Ramming Enclave Gates: A Systematic Vulnerability Assessment of TEE Shielding Runtimes

Jo Van Bulck [1], Fritz Alder [1], David Oswald [2]

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

rC3 — Remote Chaos Experience, December 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 Floating-point register sanitization	★	★	○	★	★	●	○	○
	#3 Entry stack pointer restore	○	○	★	●	○	○	○	★
	#4 Exit register leakage	○	○	○	★	○	○	○	○
Tier2 (API)	#5 Missing pointer range check	○	★	★	★	○	●	○	★
	#6 Null-terminated string handling	✯	★	○	○	○	○	○	○
	#7 Integer overflow in range check	○	○	●	○	●	○	●	●
	#8 Incorrect pointer range check	○	○	●	○	○	●	○	●
	#9 Double fetch untrusted pointer	○	○	●	○	○	○	○	○
	#10 Ocall return value not checked	○	★	★	★	○	●	★	○
	#11 Uninitialized padding leakage	[LK17]	★	○	●	○	●	★	★

Summary: > 40 enclave interface sanitization vulnerabilities across > 8 projects

Outline: How to besiege a TEE enclave?

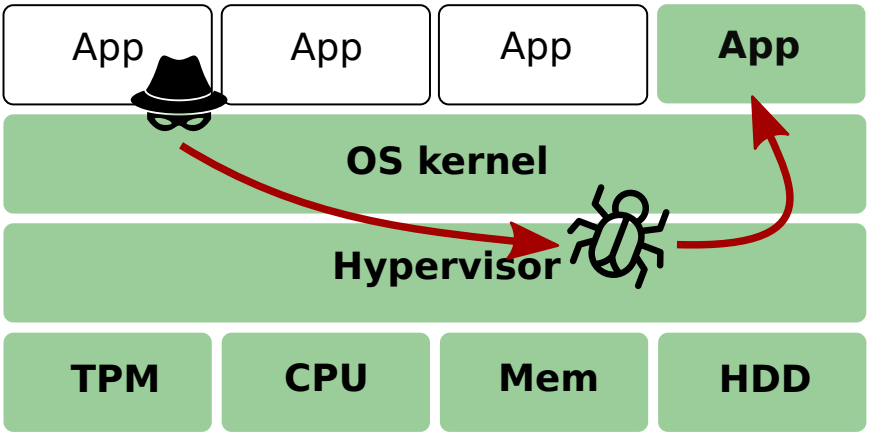
Vulnerability \ Runtime		SGX-SDK	OpenEnclave	Graphene	SGX-LKL	Rust-EDP	Asylo	Keystone	Sancus
Tier1 (ABI)	#1 Entry status flags sanitization	★	★	◐	●	◐	●	○	○
	#2 Floating-point register sanitization	★	★	○	★	○	●	○	○
	#3 Entry stack pointer restore	○	○	★	●	○	○	○	★
	#4 Exit register leakage	○	○	○	★	○	○	○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

EMBARGO

Impact: 7 CVEs . . . and lengthy embargo periods

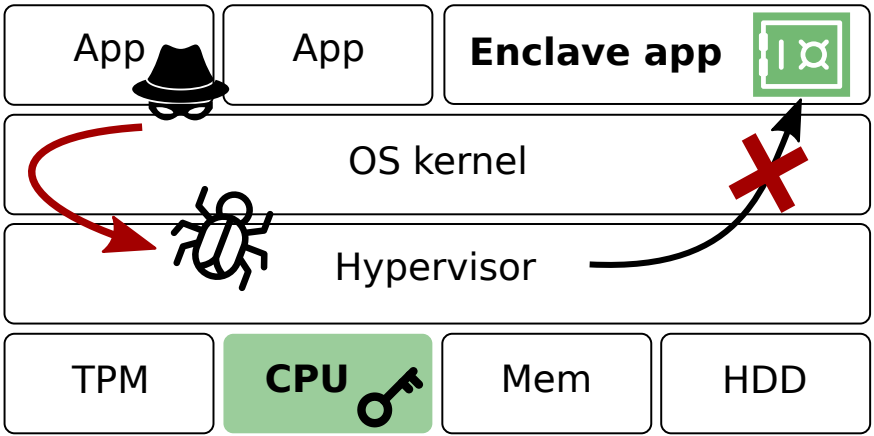
Why do we need enclave fortresses anyway?

