The Tale Continues:
Pitfalls and Best Practices for SGX Shielding Runtimes

Jo Van Bulck [1] Fritz Alder [1]

Joint work with

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

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

2nd Intel SGX Community Workshop, July 14, 2020

Outline: How to besiege a fortress?

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

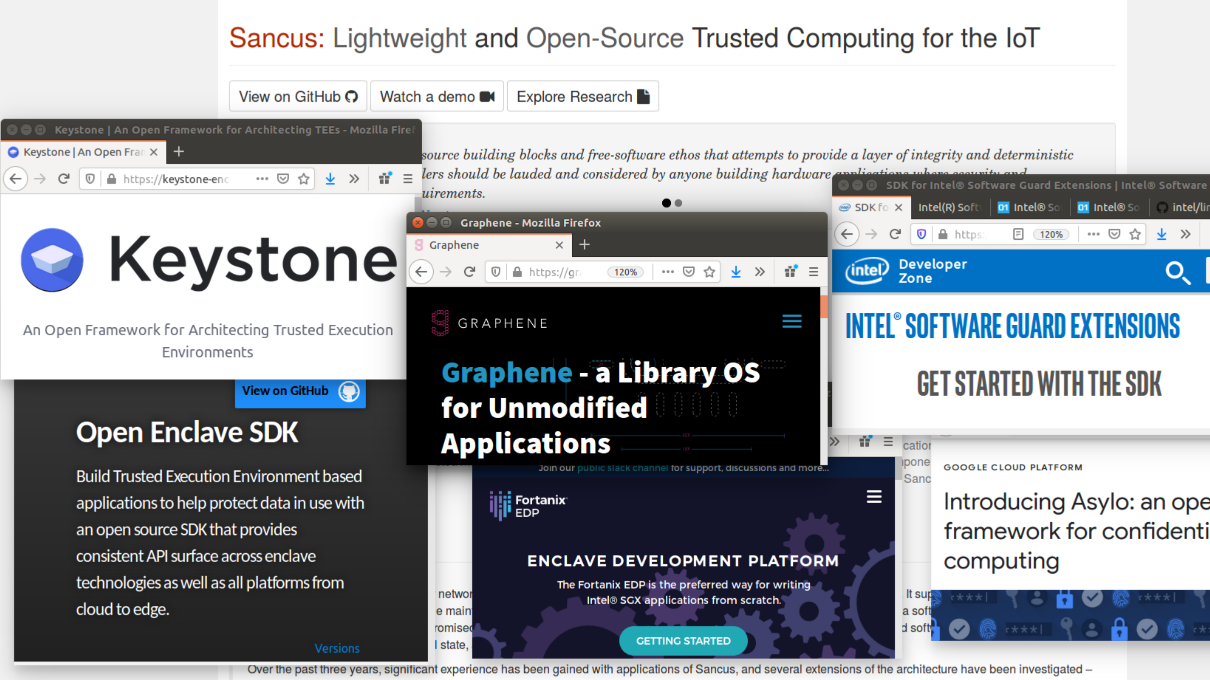
Outline: How to besiege a TEE enclave?

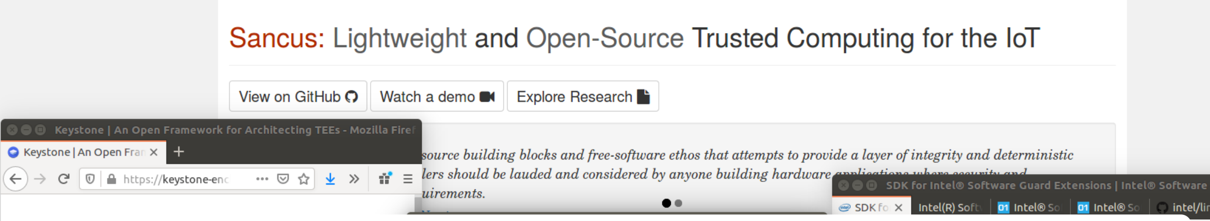
Tier	Vulnerability \ Runtime	SGX-SDK	OpenEnclave	Graphene	SGX-LKL	Rust-EDP	Asylo	Keystone	Sancus
Tier1 (ABI)	#1 Entry status flags sanitization	★	★	◐	●	◐	●	○	○
	#2 Entry stack pointer restore	○	○	★	●	○	○	○	★
	#3 Exit register leakage	○	○	○	★	○	○	○	○
Tier2 (API)	#4 Missing pointer range check	○	★	★	★	○	●	○	★
	#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

¯_(ツ)_/¯

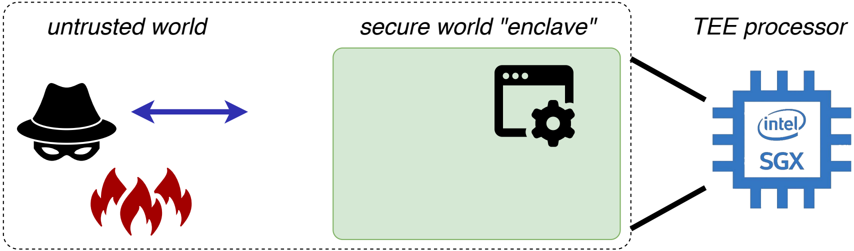
Why do we need enclave fortresses anyway?


Sancus: Lightweight and Open-Source Trusted Computing for the IoT
View on GitHub
Watch a demo
Explore Research
Keystone | An Open Framework for Architecting TEEs - Mozilla Firef
Keystone
An Open Framework for Architecting Trusted Execution Environments
View on GitHub
Open Enclave SDK
Build Trusted Execution Environment based applications to help protect data in use with an open source SDK that provides consistent API surface across enclave technologies as well as all platforms from cloud to edge.
Versions
Graphene - Mozilla Firefox
GRAPHENE
Graphene - a Library OS for Unmodified Applications
Join our public slack channel for support, discussions and more...
Fortanix EDP
ENCLAVE DEVELOPMENT PLATFORM
The Fortanix EDP is the preferred way for writing Intel® SGX applications from scratch.
GETTING STARTED
SDK for Intel® Software Guard Extensions | Intel® Software
intel Developer Zone
INTEL® SOFTWARE GUARD EXTENSIONS
GET STARTED WITH THE SDK
GOOGLE CLOUD PLATFORM
Introducing Asylo: an ope framework for confidenti computing
Over the past three years, significant experience has been gained with applications of Sancus, and several extensions of the architecture have been investigated –



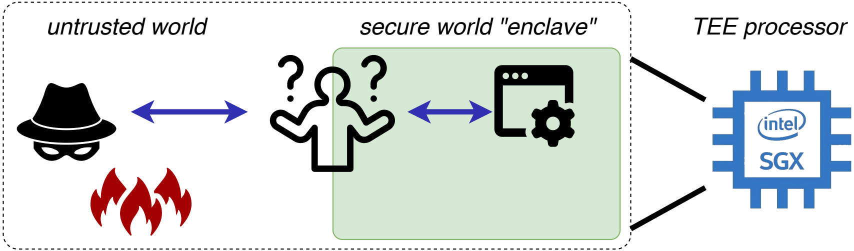
What do these projects have in common?

