

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

Jo Van Bulck¹ Fritz Alder¹

Joint work with

David Oswald² Eduard Marin² Abdulla Aldoseri² Flavio D. Garcia² Frank Piessens¹

¹imec-DistriNet, KU Leuven, Belgium ²The 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?

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 Firefox

Keystone | An Open Framework for Architecting TEEs

https://keystone-enc...

The Keystone project website features a large logo and the text "An Open Framework for Architecting Trusted Execution Environments". It includes a "View on GitHub" button and a "Open Enclave SDK" section describing the purpose of the framework.

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

Over the past three years, significant experience has been gained with applications of Sancus, and several extensions of the architecture have been investigated –

source building blocks and free-software ethos that attempts to provide a layer of integrity and deterministic timers should be lauded and considered by anyone building hardware applications where security and

Graphene - Mozilla Firefox

Graphene

https://gr...

The Graphene project website features a large logo and the text "GRAPHENE". It includes a "View on GitHub" button and a "Graphene - a Library OS for Unmodified Applications" section.

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

Over the past three years, significant experience has been gained with applications of Sancus, and several extensions of the architecture have been investigated –

SDK for Intel® Software Guard Extensions | Intel® Software

intel

intel(R) Soft... Intel(R) So... Intel(R) So... intel/lin

https://intel...

120%

intel Developer Zone

INTEL® SOFTWARE GUARD EXTENSIONS

GET STARTED WITH THE SDK

GOOGLE CLOUD PLATFORM

Introducing Asylo: an open framework for confidential computing

It sup... a soft... d soft...

The Intel Software Guard Extensions developer zone page features a "GET STARTED WITH THE SDK" button. The Google Cloud Platform page features a "Introducing Asylo: an open framework for confidential computing" section.

Sancus: Lightweight and Open-Source Trusted Computing for the IoT

[View on GitHub](#)

[Watch a demo](#)

[Explore Research](#)

The screenshot shows a web browser window with the title "Keystone | An Open Framework for Architecting TEEs - Mozilla Firefox". The main content area displays a paragraph of text: "source building blocks and free-software ethos that attempts to provide a layer of integrity and deterministic behaviors should be lauded and considered by anyone building hardware applications where security and requirements." Below this, there are several tabs open, including "SDK for Intel® Software Guard Extensions | Intel® Software" and "SDK for Intel® Software Guard Extensions | Intel® Software".

What do these projects have in common?

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](#)

for unmodified Applications

The screenshot shows the Fortanix EDP website. The header reads "ENCLAVE DEVELOPMENT PLATFORM". Below it, a sub-header says "The Fortanix EDP is the preferred way for writing Intel® SGX applications from scratch." A "GETTING STARTED" button is visible at the bottom. The background features large purple gears.

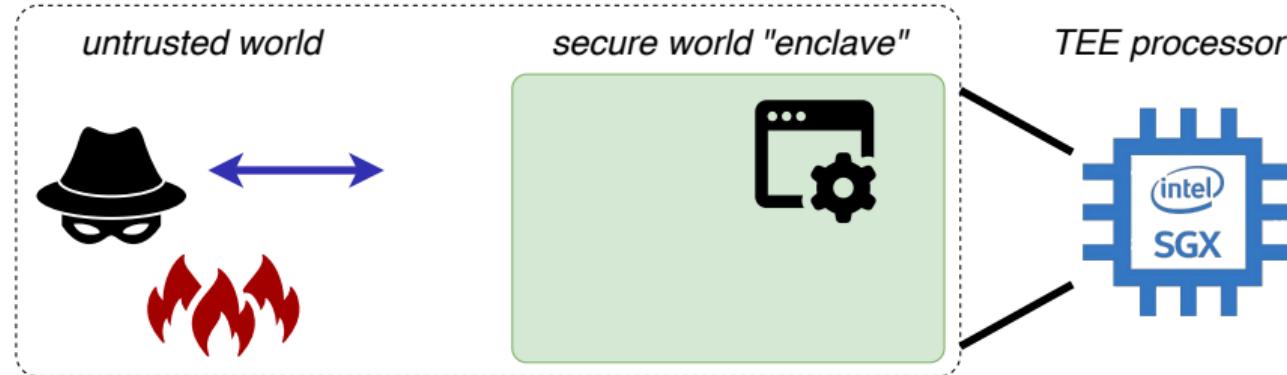
GOOGLE CLOUD PLATFORM

Introducing Asylo: an open framework for confidential computing



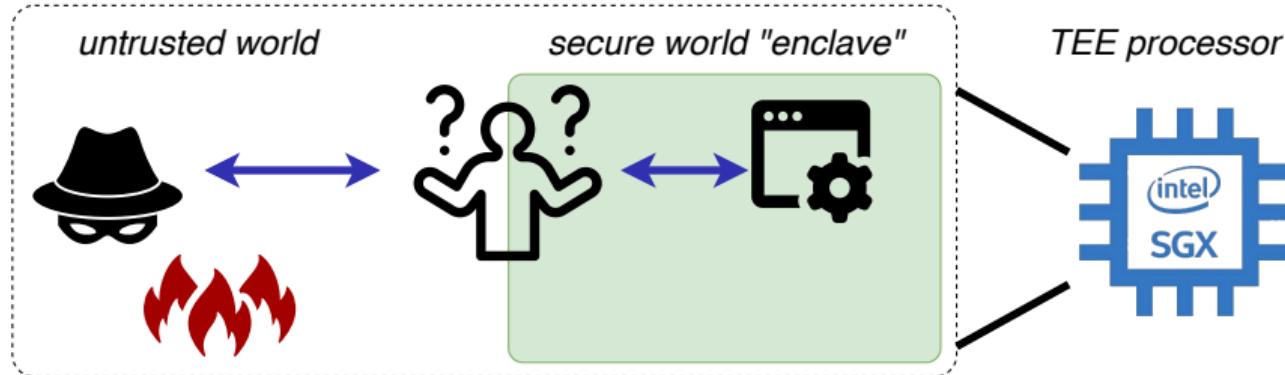
Over the past three years, significant experience has been gained with applications of Sancus, and several extensions of the architecture have been investigated –

Why isolation is not enough: Enclave shielding runtimes



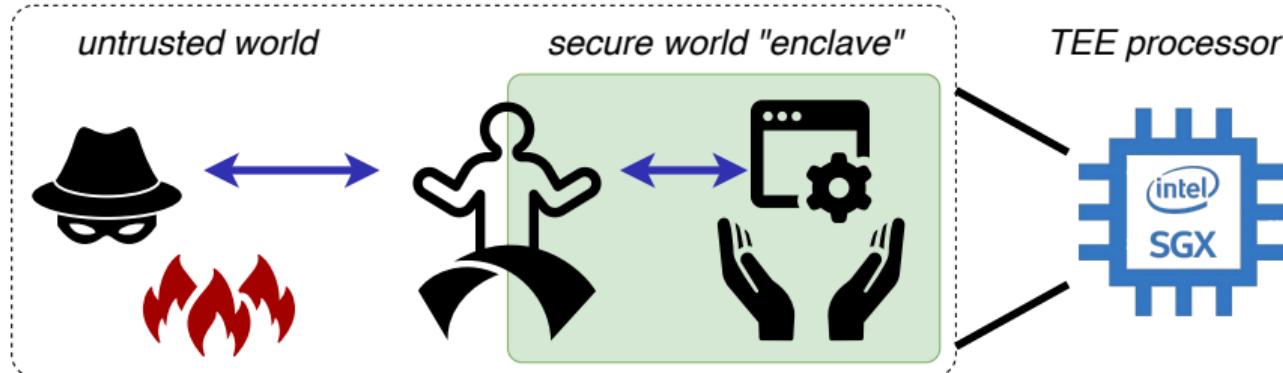
- TEE promise: enclave == “secure oasis” in a **hostile environment**

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

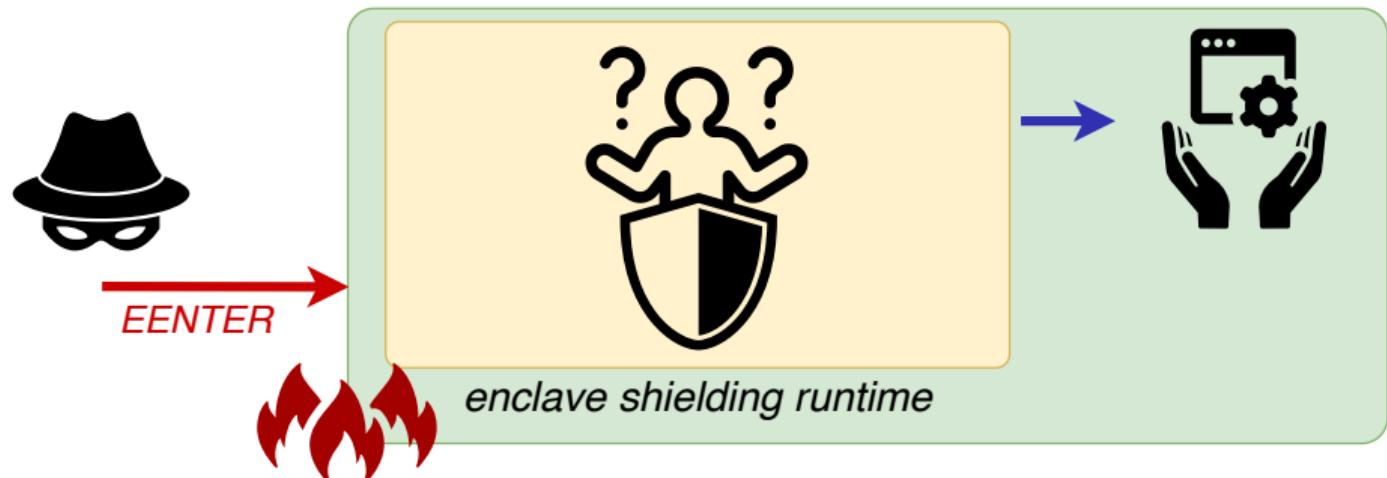


A photograph of a suspension bridge made of wood and metal cables, spanning a deep blue-green river. Several people are walking across the bridge, some carrying items. The bridge is supported by tall wooden poles and green metal towers.

