Jo Van Bulck, David Oswald, Eduard Marin,
Abdulla Aldoseri, Flavio Garcia, and Frank Piessens

— Present —

"FOSDEM '20 Famous Story"

A TALE OF TWO WORLDS

• ASSESSING THE • VULNERABILITY OF ENCLAVE SHIELDING RUNTIMES

- PRODUCED BY -
KU LEUVEN AND BIRMINGHAM
- UNIVERSITIES -

- Trusted computing **across the system stack:** hardware, compiler, OS, application
- Integrated **attack-defense** perspective and **open-source** prototypes

CPU vulnerability research
[VBMW+18, SLM+19, MOG+20]

SGX-Step framework
[VBPS17]

Sancus enclave processor
[NVBM+17]

Outline: How to besiege a fortress?

Idea: security is weakest at the input/output interface(!)

Outline: How to besiege a TEE enclave?

Tier	Vulnerability \ Runtime	SGX-SDK	OpenEnclave	Graphene	SGX-LKL	Rust-EDP	Asylo	Keystone	Sancus
Tier1 (ABI)	#1 Entry status flags sanitization	★	★	◐	●	◐	●	○	○
	#2 Entry stack pointer restore	○	○	★	●	○	○	○	★
	#3 Exit register leakage	○	○	○	★	○	○	○	○
Tier2 (API)	#4 Missing pointer range check	○	★	★	★	○	●	○	★
	#5 Null-terminated string handling	⯪	★	○	○	○	○	○	○
	#6 Integer overflow in range check	○	○	●	○	●	○	●	●
	#7 Incorrect pointer range check	○	○	●	○	○	●	○	●
	#8 Double fetch untrusted pointer	○	○	●	○	○	○	○	○
	#9 Ocall return value not checked	○	★	★	★	○	●	★	○
	#10 Uninitialized padding leakage	[LK17]	★	○	●	○	●	★	★

Summary: > 35 enclave interface sanitization vulnerabilities across 8 projects

Outline: How to besiege a TEE enclave?

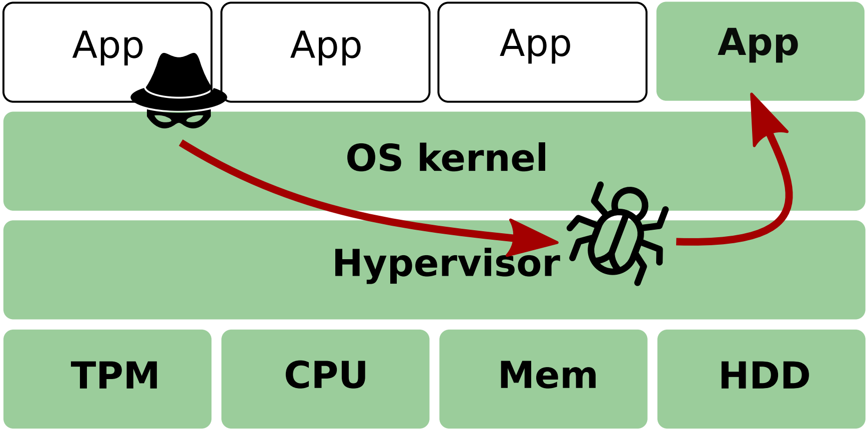
Vulnerability \ Runtime		SGX-SDK	OpenEnclave	Graphene	SGX-LKL	Rust-EDP	Asylo	Keystone	Sancus
Tier1 (ABI)	#1 Entry status flags sanitization	★	★	◐	●	◐	●	○	○
	#2 Entry stack pointer restore	○	○	★	●	○	○	○	★
	#3 Exit register leakage	○	○	○	★	○	○	○	○
Tier2 (API)	#4 Missing pointer range check	[illegible]	★	★	★	○	●	○	★
	#5 Null-terminated string handling	★	★	○	[illegible]	○	○	○	○
	#6 Integer overflow in range check	○	○	●	○	●	○	●	●
	#7 Incorrect pointer range check	○	○	●	○	○	●	○	●
	#8 Double fetch untrusted pointer	○	○	●	○	○	○	○	○
	#9 Ocall return value not checked	○	★	★	★	○	●	★	○
	#10 Uninitialized padding leakage	[LK17]	★	○	●	○	●	★	★

Impact: 5 CVEs . . . and lengthy embargo periods

¯_(ツ)_/¯

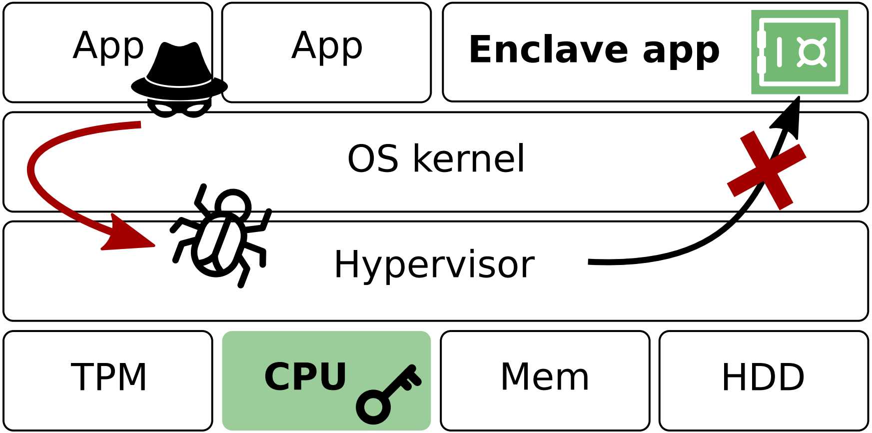
Why do we need enclave fortresses anyway?

The big picture: Enclaved execution attack surface



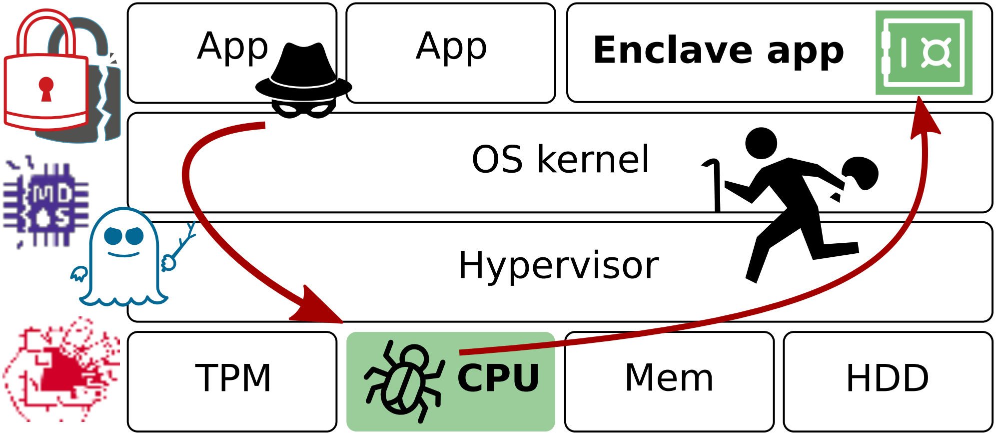
Traditional **layered designs:** large trusted computing base

The big picture: Enclaved execution attack surface



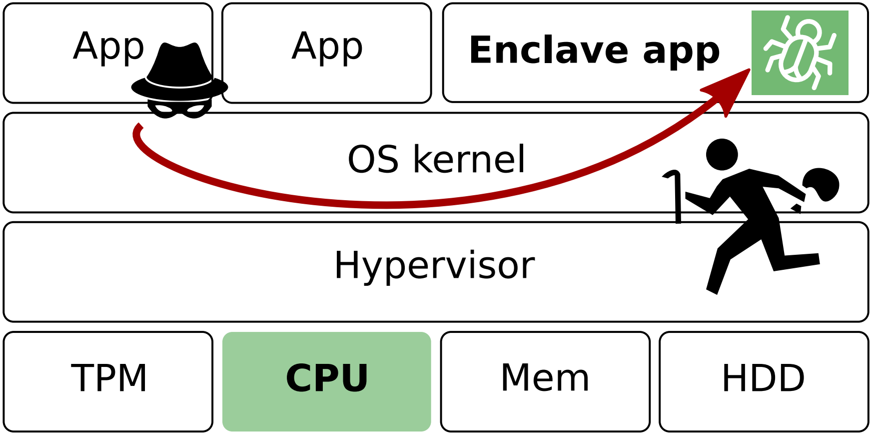
Intel SGX promise: hardware-level **isolation and attestation**

