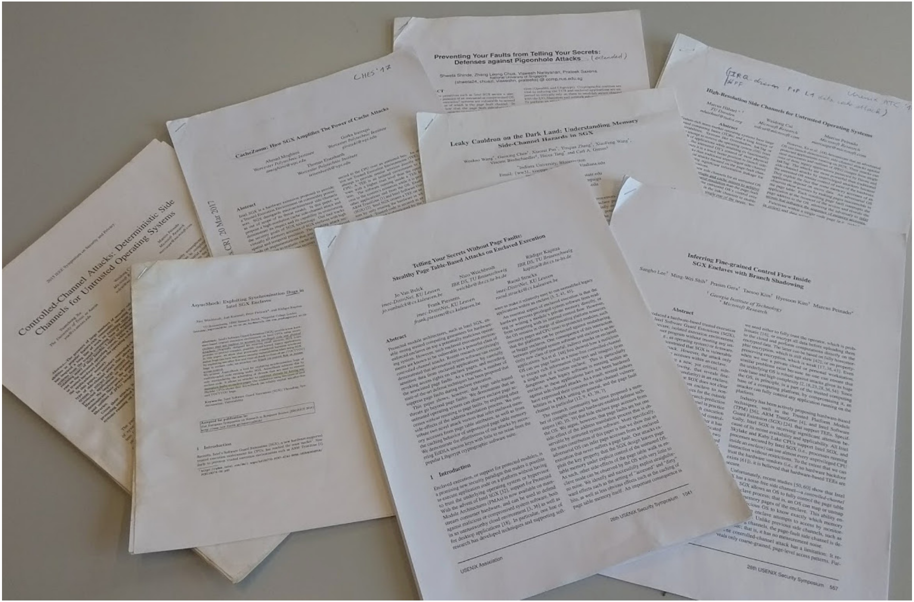
SGX-Step: A Practical Attack Framework for Precise Enclave Execution Control

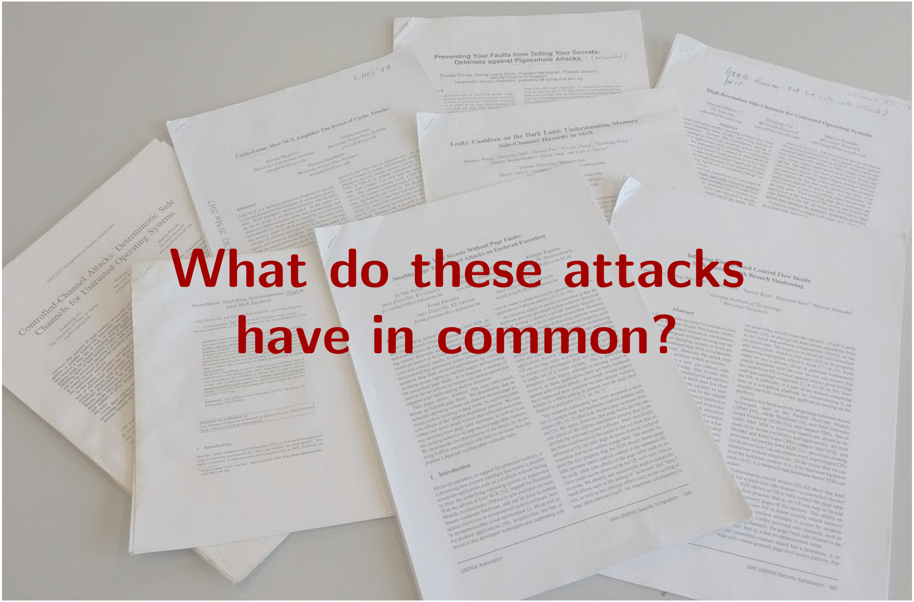
Jo Van Bulck Frank Piessens Raoul Strackx

imec-DistriNet, KU Leuven

SysTEX'17, October 28, 2017

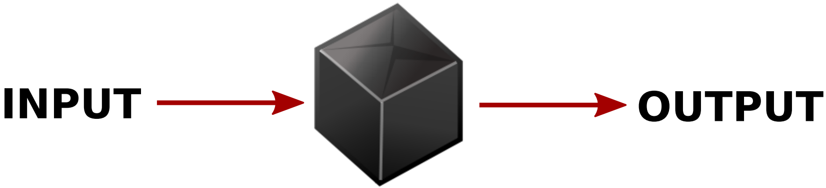
Road Map


CHES '17
Preventing Your Faults from Telling Your Secrets: Defenses against Pigeonhole Attacks
CacheZoom: How SGX Amplifies The Power of Cache Attacks
Leaky Cauldron on the Dark Land: Understanding Memory Side-Channel Hazards in SGX
High-Resolution Side Channels for Untrusted Operating Systems
Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems
AsyncShock: Exploiting Synchronization Bugs in Intel SGX Enclaves
Telling Your Secrets Without Page Faults: Stealthy Page Table-Based Attacks on Enclaved Execution
Inferring Fine-grained Control Flow Inside SGX Enclaves with Branch Shadowing