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

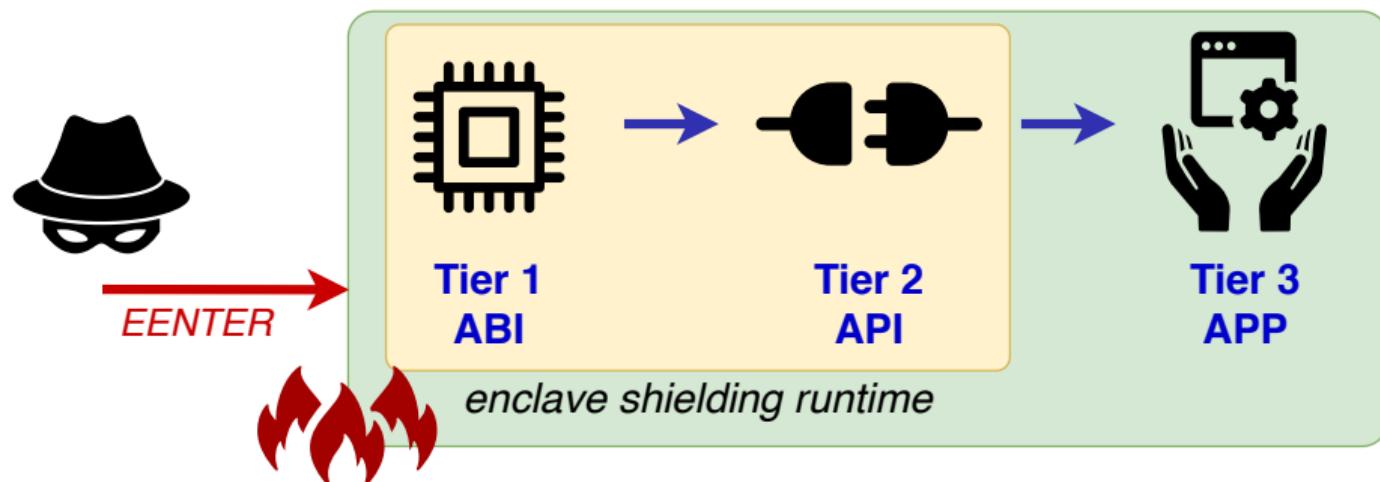
Enclave shielding responsibilities

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

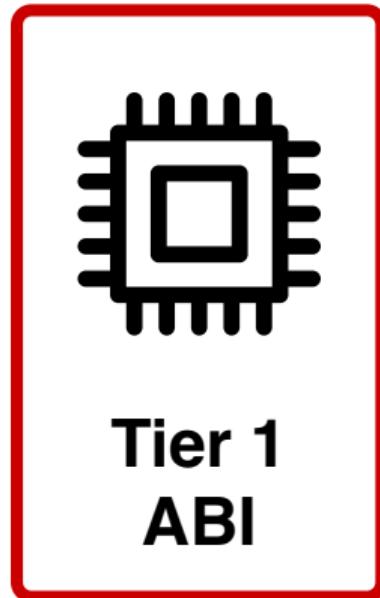


Enclave shielding responsibilities

⚠️ 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



**Tier 1
ABI**

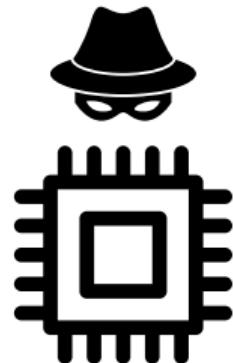


**Tier 2
API**



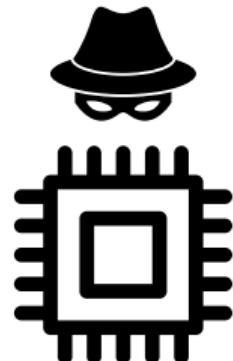
**Tier 3
APP**

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

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



```
1 lea rdi, buf
2 mov al, 0x0
3 mov ecx, 100
4 rep stos [rdi], al
```

x86 string instructions: Direction Flag (DF) operation

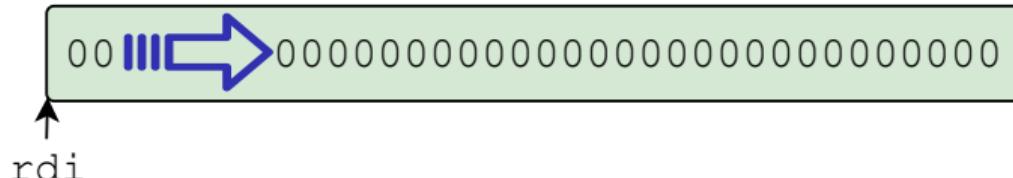


- Special x86 rep string instructions to speed up streamed memory operations
- Default operate **left-to-right**