The big picture: Enclaved execution attack surface

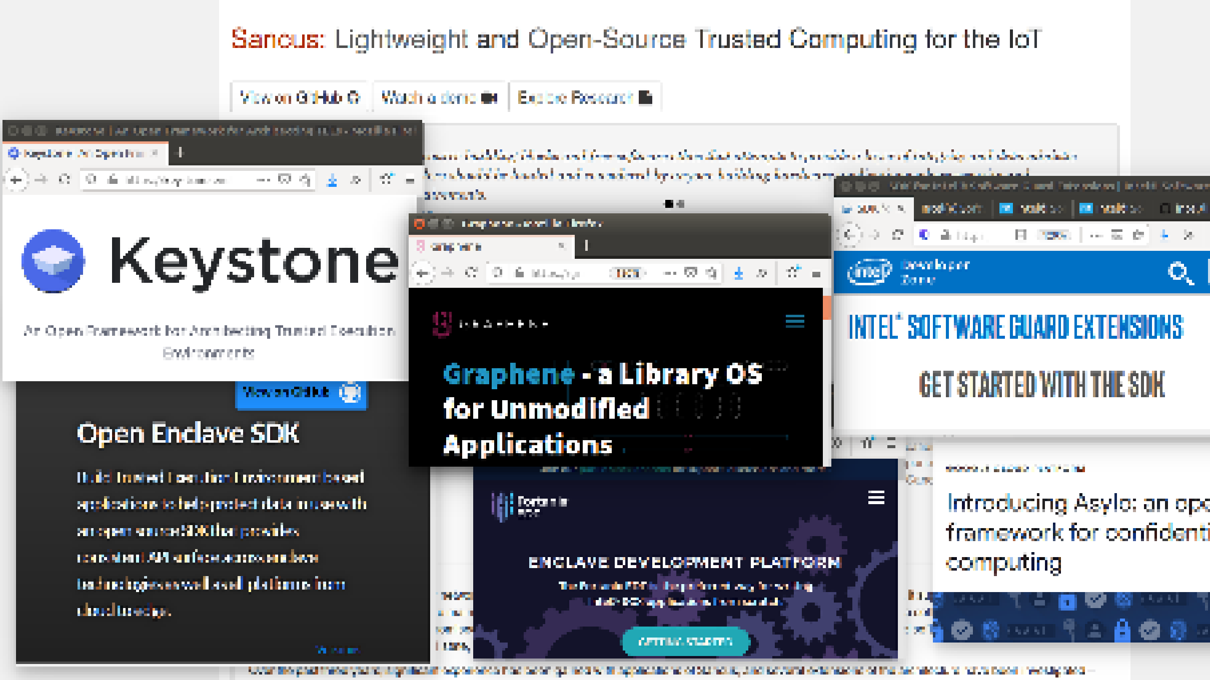


Previous attacks: exploit microarchitectural bugs or side-channels at the hardware level

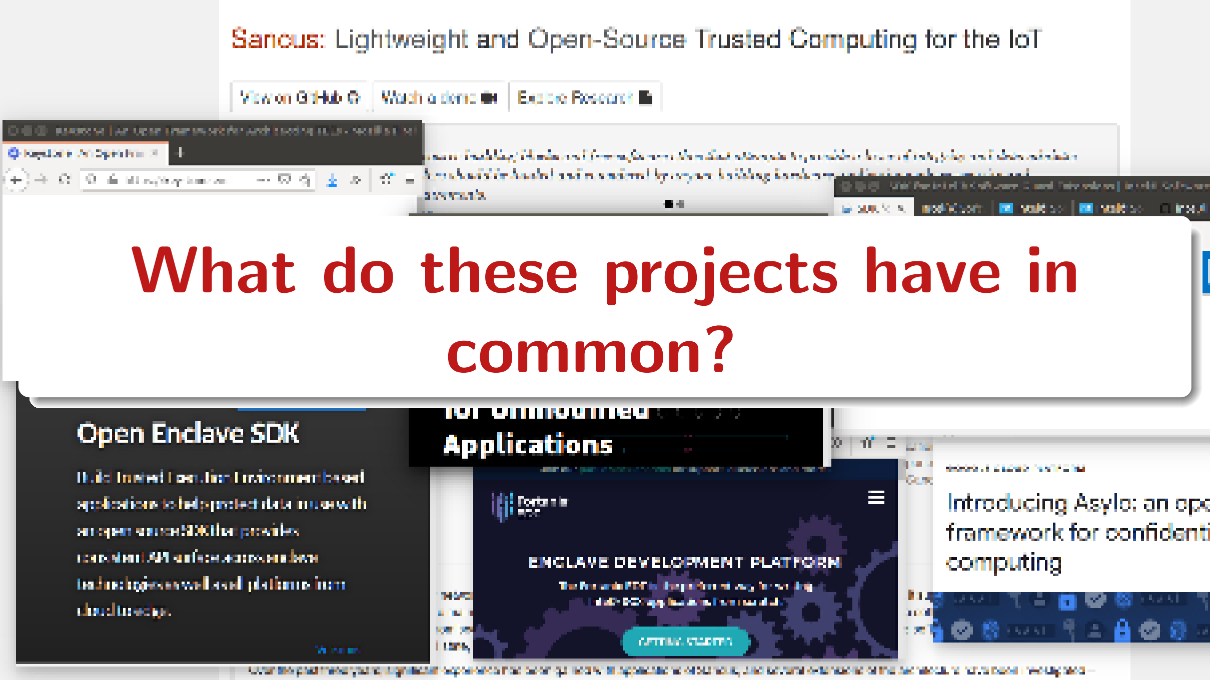
The big picture: Enclaved execution attack surface



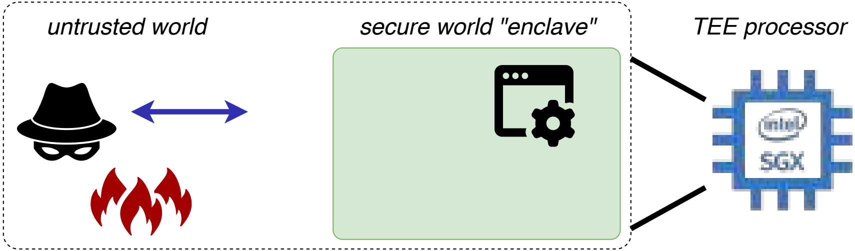
Idea: what about vulnerabilities in the trusted enclave software itself?


Sancus: Lightweight and Open-Source Trusted Computing for the IoT
Keystone
An Open Framework for Architecting Trusted Execution Environments
Open Enclave SDK
Graphene - a Library OS for Unmodified Applications
INTEL® SOFTWARE GUARD EXTENSIONS
GET STARTED WITH THE SDK
ENCLAVE DEVELOPMENT PLATFORM
Introducing Asylo: an
computing

What do these projects have in common?

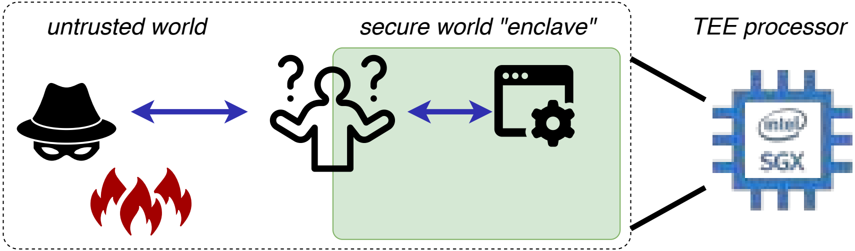


Why isolation is not enough: Enclave shielding runtimes



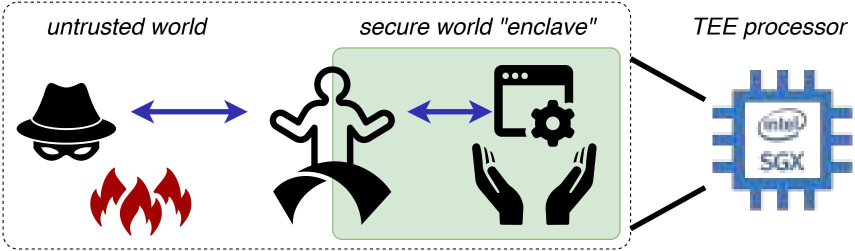
- TEE promise: enclave == "secure oasis" in a **hostile environment**

Why isolation is not enough: Enclave shielding runtimes



- TEE promise: enclave == "secure oasis" in a **hostile environment**
- . . . but application writers and compilers are largely unaware of **isolation boundaries**

Why isolation is not enough: Enclave shielding runtimes



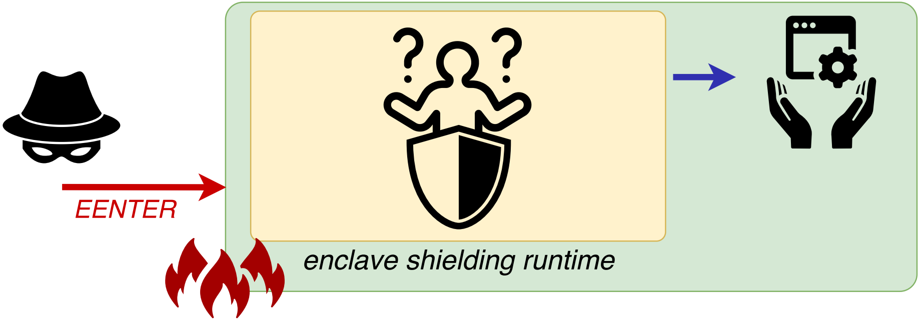
- TEE promise: enclave == "secure oasis" in a **hostile environment**
- . . . but application writers and compilers are largely unaware of **isolation boundaries**

Trusted **shielding runtime** transparently acts as a secure bridge on enclave entry/exit


…but what if the bridge itself is flawed?