Why isolation is not enough: Enclave shielding runtimes



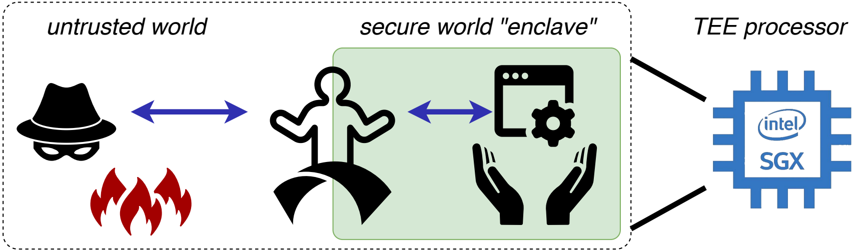
- TEE promise: enclave == "secure oasis" in a **hostile environment**

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



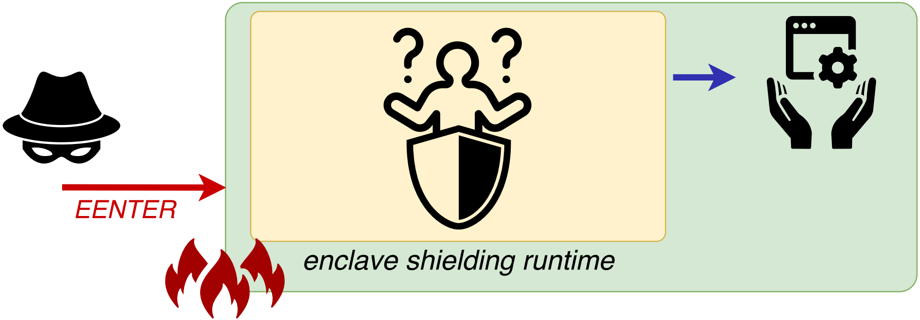
- TEE promise: enclave == "secure oasis" in a **hostile environment**
- ...but application writers and compilers are largely unaware of **isolation boundaries**


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


. . . but what if the bridge itself is flawed?

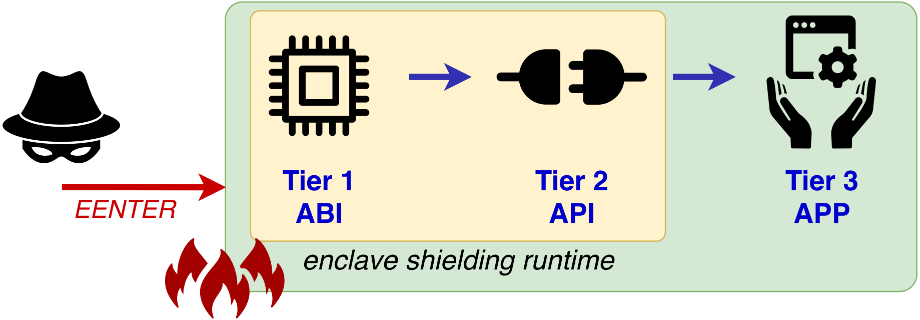
Enclave shielding responsibilities

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

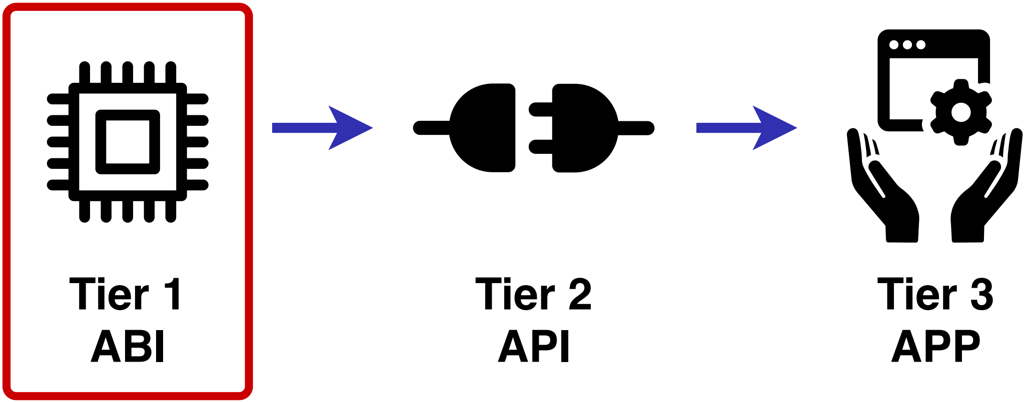


Enclave shielding responsibilities

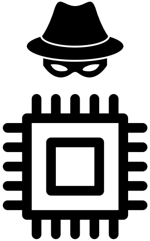
🔔 **Key insight:** split sanitization responsibilities across the ABI and API tiers: *machine state* vs. higher-level *programming language interface*



Tier1: Establishing a trustworthy enclave ABI

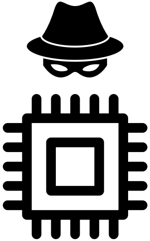


Tier1: Establishing a trustworthy enclave ABI



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

Tier1: Establishing a trustworthy enclave ABI



- $\rightsquigarrow$ Attacker controls CPU register contents on enclave entry/exit
- $\leftrightarrow$ Compiler expects well-behaved **calling convention** (e.g., stack)
- $\Longrightarrow$ 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
```

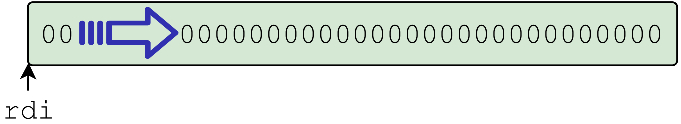
x86 string instructions: Direction Flag (DF) operation

- 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
```

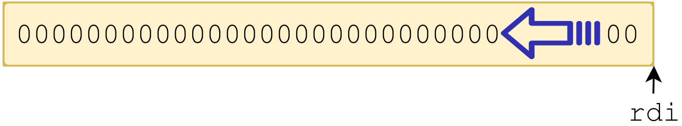


x86 string instructions: Direction Flag (DF) operation

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

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

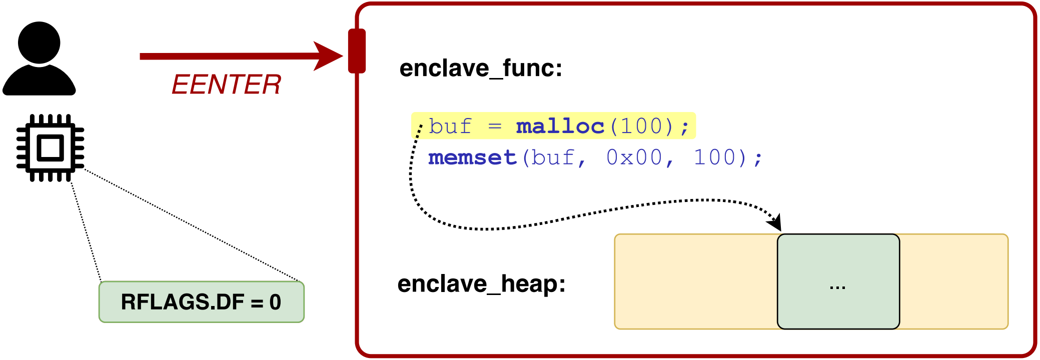
```asm
lea rdi, buf+100
mov al, 0x0
mov ecx, 100
std ; set direction flag
rep stos [rdi], al
```



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.