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



```
1 lea rdi, buf
2 mov al, 0x0
3 mov ecx, 100
4 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*

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



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

00000000000000000000000000000000

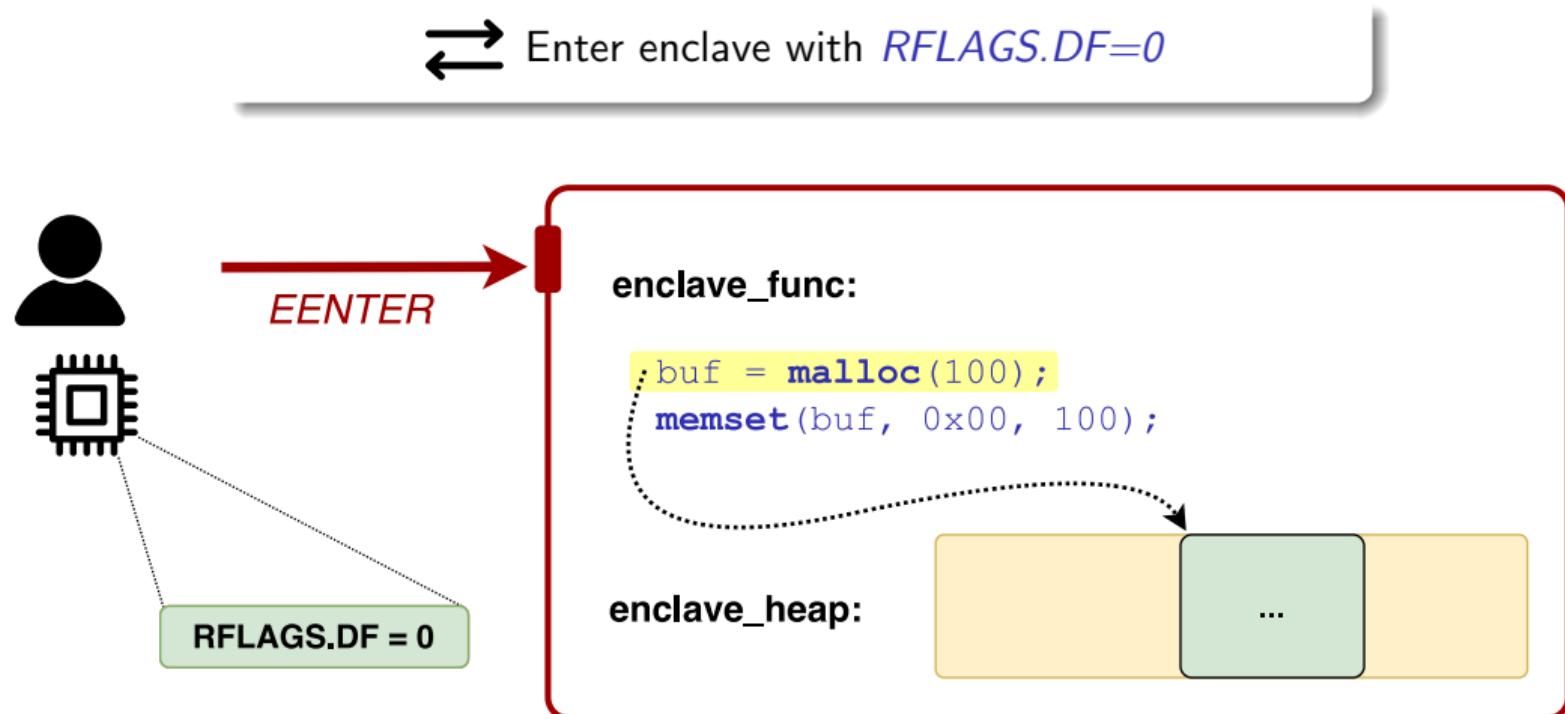


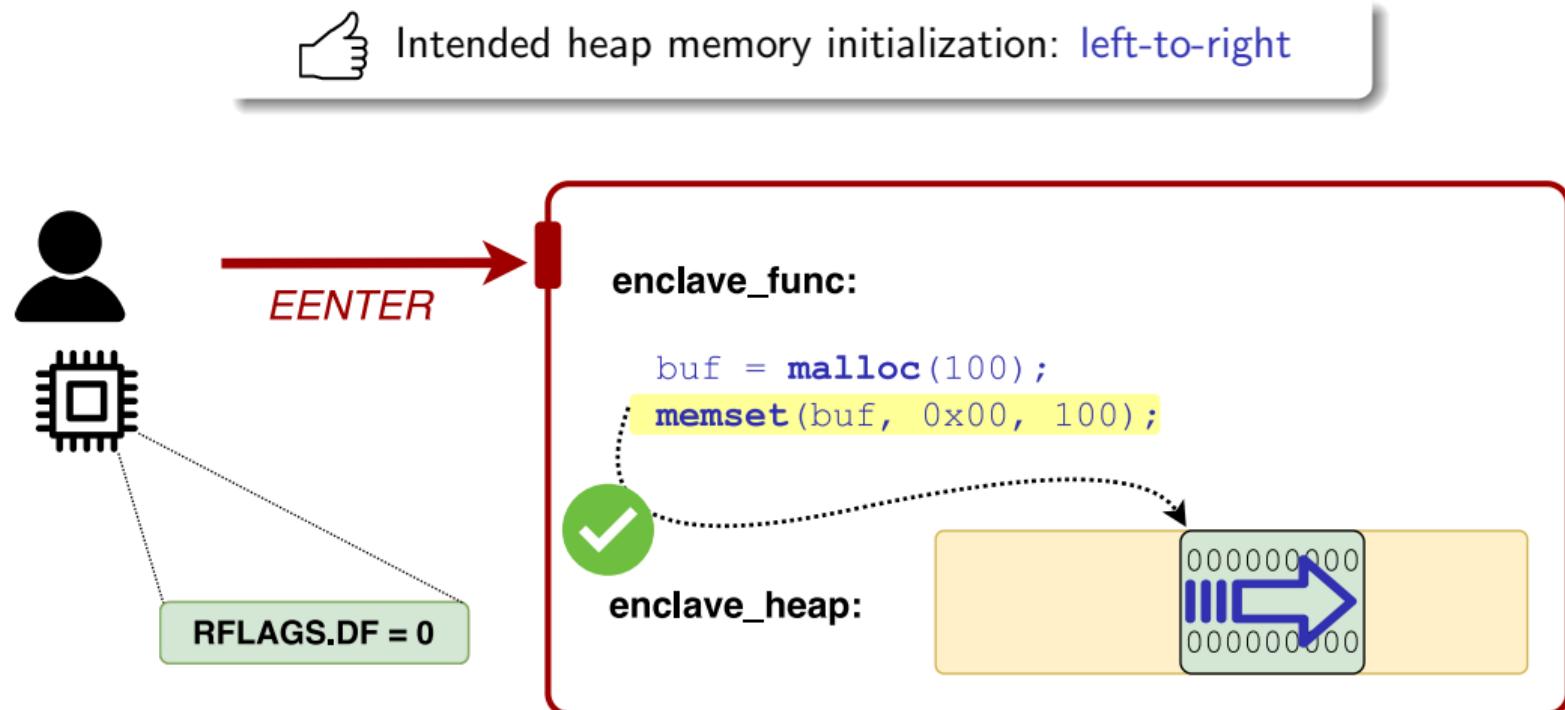
rdi

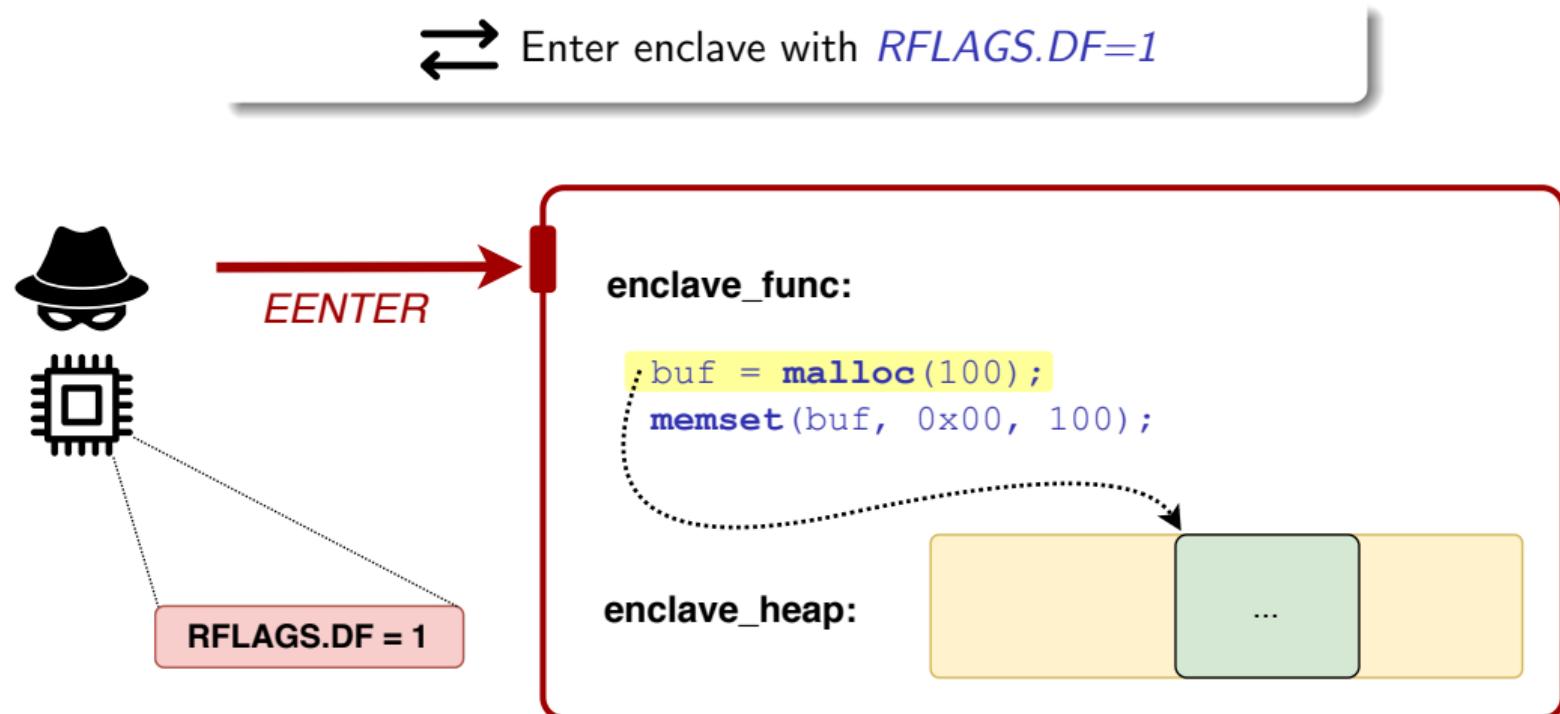
x86 System-V ABI



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.

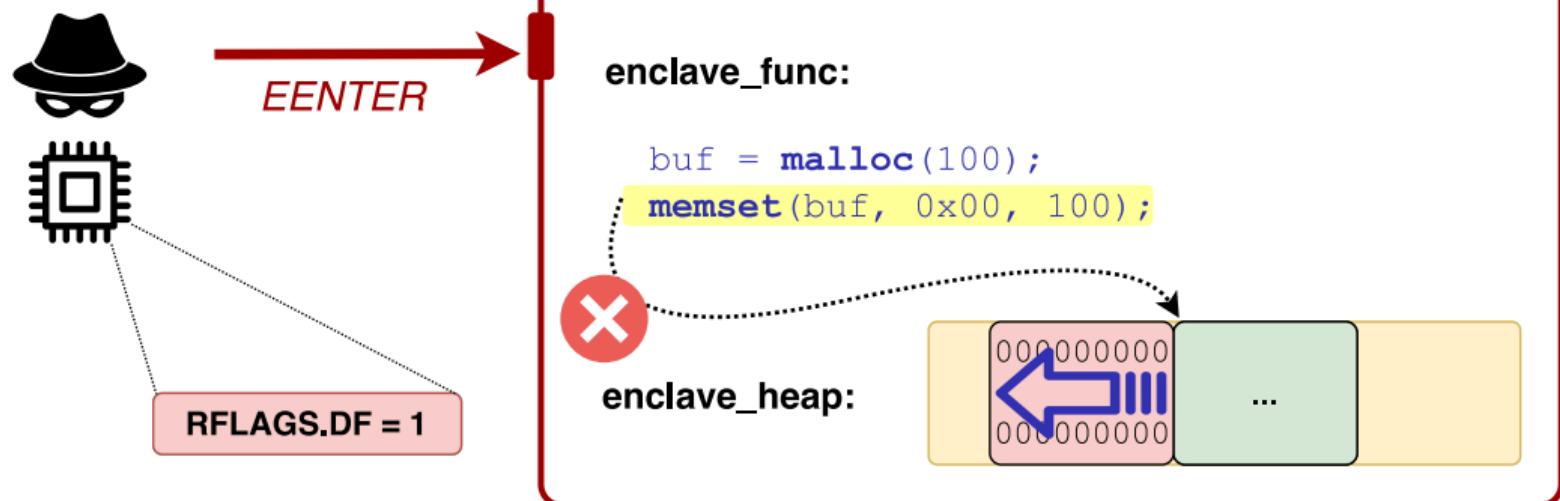








Enclave heap **memory corruption**: right-to-left...



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](#)

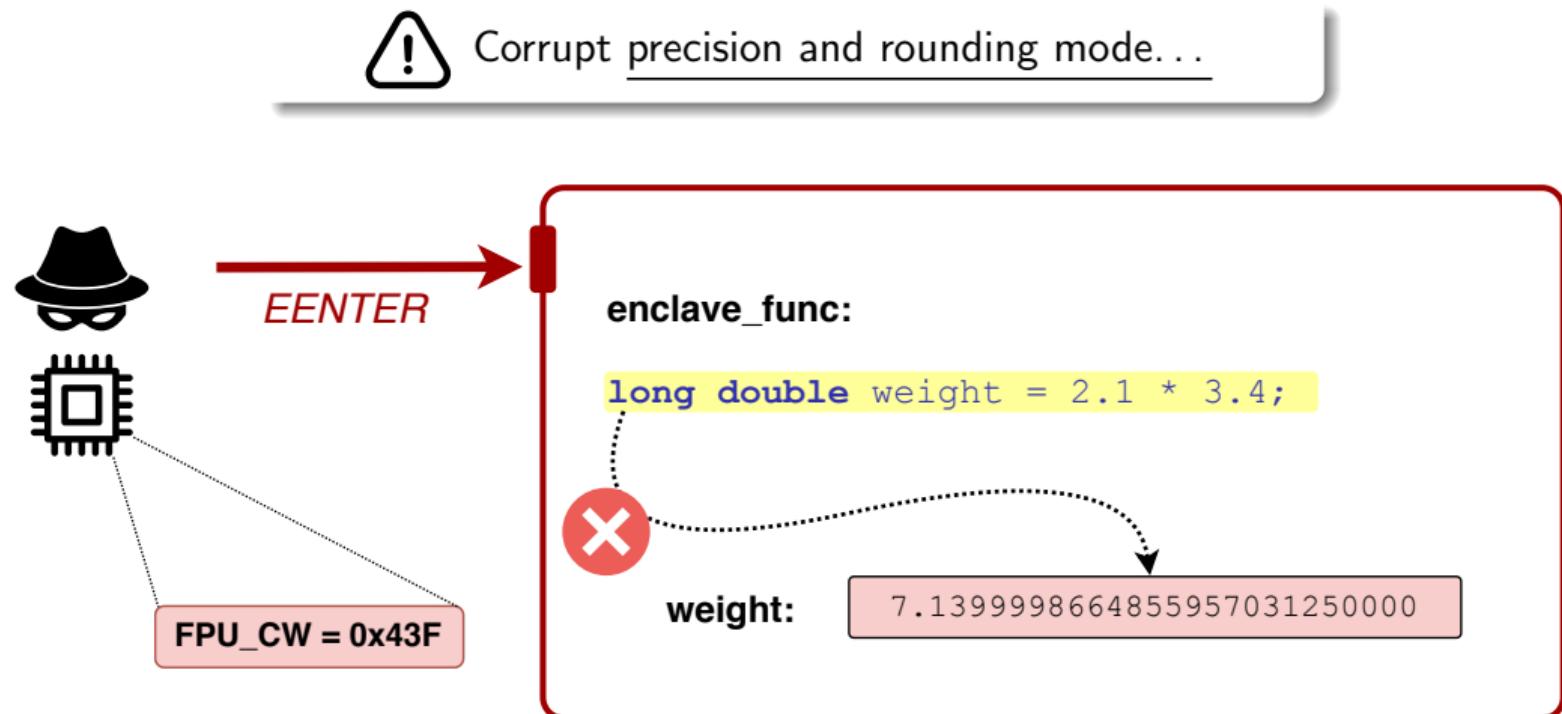
x86 System-V ABI



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

CVE-2020-0561



Controlling FPU precision and rounding modes