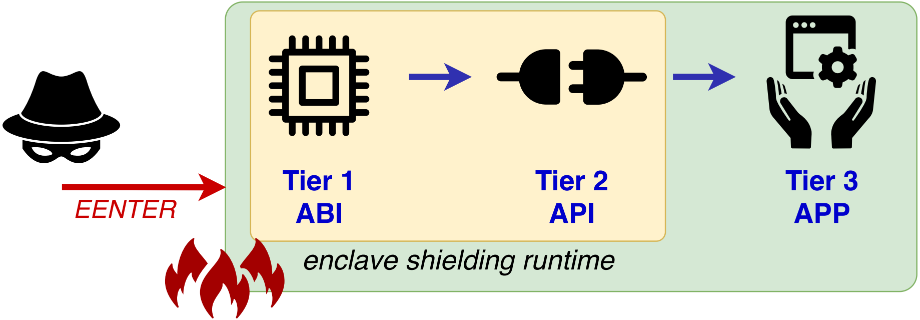
Enclave shielding responsibilities

Key questions: how to securely bootstrap from the untrusted world to the enclaved application binary (and back)? Which sanitizations to apply?

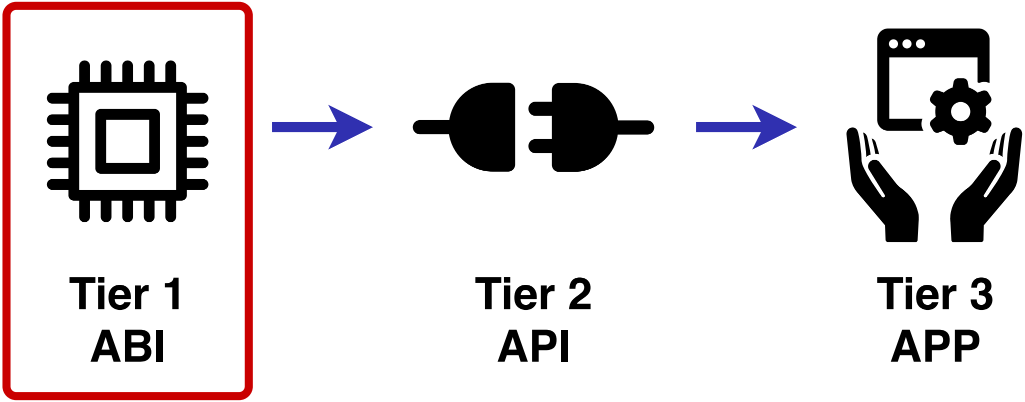


Enclave shielding responsibilities

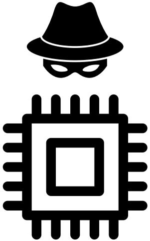
Key insight: split sanitization responsibilities across the ABI and API tiers: *machine state* vs. higher-level *programming language interface*



Tier1: Establishing a trustworthy enclave ABI

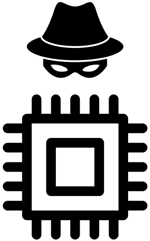


Tier1: Establishing a trustworthy enclave ABI



- ⇝ Attacker controls CPU register contents on enclave entry/exit
- ↔ Compiler expects well-behaved **calling convention** (e.g., stack)
- ⟹ Need to **initialize CPU registers** on entry and **scrub** before exit!

Tier1: Establishing a trustworthy enclave ABI



- ⇝ Attacker controls CPU register contents on enclave entry/exit
- ↔ Compiler expects well-behaved **calling convention** (e.g., stack)
- ⟹ Need to **initialize CPU registers** on entry and **scrub** before exit!

ABI vulnerability analysis

Relatively well-understood, but special care for **stack pointer** + **status register**

Summary: ABI-level attack surface

Vulnerability \ Runtime		SGX-SDK	OpenEnclave	Graphene	SGX-LKL	Rust-EDP	Asylo	Keystone	Sancus
Tier1 (ABI)	#1 Entry status flags sanitization	★	★	◐	●	◐	●	○	○
	#2 Entry stack pointer restore	○	○	★	●	○	○	○	★
	#3 Exit register leakage	○	○	○	★	○	○	○	○

Read the paper for several exploitable ABI vulnerabilities!

Summary: ABI-level attack surface

Vulnerability \ Runtime		SGX-SDK	OpenEnclave	Graphene	SGX-LKL	Rust-EDP	Asylo	Keystone	Sancus
Tier1 (ABI)	#1 Entry status flags sanitization	★	★	◐	●	◐	●	○	○
	#2 Entry stack pointer restore	○	○	★	●	○	○	○	★
	#3 Exit register leakage	○	○	○	★	○	○	○	○
		x86 CISC (Intel SGX)						**RISC**	

A lesson on complexity

🔔 Attack surface **complex x86 ABI** (Intel SGX) >> simpler RISC designs

x86 string instructions: Direction Flag (DF) operation

- Special x86 rep string instructions to speed up streamed memory operations

```c
/* memset(buf, 0x0, 100) */
for (int i=0; i < 100; i++)
    buf[i] = 0x0;
```

→

```asm
lea rdi, buf
mov al, 0x0
mov ecx, 100
rep stos [rdi], al
```

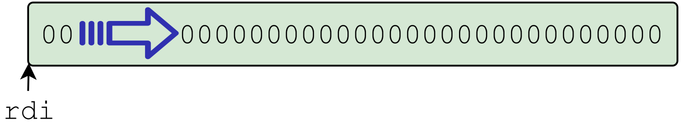
x86 string instructions: Direction Flag (DF) operation

- Special x86 rep string instructions to speed up streamed memory operations
- Default operate **left-to-right**

```c
/* memset(buf, 0x0, 100) */
for (int i=0; i < 100; i++)
    buf[i] = 0x0;
```

→

```asm
lea rdi, buf
mov al, 0x0
mov ecx, 100
rep stos [rdi], al
```



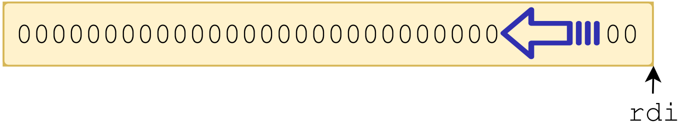
x86 string instructions: Direction Flag (DF) operation

- Special x86 rep string instructions to speed up streamed memory operations
- Default operate **left-to-right**, unless software sets *RFLAGS.DF=1*

```
/* memset(buf, 0x0, 100) */
for (int i=0; i < 100; i++)
   buf[i] = 0x0;
```

→

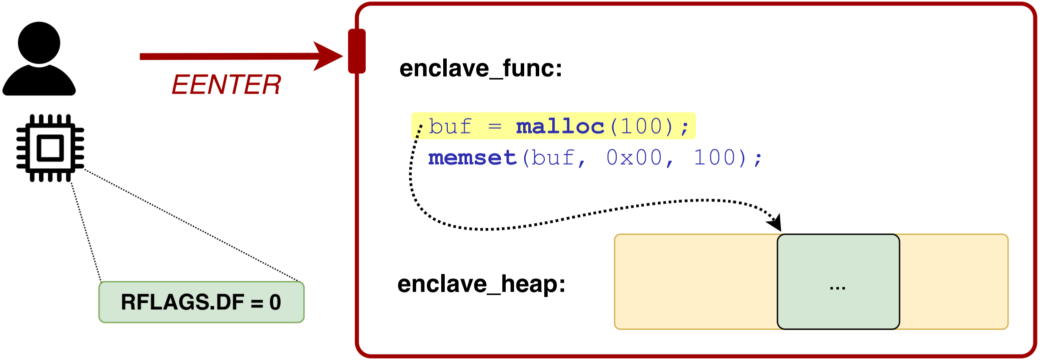
```
lea rdi, buf+100
mov al, 0x0
mov ecx, 100
std ; set direction flag
rep stos [rdi], al
```



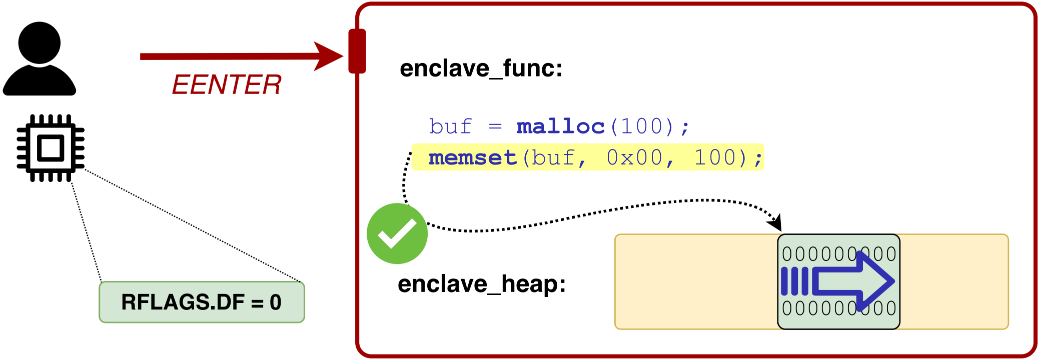
x86 System-V ABI

[8] The direction flag DF in the %rFLAGS register must be clear (set to "forward" direction) on function entry and return. Other user flags have no specified role in the standard calling sequence and are *not* preserved across calls.