The big picture: Enclaved execution attack surface



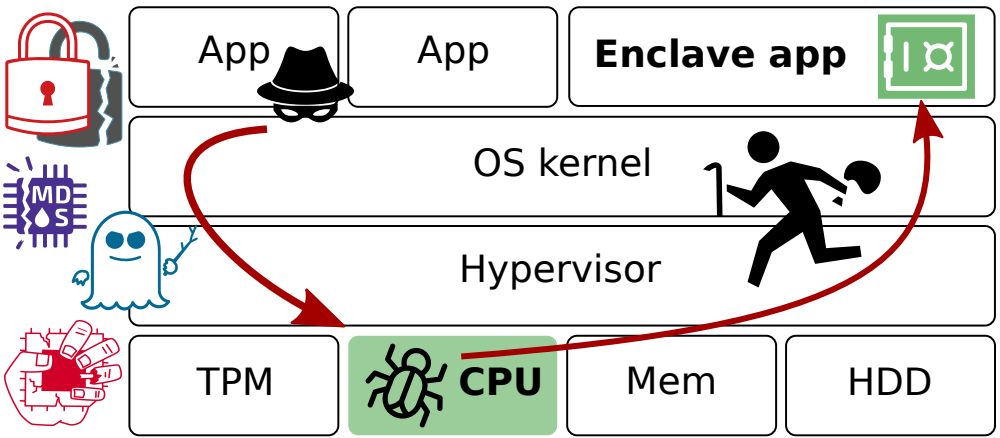
Traditional **layered designs:** large trusted computing base

The big picture: Enclaved execution attack surface



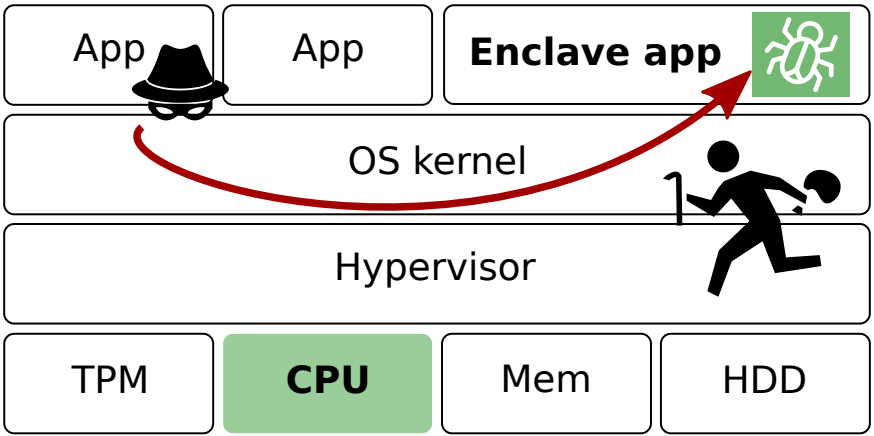
Intel SGX promise: hardware-level **isolation and attestation**

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

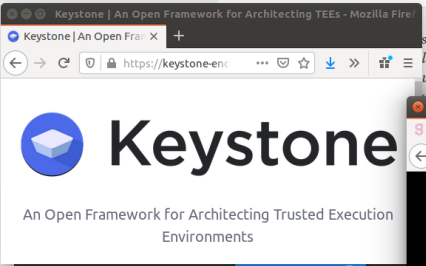


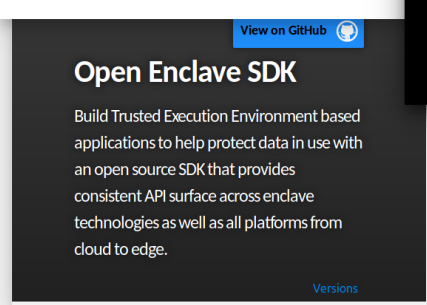
Idea: what about vulnerabilities in the trusted enclave software itself?

