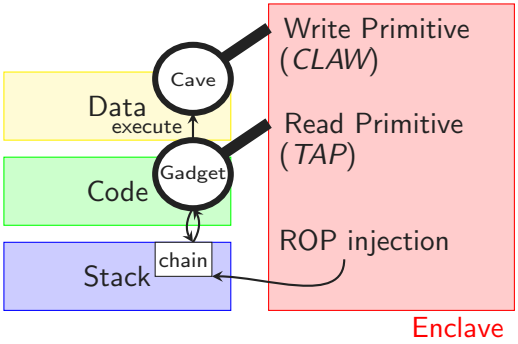

Data
Code
Gadget
Stack
Read Primitive
(TAP)
Enclave


Write Primitive
(*CLAW*)
Cave
Data
Read Primitive
(*TAP*)
Gadget
Code
Stack
Enclave


Write Primitive
(*CLAW*)
Cave
Data
Read Primitive
(*TAP*)
Gadget
Code
ROP injection
chain
Stack
Enclave


Write Primitive
(CLAW)
Cave
Data
Read Primitive
(TAP)
Gadget
Code
ROP injection
chain
Stack
Enclave


Write Primitive
(CLAW)
Cave
Data
execute
Read Primitive
(TAP)
Gadget
Code
ROP injection
chain
Stack
Enclave

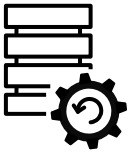

Write Primitive
(CLAW)
Cave
Data
execute
Read Primitive
(TAP)
Gadget
Code
ROP injection
chain
Stack
Enclave

- Enclave can access host memory...

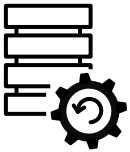
- Enclave can access host memory...
- ...but crashes on invalid access

Problems

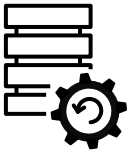
- Enclave can access host memory...
- ...but crashes on invalid access
- No syscall or exception handler available



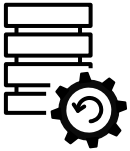
- Intel TSX: hardware transactional memory



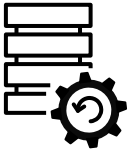
- Intel TSX: hardware transactional memory
- Multiple reads and writes are atomic



- Intel TSX: hardware transactional memory
- Multiple reads and writes are atomic
- Operations in a transaction



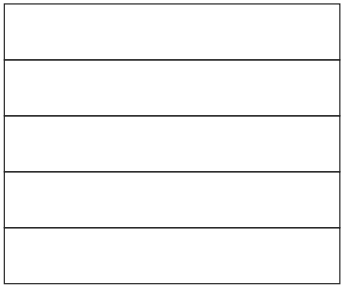
- Intel TSX: hardware transactional memory
- Multiple reads and writes are atomic
- Operations in a transaction
- Conflict → abort and roll back



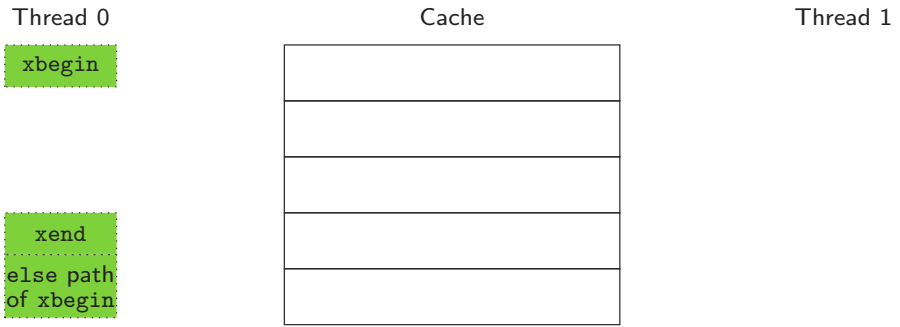
- Intel TSX: hardware transactional memory
- Multiple reads and writes are atomic
- Operations in a transaction
- Conflict $\rightarrow$ abort and roll back
- Faults are suppressed

