

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

— Present —

"" ACM CCS19 Famous Story ""

A TALE OF TWO WORLDS

• ASSESSING THE •
VULNERABILITY OF ENCLAVE
SHIELDING RUNTIMES

— PRODUCED BY —
KU LEUVEN AND BIRMINGHAM
— UNIVERSITIES —





Outline: How to besiege a fortress?



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

Outline: How to besiege a TEE enclave?

		Runtime	SGX-SDK	OpenEnclave	Graphene	SGX-LKL	Rust-EDP	Asylo	Keystone	Sancus
Vulnerability										
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?

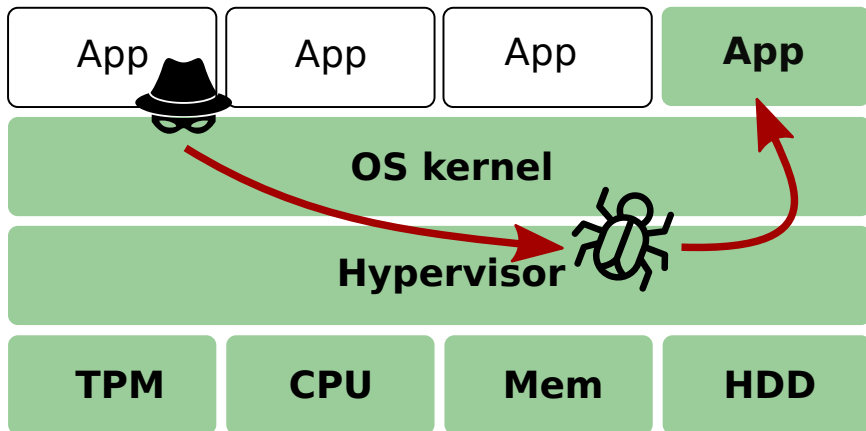
	Runtime	SGX-SDK	OpenEnclave	Graphene	SGX-LKL	Rust-EDP	Asylo	Keystone	Sancus
Tier1 (ABI)	#1 Entry status flags sanitization	★	★	◐	●	◐	●	○	○
	#2 Entry stack pointer restore	○	○	★	●	○	○	○	★
	#3 Exit register leakage	○	○	○	★	○	○	○	○
Tier2 (API)	#4 Missing pointer range check	★	★	★	★	○	●	○	★
	#5 Null-terminated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

Impact: 5 CVEs ... and lengthy embargo periods

שְׁטֵט (צ"י)

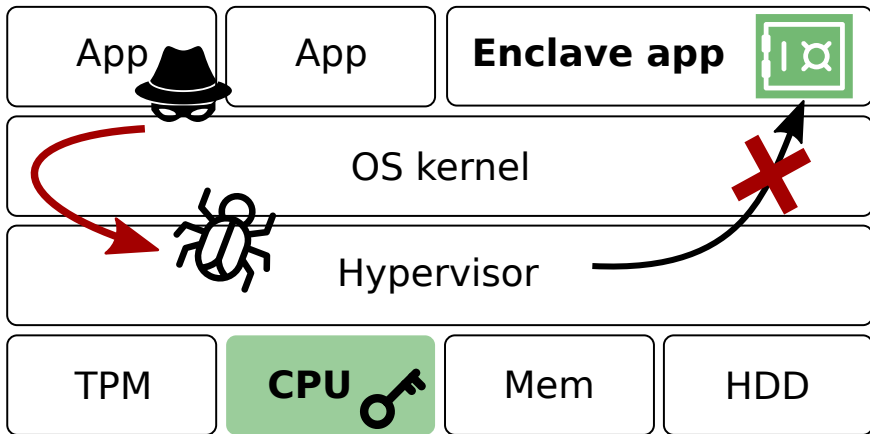
Why do we need enclave fortresses anyway?

The big picture: Enclaved execution attack surface



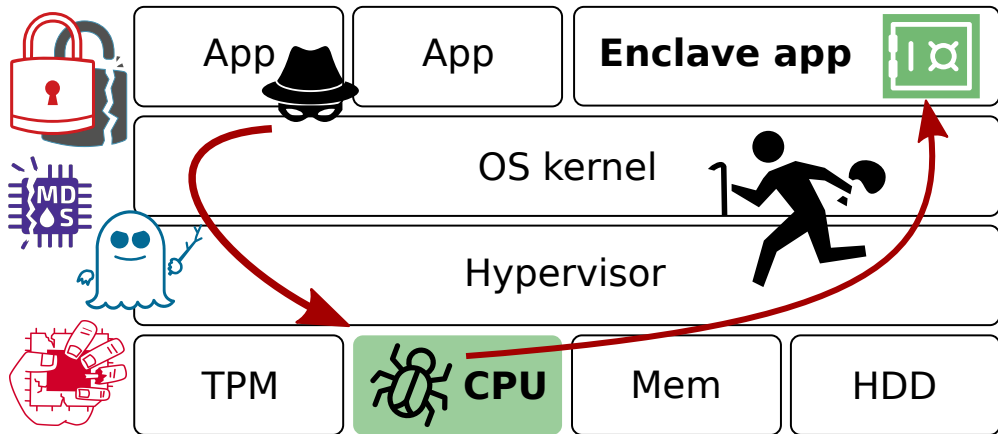
Traditional **layered designs**: large **trusted computing base**