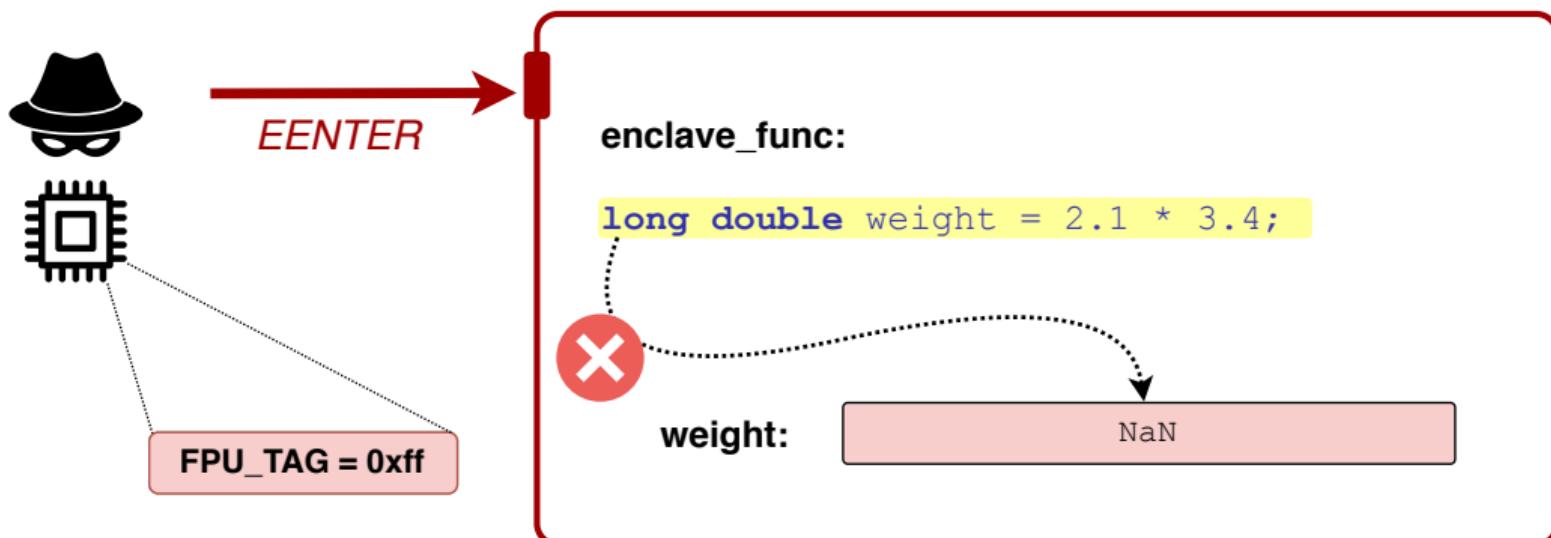
CVE-2020-0561

	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 XMM registers before entering the enclave

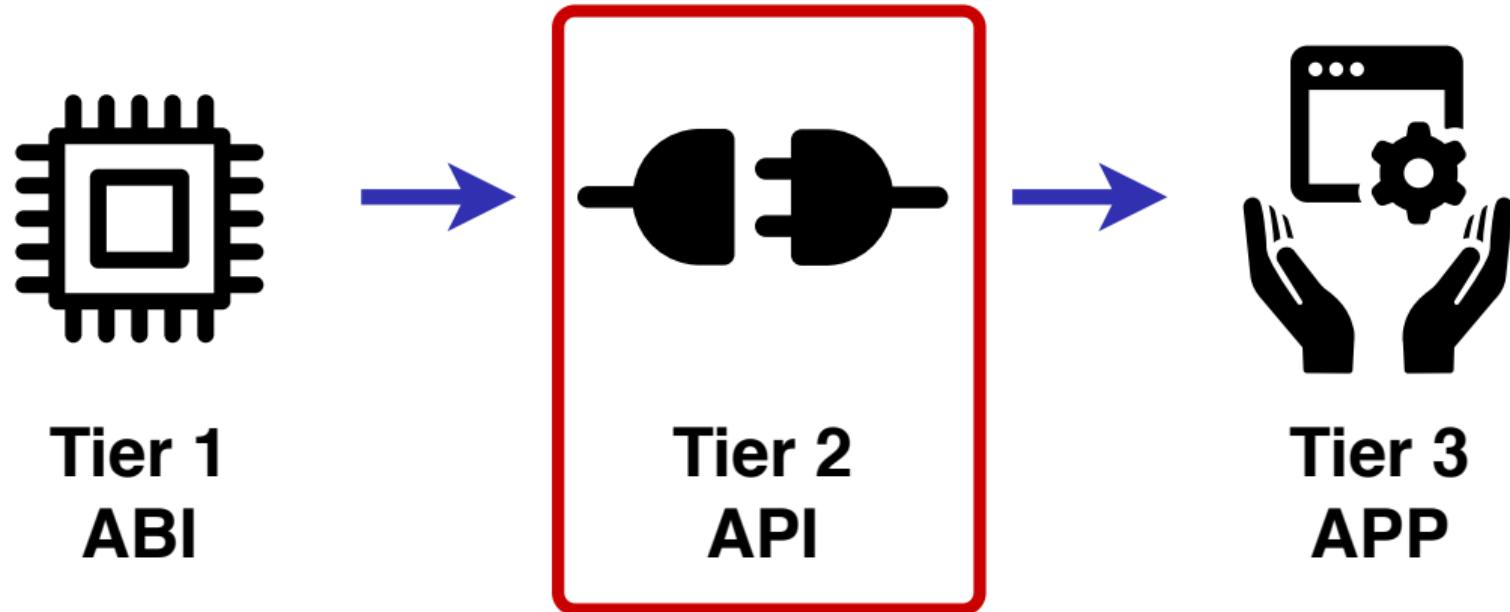


Summary: ABI-level FPU attack surface today

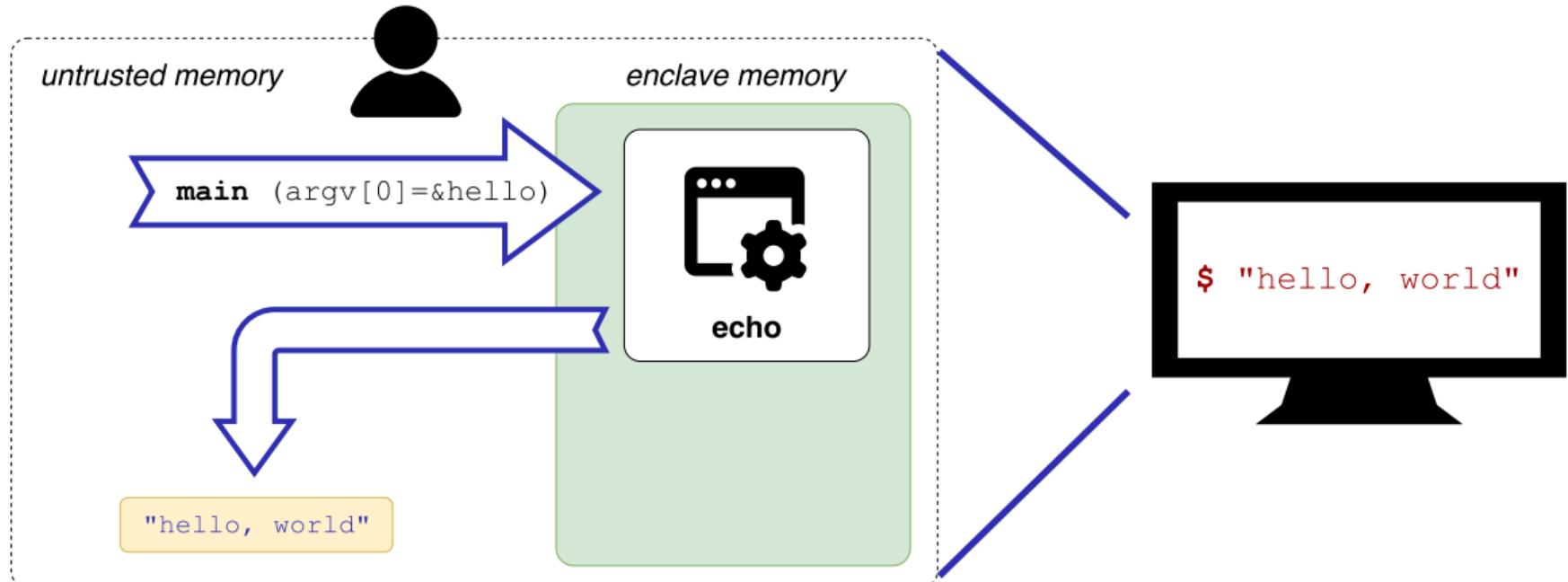
	SGX-SDK*	OpenEnclave	Graphene	Rust-EDP	Enary
Exploit	★	★	○	★	○
Patch 1	xrstor	ldmxcsr/cw	fxrstor	ldmxcsr/cw	xrstor
Patch 2		xrstor		xrstor	

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

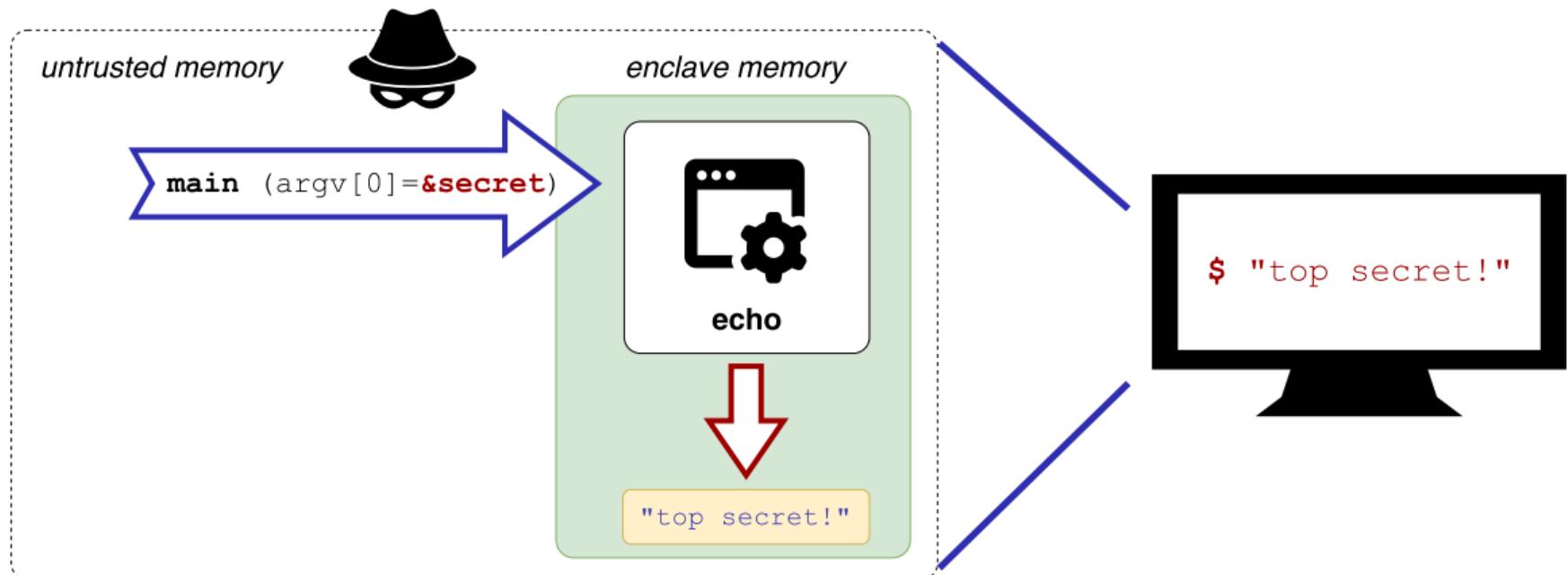
Tier 2: Sanitizing the enclave API



Validating pointer arguments: Confused deputy attacks



Validating pointer arguments: Confused deputy attacks



Validating pointer arguments: Confused deputy attacks

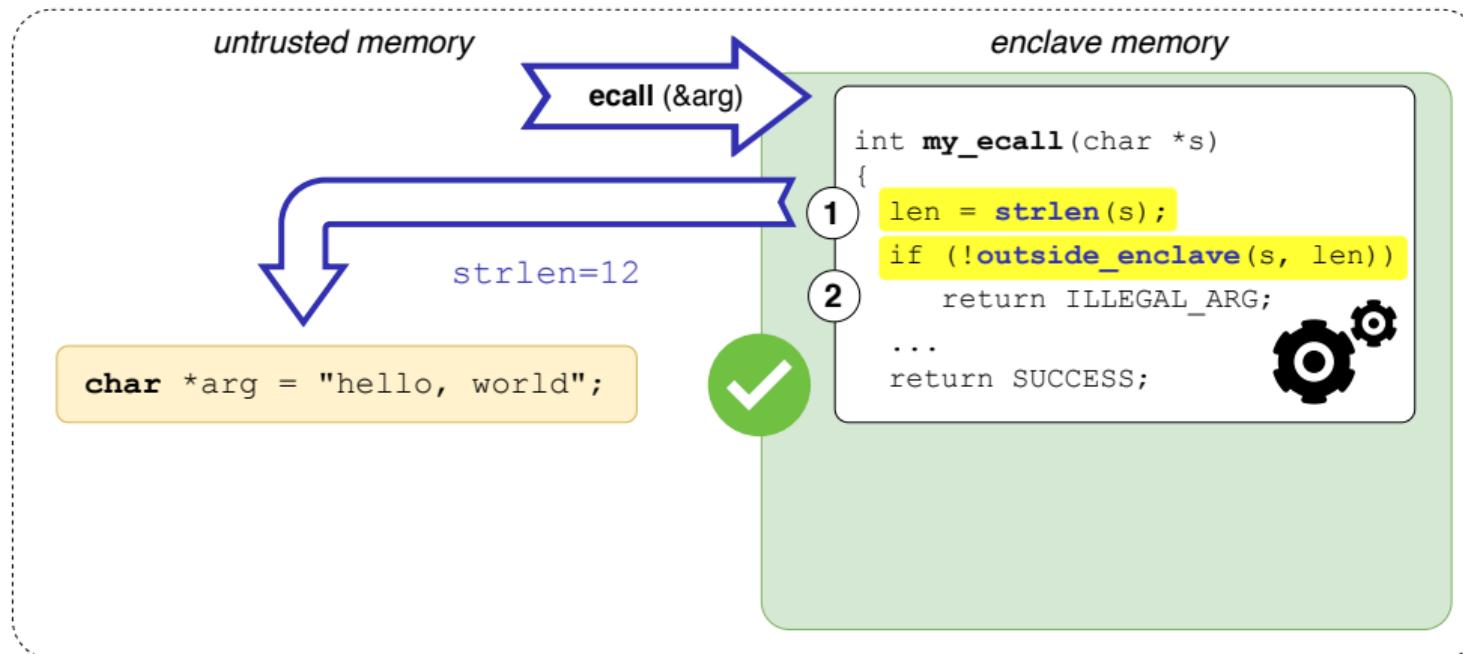
```
Hello world from enclaved application binary!
--> enclave secret at 0x400688