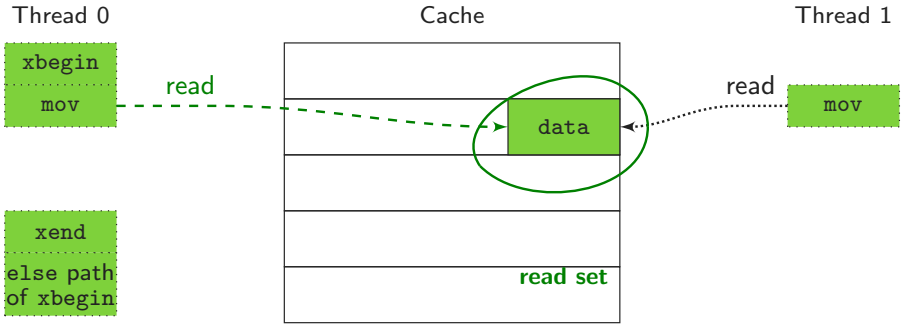
Thread 0

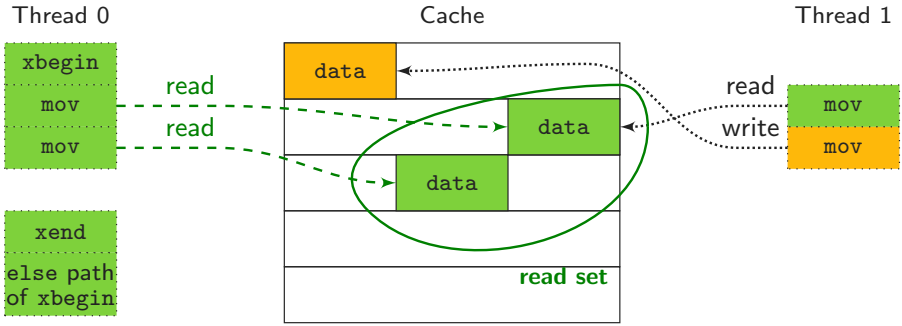
Cache

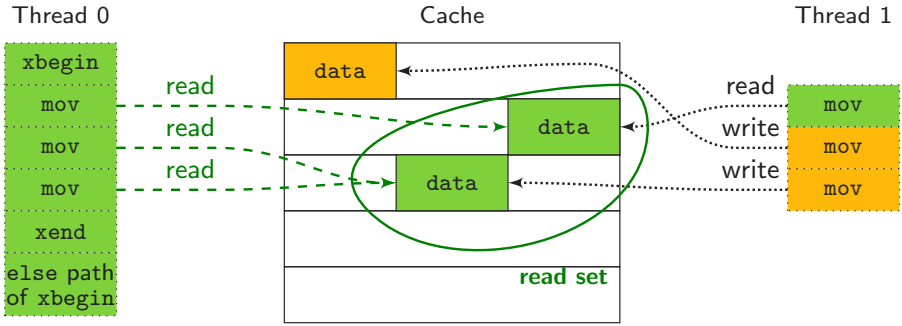


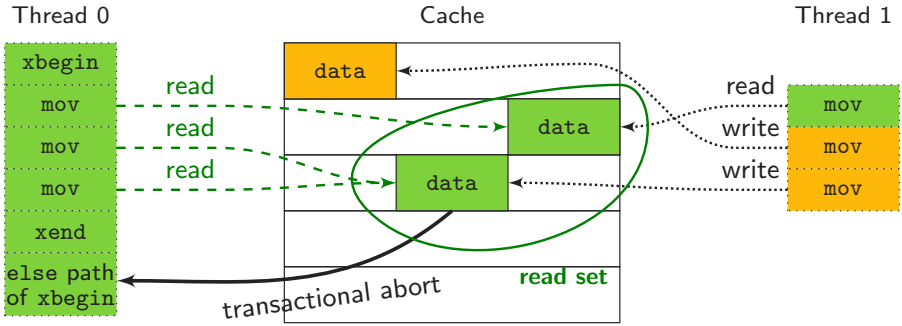
Thread 1


Thread 0
Cache
Thread 1
xbegin
xend
else path
of xbegin


Thread 0
Cache
Thread 1
xbegin
mov
read
read
mov
data
xend
else path
of xbegin
read set


Thread 0
Cache
Thread 1
xbegin
mov
mov
read
read
data
data
data
read
write
mov
mov
xend
else path
of xbegin
read set


Thread 0
Cache
Thread 1
xbegin
mov
mov
mov
xend
else path of xbegin
read
read
read
data
data
data
read set
read
write
write
mov
mov
mov


Thread 0
Cache
Thread 1
xbegin
mov
mov
mov
xend
else path of xbegin
read
read
read
data
data
data
read
write
write
mov
mov
mov
transactional abort
read set