The big picture: Enclaved execution attack surface



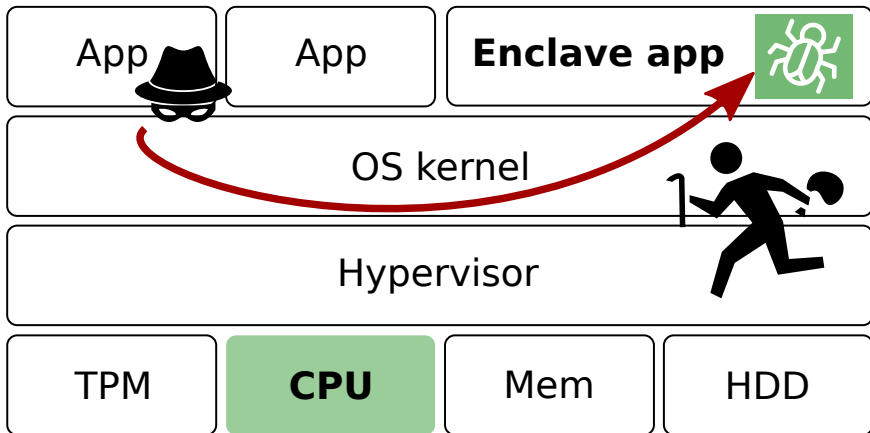
Intel SGX promise: hardware-level **isolation and attestation**

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

[View on GitHub](#)

[Watch a demo](#)

[Explore Research](#)

Keystone | An Open Framework for Architecting TEEs - Mozilla Firefox

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 - a Library OS for Unmodified Applications

Join our public stack 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](#)

Intel Developer Zone

INTEL® SOFTWARE GUARD EXTENSIONS

GET STARTED WITH THE SDK

GOOGLE CLOUD PLATFORM

Introducing Asylo: an open framework for confidential computing

It supports a software-defined state, network, and more...

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

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

[View on GitHub](#)

[Watch a demo](#)

[Explore Research](#)

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 Untrusted Applications

Fortanix
EDP

ENCLAVE DEVELOPMENT PLATFORM

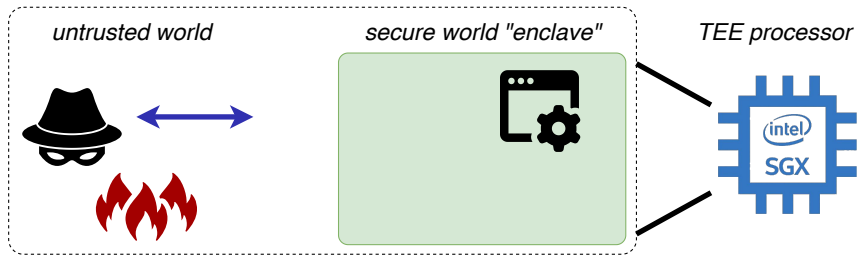
The Fortanix EDP is the preferred way for writing Intel® SGX applications from scratch.

[GETTING STARTED](#)

GOOGLE CLOUD PLATFORM

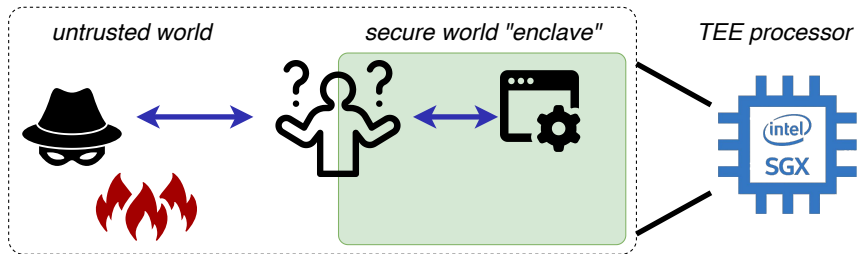
Introducing Asylo: an open source framework for confidential computing

Why isolation is not enough: Enclave shielding runtimes



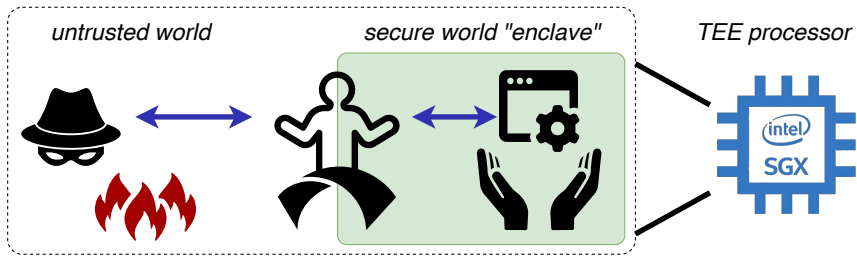
- TEE promise: enclave == “secure oasis” in a **hostile environment**

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




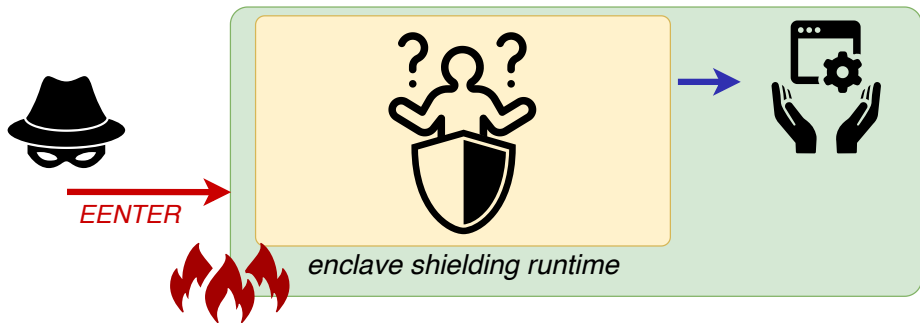


... but what if the bridge itself is flawed?