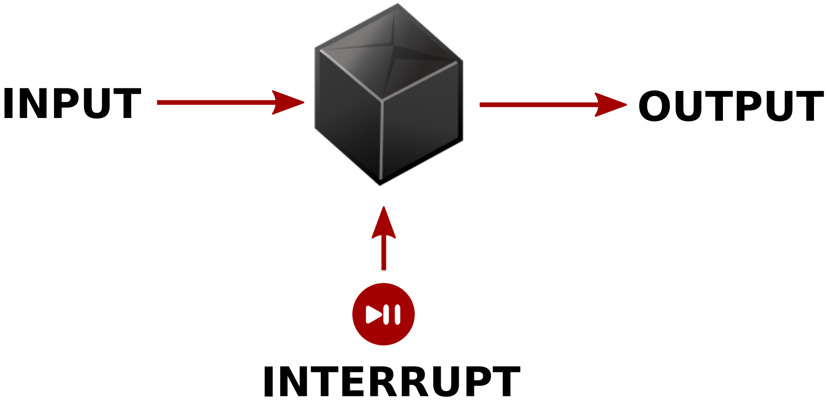


What do these attacks have in common?

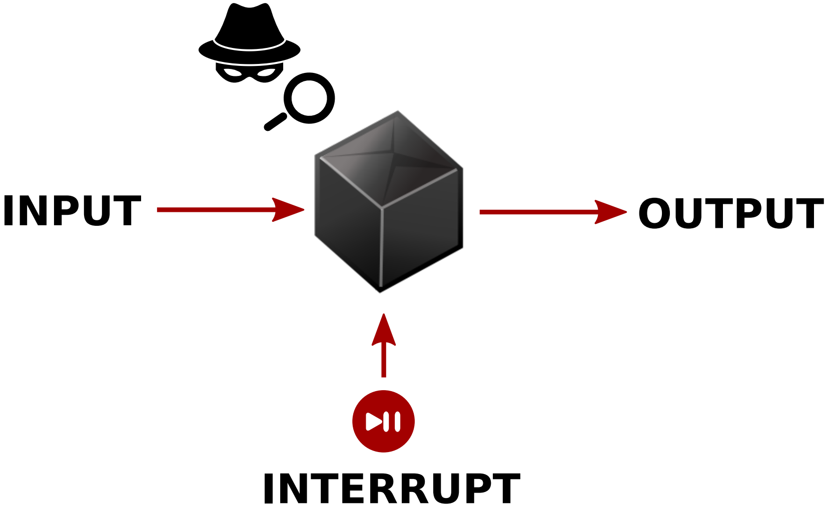
Enclaves as a Black Box



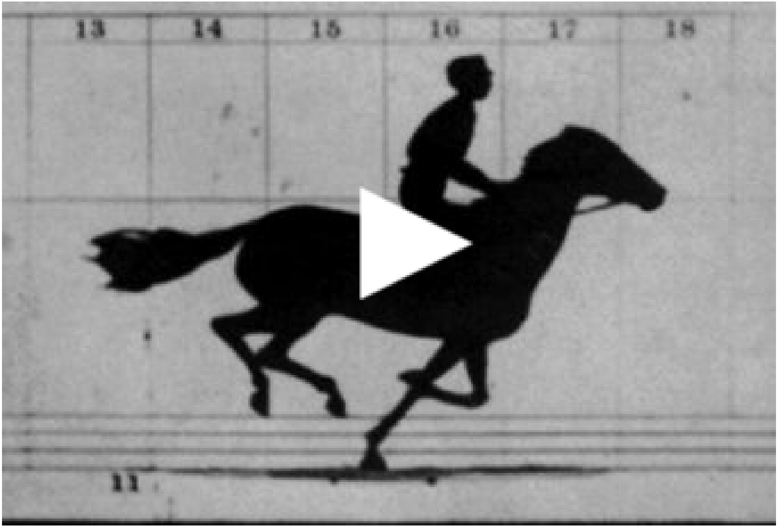
Enclaves as a Black Box



Enclaves as a Black Box

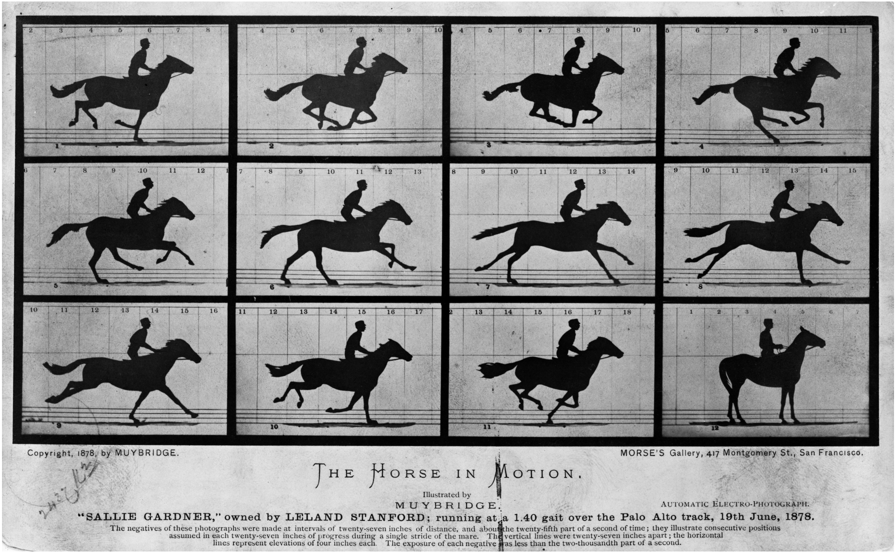


The Galloping Enclave Analogy

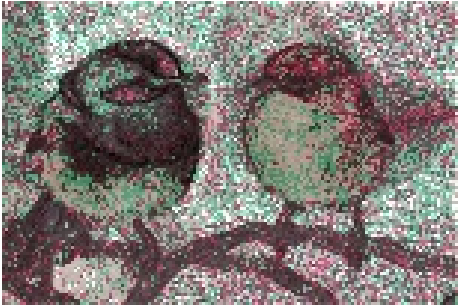


Source: https://en.wikipedia.org/wiki/Sallie_Gardner_at_a_Gallop

The Galloping Enclave Analogy



High Resolution Side-Channels in Practice



Xu et al.: "Controlled-channel attacks: Deterministic side channels for untrusted operating systems", Oakland 2015 [XCP15]

⇒ *Coarse-grained preemption (4 KB page leakage)*

High Resolution Side-Channels in Practice

Hähnel et al.: "High-resolution side channels for untrusted operating systems", ATC 2017 [HCP17]

⇒ *Fine-grained preemption (64 B cache line leakage)*

Road Map

Timer-Based Attacks

Goal: interrupt each enclave instruction sequentially

Timer-Based Attacks

Goal: interrupt each enclave instruction sequentially

Frequent enclave preemption challenge:

- ☹ x86 HW *debug features* disabled in enclave mode
- ☺ . . . but full control over **timer devices**/scheduling