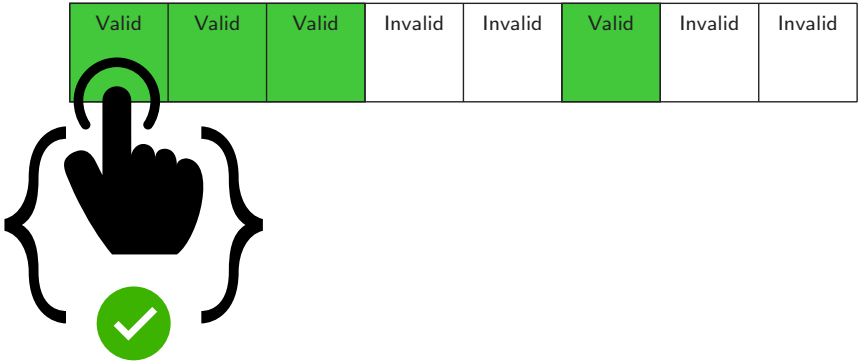
- Segmentation fault is a fault

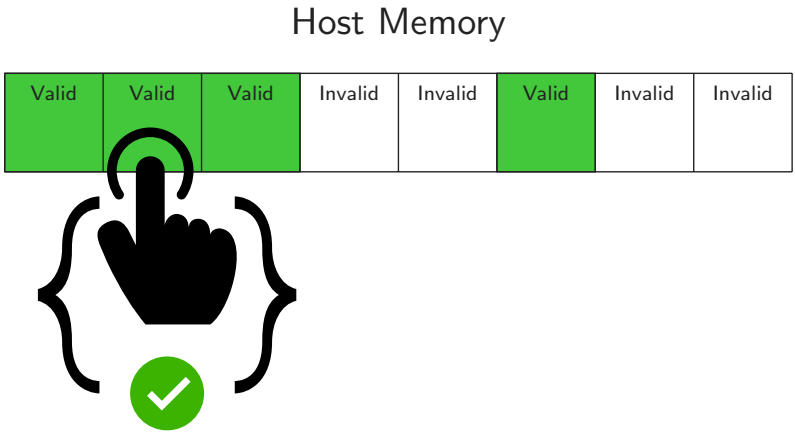
- Segmentation fault is a fault
- Suppressed in TSX transaction

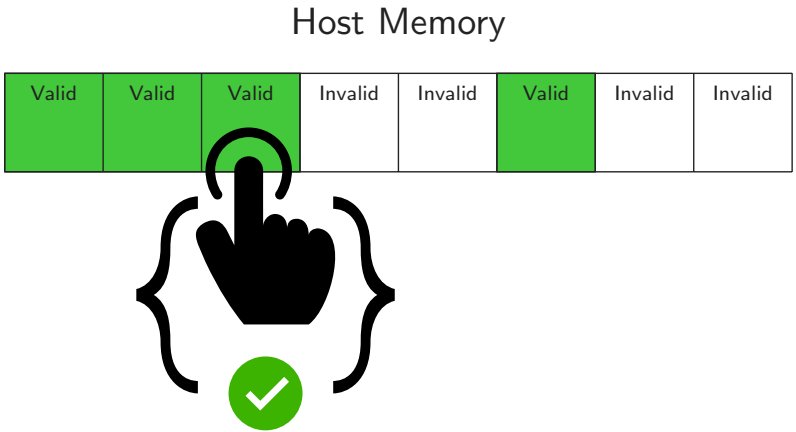
- Segmentation fault is a fault
- Suppressed in TSX transaction
- Abort code → "don't try again"

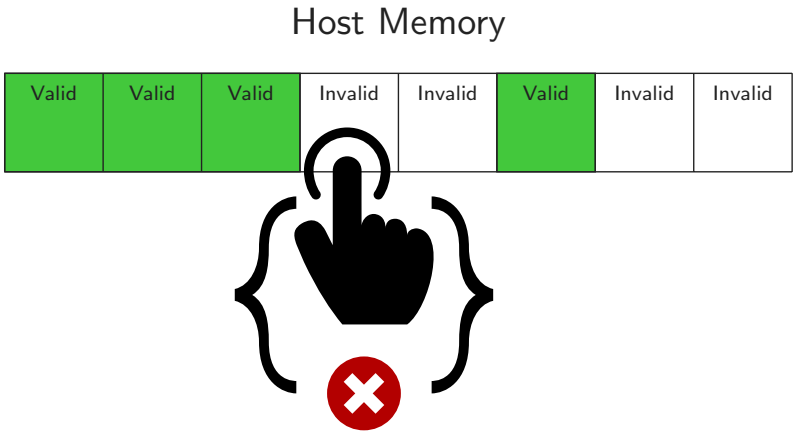
- Segmentation fault is a fault
- Suppressed in TSX transaction
- Abort code → "don't try again"
- Valid page → transaction succeeds