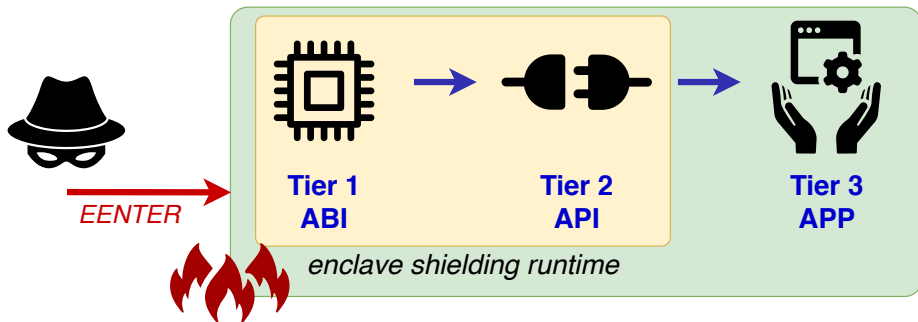
Enclave shielding responsibilities

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

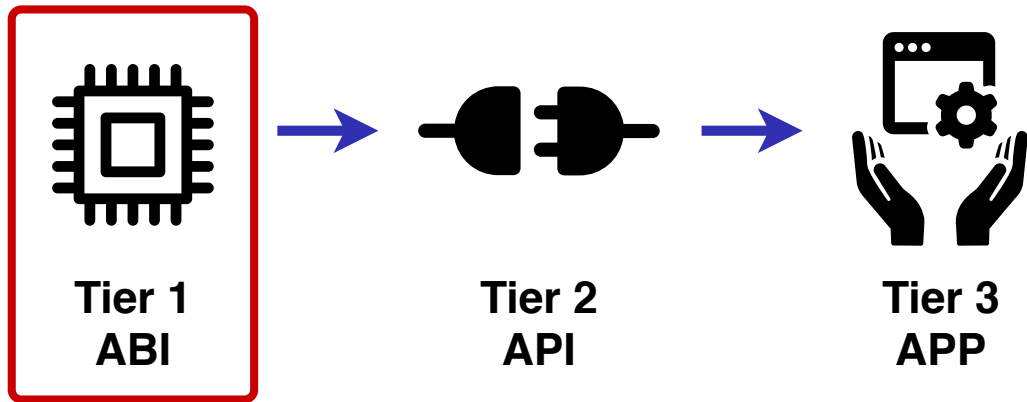


Enclave shielding responsibilities

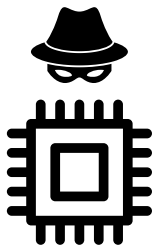
 **Key insight:** split sanitization responsibilities across the ABI and API tiers:
machine state vs. higher-level *programming language interface*



Tier1: Establishing a trustworthy enclave ABI

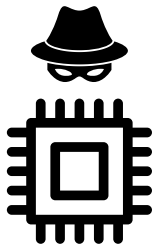


Tier1: Establishing a trustworthy enclave ABI



- ~> 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

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

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

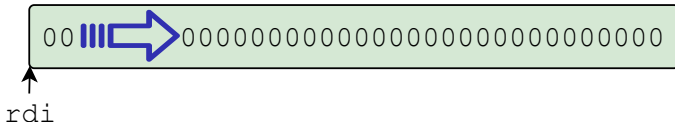


- Special **x86 rep string instructions** to speed up streamed memory operations
- Default operate **left-to-right**

```
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
```

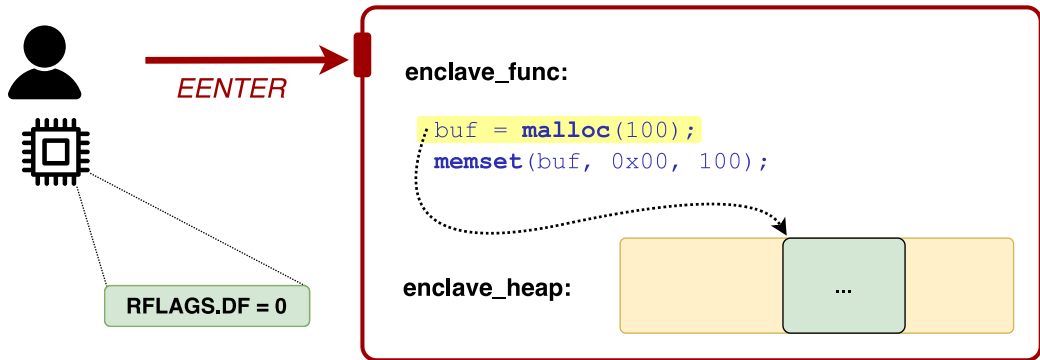


x86 System-V ABI



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**



EENTER →



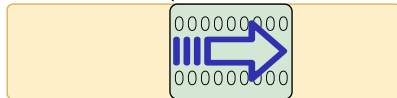
RFLAGS.DF = 0

enclave_func:

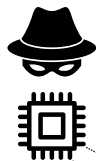
```
buf = malloc(100);  
memset(buf, 0x00, 100);
```



enclave_heap:



↔ Enter enclave with *RFLAGS.DF=1*



RFLAGS.DF = 1

EENTER →

enclave_func:

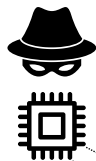
```
buf = malloc(100);  
memset(buf, 0x00, 100);
```

enclave_heap:





Enclave heap **memory corruption**: [right-to-left...](#)



EENTER

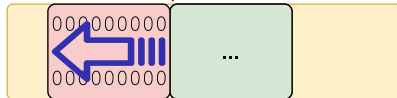
RFLAGS.DF = 1

enclave_func:

```
buf = malloc(100);  
memset(buf, 0x00, 100);
```



enclave_heap:



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:H/PR:L/UI:N/S:C/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:C/C:L/I:L/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* 😊

enclave_func:

```
uint16_t d = lookup_table[secret];
```

enclave_data:

A

B

C

D

64B cacheline

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



Enter enclave with *RFLAGS.AC=1* and secret index=0

→ well-aligned data access: **no exception**



EENTER

RFLAGS.AC = 1

enclave_func:

```
uint16_t d = lookup_table[secret];
```

secret = 0



enclave_data:

A

B

C

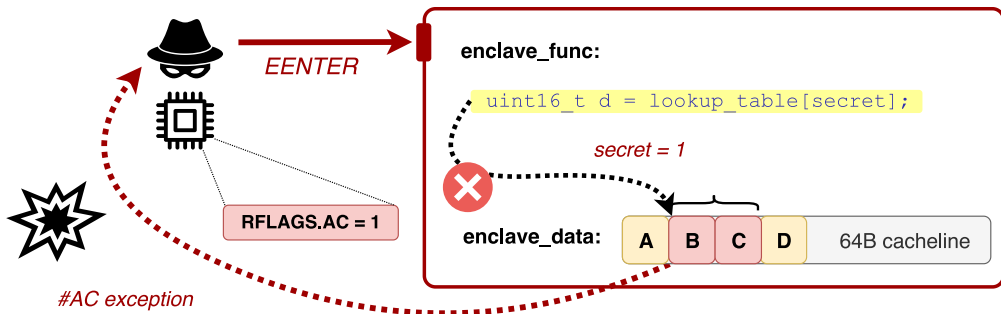
D

64B cacheline

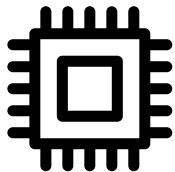
SGX-AC: Building an intra-cacheline side-channel



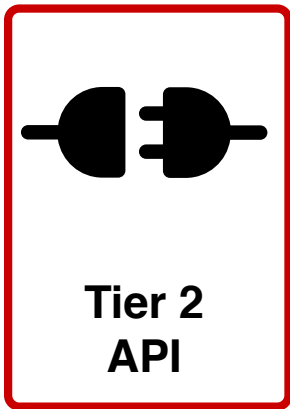
Enter enclave with `RFLAGS.AC=1` and secret index=1
→ unaligned data access: **alignment-check exception...**