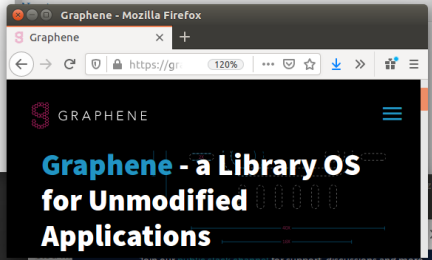
Sancus: Lightweight and Open-Source Trusted Computing for the IoT

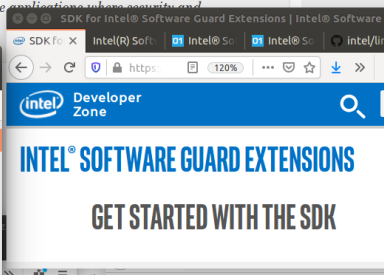
View on GitHub | Watch a demo | Explore Research

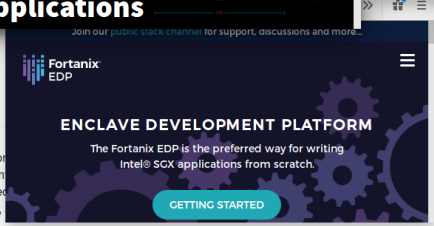
source building blocks and free-software ethos that attempts to provide a layer of integrity and deterministic
lers should be lauded and considered by anyone building hardware applications where security and
uirements.