⇄ Enter enclave with *RFLAGS.DF=0*



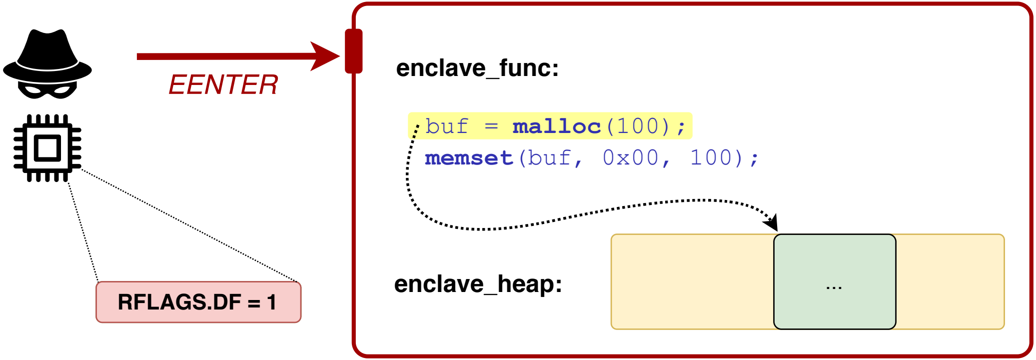
Intended heap memory initialization: left-to-right



SGX-DF: Inverting enclaved string memory operations

CVE-2019-14565

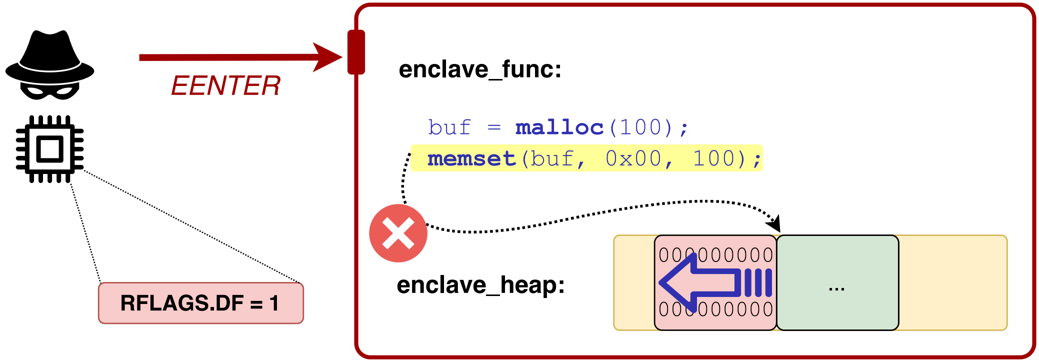
Enter enclave with *RFLAGS.DF=1*



SGX-DF: Inverting enclaved string memory operations

CVE-2019-14565

⚠ Enclave heap **memory corruption:** right-to-left. . .



Summary:

A potential security vulnerability in Intel SGX SDK may allow for information disclosure, escalation of privilege or denial of service. Intel is releasing software updates to mitigate this potential vulnerability. This potential vulnerability is present in all SGX enclaves built with the affected SGX SDK versions.

Vulnerability Details:

CVEID: CVE-2019-14566

Description: Insufficient input validation in Intel(R) SGX SDK versions shown below may allow an authenticated user to enable information disclosure, escalation of privilege or denial of service via local access.

CVSS Base Score: 7.8 (High)

CVSS Vector: CVSS:3.1/AV:L/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H

CVEID: CVE-2019-14565

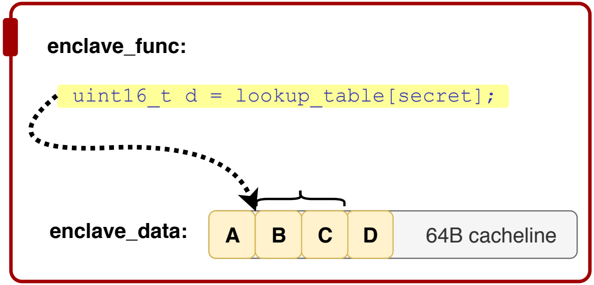
Description: Insufficient initialization in Intel(R) SGX SDK versions shown below may allow an authenticated user to enable information disclosure, escalation of privilege or denial of service via local access.

CVSS Base Score: 7.0 (High)

CVSS Vector: CVSS:3.1/AV:L/AC:H/PR:L/UI:N/S:U/C:H/I:H/A:H

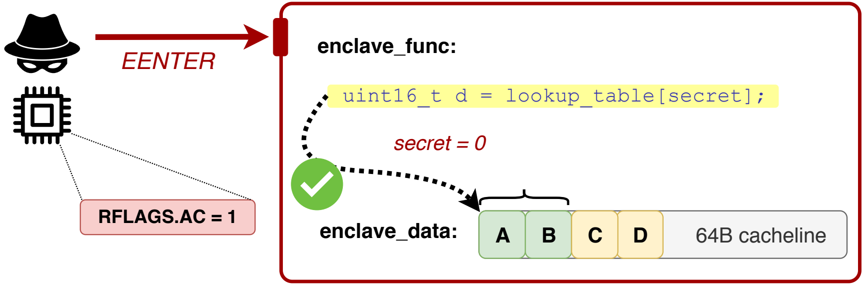
SGX-AC: Building an intra-cacheline side-channel

There's more! Alignment Check (AC) flag enables **exceptions for unaligned data accesses** → *intra-cacheline side-channel* ☺



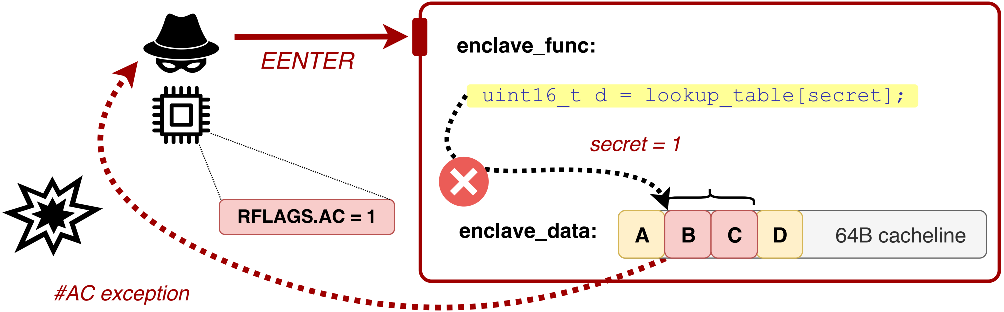
SGX-AC: Building an intra-cacheline side-channel

Enter enclave with *RFLAGS.AC=1* and secret index=0
→ well-aligned data access: **no exception**

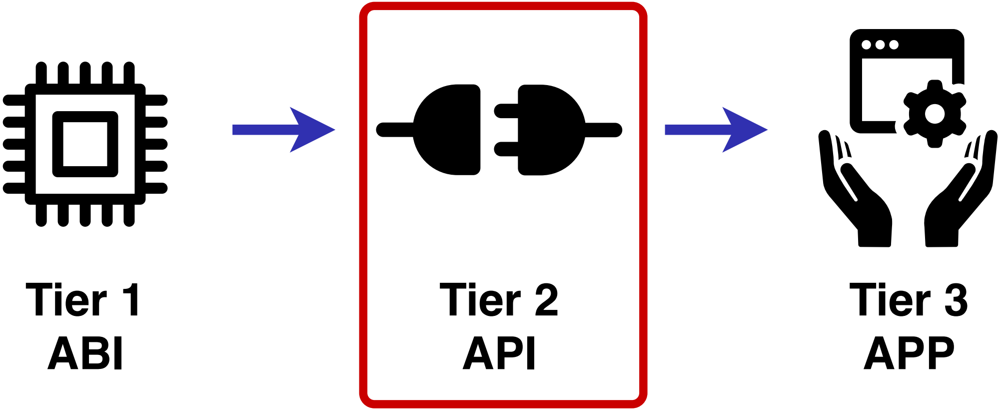


SGX-AC: Building an intra-cacheline side-channel

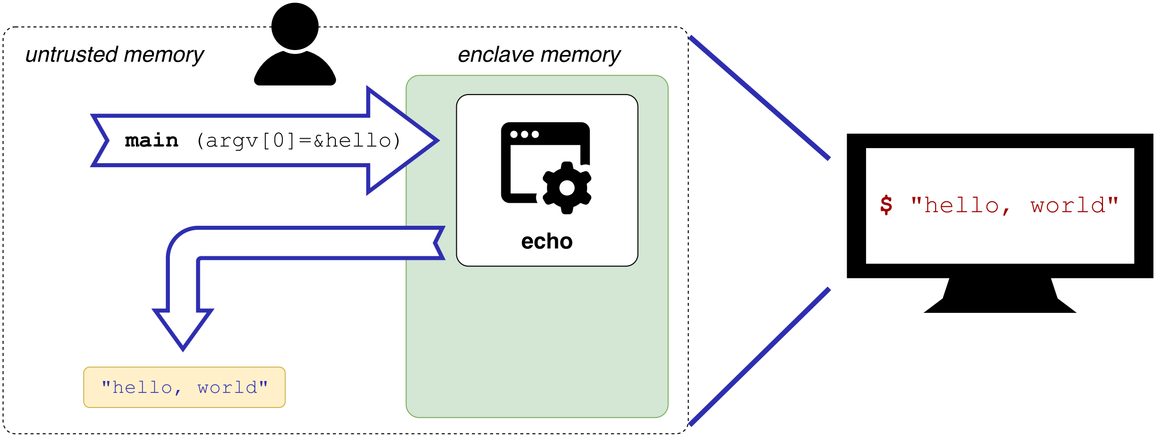
Enter enclave with *RFLAGS.AC=1* and secret index=1
→ unaligned data access: **alignment-check exception. . .**