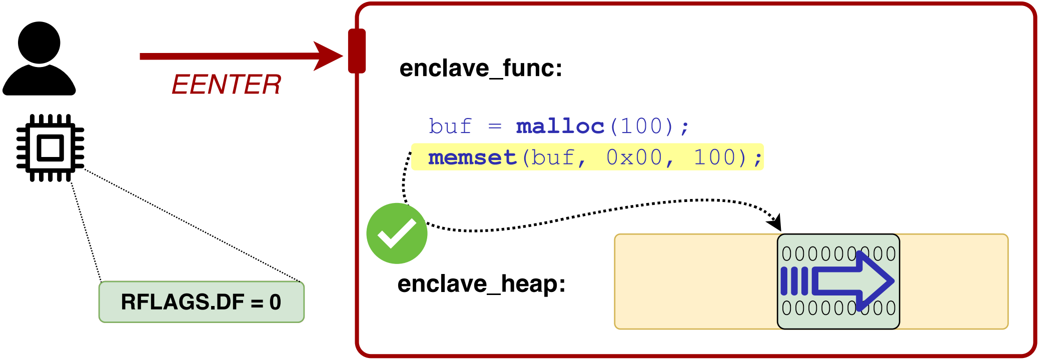
⇄ Enter enclave with *RFLAGS.DF=0*



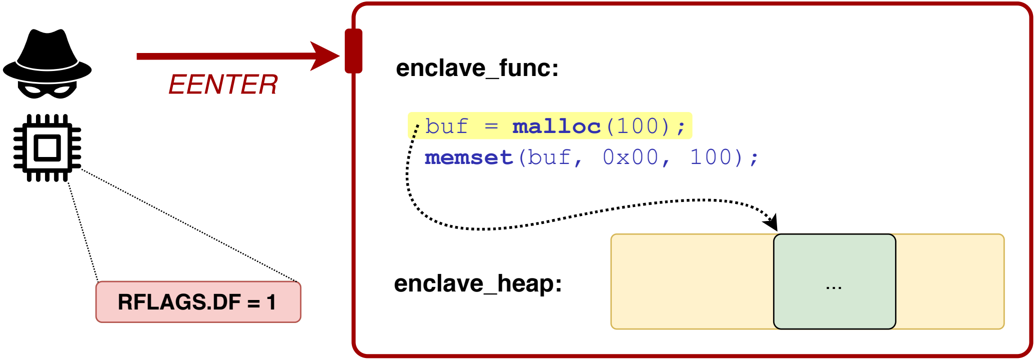
SGX-DF: Inverting enclaved string memory operations

CVE-2019-14565

Intended heap memory initialization: left-to-right

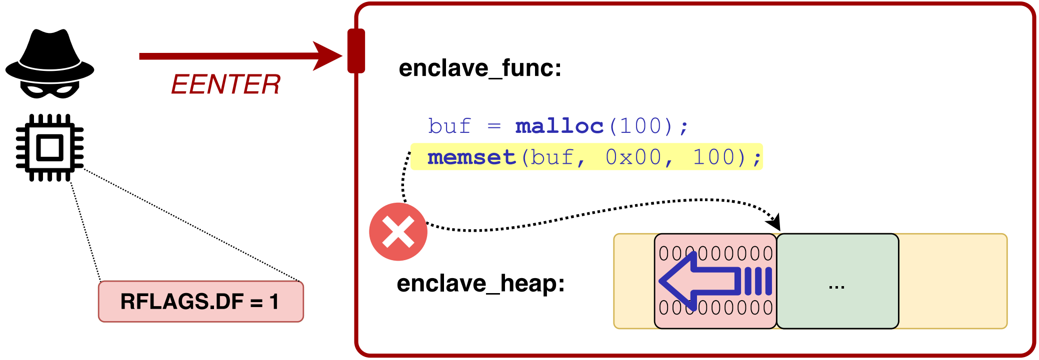


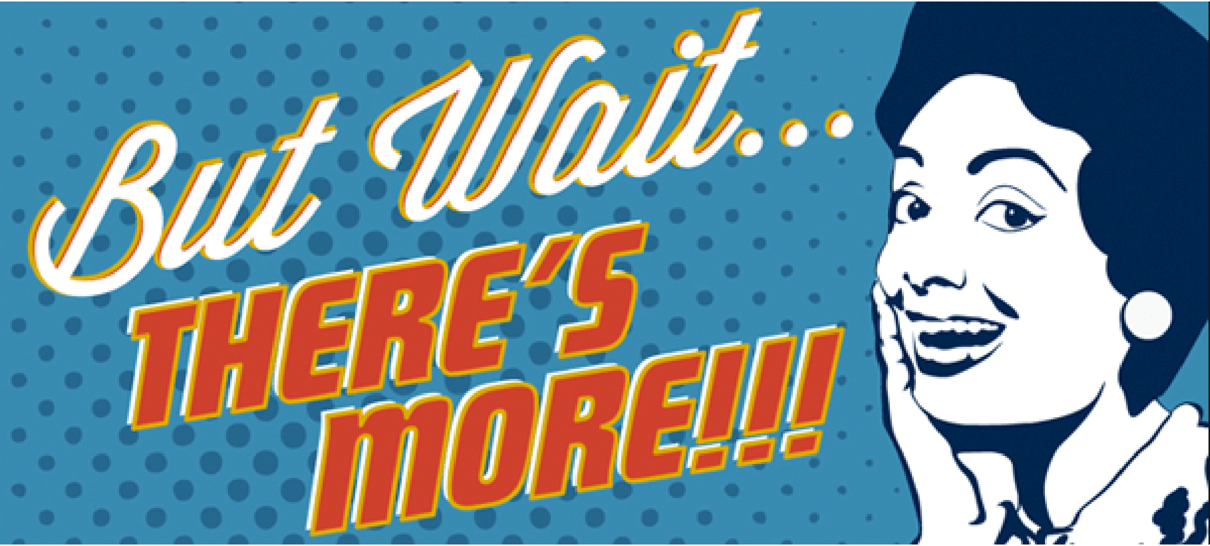
Enter enclave with *RFLAGS.DF=1*



SGX-DF: Inverting enclaved string memory operations

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




But Wait...
THERE'S MORE!!!

x87 Floating Point Unit (FPU) and Streaming SIMD Extensions (SSE)

- Older **x87** high-precision floating-point unit: FPU control word
- Newer **SSE** vector floating-point operations: MXCSR register

x87 Floating Point Unit (FPU) and Streaming SIMD Extensions (SSE)