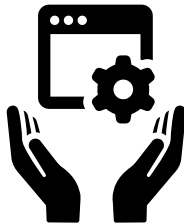
Tier 2: Sanitizing the enclave API



**Tier 1
ABI**

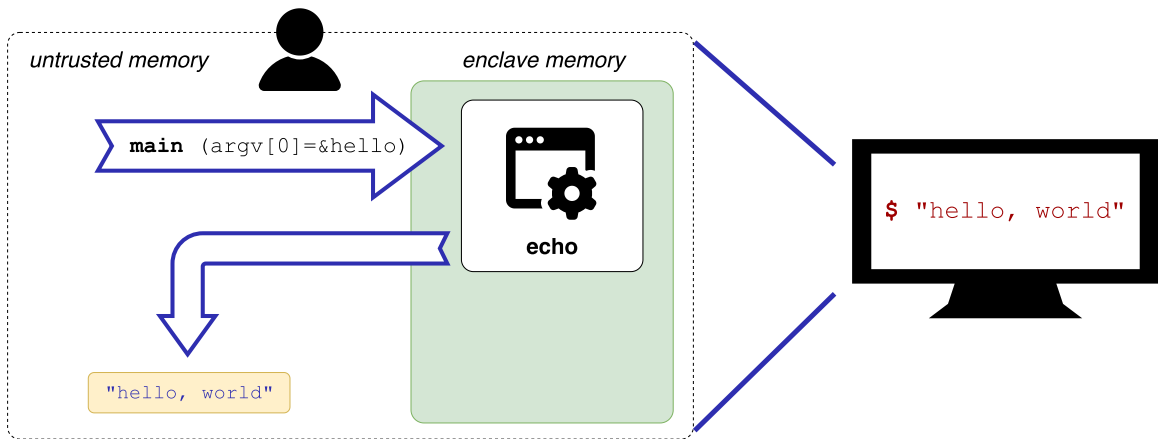


**Tier 2
API**

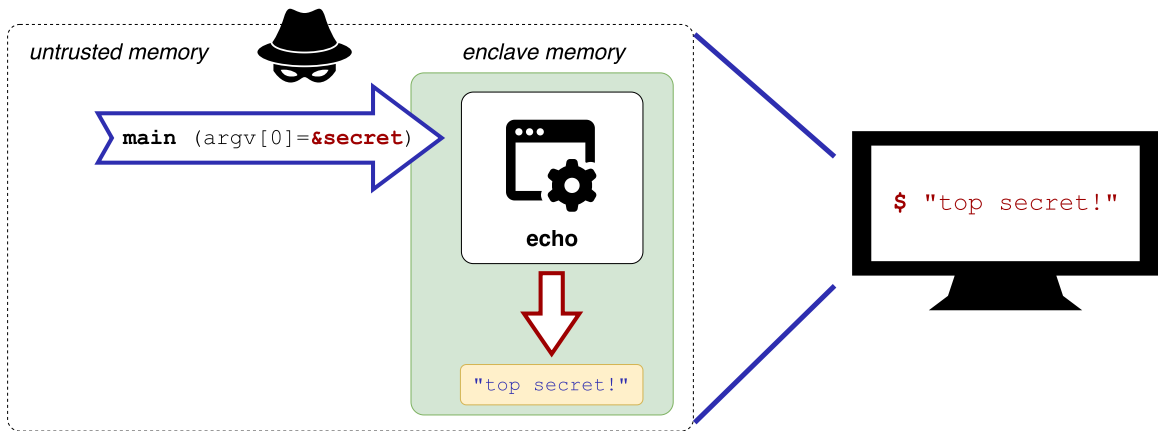


**Tier 3
APP**

Validating pointer arguments: Confused deputy attacks



Validating pointer arguments: Confused deputy attacks



Validating pointer arguments: Confused deputy attacks

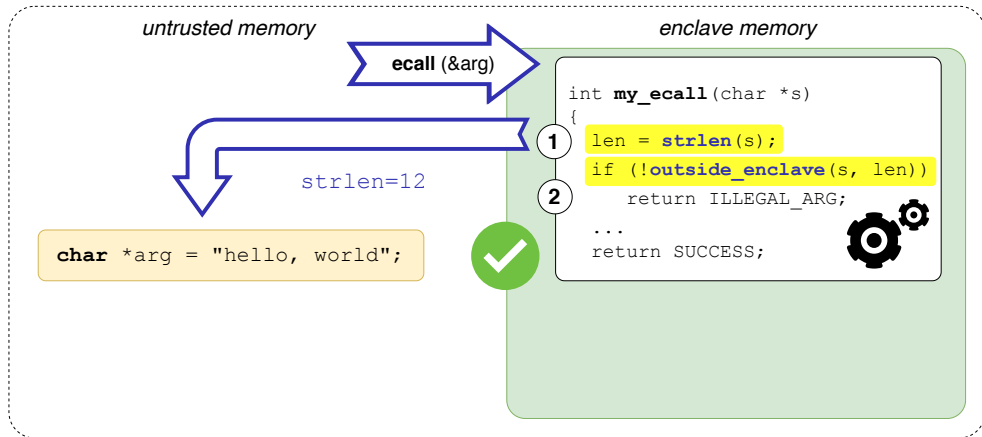
```
Hello world from enclaved application binary!
--> enclave secret at 0x400688

Echoing user-provided command line arguments
  argv[0] @0x4dfdff0 = 'file:helloworld'
  argv[1] @0x4dfdfd4 = 'super secret enclave string'
  argv[2] @0x4dfdfce = 'test2'

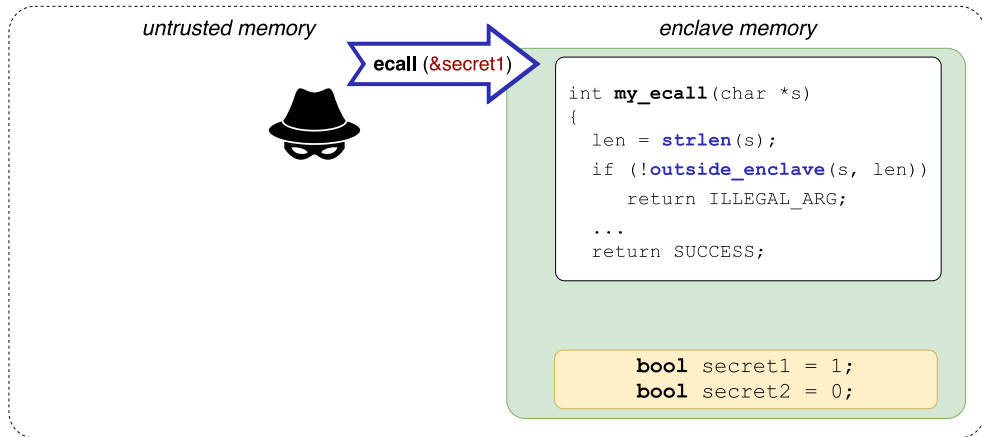
[ 1] ---- return from shim_write(...) = 249
[ 1] ---- shim_exit_group (returning 0)
[ 1] now kill other threads in the process
[ 1] walk_thread_list(callback=0xbb2cb72)
[ 1] now exit the process
[ 1] ipc broadcast: IPC_CLD_EXIT(1, 1, 0)
[ 1] found port 0xba720c0 (handle 0xbfaa5b0) for process 0 (type 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
$
```



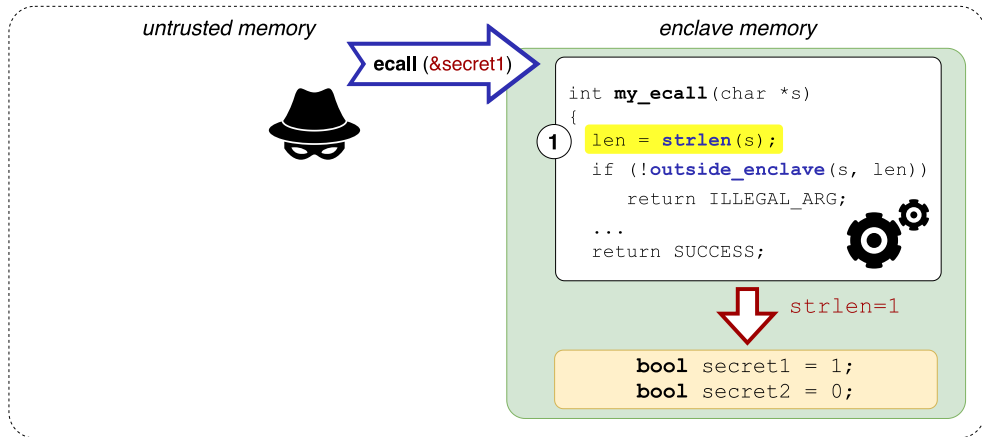

 **Idea:** 2-stage approach ensures *string arguments fall entirely outside enclave*



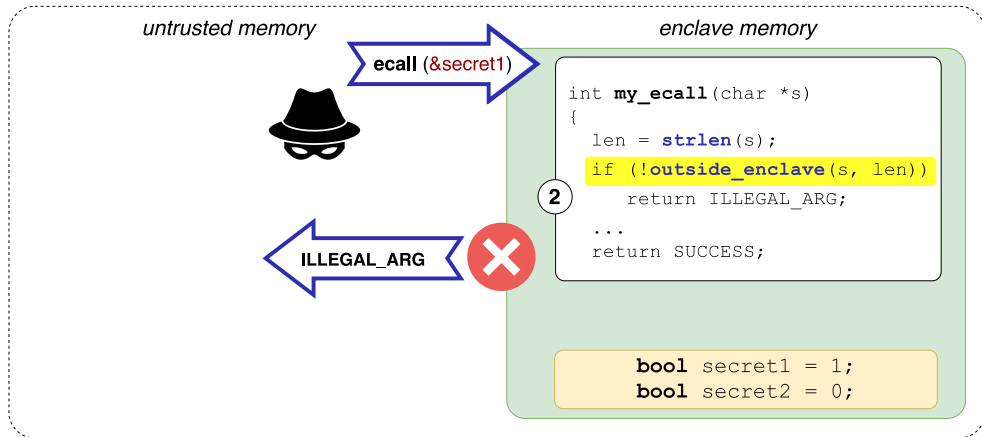
❌ ... **but** what if we try passing an **illegal, in-enclave pointer** anyway?