Timer-Based Attacks

Goal: interrupt each enclave instruction sequentially

Frequent enclave preemption challenge:

- ☹ x86 HW *debug features* disabled in enclave mode
- ☺ ... but full control over **timer devices**/scheduling

Timer interval prediction challenge:

- ☹ considerable *jitter* when configuring timer in kernel space

> We also counted how many CPU instructions can be executed between such frequent timer interrupts by running a loop with an ADD instruction. On average, about 48.76 ADD instructions were executed between two timer interrupts (standard deviation: 2.75)[1]. This implies that, by using this frequent timer, we can

Inferring fine-grained control flow inside SGX enclaves with branch shadowing [LSG+17a]

Timer-Based Attacks

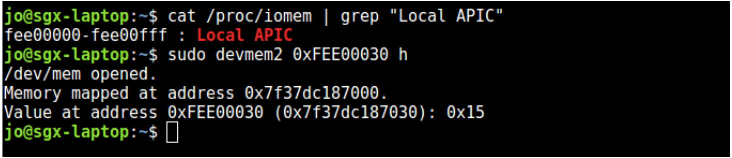
Goal: interrupt each enclave instruction sequentially

Frequent enclave preemption challenge:

- ☹ x86 HW *debug features* disabled in enclave mode
- ☺ ... but full control over **timer devices**/scheduling

Timer interval prediction challenge:

- ☹ considerable *jitter* when configuring timer in kernel space
- ☺ ... but APIC can be **memory-mapped** in user space (!)