- Older **x87** high-precision floating-point unit: FPU control word
- Newer **SSE** vector floating-point operations: MXCSR register

gcc utilizes the **x87 for extended precision** when calculating on `long double`

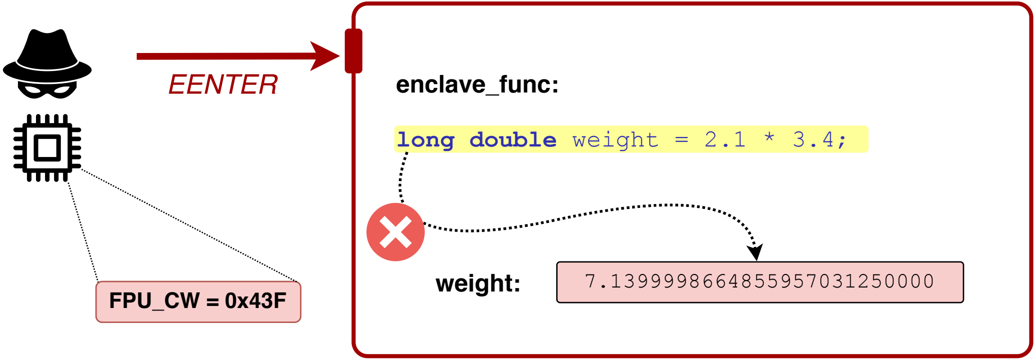
Controlling FPU precision and rounding modes

x86 System-V ABI

The control bits of the `MXCSR` register are callee-saved (preserved across calls), while the status bits are caller-saved (not preserved). The x87 status word register is caller-saved, whereas the x87 control word is callee-saved.

Controlling FPU precision and rounding modes

Corrupt precision and rounding mode. . .



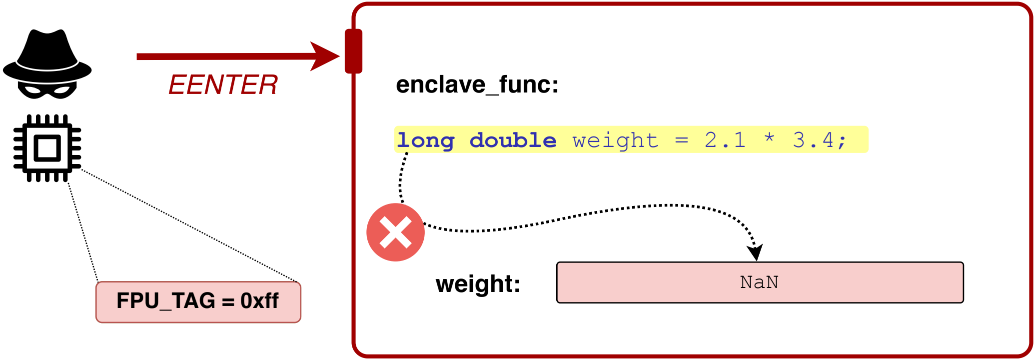
Controlling FPU precision and rounding modes

	SGX-SDK*	OpenEnclave	Graphene	Rust-EDP	Enarx
Exploit Patch 1	★ `xrstor`	○ `ldmxcsr/cw`	○ `fxrstor`	★ `ldmxcsr/cw`	○ `xrstor`

* Includes derived runtimes such as Baidu's Rust-SGX and Google's Asylo.

Fill MMX register to fault FPU operations

Fill XMM registers before entering the enclave

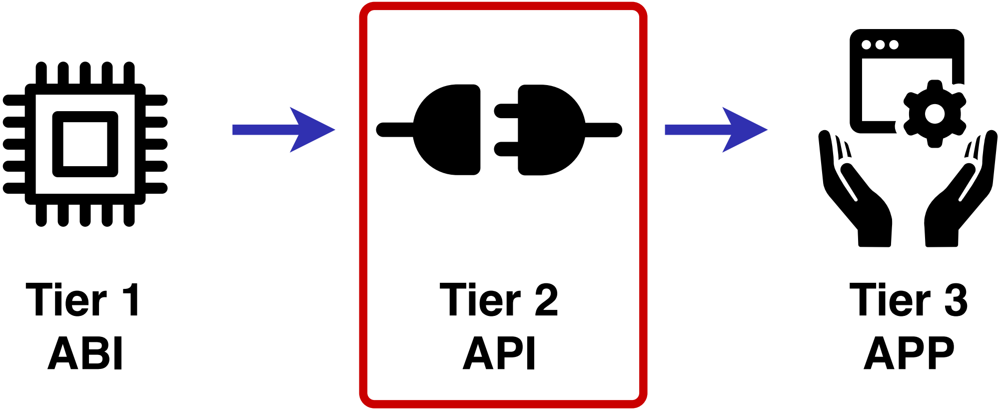


Summary: ABI-level FPU attack surface today

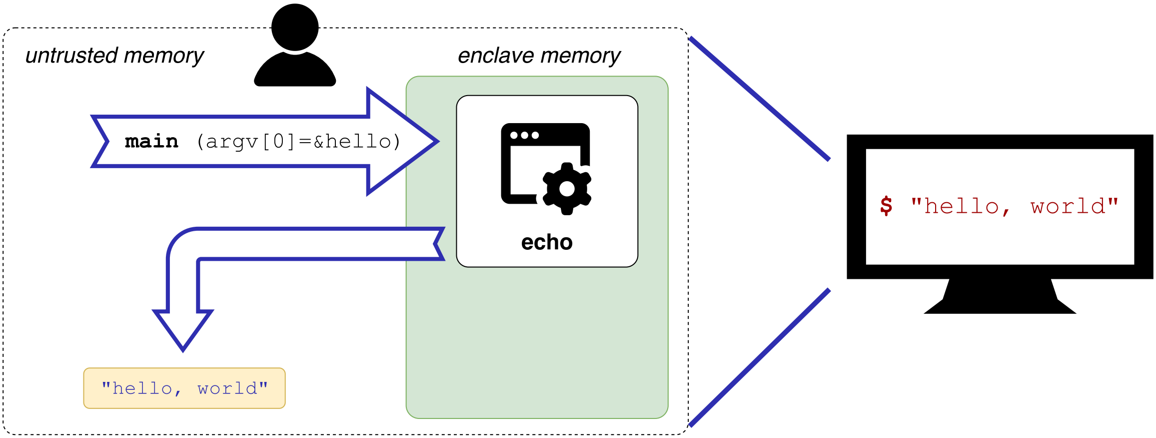
	SGX-SDK*	OpenEnclave	Graphene	Rust-EDP	Enarx
Exploit	★	⯪	○	★	○
Patch 1	`xrstor`	~~`ldmxcsr/cw`~~	`fxrstor`	~~`ldmxcsr/cw`~~	`xrstor`
Patch 2		`xrstor`		`xrstor`	

* Includes derived runtimes such as Baidu's Rust-SGX and Google's Asylo.