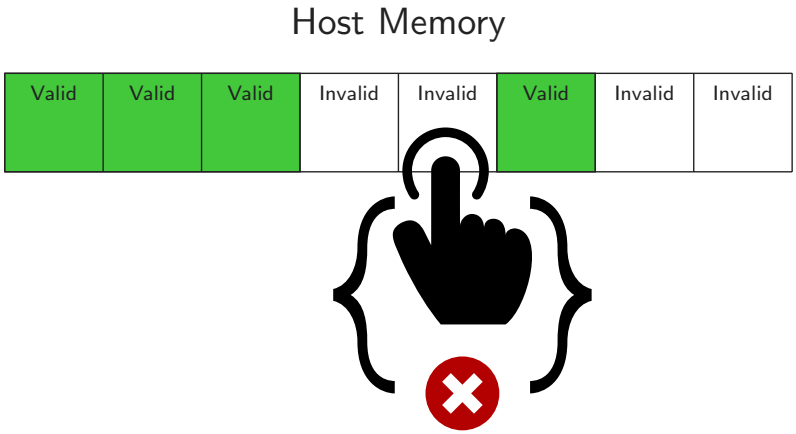
Host Memory

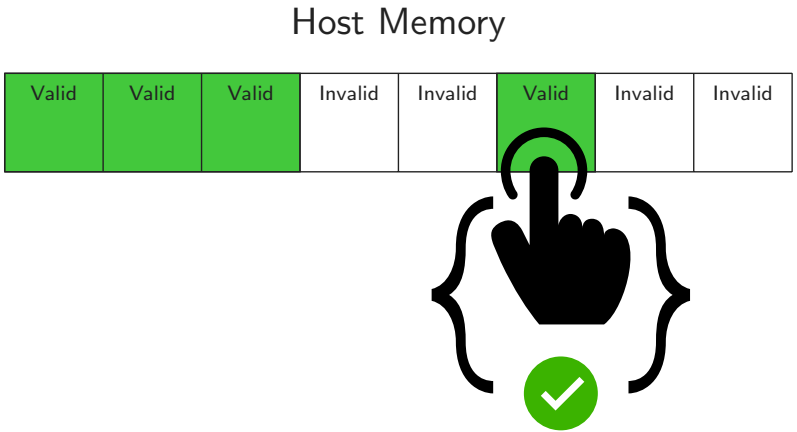



Host Memory
Valid
Valid
Valid
Invalid
Invalid
Valid
Invalid
Invalid

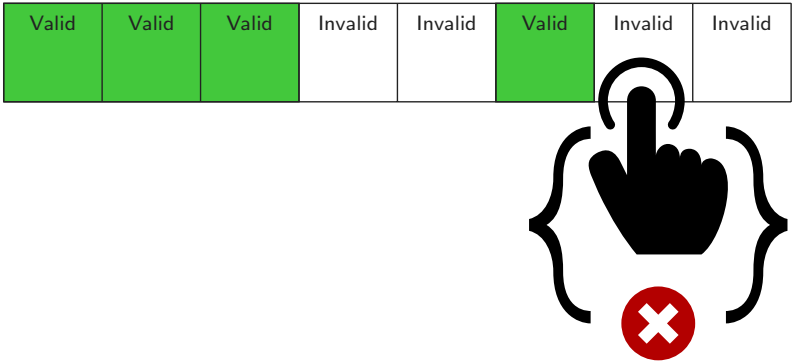

Host Memory
Valid
Valid
Valid
Invalid
Invalid
Valid
Invalid
Invalid


Host Memory
Valid
Valid
Valid
Invalid
Invalid
Valid
Invalid
Invalid

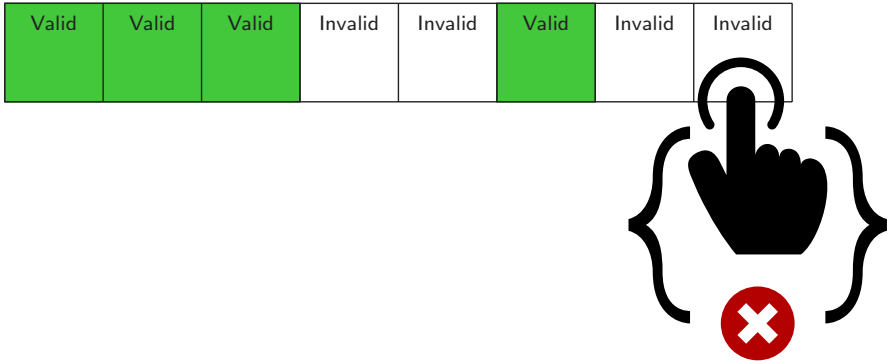

Host Memory
Valid
Valid
Valid
Invalid
Invalid
Valid
Invalid
Invalid


Host Memory
Valid
Valid
Valid
Invalid
Invalid
Valid
Invalid
Invalid

Host Memory



Host Memory



- Entire memory: 45 min

- Entire memory: 45 min
- Start from saved RIP/RSP: few seconds

- Entire memory: 45 min
- Start from saved RIP/RSP: few seconds
- Undetectable by OS

- Entire memory: 45 min
- Start from saved RIP/RSP: few seconds
- Undetectable by OS
- Used to find ROP gadgets

- Write to mapped page...