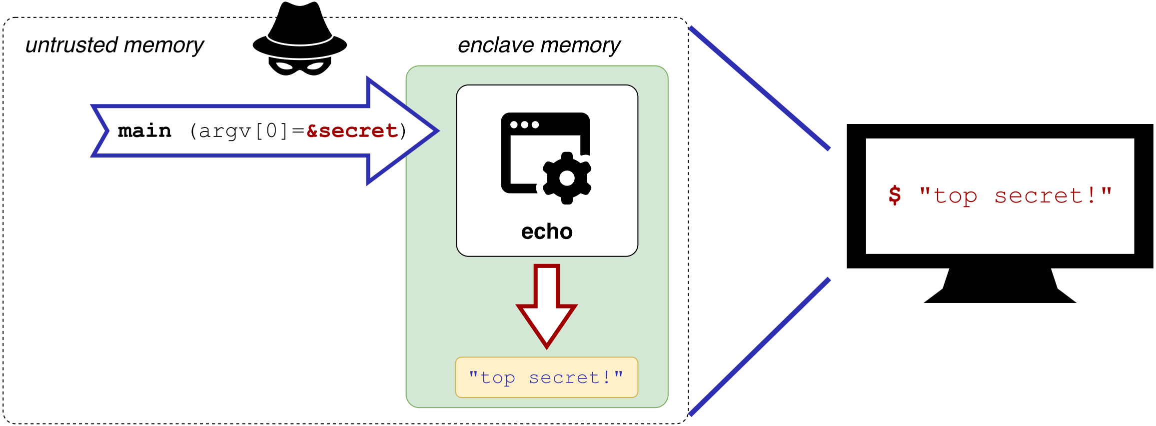
Tier 2: Sanitizing the enclave API



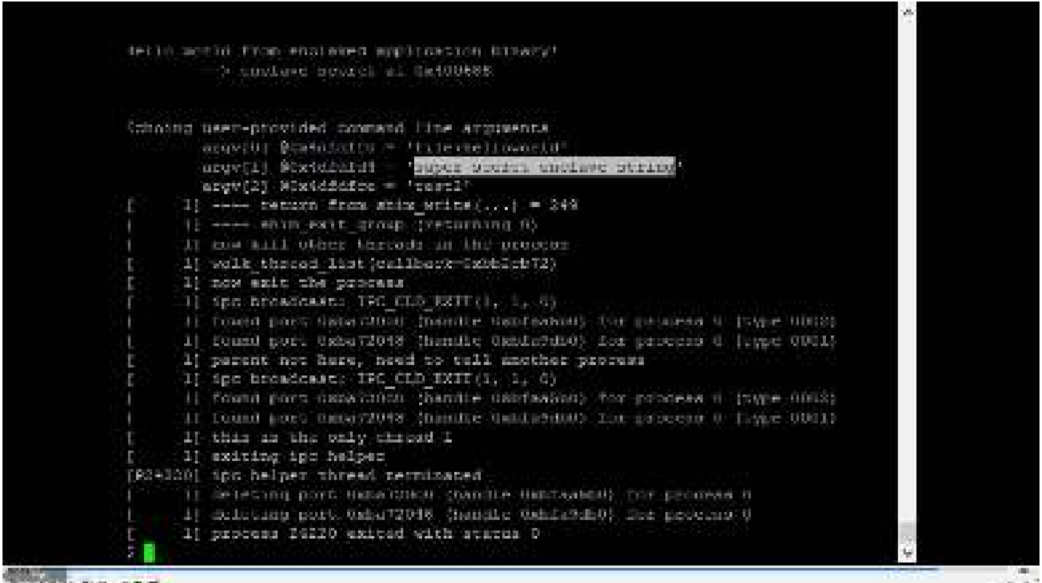
Validating pointer arguments: Confused deputy attacks



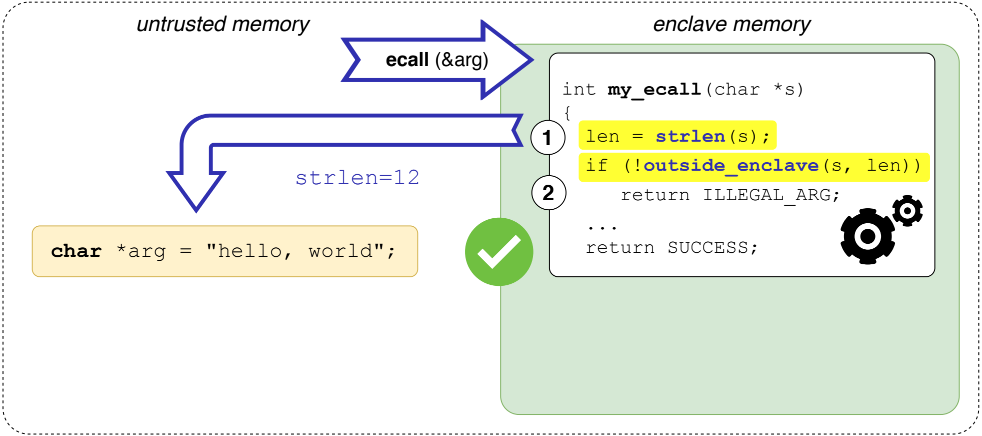
Validating pointer arguments: Confused deputy attacks



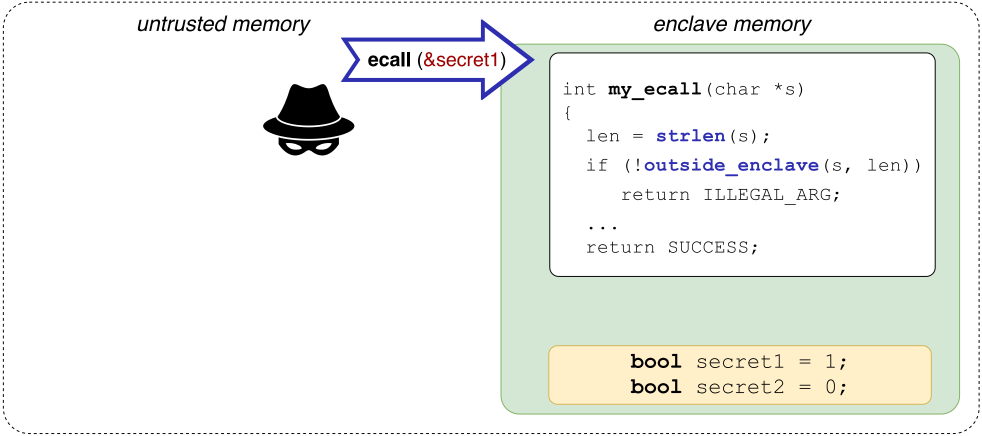
Validating pointer arguments: Confused deputy attacks



Idea: 2-stage approach ensures string arguments fall *entirely* outside enclave



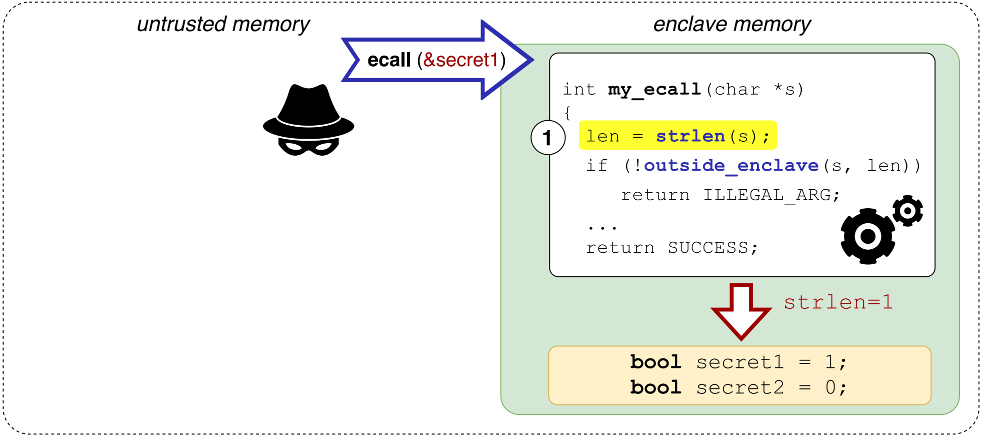
. . . **but** what if we try passing an illegal, in-enclave pointer anyway?



Intel SGX-SDK: Null-terminated strings are hard. . .