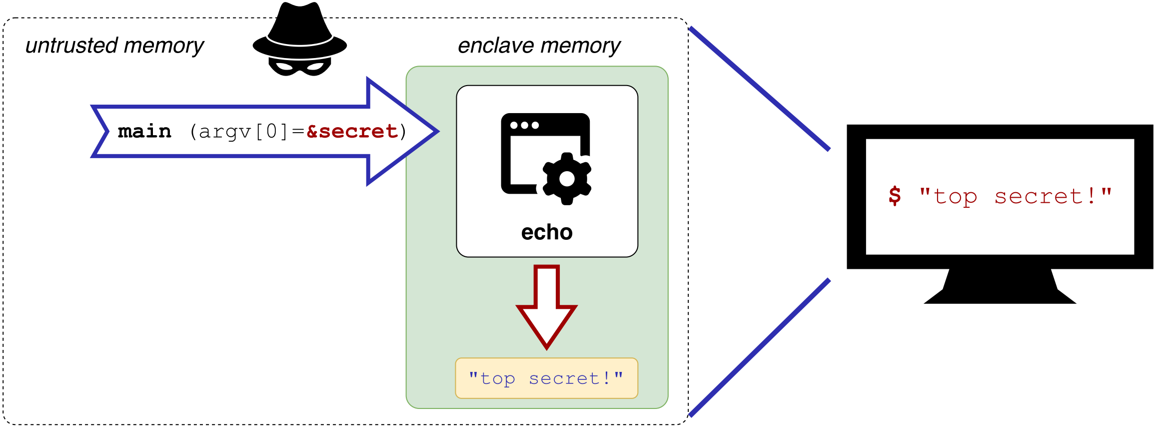
Tier 2: Sanitizing the enclave API



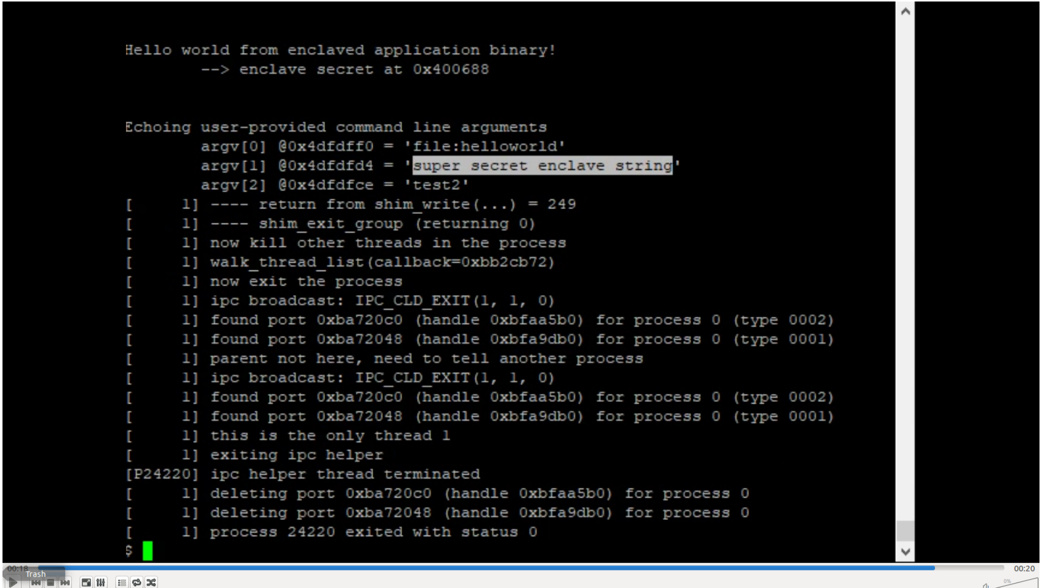
Validating pointer arguments: Confused deputy attacks



Validating pointer arguments: Confused deputy attacks



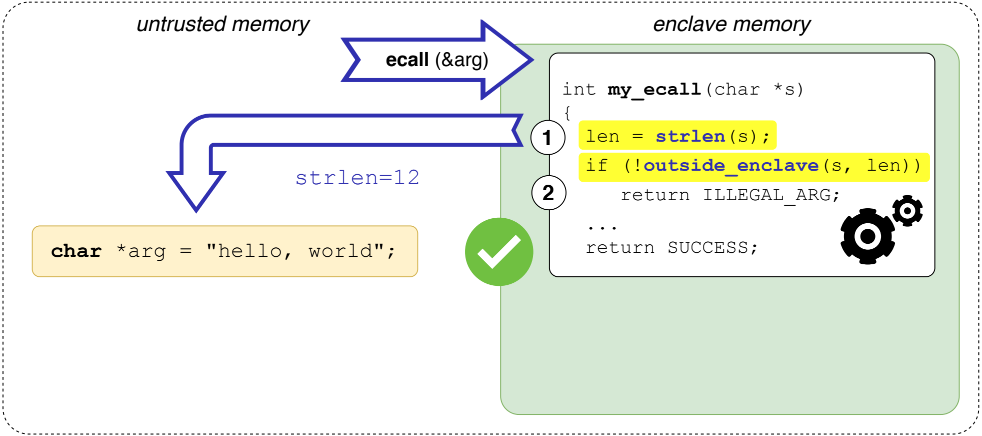
Validating pointer arguments: Confused deputy attacks



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

CVE-2018-3626

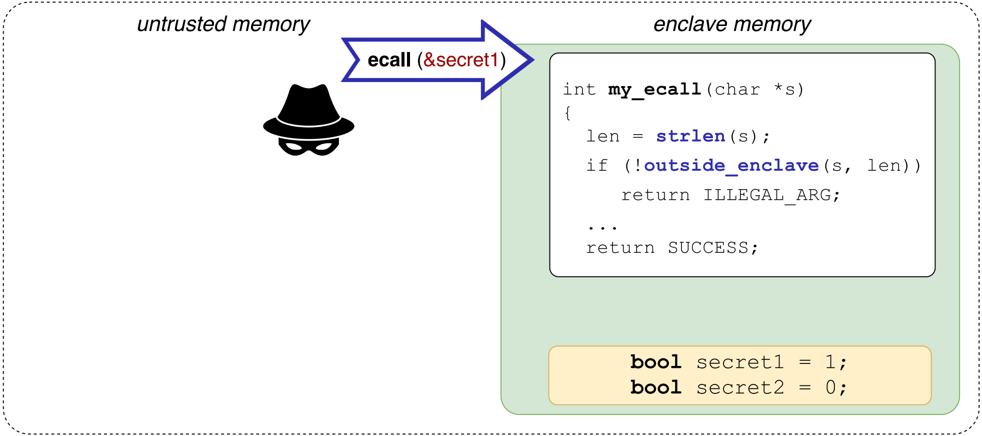
Idea: 2-stage approach ensures string arguments fall *entirely* outside enclave