- Write to mapped page...
- ...abort immediately

- Write to mapped page...
- ...abort immediately

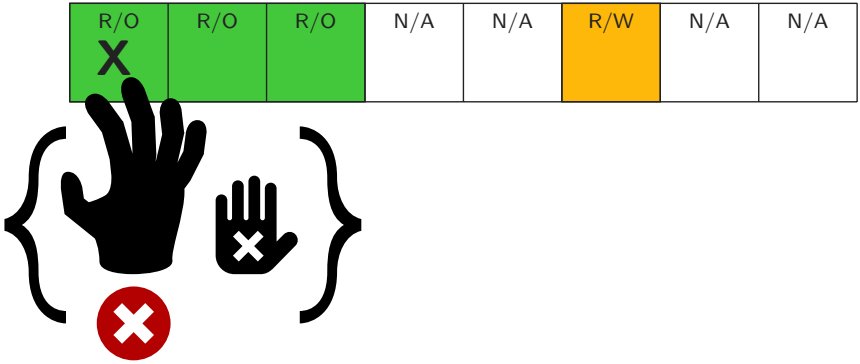
$\rightarrow$ No architectural write

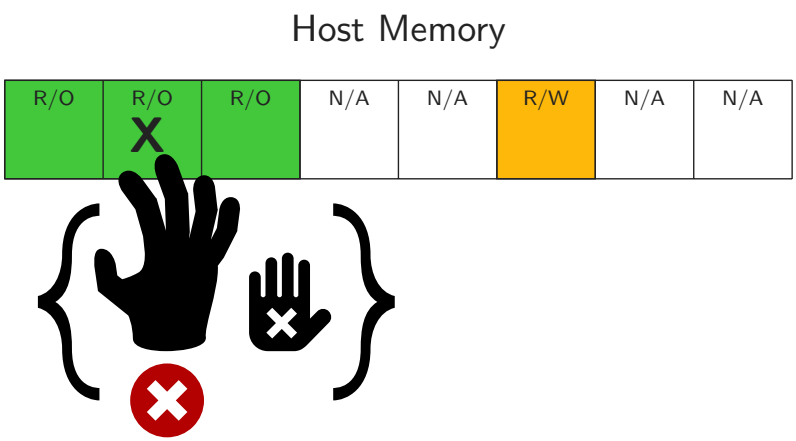
- Write to mapped page...
- ...abort immediately