CVE-2018-3626

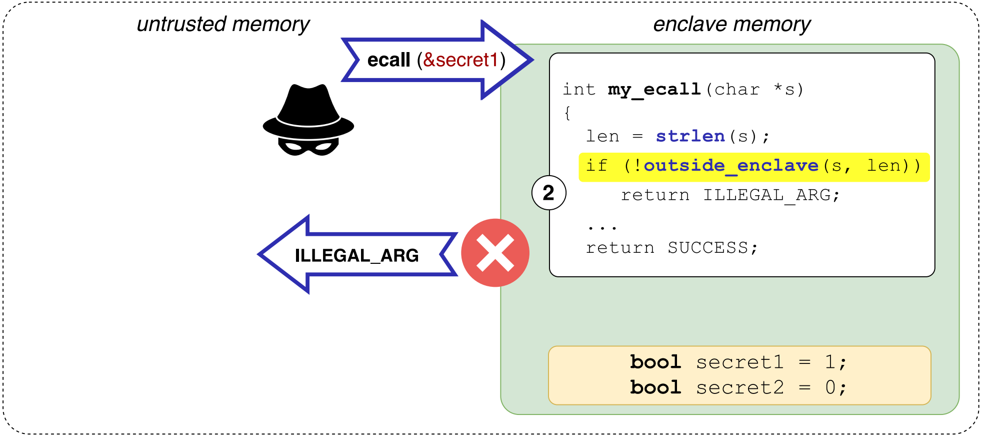
Enclave **first** computes length of secret, in-enclave buffer!



Intel SGX-SDK: Null-terminated strings are hard. . .

CVE-2018-3626

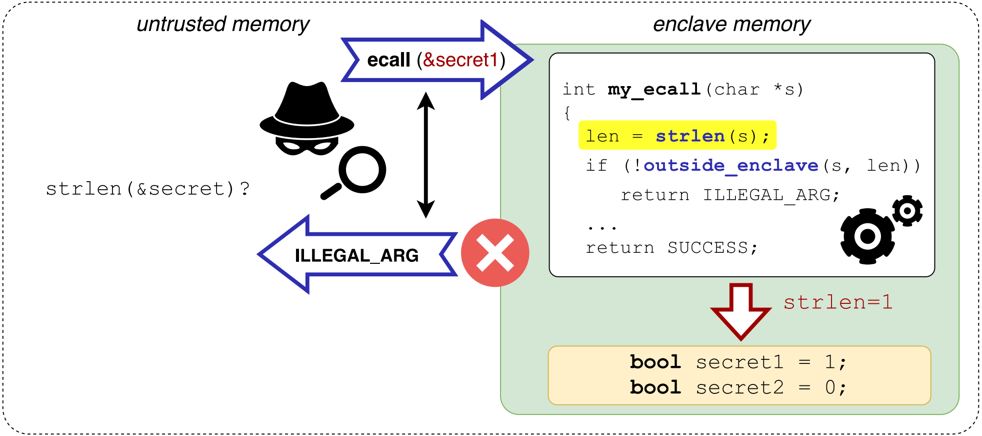
. . . and only **afterwards verifies** whether *entire string* falls outside enclave



Intel SGX-SDK: Null-terminated strings are hard. . .

CVE-2018-3626

Idea: strlen() timing as a side-channel oracle for in-enclave null bytes ☺

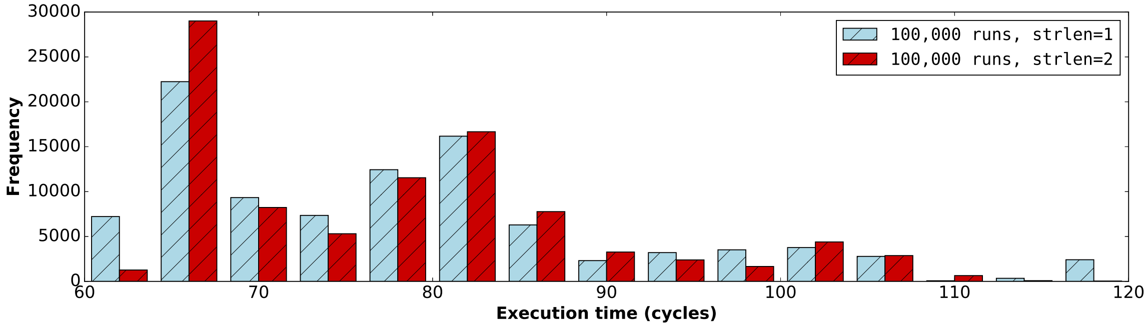


What about measuring execution time?

Building the oracle with `strlen()` timing?

Execution timing side-channel?

✖ **Too noisy:** we need to measure timing of a single x86 increment instruction. . .



What about measuring page faults?

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.

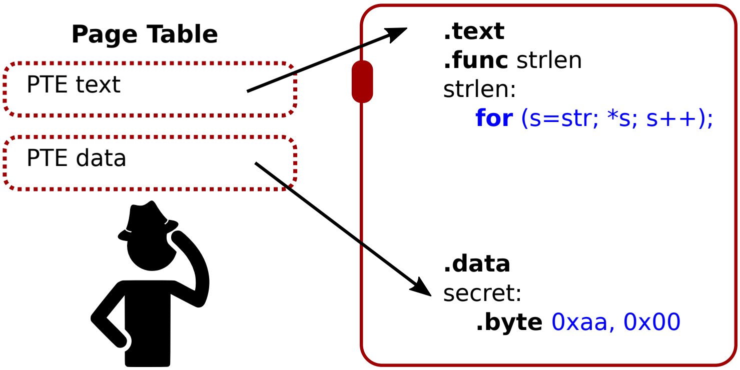
An attacker with access to the platform can see what pages are being executed or accessed. This side-channel vulnerability can be mitigated by aligning specific code and data blocks to exist entirely within a single page.

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.

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

Counting `strlen()` loop iterations with page faults?

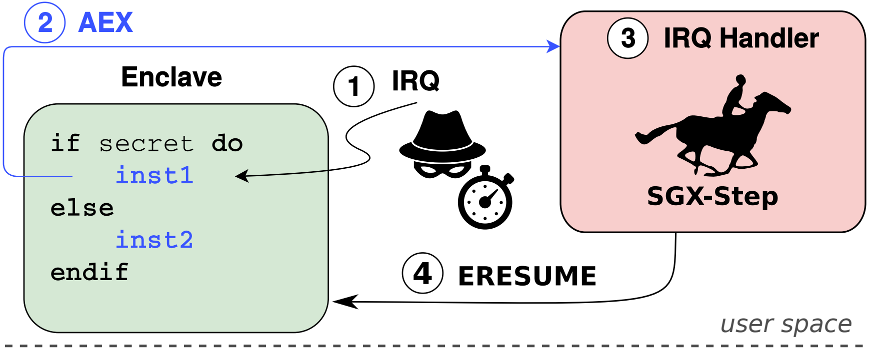
❌ **Temporal resolution:** progress requires both code + data pages mapped in



Challenge: Counting `strlen()` loop iterations

What about leveraging interrupts?

SGX-Step: Executing enclaves one instruction at a time

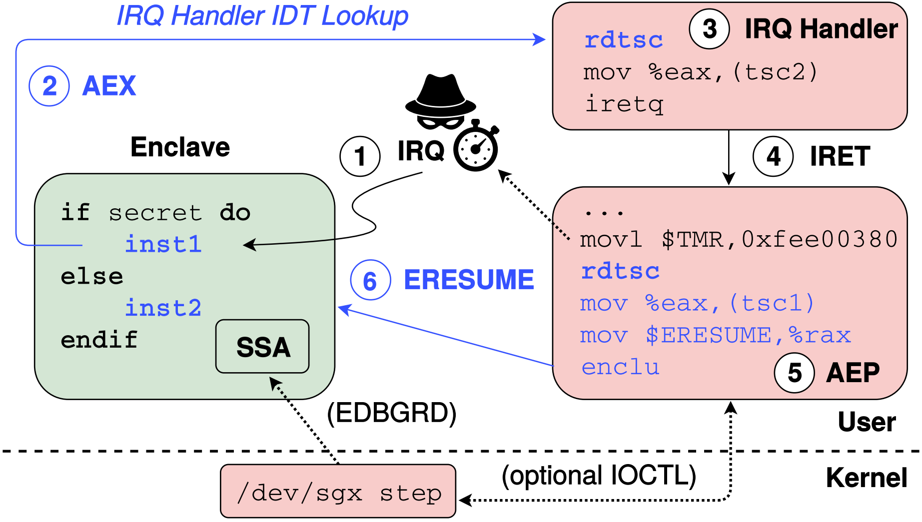


Van Bulck et al. "SGX-Step: A practical attack framework for precise enclave execution control", SysTEX 2017 [VBPS17]

Van Bulck et al. "Nemesis: Studying Microarchitectural Timing Leaks in Rudimentary CPU Interrupt Logic", CCS 2018 [VBPS18]

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

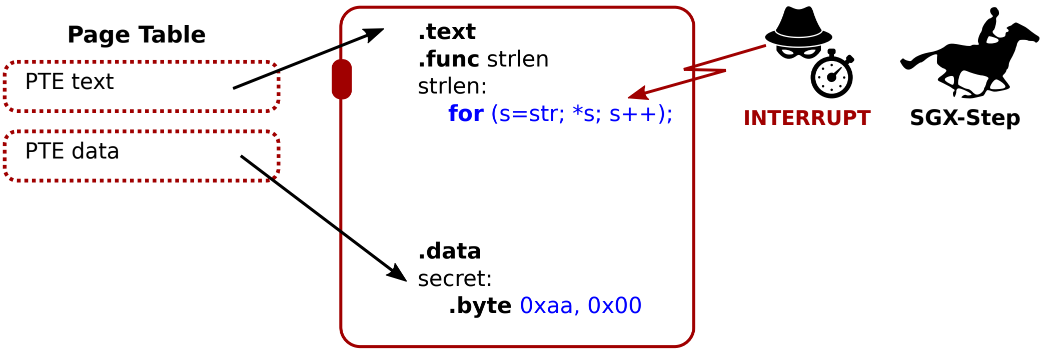
SGX-Step: Executing enclaves one instruction at a time