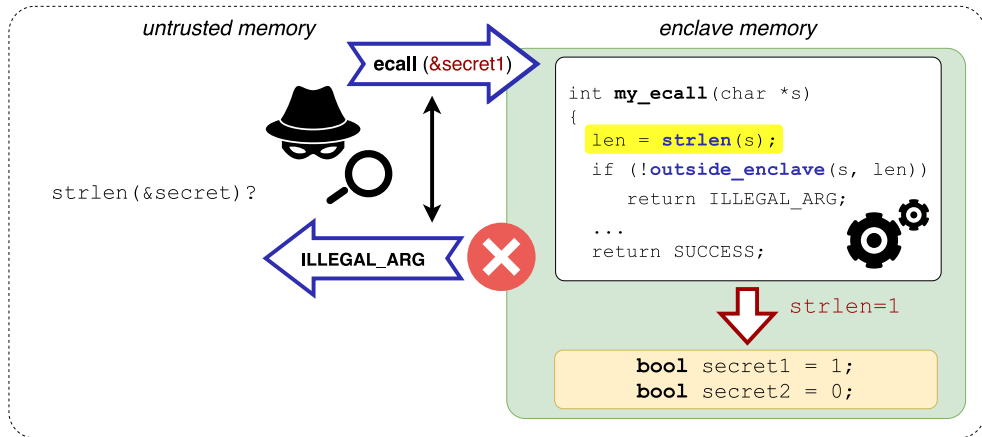
⚠ Enclave **first** computes length of secret, in-enclave buffer!



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



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



Challenge: Building a precise null byte oracle

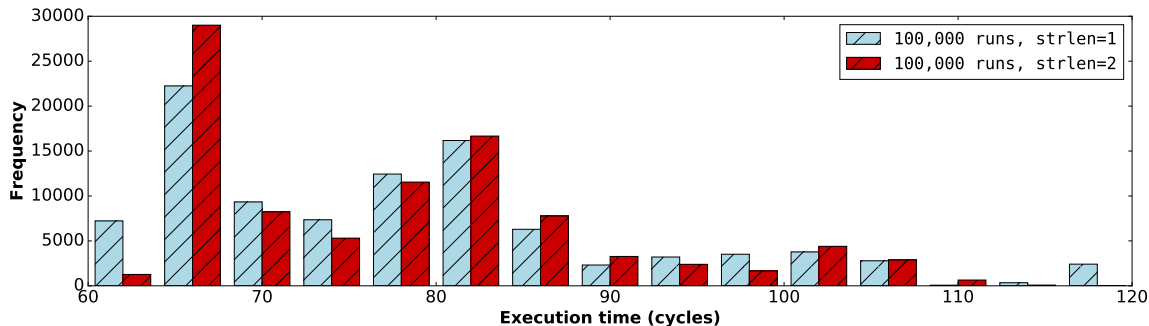


What about measuring execution time?

Building the oracle with strlen() timing?

Execution timing side-channel?

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



Challenge: Building a precise null byte oracle



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.

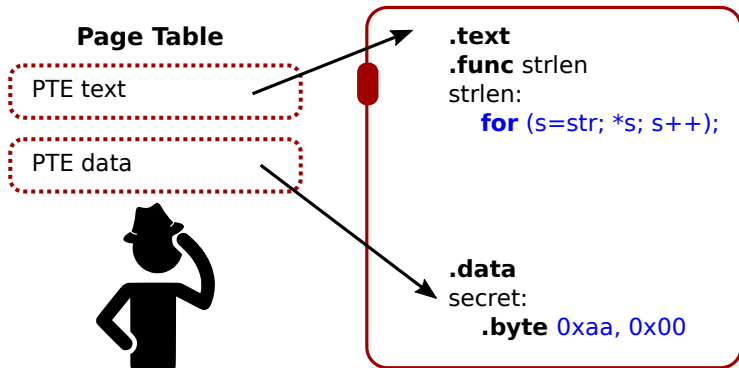
An attacker with access to the platform can see what pages are being executed or accessed. This side-channel vulnerability can be mitigated by aligning specific code and data blocks to exist entirely within a single page.

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



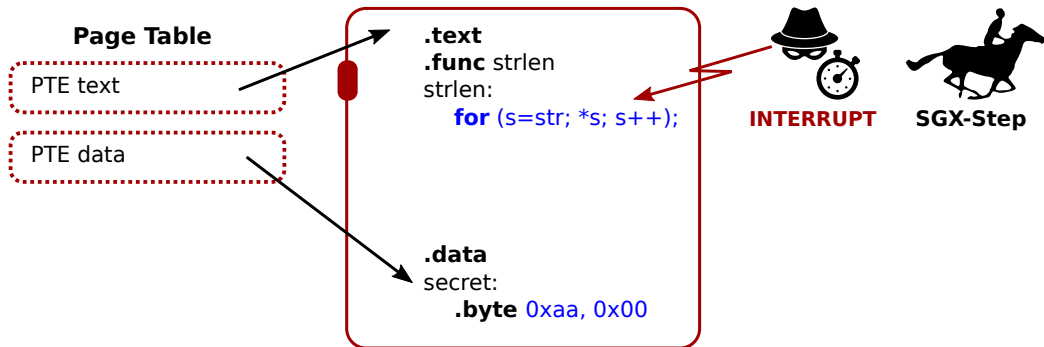
Challenge: Counting `strlen()` loop iterations



What about leveraging interrupts?