Echoing user-provided command line arguments
    argv[0] @0x4dfdfff0 = 'file:helloworld'
    argv[1] @0x4dfdfdf4 = 'super secret enclave string'
    argv[2] @0x4dfdffce = 'test2'
[ 1] ---- return from shim_write(...) = 249
[ 1] ---- shim_exit_group (returning 0)
[ 1] now kill other threads in the process
[ 1] walk_thread_list(callback=0xbb2cb72)
[ 1] now exit the process
[ 1] ipc broadcast: IPC_CLD_EXIT(1, 1, 0)
[ 1] found port 0xba720c0 (handle 0xbfaa5b0) for process 0 (type 0002)
[ 1] found port 0xba72048 (handle 0xbfa9db0) for process 0 (type 0001)
[ 1] parent not here, need to tell another process
[ 1] ipc broadcast: IPC_CLD_EXIT(1, 1, 0)
[ 1] found port 0xba720c0 (handle 0xbfaa5b0) for process 0 (type 0002)
[ 1] found port 0xba72048 (handle 0xbfa9db0) for process 0 (type 0001)
[ 1] this is the only thread 1
[ 1] exiting ipc helper
[P24220] ipc helper thread terminated
[ 1] deleting port 0xba720c0 (handle 0xbfaa5b0) for process 0
[ 1] deleting port 0xba72048 (handle 0xbfa9db0) for process 0
[ 1] process 24220 exited with status 0
$
```





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



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

untrusted memory



ecall (&secret1)

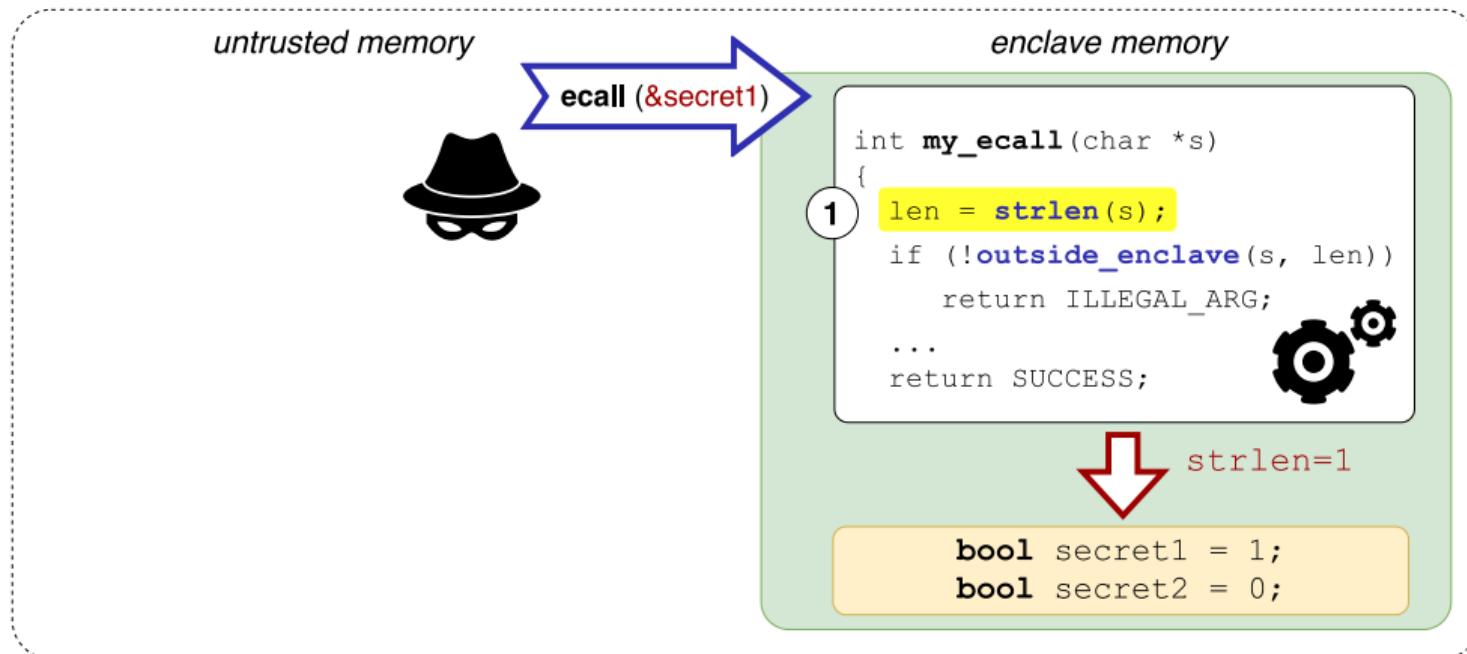
enclave memory