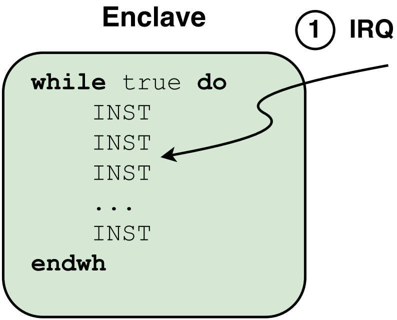


Interrupting and Resuming Enclaves

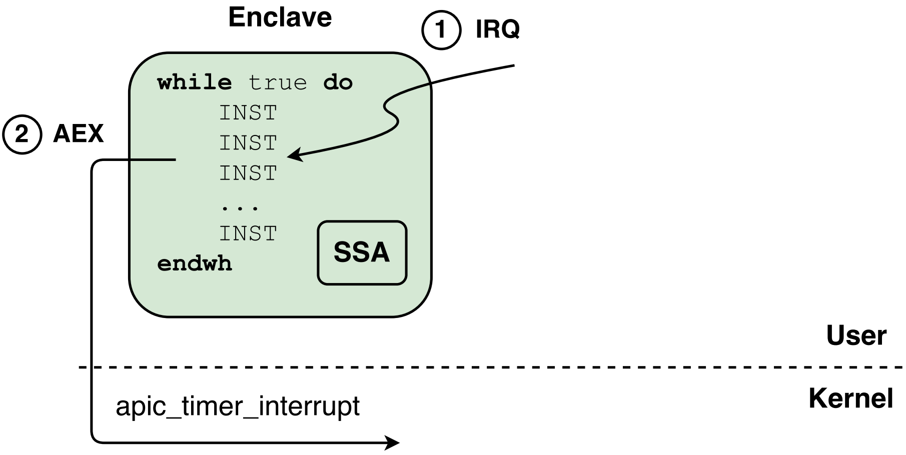
Enclave

```
while true do
    INST
    INST
    INST
    ...
    INST
endwh
```