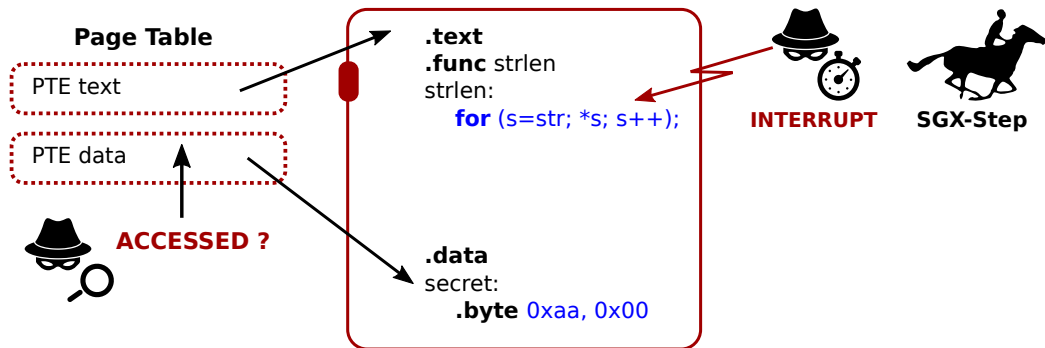
Building a deterministic strlen() null byte oracle with SGX-Step

 Execute *exactly one* enclave instruction → timer interrupt



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

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



CVE-2018-3626: ALL YOUR ZERO BYTES

A close-up photograph of a raccoon with its hands clasped together in a prayer-like gesture. The raccoon has a black and white face with a white patch around its eyes and a black mask. It is looking directly at the camera with a slight smile. The background is a blurred forest floor with brown leaves and green grass.

ARE BELONG TO US

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



```
...  
aesenc xmm0  
aesenc xmm0  
aesenclast xmm0  
...
```



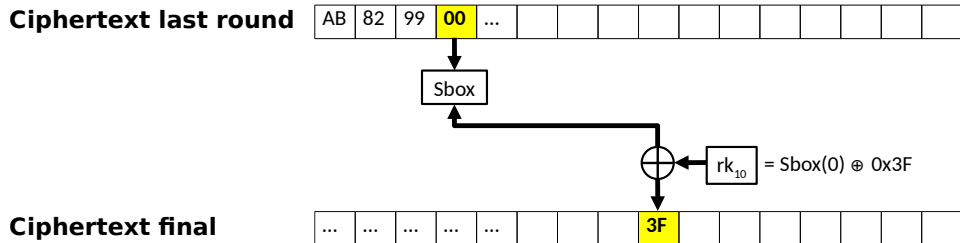
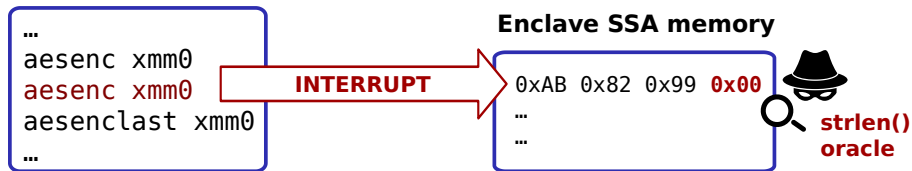
INTERRUPT

(store registers)