```
int my_ecall(char *s)
{
    len = strlen(s);
    if (!outside_enclave(s, len))
        return ILLEGAL_ARG;
    ...
    return SUCCESS;
```

```
bool secret1 = 1;
bool secret2 = 0;
```

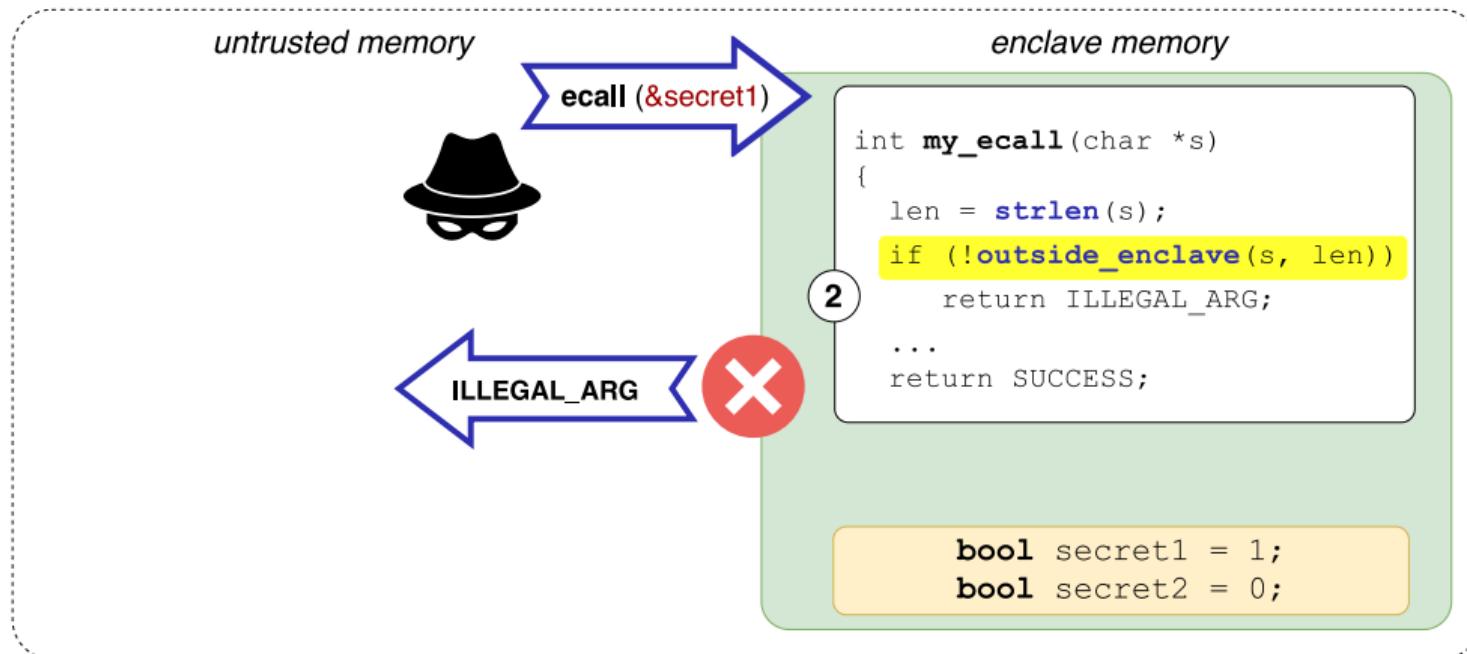


Enclave **first** computes length of secret, in-enclave buffer!



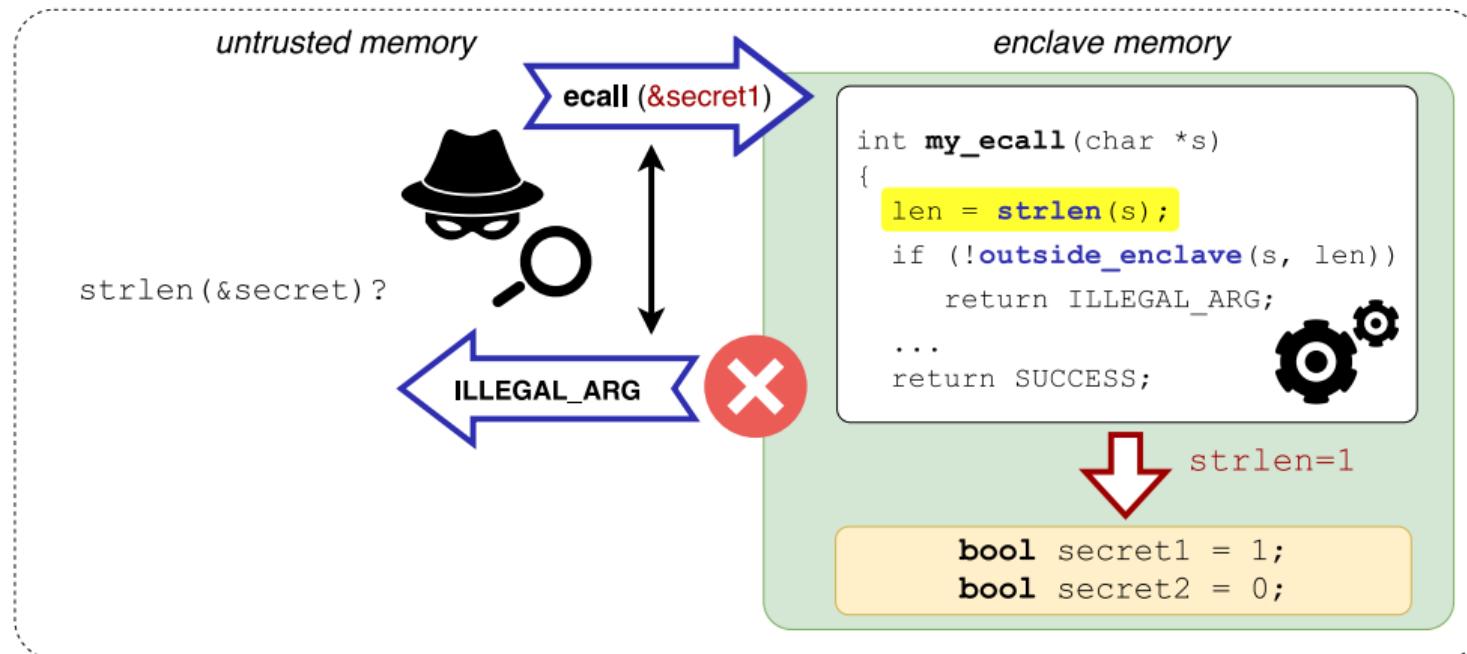


... and only **afterwards verifies** whether *entire string falls outside enclave*





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

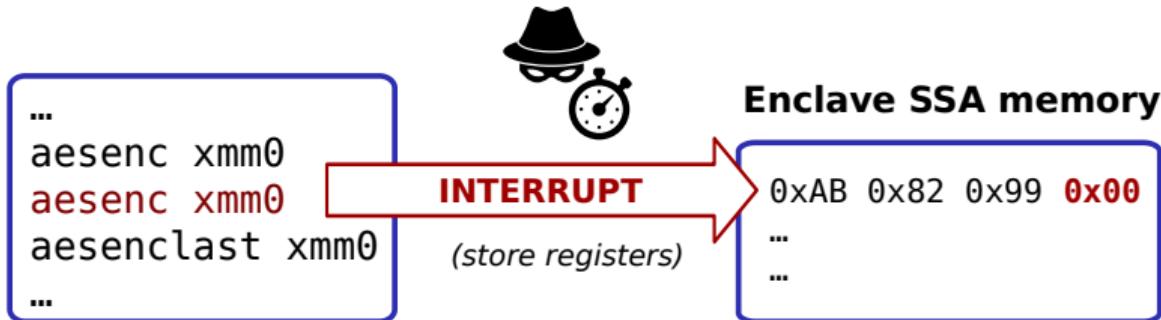


CVE-2018-3626: ALL YOUR ZERO BYTES

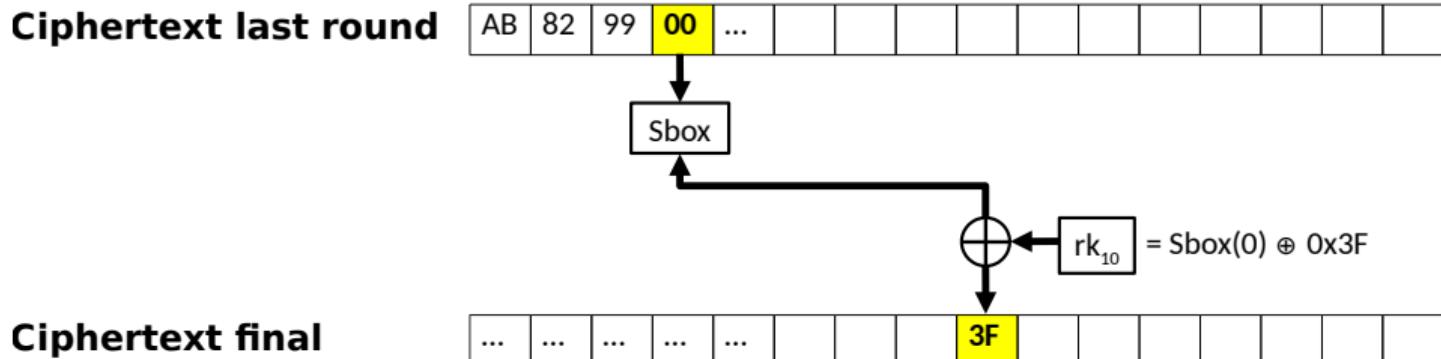
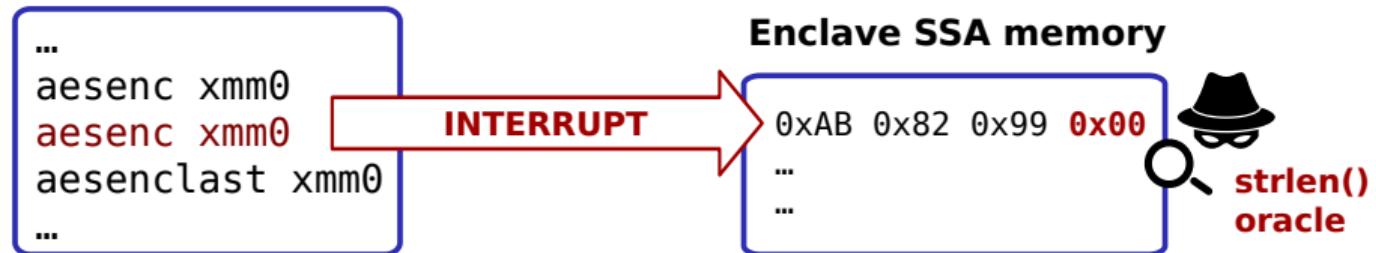


ARE BELONG TO US

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