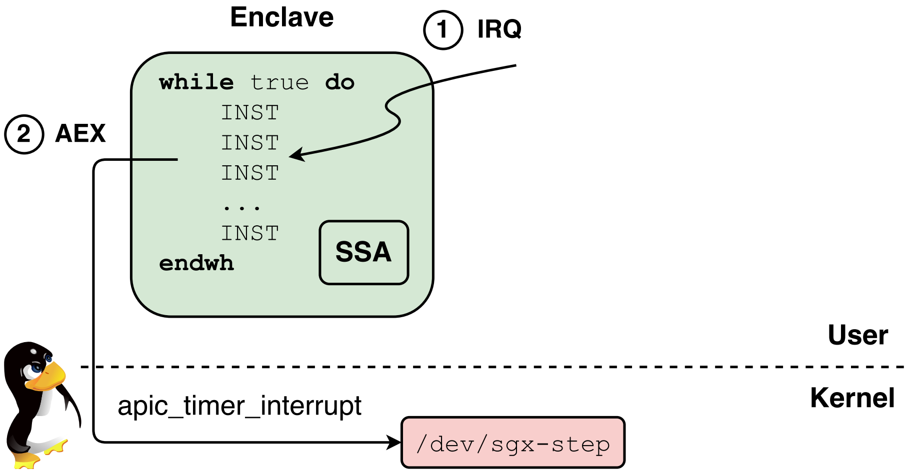
Interrupting and Resuming Enclaves



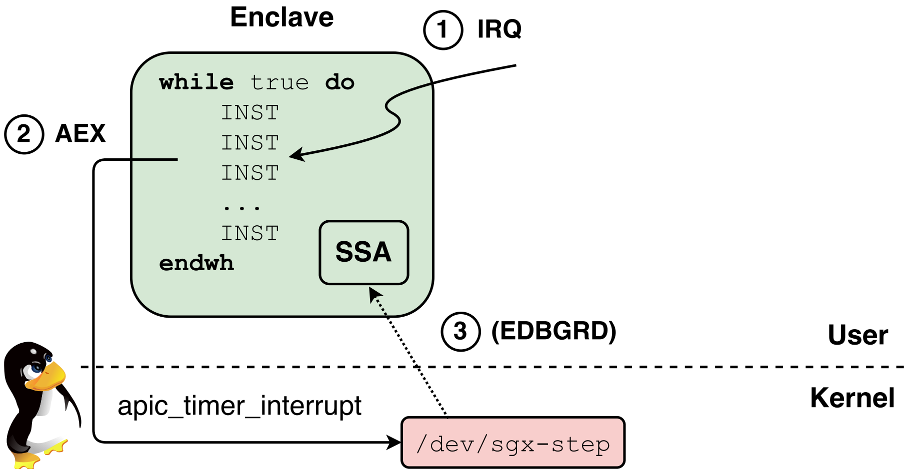
Interrupting and Resuming Enclaves



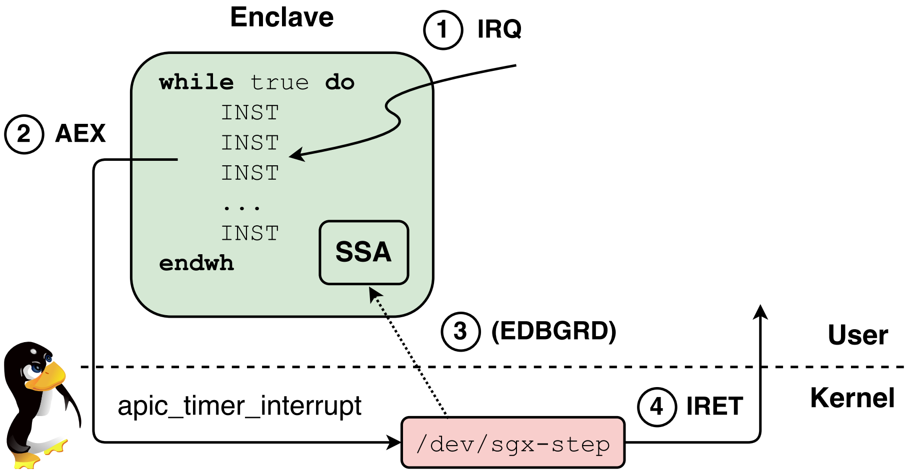
Interrupting and Resuming Enclaves



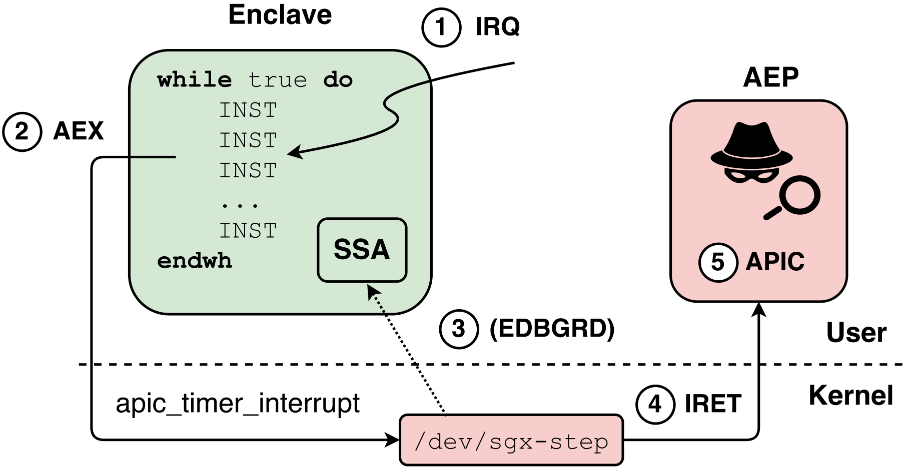
Interrupting and Resuming Enclaves



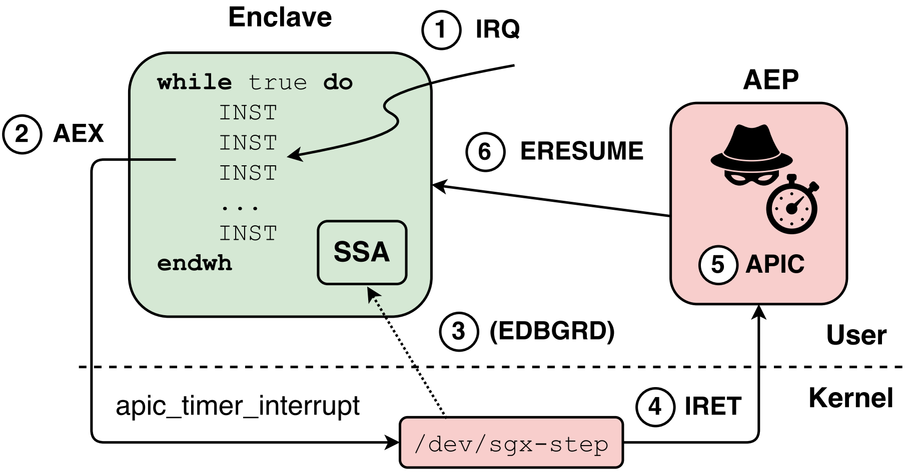
Interrupting and Resuming Enclaves



Interrupting and Resuming Enclaves



Interrupting and Resuming Enclaves



Road Map

Determining String Length

`strlen` loop

Note: page fault-driven attacks cannot make progress

```c
size_t strlen (char *str)
{
  char *s;

  for (s = str; *s; ++s);
  return (s - str);
}
```

```asm
    mov  %rdi,%rax
1:  cmpb $0x0,(%rax)
    je   2f
    inc  %rax
    jmp  1b
2:  sub  %rdi,%rax
    retq
```

Determining String Length

`strlen` loop

Note: page fault-driven attacks cannot make progress

```c
size_t strlen (char *str)
{
  char *s;

  for (s = str; *s; ++s);
  return (s - str);
}
```

```
    mov   %rdi,%rax
1:  cmpb  $0x0,(%rax)
    je    2f
    inc   %rax
    jmp   1b
2:  sub   %rdi,%rax
    retq
```

⇒ tight loop: 4 instructions, single memory operand

Determining String Length

Protection from Side-Channel Attacks