Enclave SSA memory

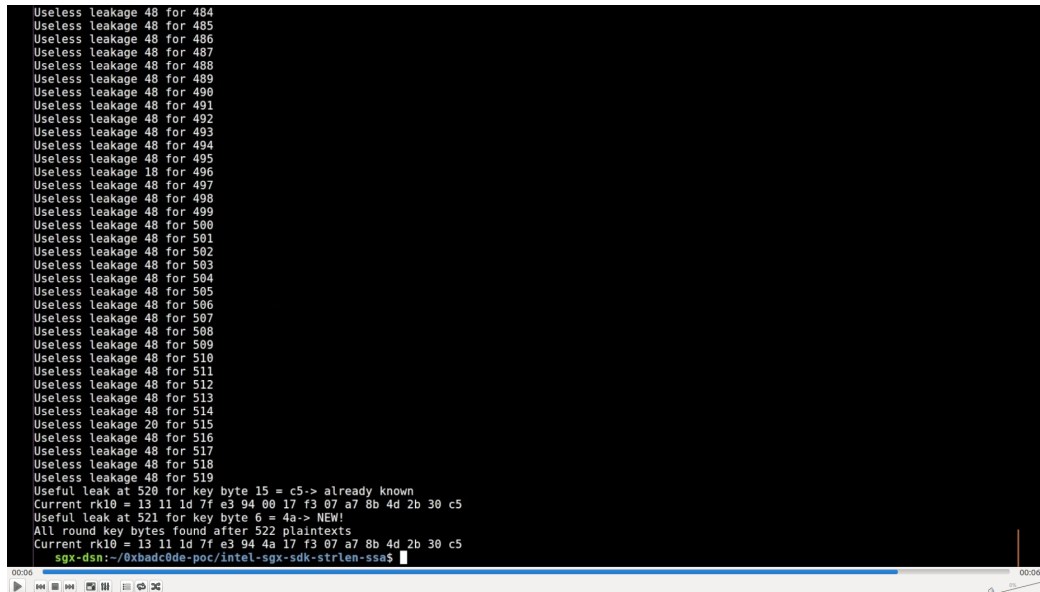
```
0xAB 0x82 0x99 0x00  
...  
...
```


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



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

```
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

	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)

Summary: API-level attack surface

Runtime		SGX-SDK	OpenEnclave	Graphene	SGX-LKL	Rust-EDP	Asylo	Keystone	Sancus
Vulnerability									
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.*



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

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



ACM CCS, November 2019

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.



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.

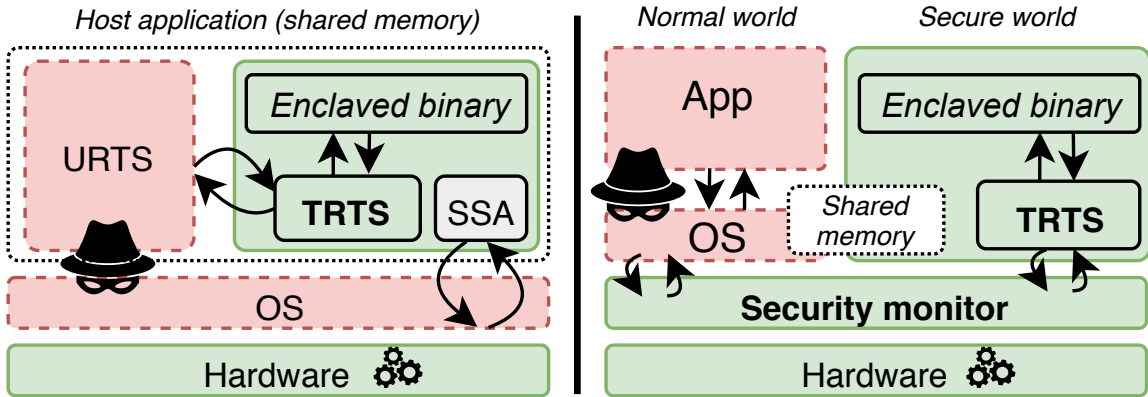


J. Van Bulck, N. Weichbrodt, R. Kapitza, F. Piessens, and R. Strackx.

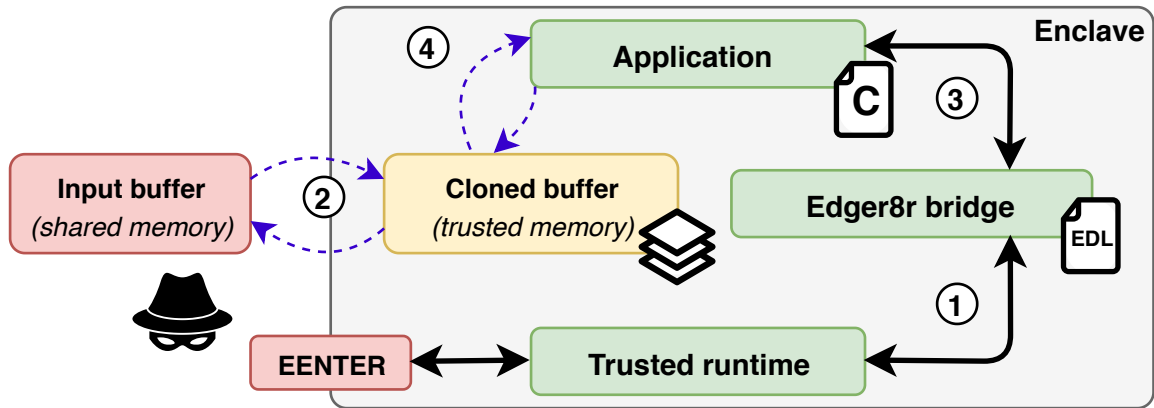
Telling your secrets without page faults: Stealthy page table-based attacks on enclaved execution.

In *Proceedings of the 26th USENIX Security Symposium*, pp. 1041–1056, 2017.

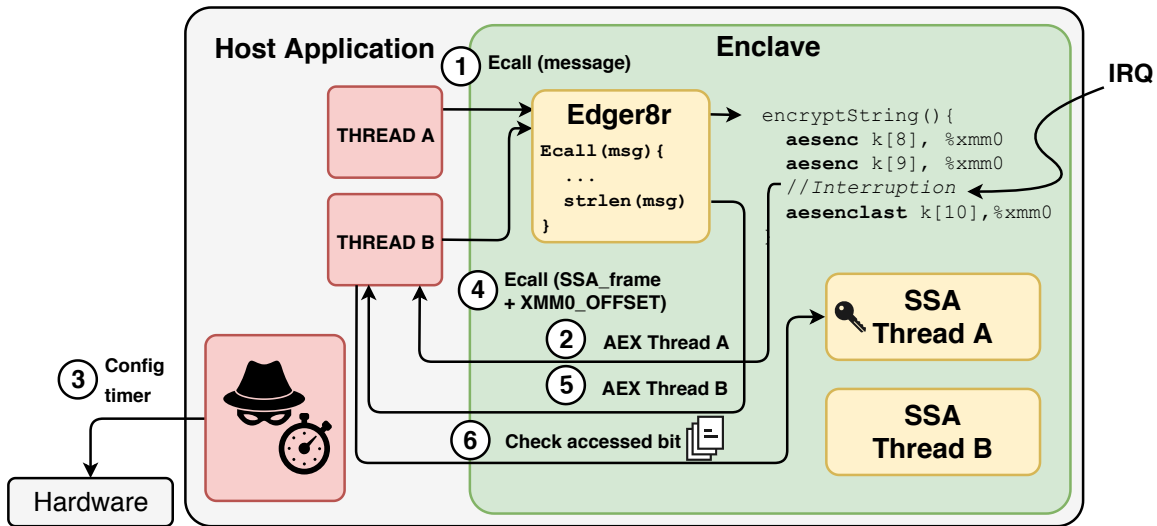
TEE design: Single-address-space vs. world-shared memory approaches



edger8r: Input/output buffer cloning



Intel SGX strlen oracle attack



Exploitation challenges: Building a precise null byte oracle



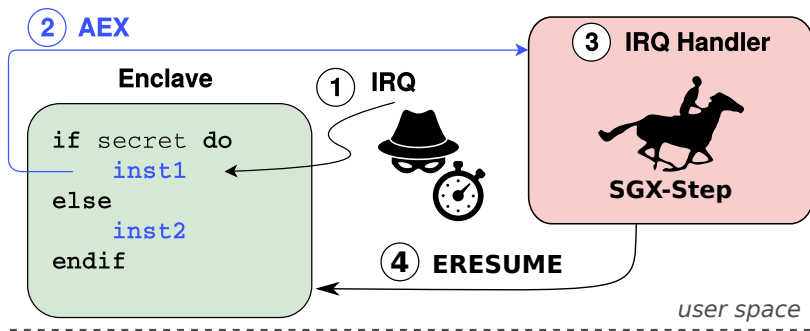
Goal: Precisely count number of executed strlen() loop iterations?

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

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