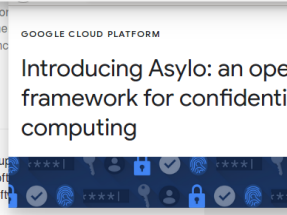




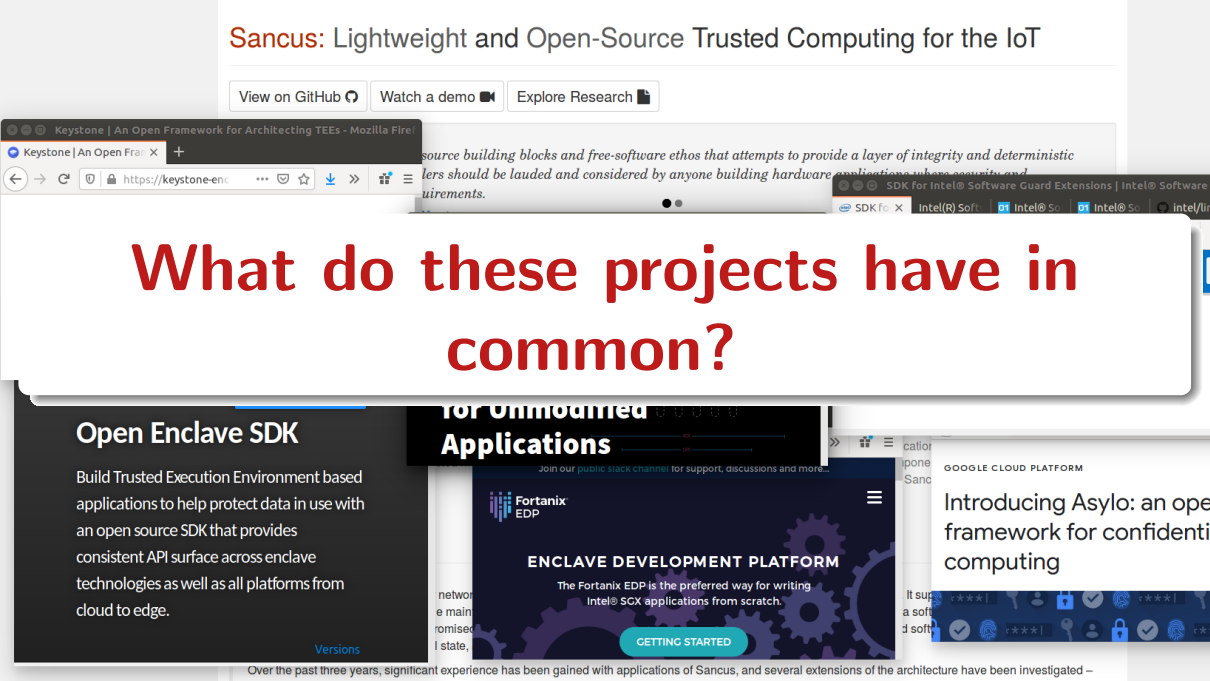




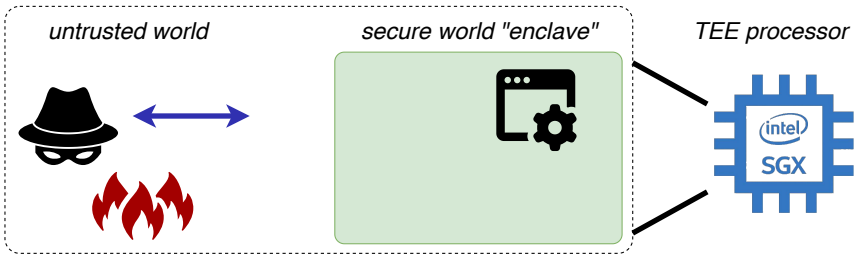




Over the past three years, significant experience has been gained with applications of Sancus, and several extensions of the architecture have been investigated –

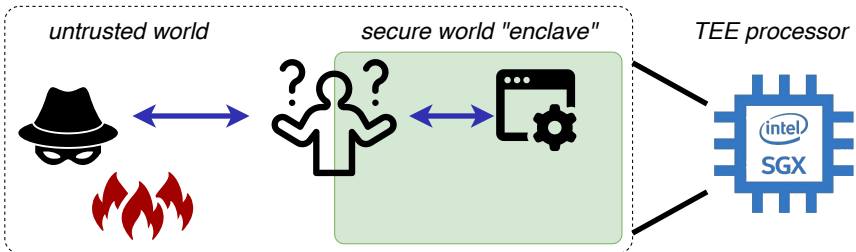

Sancus: Lightweight and Open-Source Trusted Computing for the IoT
View on GitHub
Watch a demo
Explore Research
Keystone | An Open Framework for Architecting TEEs - Mozilla Firef
Keystone | An Open Fra
https://keystone-enc
source building blocks and free-software ethos that attempts to provide a layer of integrity and deterministic
lers should be lauded and considered by anyone building hardware
uirements.
SDK for Intel® Software Guard Extensions | Intel® Software
SDK fo
Intel(R) Softu
Intel® So
Intel® So
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
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
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 –

Why isolation is not enough: Enclave shielding runtimes



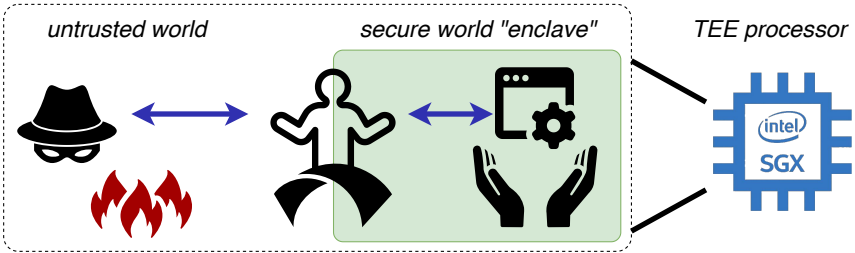
- TEE promise: enclave == "secure oasis" in a **hostile environment**

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



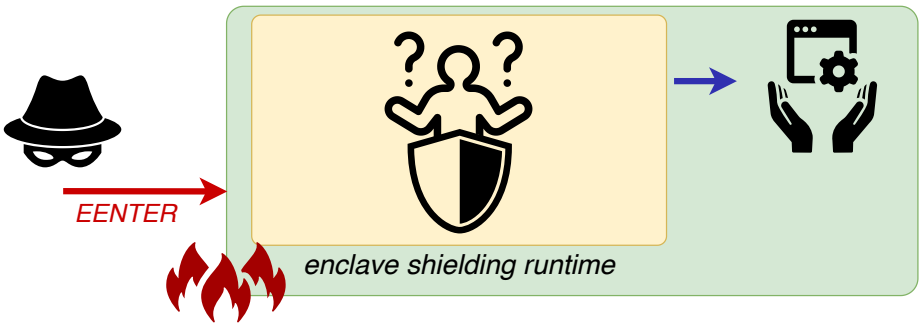
- TEE promise: enclave == "secure oasis" in a **hostile environment**
- . . . but application writers and compilers are largely unaware of **isolation boundaries**