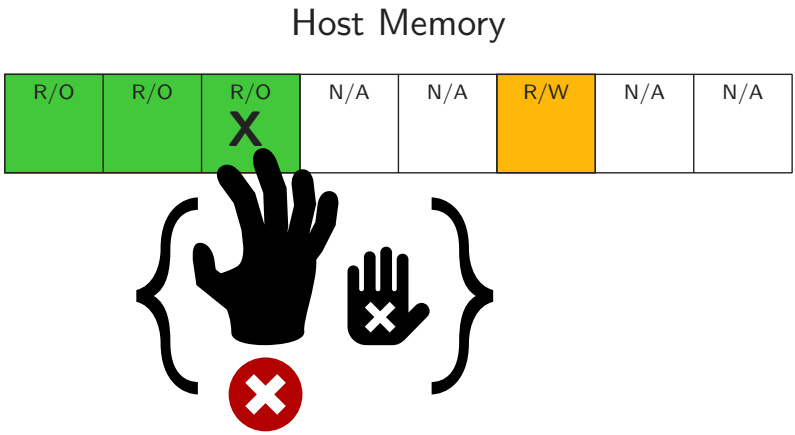
→ No architectural write

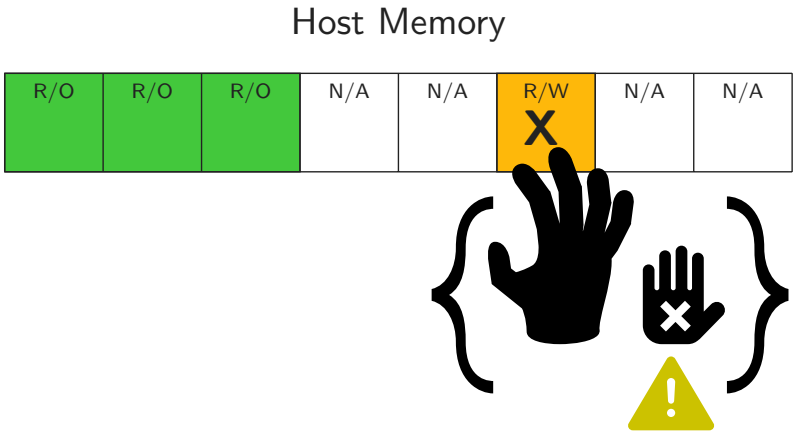
- Abort code → explicit or implicit

Host Memory




Host Memory
R/O
R/O
X
R/O
N/A
N/A
R/W
N/A
N/A


Host Memory
R/O
R/O
R/O
X
N/A
N/A
R/W
N/A
N/A


Host Memory
R/O
R/O
R/O
N/A
N/A
R/W
X
N/A
N/A

- TAP+CLAW $\rightarrow$ find writable memory

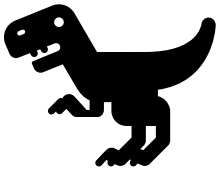
- TAP+CLAW → find writable memory

→ Robust write-anything-anywhere primitive

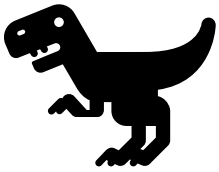
- TAP+CLAW → find writable memory

→ Robust write-anything-anywhere primitive

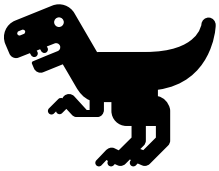
→ Store malicious payload



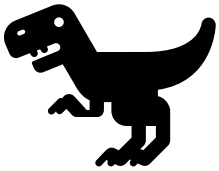
1. TAP: find ROP gadgets



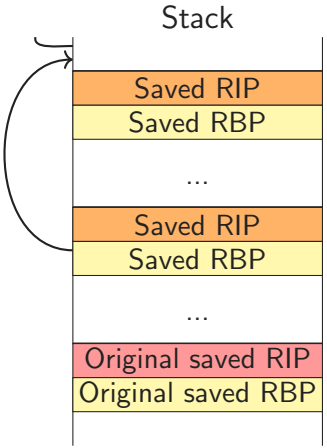
1. TAP: find ROP gadgets
2. CLAW: find writable memory (data cave)



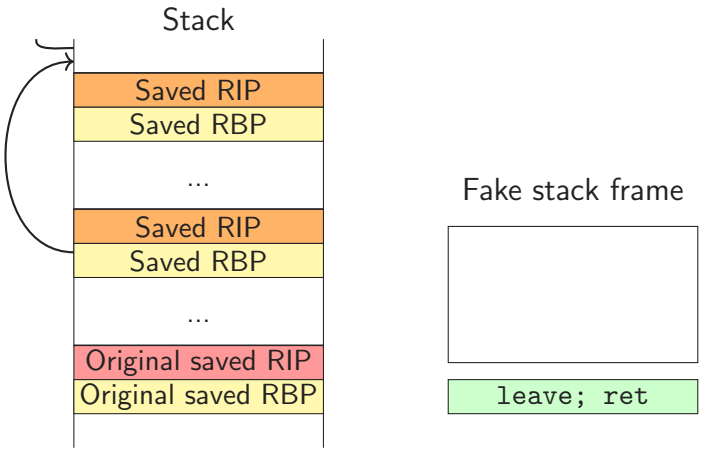
1. TAP: find ROP gadgets
2. CLAW: find writable memory (data cave)
3. Inject ROP gadgets into host stack