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

Building a deterministic `strlen()` null byte oracle with SGX-Step

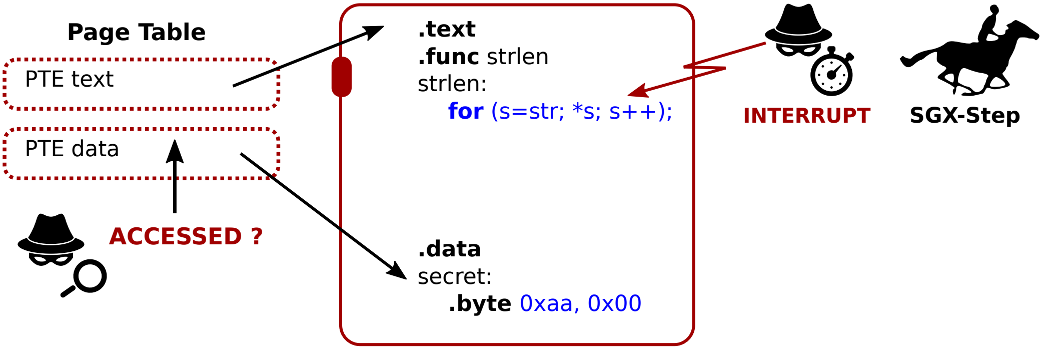
Execute *exactly* one enclave instruction → **timer interrupt**



Van Bulck et al. "SGX-Step: A practical attack framework for precise enclave execution control", SysTEX 2017 [VBPS17]

Building a deterministic `strlen()` null byte oracle with SGX-Step

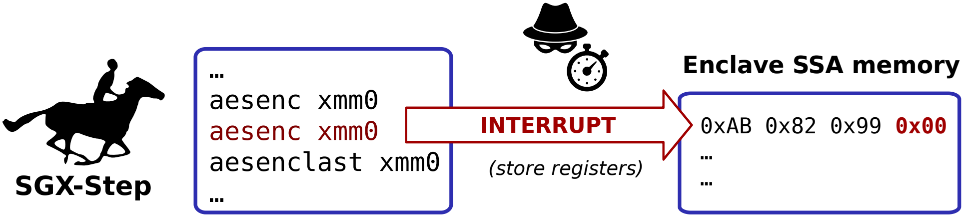
Page table accessed bit set? → **strlen++** → resume



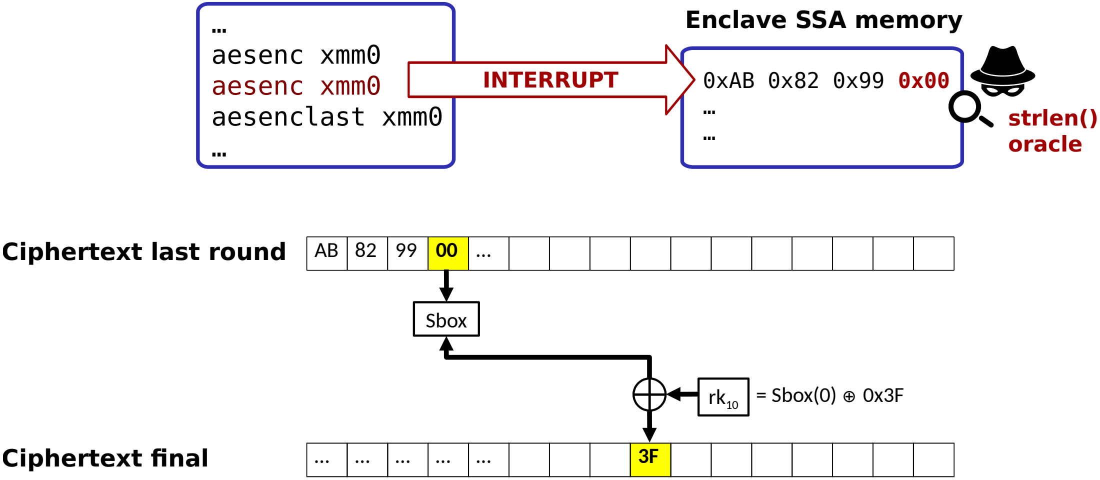
Van Bulck et al. "Telling your secrets without page faults: Stealthy page table-based attacks on enclaved execution", USENIX 2017 [VBWK+17]


CVE-2018-3626: ALL YOUR ZERO BYTES
ARE BELONG TO US
imgflip.com

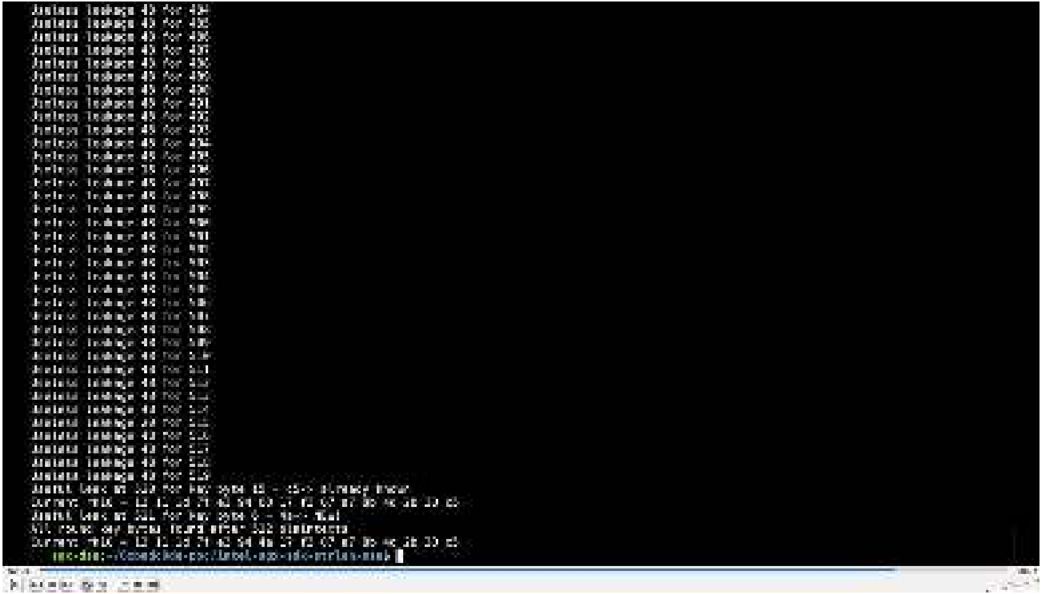
Breaking AES-NI with the `strlen()` null byte oracle



Breaking AES-NI with the `strlen()` null byte oracle



Breaking AES-NI with the `strlen()` null byte oracle



Summary: API-level attack surface

Vulnerability \ Runtime		SGX-SDK	OpenEnclave	Graphene	SGX-LKL	Rust-EDP	Asylo	Keystone	Sancus
Tier2 (API)	#4 Missing pointer range check	○	★	★	★	○	●	○	★
	#5 Null-terminated string handling	⯪	★	○	○	○	○	○	○
	#6 Integer overflow in range check	○	○	●	○	●	○	●	●
	#7 Incorrect pointer range check	○	○	●	○	○	●	○	●
	#8 Double fetch untrusted pointer	○	○	●	○	○	○	○	○
	#9 Ocall return value not checked	○	★	★	★	○	●	★	○
	#10 Uninitialized padding leakage	[LK17]	★	○	●	○	●	★	★

Read the paper for more API attacks!