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

CVE-2018-3626

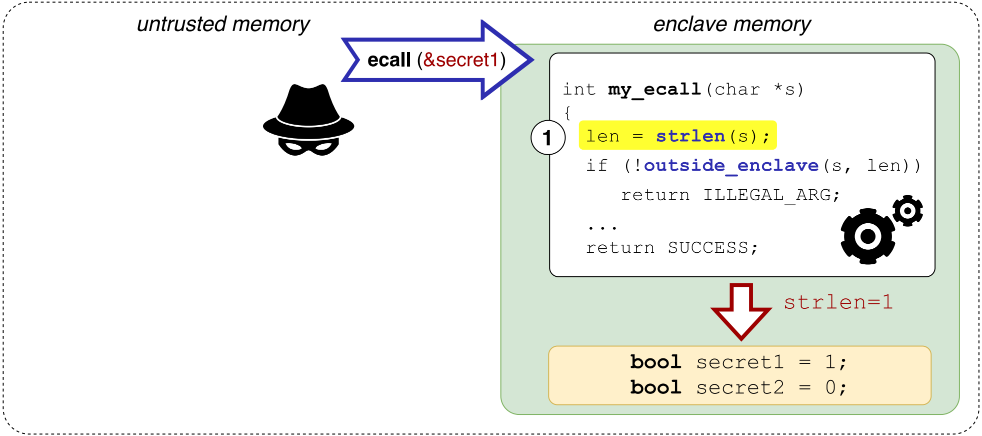
. . . **but** what if we try passing an illegal, in-enclave pointer anyway?



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

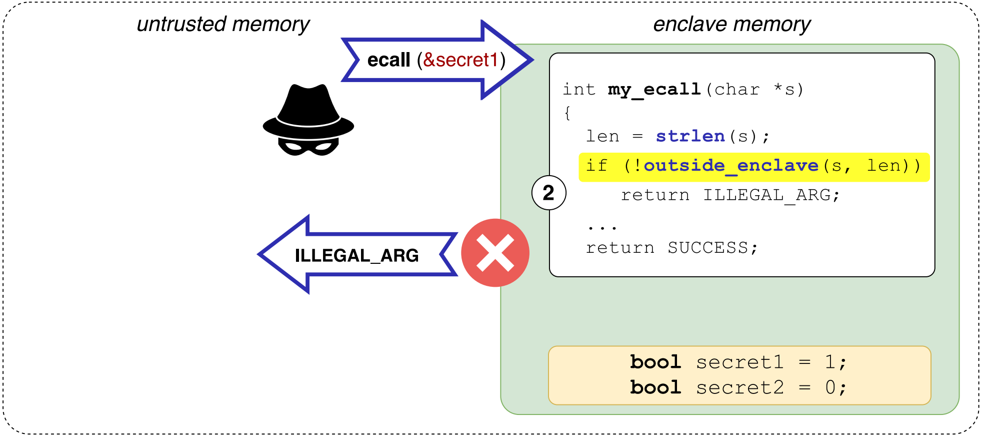
CVE-2018-3626

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



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

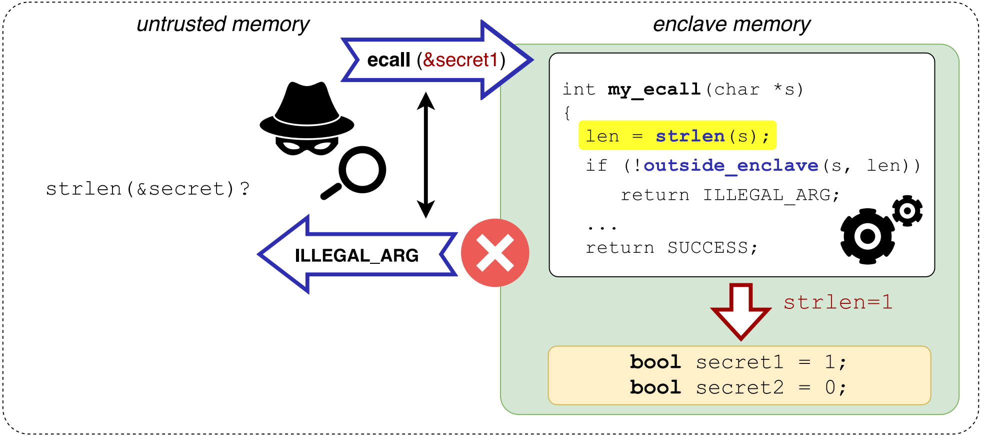
. . . and only **afterwards verifies** whether *entire string* falls outside enclave



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

CVE-2018-3626

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




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

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

SGX-Step

```
…
aesenc xmm0
aesenc xmm0
aesenclast xmm0
…
```

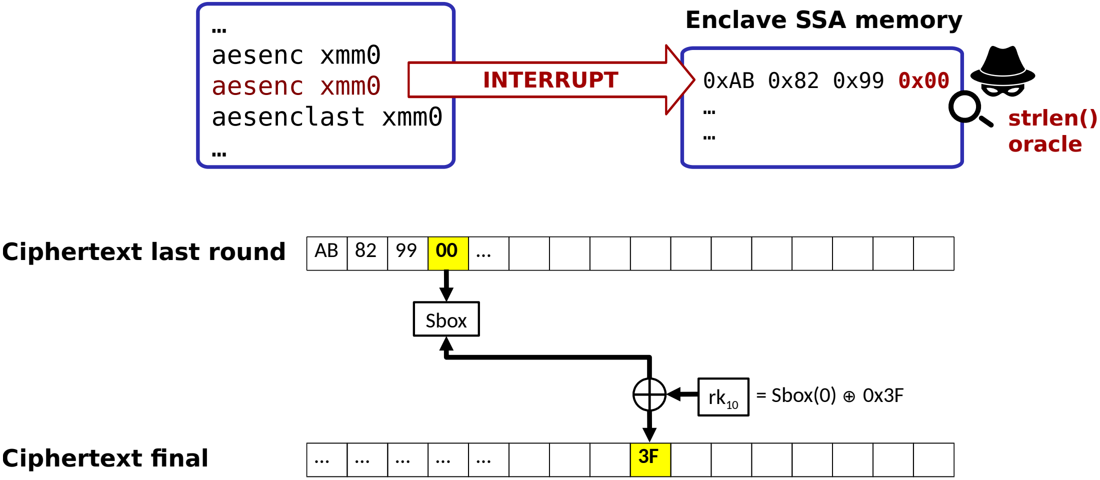
INTERRUPT

(store registers)