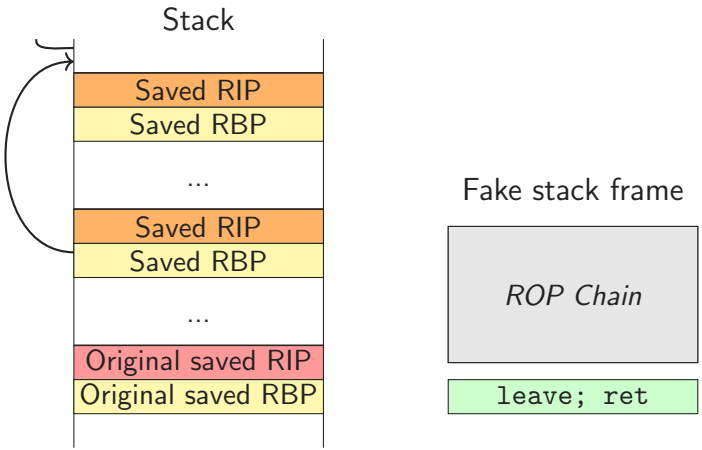


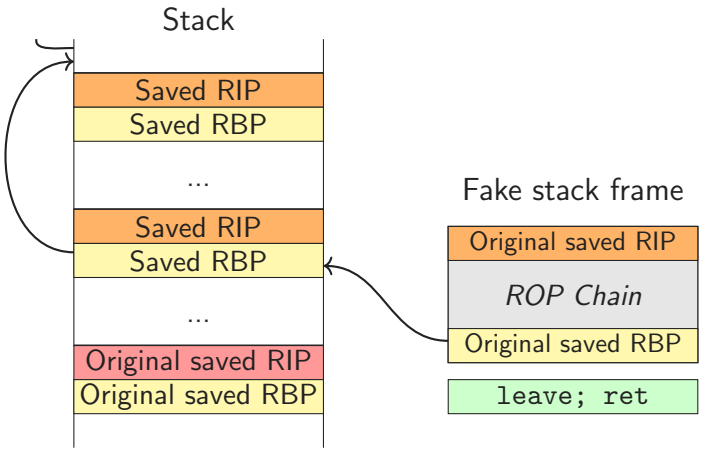
1. TAP: find ROP gadgets
2. CLAW: find writable memory (data cave)
3. Inject ROP gadgets into host stack
4. Profit!

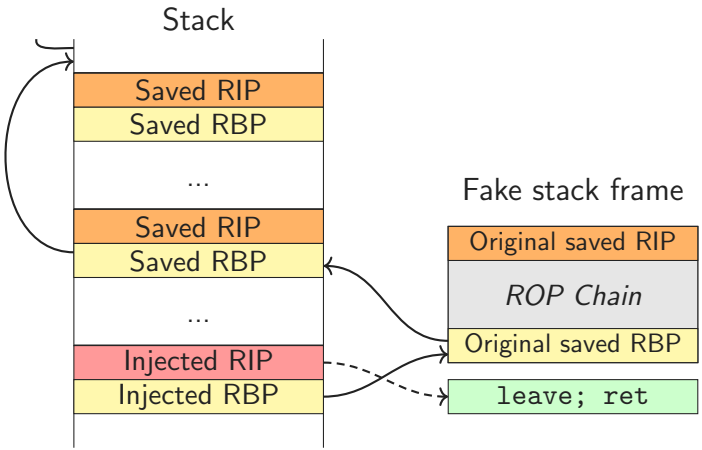

Stack
Saved RIP
Saved RBP
...
Saved RIP
Saved RBP
...
Original saved RIP
Original saved RBP

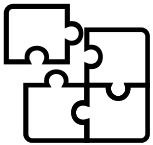
```
leave; ret
```


Stack
Saved RIP
Saved RBP
...
Saved RIP
Saved RBP
...
Original saved RIP
Original saved RBP
Fake stack frame
leave; ret


Stack
Saved RIP
Saved RBP
...
Saved RIP
Saved RBP
...
Original saved RIP
Original saved RBP
Fake stack frame
ROP Chain
leave; ret

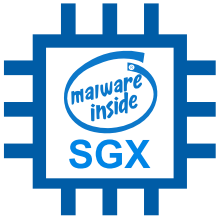

Stack
Saved RIP
Saved RBP
...
Saved RIP
Saved RBP
...
Original saved RIP
Original saved RBP
Fake stack frame
Original saved RIP
ROP Chain
Original saved RBP
leave; ret


Stack
Saved RIP
Saved RBP
...
Saved RIP
Saved RBP
...
Injected RIP
Injected RBP
Fake stack frame
Original saved RIP
ROP Chain
Original saved RBP
leave; ret



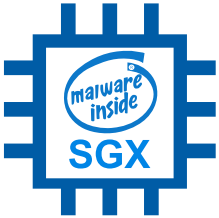
64.8 MB writable data
`mprotect` ROP gadgets

Several pages writable data
`mprotect` ROP gadgets

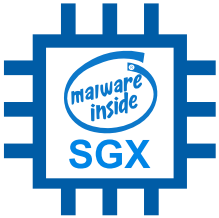


- Remote attestation + dynamic loading → no emulation, no binary

Full Exploit

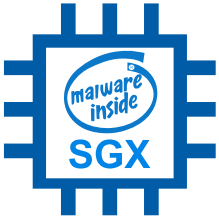


- Remote attestation + dynamic loading → no emulation, no binary
- Host continues normally → (nearly) no traces