Intel® SGX does not provide explicit protection from side-channel attacks. It is the enclave developer's responsibility to address side-channel attack concerns.

In general, enclave operations that require an OCall, such as thread synchronization, I/O, etc., are exposed to the untrusted domain. If using an OCall would allow an attacker to gain insight into enclave secrets, then there would be a security concern. This scenario would be classified as a side-channel attack, and it would be up to the ISV to design the enclave in a way that prevents the leaking of side-channel information.

An attacker with access to the platform can see what pages are being executed or accessed. This side-channel vulnerability can be mitigated by aligning specific code and data blocks to exist entirely within a single page.

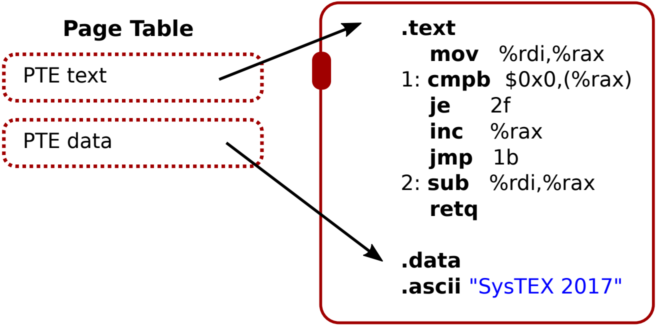
More important, the application enclave should use an appropriate crypto implementation that is side channel attack resistant inside the enclave if side-channel attacks are a concern.

Source: `https://software.intel.com/en-us/node/703016`

Attacking `strlen`

Page fault adversary

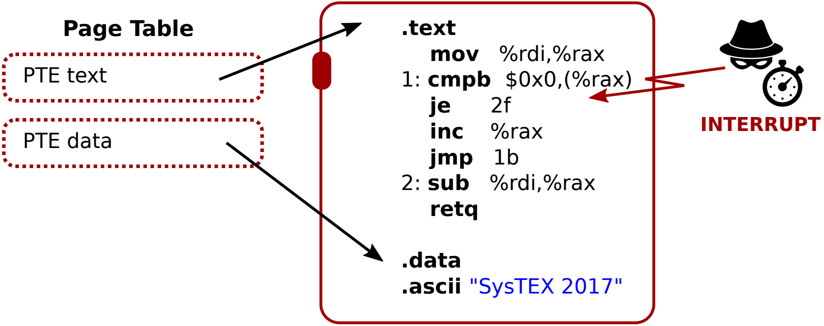
Progress ⇒ both code + data pages present ☹



Attacking `strlen`

Single-stepping adversary

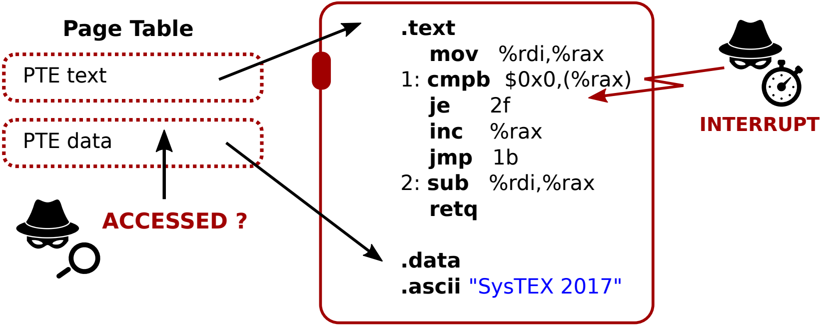
Execute + interrupt ⇒ data page accessed ? ☺