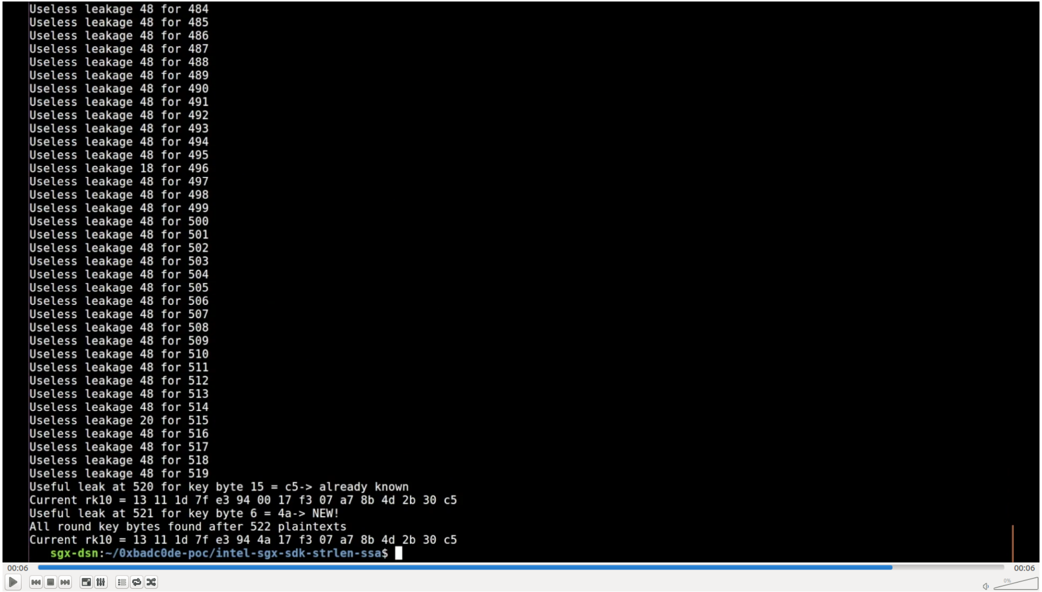
Enclave SSA memory

```
0xAB 0x82 0x99 0x00
…
…
```

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



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



Summary: API-level attack surface

	Runtime / Vulnerability	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!


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

The Tale Continues: Pitfalls and Best Practices for SGX Shielding Runtimes

Jo Van Bulck[1] Fritz Alder[1]

Joint work with

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

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

2nd Intel SGX Community Workshop, July 14, 2020

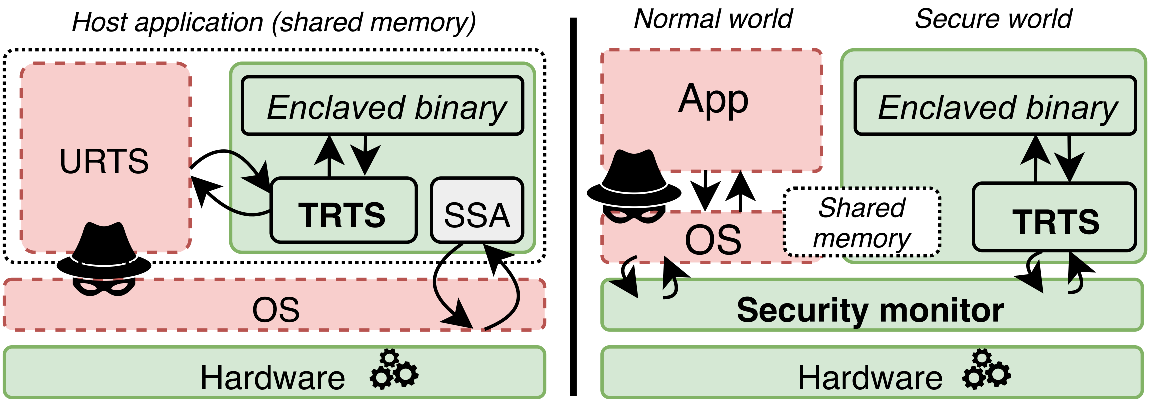
References I

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

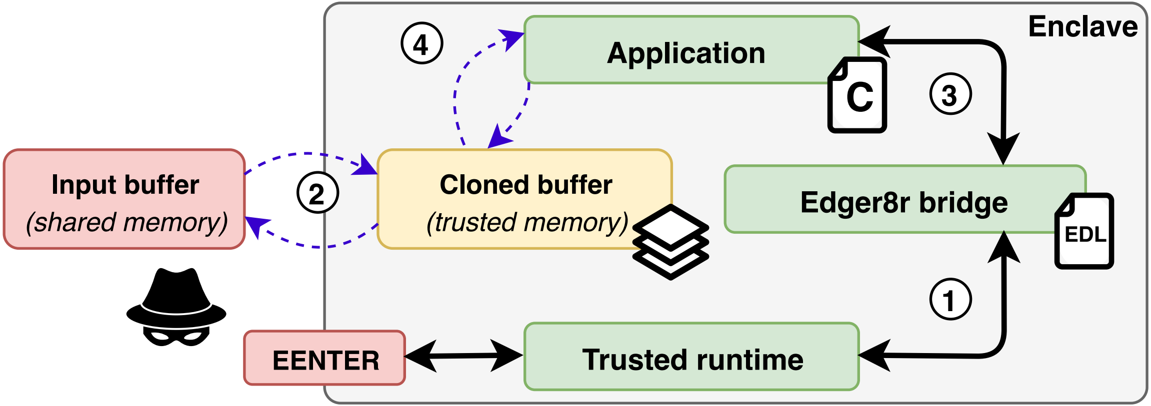
J. Van Bulck, F. Piessens, and R. Strackx.
SGX-Step: A practical attack framework for precise enclave execution control.
In *SysTEX*, pp. 4:1–4:6, 2017.

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

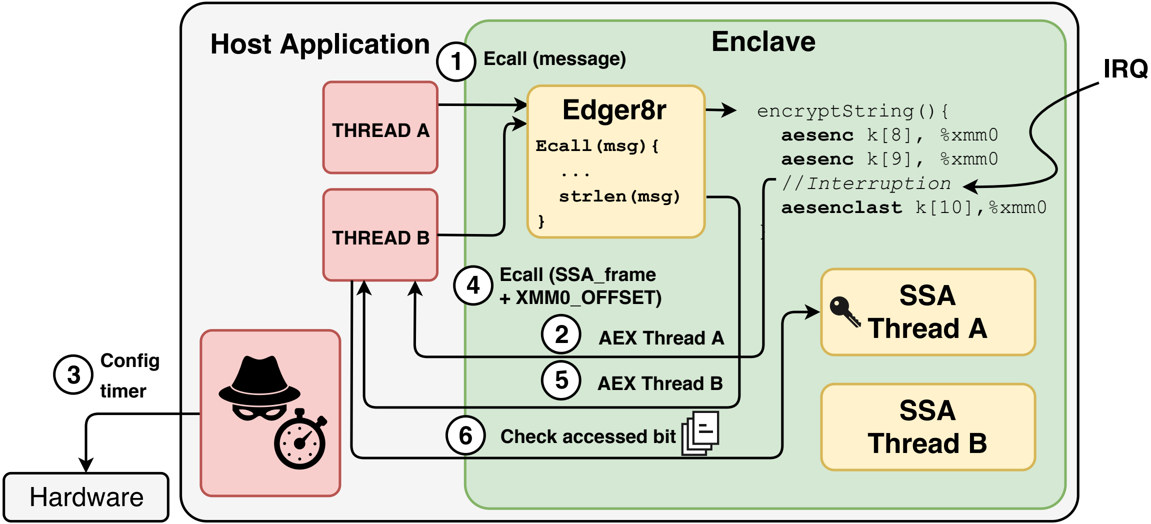
TEE design: Single-address-space vs. world-shared memory approaches



edger8r: Input/output buffer cloning



Intel SGX strlen oracle attack



Exploitation challenges: Building a precise null byte oracle

Goal: Precisely count number of executed `strlen()` loop iterations?