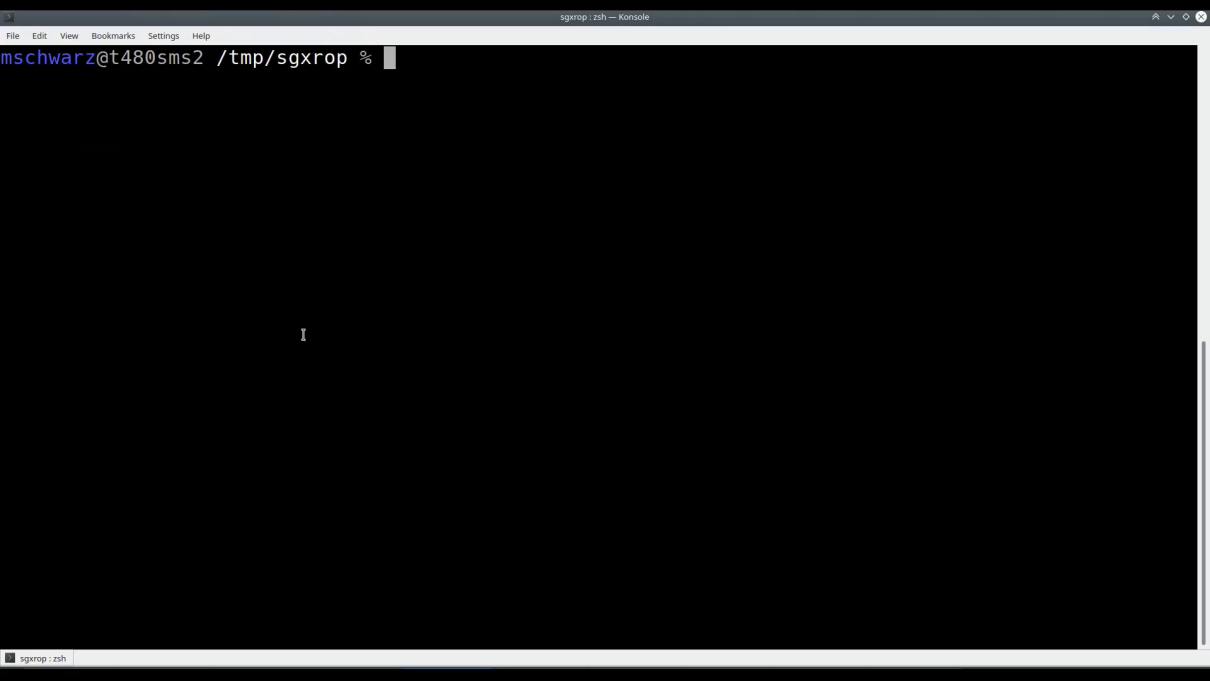
- Remote attestation + dynamic loading → no emulation, no binary
- Host continues normally → (nearly) no traces
- Trigger-based → plausible deniability

Full Exploit

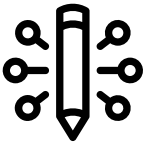


- Remote attestation + dynamic loading → no emulation, no binary
- Host continues normally → (nearly) no traces
- Trigger-based → plausible deniability

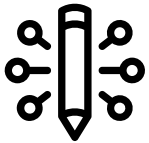
→ Securely and stealthily deploying zero days


sgxrop : zsh — Konsole
File Edit View Bookmarks Settings Help
mschwarz@t480sms2 /tmp/sgxrop %
sgxrop : zsh

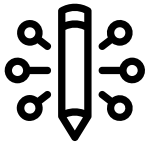
https://github.com/IAIK/SGXROP



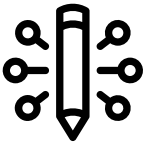
- Asymmetric threat model



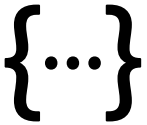
- Asymmetric threat model
- Enclaves assumed always benign



- Asymmetric threat model
- Enclaves assumed always benign
- Not realistic in most scenarios



- Asymmetric threat model
- Enclaves assumed always benign
- Not realistic in most scenarios
- Full memory access avoidable → reduce attack surface



Takeaways

- Asymmetric threat model in SGX fosters malware
- SGX hides and protects malware
- Easy to port existing malware to SGX ROP

Thank you!

SGX - Secure Enclaves als Angriffsvektor

Michael Schwarz (@misc0110), Samuel Weiser, Daniel Gruss

October 1, 2019 - IKT-Sicherheitskonferenz

Graz University of Technology

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

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

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