Attacking `strlen`

Single-stepping adversary

Execute + interrupt ⇒ data page accessed ? ☺



Van Bulck et al. "Telling your secrets without page faults: Stealthy page table-based attacks on enclaved execution", USENIX 2017 [VBWK+17]

Defeating Zigzagger Branch Obfuscation

Conditional control flow

$\Rightarrow$ vulnerable to *interrupt-driven BTB probing* attacks [LSG+17a]

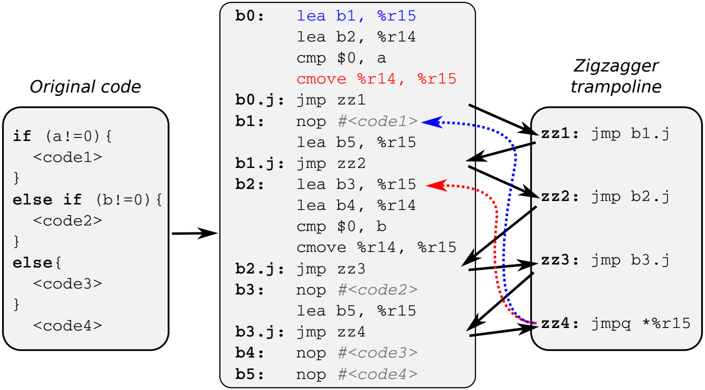
Original code

```
if (a!=0){
  <code1>
}
else if (b!=0){
  <code2>
}
else{
  <code3>
}
  <code4>
```

Defeating Zigzagger Branch Obfuscation

Zigzagger compile-time hardening

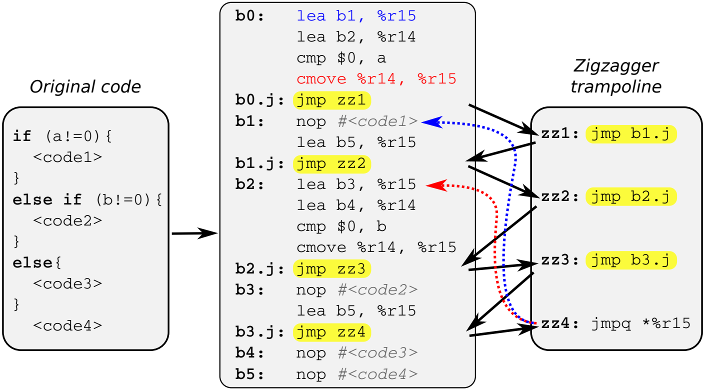
⇒ obfuscate target with `cmov` + tight `jmp` sequence



Defeating Zigzagger Branch Obfuscation

Zigzagger security argument

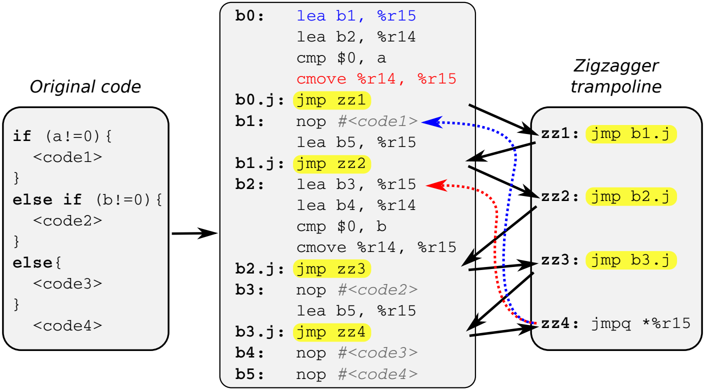
"our APIC *timer trick is not fine-grained enough* to distinguish each branches in practice" [LSG+17b]



Defeating Zigzagger Branch Obfuscation

Zigzagger security argument

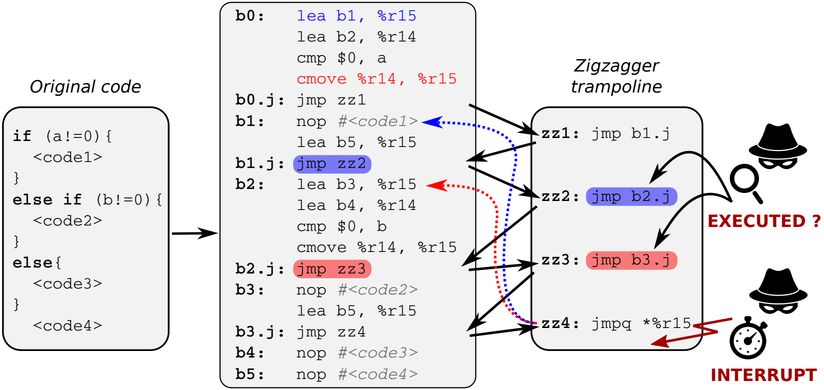
. . . but SGX-Step breaks at *every* single instruction (!)