```
Useless leakage 48 for 484
Useless leakage 48 for 485
Useless leakage 48 for 486
Useless leakage 48 for 487
Useless leakage 48 for 488
Useless leakage 48 for 489
Useless leakage 48 for 490
Useless leakage 48 for 491
Useless leakage 48 for 492
Useless leakage 48 for 493
Useless leakage 48 for 494
Useless leakage 48 for 495
Useless leakage 18 for 496
Useless leakage 48 for 497
Useless leakage 48 for 498
Useless leakage 48 for 499
Useless leakage 48 for 500
Useless leakage 48 for 501
Useless leakage 48 for 502
Useless leakage 48 for 503
Useless leakage 48 for 504
Useless leakage 48 for 505
Useless leakage 48 for 506
Useless leakage 48 for 507
Useless leakage 48 for 508
Useless leakage 48 for 509
Useless leakage 48 for 510
Useless leakage 48 for 511
Useless leakage 48 for 512
Useless leakage 48 for 513
Useless leakage 48 for 514
Useless leakage 20 for 515
Useless leakage 48 for 516
Useless leakage 48 for 517
Useless leakage 48 for 518
Useless leakage 48 for 519
Useful leak at 520 for key byte 15 = c5-> already known
Current rk10 = 13 11 1d 7f e3 94 00 17 f3 07 a7 8b 4d 2b 30 c5
Useful leak at 521 for key byte 6 = 4a-> NEW!
All round key bytes found after 522 plaintexts
Current rk10 = 13 11 1d 7f e3 94 4a 17 f3 07 a7 8b 4d 2b 30 c5
sgx-dsn:~/0xbadc0de-poc/intel-sgx-sdk-strlen-ssa$
```

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!



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 Bulck¹ Fritz Alder¹

Joint work with

David Oswald² Eduard Marin² Abdulla Aldoseri² Flavio D. Garcia² Frank Piessens¹

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

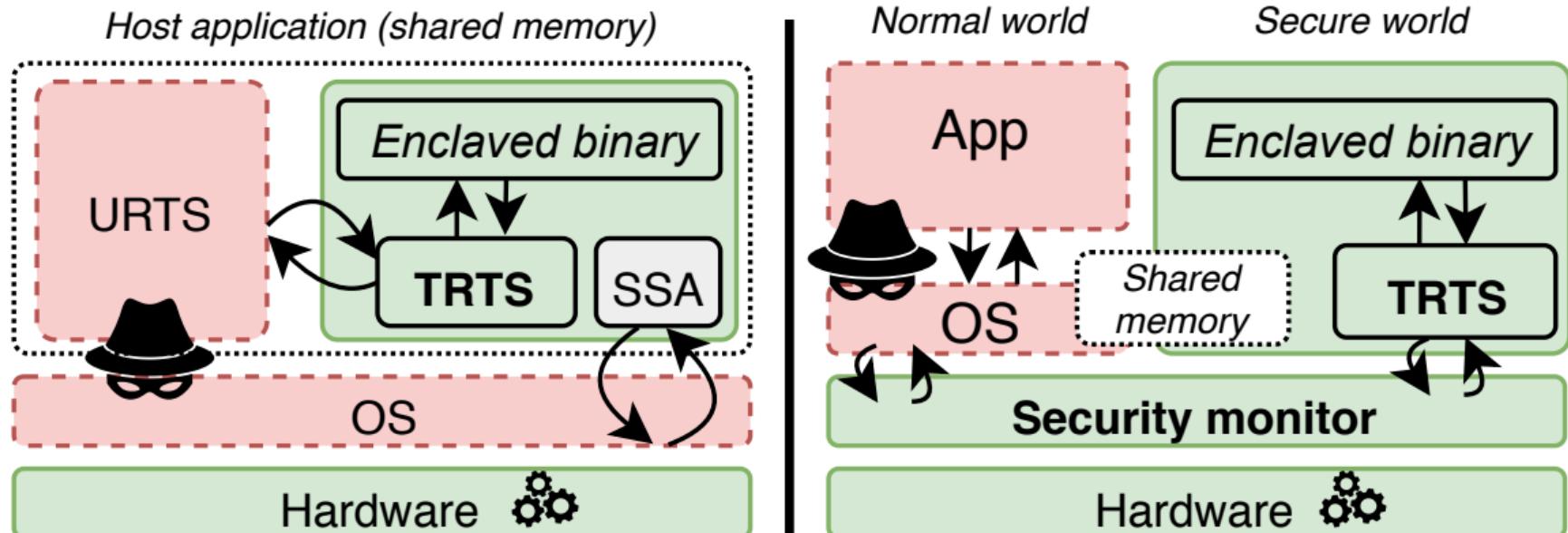


2nd Intel SGX Community Workshop, July 14, 2020

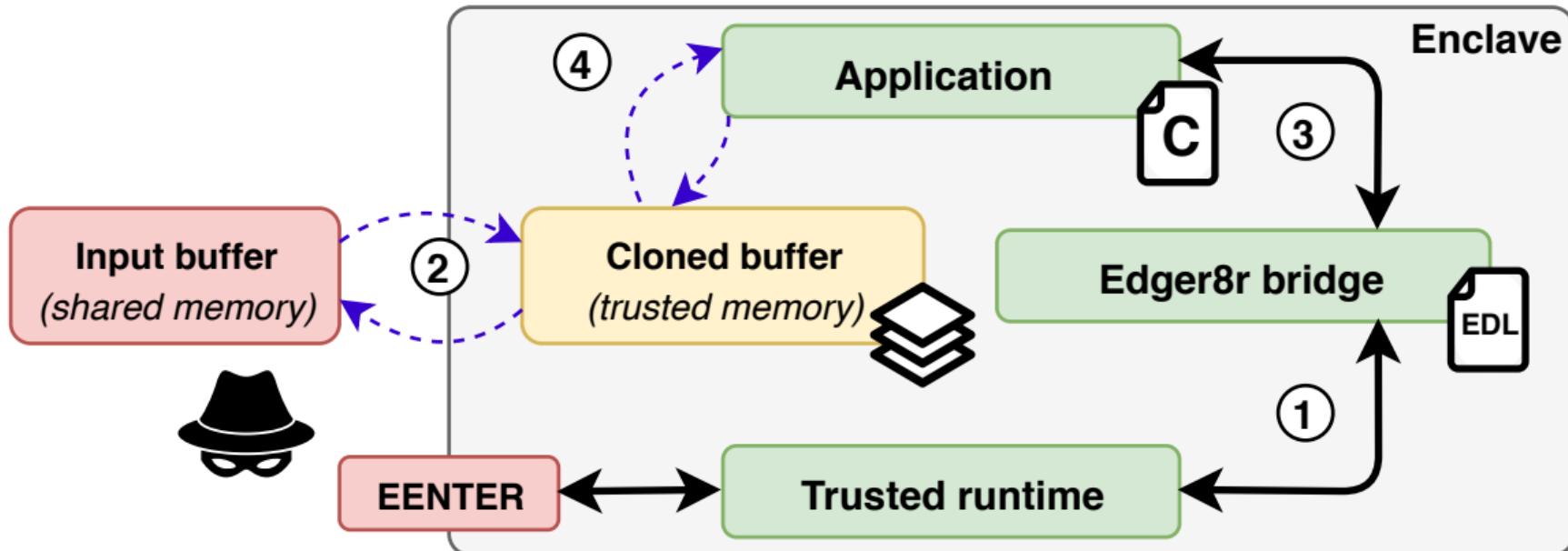
References I

-  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.

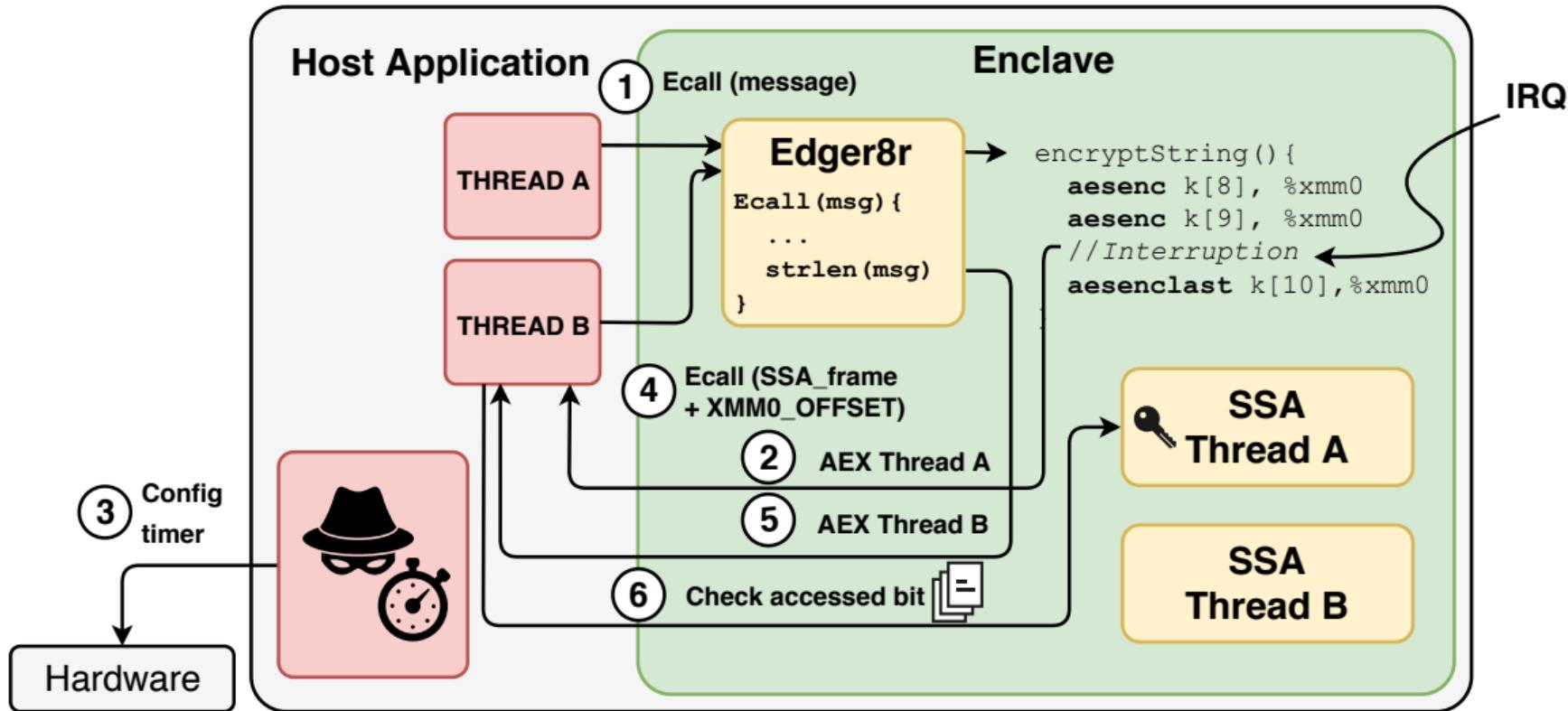
TEE design: Single-address-space vs. world-shared memory approaches



edger8r: Input/output buffer cloning



Intel SGX strlen oracle attack



Exploitation challenges: Building a precise null byte oracle



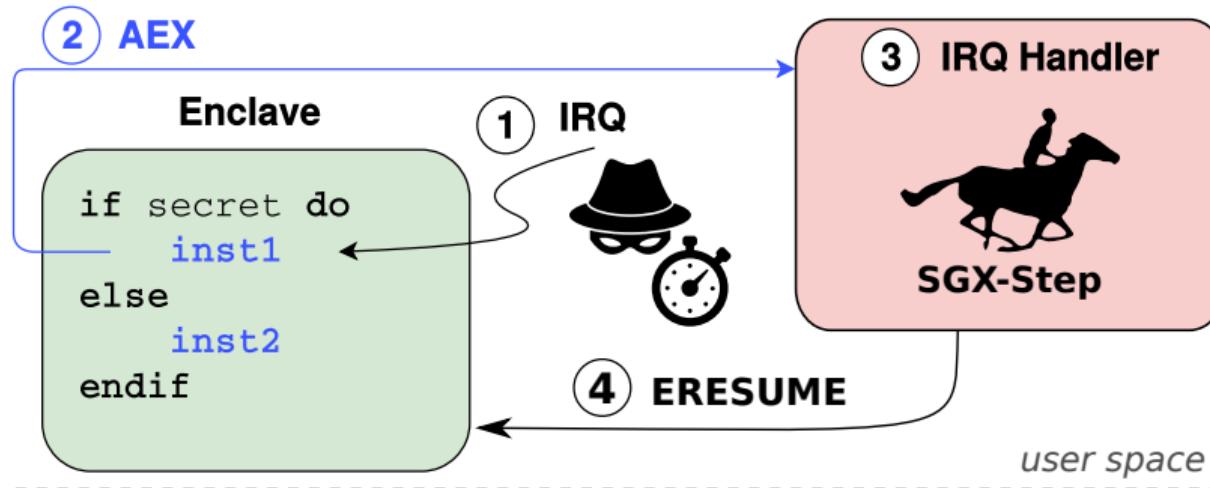
Goal: Precisely count number of executed strlen() loop iterations?