Summary: API-level attack surface

	Vulnerability \ Runtime	SGX-SDK	OpenEnclave	Graphene	SGX-LKL	Rust-EDP	Asylo	Keystone	Sancus
Tier2 (API)	#4 Missing pointer range check	○	★	★	★	○	●	○	★
	#5 Null-terminated string handling	⯪	★	○	○	○	○	○	○
	#6 Integer overflow in range check	○	○	●	○	●	○	●	●
	#7 Incorrect pointer range check	○	○	●	○	○	●	○	●
	#8 Double fetch untrusted pointer	○	○	●	○	○	○	○	○
	#9 Ocall return value not checked	○	★	★	★	○	●	★	○
	#10 Uninitialized padding leakage	[LK17]	★	○	●	○	●	★	★

❗ **Critical oversights** in production and research code

→ across TEEs and programming languages (incl. safe langs like Rust)

Vulnerability \ Runtime		SGX-SDK	OpenEnclave	Graphene	SGX-LKL	Rust-EDP	Asylo	Keystone	Sancus
Tier2 (API)	#4 Missing pointer range check	○	★	★	★	○	●	○	★
	#5 Null-terminated string handling	⯪	★	○	○	○	○	○	○
	#6 Integer overflow in range check	○	○	●	○	●	○	●	●
	#7 Incorrect pointer range check	○	○	●	○	○	●	○	●
	#8 Double fetch untrusted pointer	○	○	●	○	○	○	○	○
	#9 Ocall return value not checked	○	★	★	★	○	●	★	○
	#10 Uninitialized padding leakage	[LK17]	★	○	●	○	●	★	★

Generally understood (Iago attacks) but **still widespread,** not exclusive to library OSs

Checkoway et al. “Iago Attacks: Why the System Call API is a Bad Untrusted RPC Interface” ASPLOS 2013 [CS13]


Washes away Bacteria
Frequent hand washing helps keep your family healthy.
Safeguard®
White with touch of Aloe

Conclusions and outlook

Take-away message

Secure enclave interactions require proper **ABI and API sanitizations!**

Conclusions and outlook

Take-away message

Secure enclave interactions require proper **ABI and API sanitizations!**

- Large attack surface, including subtle **side-channel oversights. . .**
- **Defenses:** need to research more principled sanitization strategies
- **User-to-kernel analogy:** learn from experience with secure OS development

https://github.com/jovanbulck/0xbadc0de

A Tale of Two Worlds: Assessing the Vulnerability of Enclave Shielding Runtimes

Jo Van Bulck[1] David Oswald[2] Eduard Marin[2] Abdulla Aldoseri[2]
Flavio D. Garcia[2] Frank Piessens[1]

[1]imec-DistriNet, KU Leuven [2]The University of Birmingham, UK

Hardware-aided trusted computing devroom, FOSDEM, February 1, 2020

References I

S. Checkoway and H. Shacham.
Iago Attacks: Why the System Call API is a Bad Untrusted RPC Interface.
In *International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS, pp. 253–264, 2013.

S. Lee and T. Kim.
Leaking uninitialized secure enclave memory via structure padding.
arXiv preprint arXiv:1710.09061, 2017.

K. Murdock, D. Oswald, F. D. Garcia, J. Van Bulck, D. Gruss, and F. Piessens.
Plundervolt: Software-based fault injection attacks against intel sgx.
In *Proceedings of the 41st IEEE Symposium on Security and Privacy (S&P'20)*, 2020.

J. Noorman, J. Van Bulck, J. T. Mühlberg, F. Piessens, P. Maene, B. Preneel, I. Verbauwhede, J. Götzfried, T. Müller, and F. Freiling.
Sancus 2.0: A low-cost security architecture for IoT devices.
ACM Transactions on Privacy and Security (TOPS), 20(3):7:1–7:33, 2017.

M. Schwarz, M. Lipp, D. Moghimi, J. Van Bulck, J. Stecklina, T. Prescher, and D. Gruss.
ZombieLoad: Cross-privilege-boundary data sampling.
In *CCS*, 2019.

J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx.
Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution.
In *Proceedings of the 27th USENIX Security Symposium*, 2018.

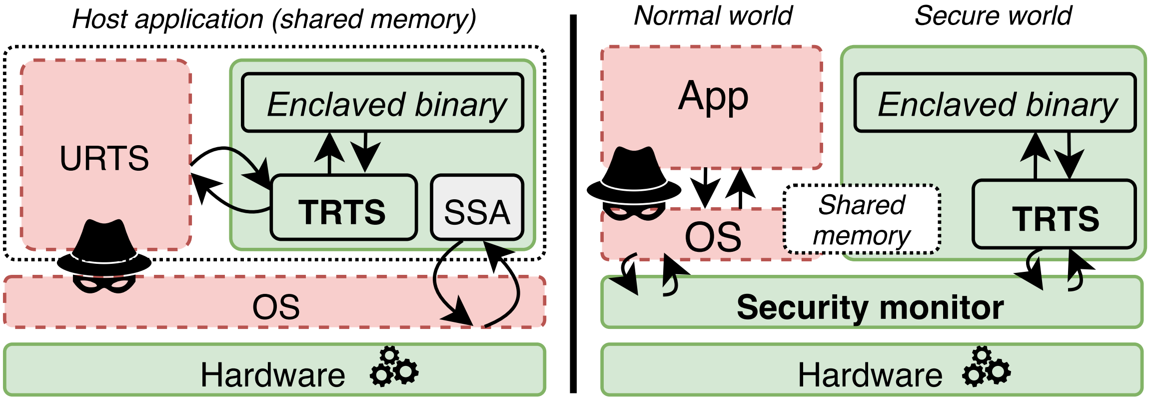
J. Van Bulck, F. Piessens, and R. Strackx.
SGX-Step: A practical attack framework for precise enclave execution control.
In *SysTEX*, pp. 4:1–4:6, 2017.

References II

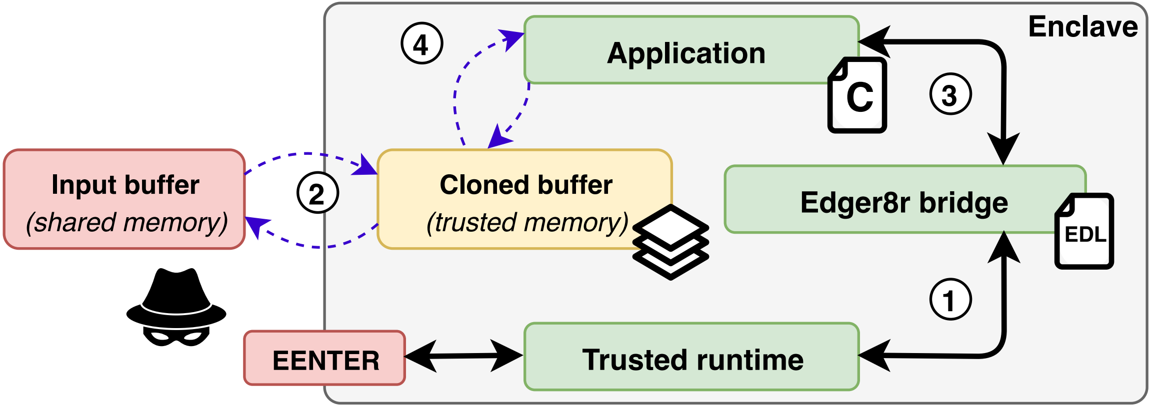
J. Van Bulck, F. Piessens, and R. Strackx.
Nemesis: Studying microarchitectural timing leaks in rudimentary cpu interrupt logic.
In *ACM CCS 2018*, 2018.

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*, pp. 1041–1056, 2017.

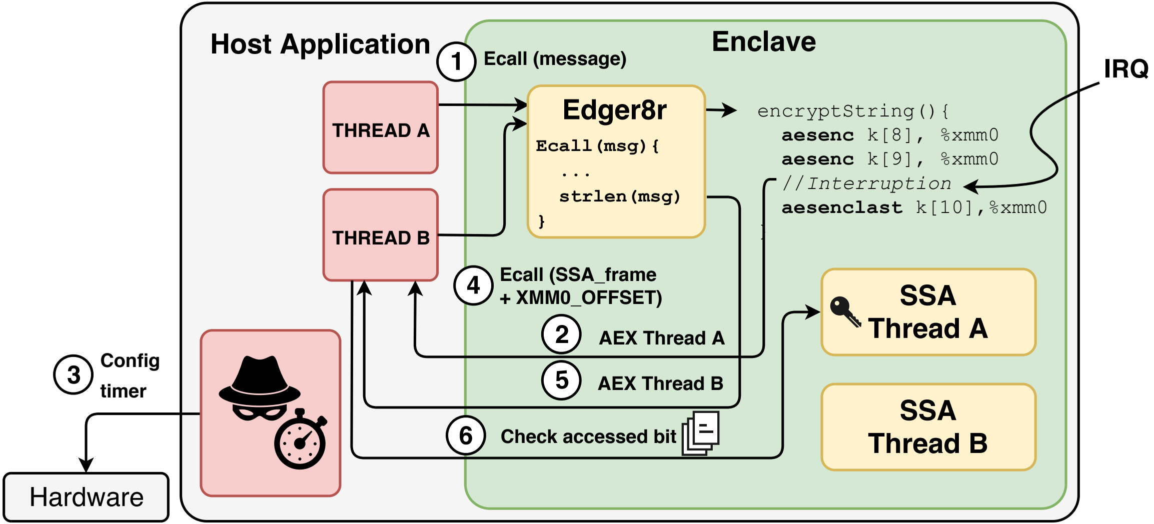
TEE design: Single-address-space vs. world-shared memory approaches



edger8r: Input/output buffer cloning



Intel SGX strlen oracle attack



Exploitation challenges: Building a precise null byte oracle

🔔 **Goal:** Precisely measure `strlen()` loop iterations?

```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
```

⇒ tight loop: 4 asm instructions, single memory operand, single code + data page

Reconstructing the full AES-NI round key

Algorithm 1 `strlen()` oracle AES key recovery where $S(\cdot)$ denotes the AES SBox and $SR(p)$ the position of byte p after AES ShiftRows.

while not full key K recovered **do**
  $(P, C, L) \leftarrow$ random plaintext, associated ciphertext, strlen oracle
  if $L < 16$ **then**
    $K[SR(L)] \leftarrow C[SR(L)] \oplus S(0)$
  end if
end while