Defeating Zigzagger Branch Obfuscation

Zigzagger security argument

. . . but SGX-Step breaks at *every* single instruction (!)



Single-Stepping Microbenchmarks

Table: Interrupts categorized according to the number of instructions executed in the victim enclave (i.e., zero-step, single-step, or multi-step). When laptop/desktop experimental results differ, we present the laptop measurements first.

Experiment	**0-Step**	**1-Step**	**$>$1**	**1-Step Ratio**
`nop`	2,083 / 1,617	100,000	0	97.96 / 98.41%
strlen	8,829 / 4,982	460,000	0	98.12 / 98.93%
Zigzagger	5,739 / 2,872	210,000	0	97.34 / 98.65%

Single-Stepping Microbenchmarks

Accuracy/temporal resolution

$\Rightarrow$ *never* execute $+1$ instruction per step

Table: Interrupts categorized according to the number of instructions executed in the victim enclave (i.e., zero-step, single-step, or multi-step). When laptop/desktop experimental results differ, we present the laptop measurements first.

Experiment	**0-Step**	**1-Step**	**$>$1**	**1-Step Ratio**
`nop`	2,083 / 1,617	100,000	0	97.96 / 98.41%
strlen	8,829 / 4,982	460,000	0	98.12 / 98.93%
Zigzagger	5,739 / 2,872	210,000	0	97.34 / 98.65%

Single-Stepping Microbenchmarks

Practicality

⇒ Low superfluous (zero-step) IRQ rate

Table: Interrupts categorized according to the number of instructions executed in the victim enclave (i.e., zero-step, single-step, or multi-step). When laptop/desktop experimental results differ, we present the laptop measurements first.

Experiment	0-Step	1-Step	>1	1-Step Ratio
`nop`	2,083 / 1,617	100,000	0	97.96 / 98.41%
strlen	8,829 / 4,982	460,000	0	98.12 / 98.93%
Zigzagger	5,739 / 2,872	210,000	0	97.34 / 98.65%

Road Map

Conclusion

Take-Away message

Enclaves can be executed *one* instruction at a time

Conclusion

Take-Away message

Enclaves can be executed *one* instruction at a time

⇒ **Practical** and **precise** attack framework

Conclusion

Take-Away message

Enclaves can be executed *one* instruction at a time

⇒ **Practical** and **precise** attack framework

⇒ Defeat defenses based on **atomic behavior**:

- *temporal* dimension: Zigzagger branch obfuscation [LSG+17a]
- *spatial* dimension: page-aligned code/data [Int17]

Thank you! Questions?

SGX-Step

`https://github.com/jovanbulck/sgx-step`

References

M. Hähnel, W. Cui, and M. Peinado.
High-resolution side channels for untrusted operating systems.
In *2017 USENIX Annual Technical Conference*, ATC '17. USENIX Association, 2017.

Intel Corporation.
Intel software guard extensions developer guide, June 2017.
software.intel.com/en-us/documentation/sgx-developer-guide.

S. Lee, M.-W. Shih, P. Gera, T. Kim, H. Kim, and M. Peinado.
Inferring fine-grained control flow inside SGX enclaves with branch shadowing.
In *Proceedings of the 26th USENIX Security Symposium*. USENIX Association, August 2017.

S. Lee, M.-W. Shih, P. Gera, T. Kim, H. Kim, and M. Peinado.
Inferring fine-grained control flow inside SGX enclaves with branch shadowing.
arXiv:1611.06952v3, June 2017.

J. Van Bulck, N. Weichbrodt, R. Kapitza, F. Piessens, and R. Strackx.
Telling your secrets without page faults: Stealthy page table-based attacks on enclaved execution.
In *Proceedings of the 26th USENIX Security Symposium*. USENIX Association, August 2017.

Y. Xu, W. Cui, and M. Peinado.
Controlled-channel attacks: Deterministic side channels for untrusted operating systems.
In *IEEE Symposium on Security and Privacy*, pp. 640–656. IEEE, 2015.