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

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

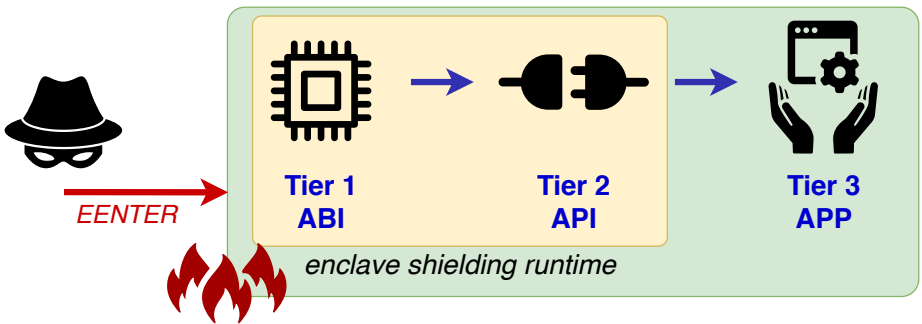
Enclave shielding responsibilities

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

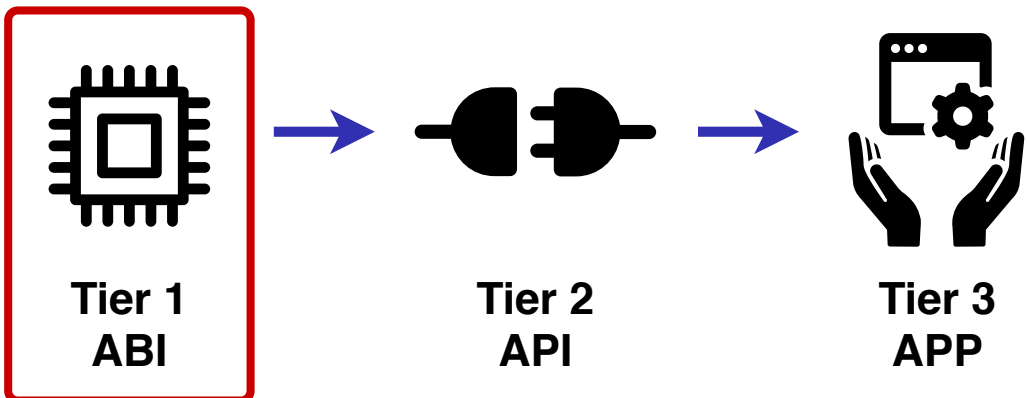


Enclave shielding responsibilities

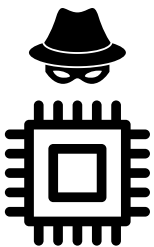
Key insight: split sanitization responsibilities across the ABI and API tiers: *machine state* vs. higher-level *programming language interface*



Tier1: Establishing a trustworthy enclave ABI



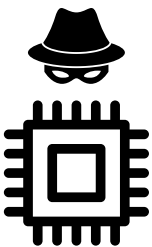
Tier1: Establishing a trustworthy enclave ABI



$\rightsquigarrow$ Attacker controls CPU register contents on enclave entry/exit

$\leftrightarrow$ Compiler expects well-behaved **calling convention** (e.g., stack)

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)

$\Rightarrow$ Need to **initialize CPU registers** on entry and **scrub** before exit!

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 Floating-point register sanitization	★	★	○	★	★	●	○	○
	#3 Entry stack pointer restore	○	○	★	●	○	○	○	★
	#4 Exit register leakage	○	○	○	★	○	○	○	○

ABI vulnerability analysis

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

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 Floating-point register sanitization	★	★	○	★	★	●	○	○
	#3 Entry stack pointer restore	○	○	★	●	○	○	○	★
	#4 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;
```

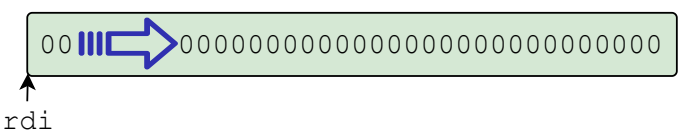
```
1 lea rdi, buf
2 mov al, 0x0
3 mov ecx, 100
4 rep stos [rdi], al
```

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
```