```
1 size_t strlen (char *str)
2 {
3     char *s;
4
5     for (s = str; *s; ++s);
6     return (s - str);
7 }
```

```
1      mov  %ordi,%rax
2:  cmpb $0x0 ,(%rax)
3:  je   2f
4:  inc  %rax
5:  jmp  1b
6: 2:  sub  %ordi,%rax
7      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]

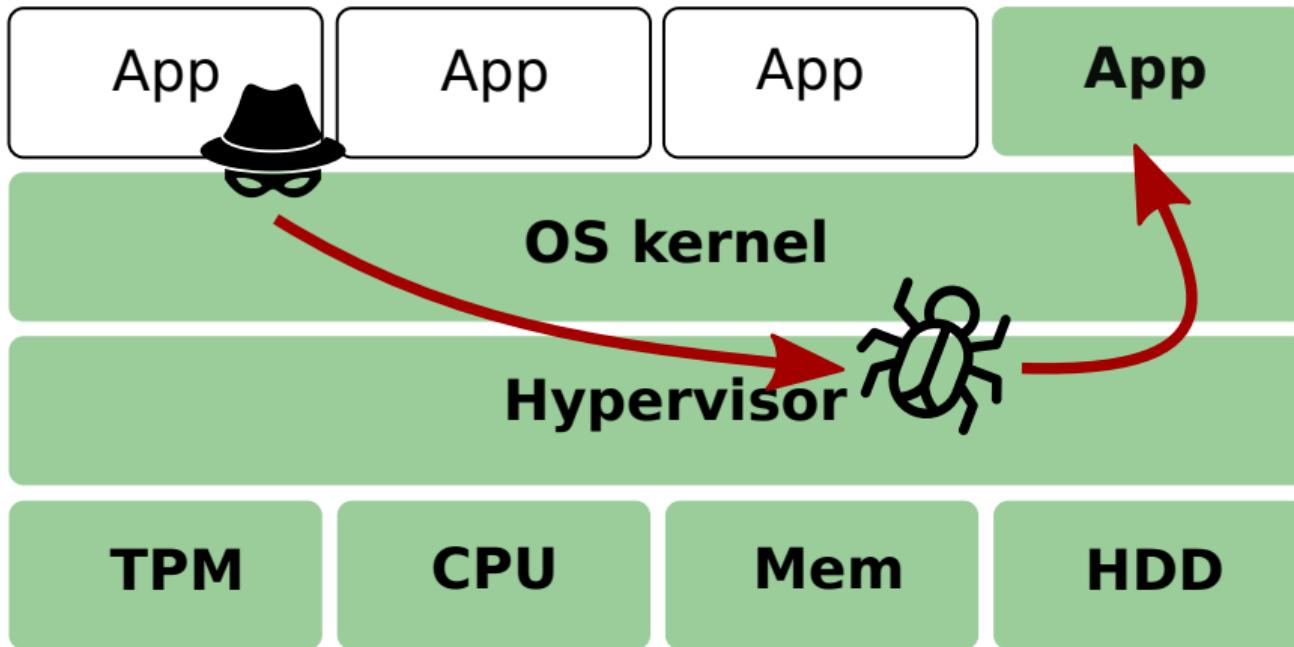
🔗 <https://github.com/jovanbulck/sgx-step>

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.

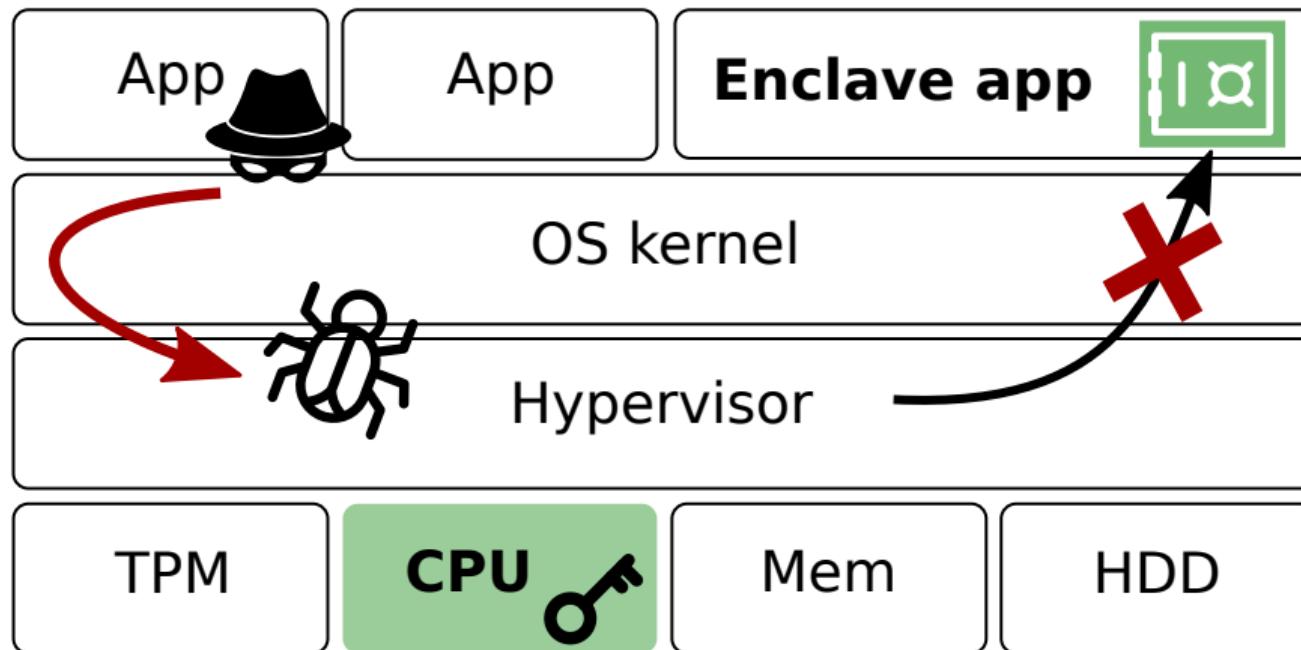
```
while not full key  $K$  recovered do
     $(P, C, L) \leftarrow$  random plaintext, associated ciphertext, strlen oracle
    if  $L < 16$  then
         $K[SR(L)] \leftarrow C[SR(L)] \oplus S(0)$ 
    end if
end while
```

The big picture: Enclaved execution attack surface



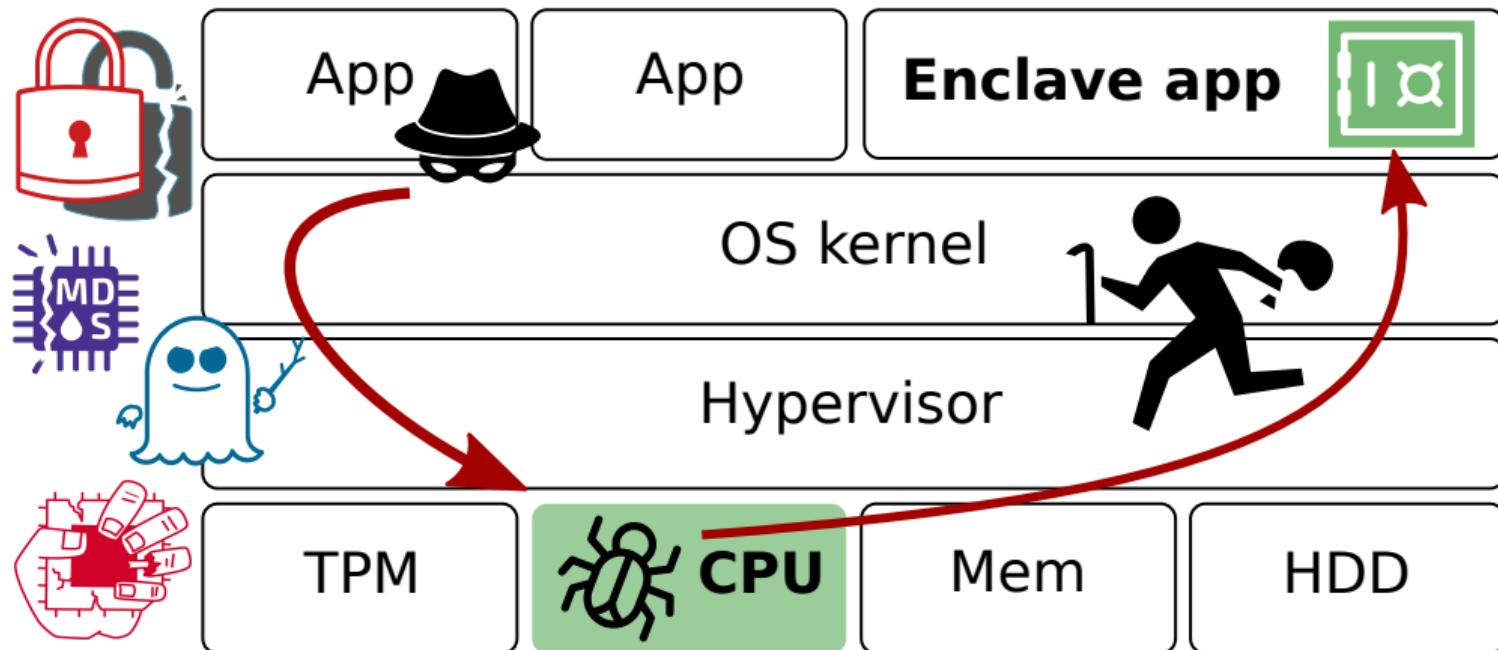
Traditional **layered designs**: large trusted computing base

The big picture: Enclaved execution attack surface



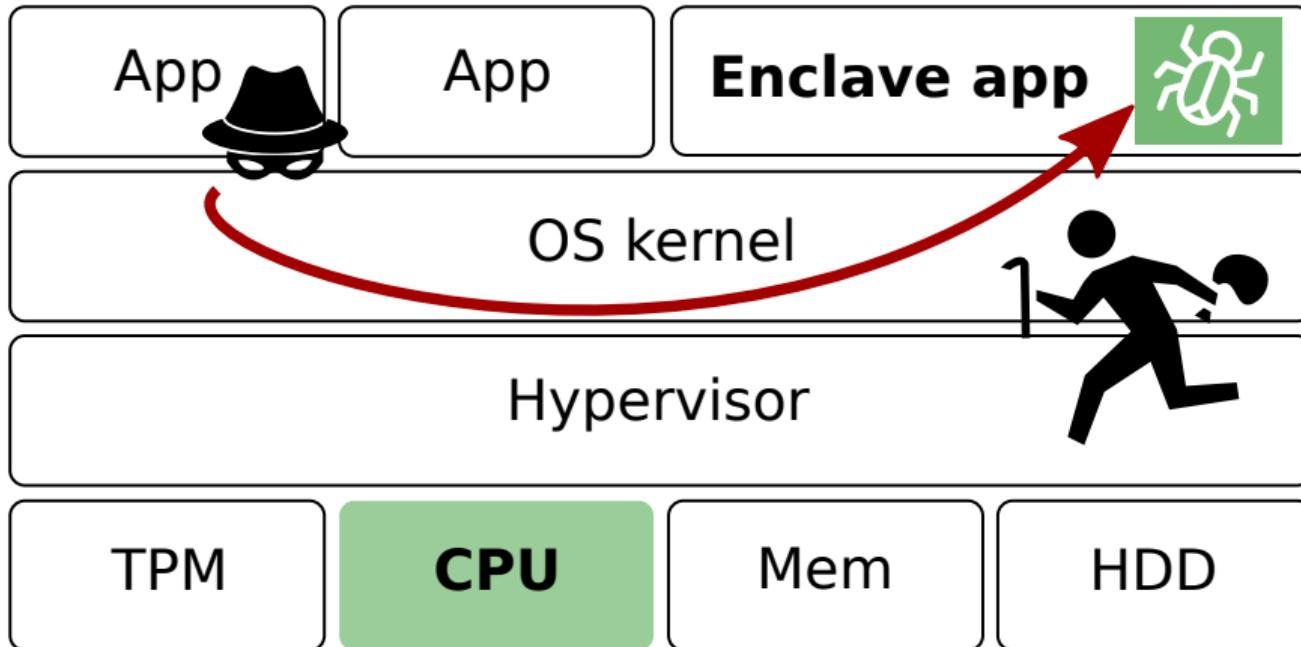
Intel SGX promise: hardware-level **isolation and attestation**

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?