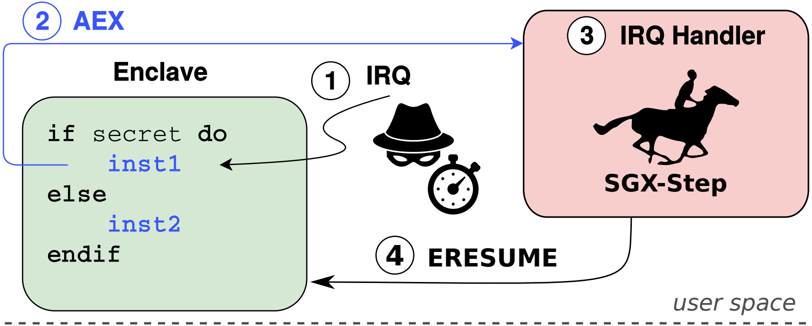
```c
size_t strlen (char *str)
{
    char *s;

    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

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]

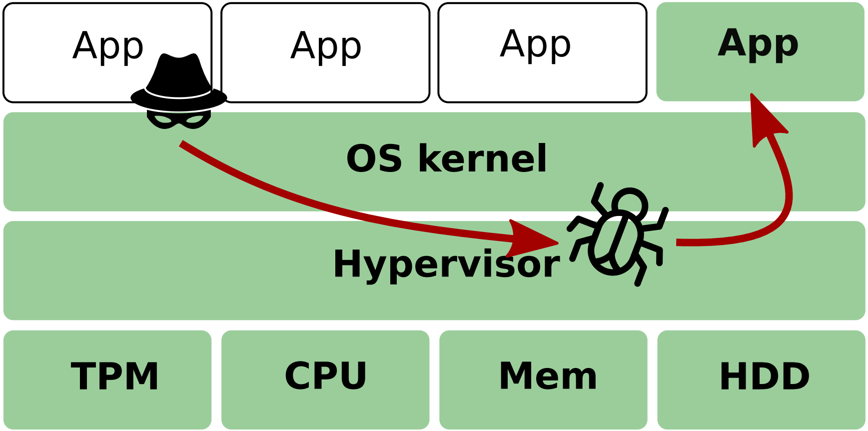
https://github.com/jovanbulck/sgx-step

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.

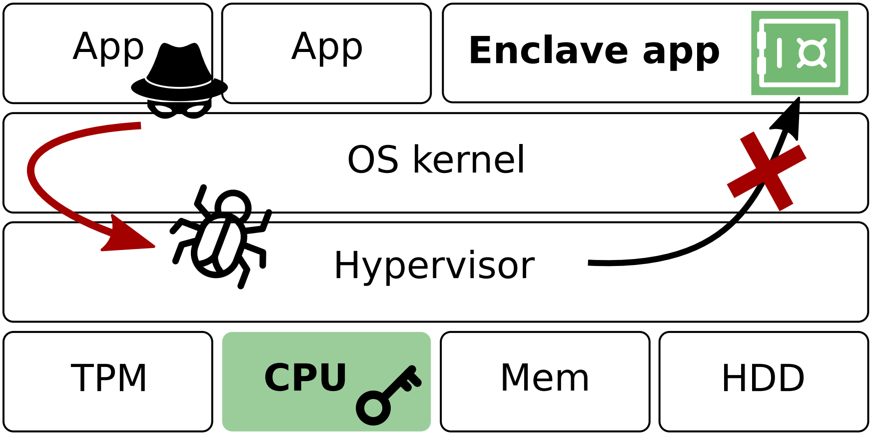
while not full key K recovered **do**
  $(P, C, L) \leftarrow$ random plaintext, associated ciphertext, strlen oracle
  if $L < 16$ **then**
    $K[SR(L)] \leftarrow C[SR(L)] \oplus S(0)$
  end if
end while

The big picture: Enclaved execution attack surface



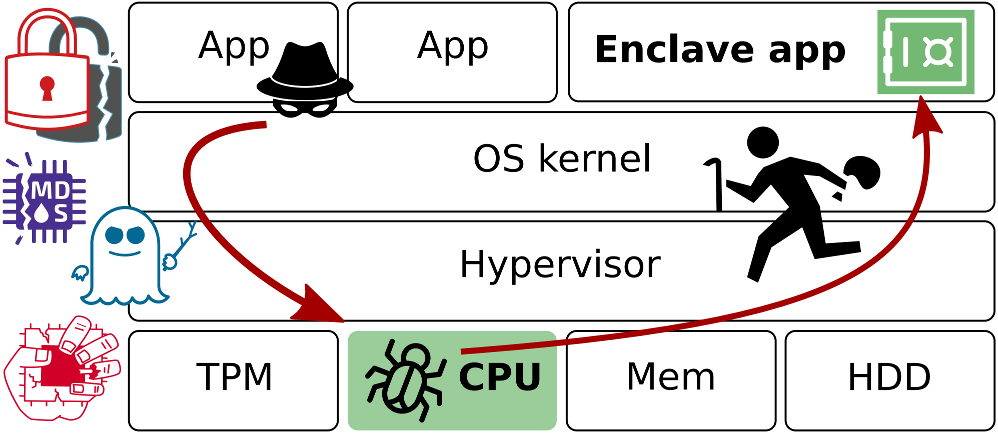
Traditional **layered designs:** large trusted computing base

The big picture: Enclaved execution attack surface



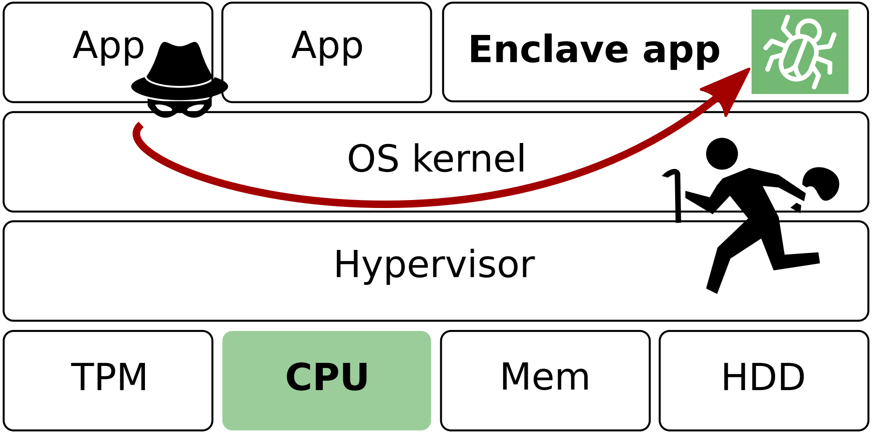
Intel SGX promise: hardware-level **isolation and attestation**

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?