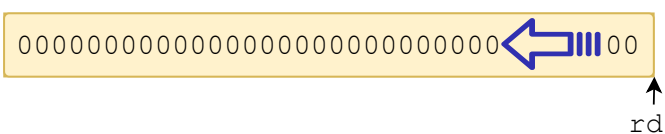


x86 string instructions: Direction Flag (DF) operation

- 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
```



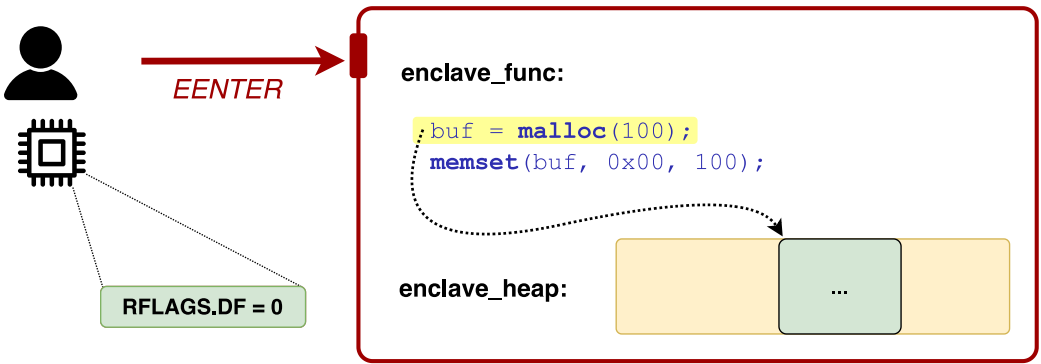
x86 System-V ABI

[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.

SGX-DF: Inverting enclaved string memory operations

CVE-2019-14565

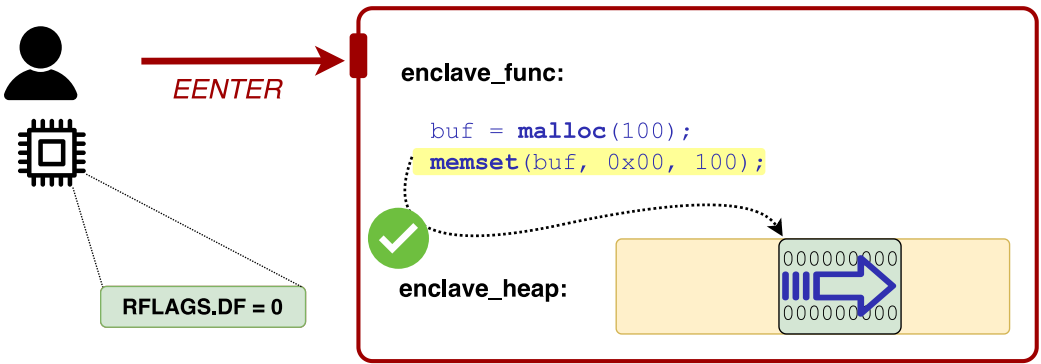
Enter enclave with *RFLAGS.DF=0*



SGX-DF: Inverting enclaved string memory operations

CVE-2019-14565

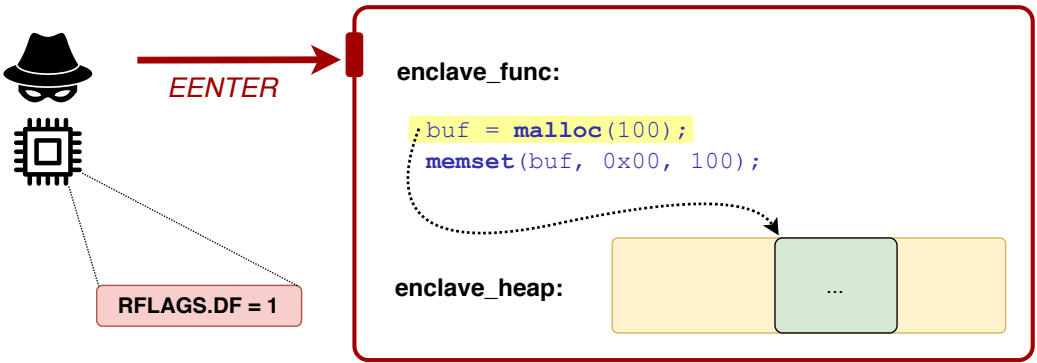
Intended heap memory initialization: left-to-right



SGX-DF: Inverting enclaved string memory operations

CVE-2019-14565

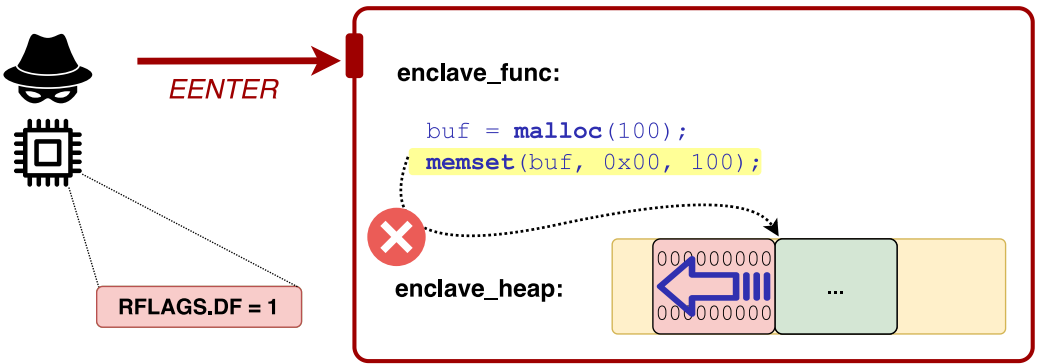
Enter enclave with *RFLAGS.DF=1*



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: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


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

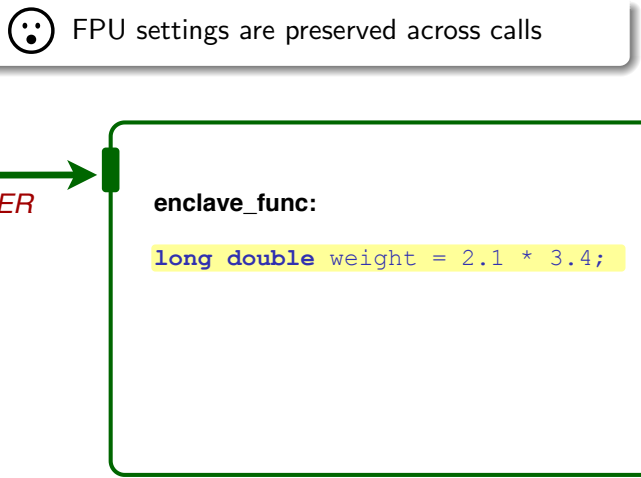
CVE-2020-0561

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

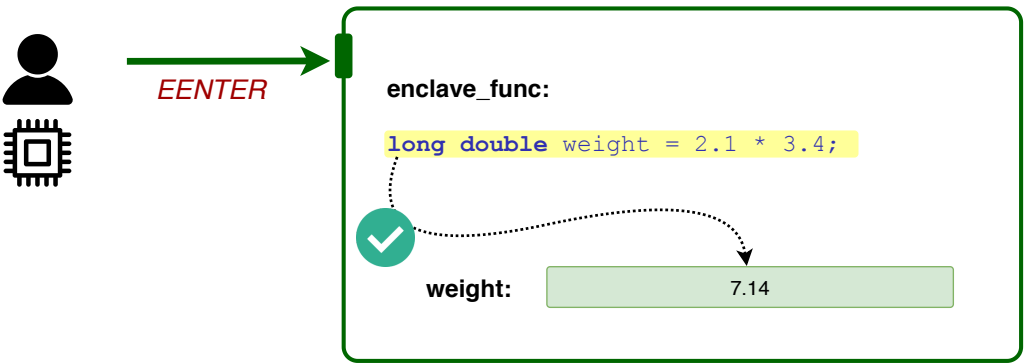
CVE-2020-0561



Controlling FPU precision and rounding modes

CVE-2020-0561

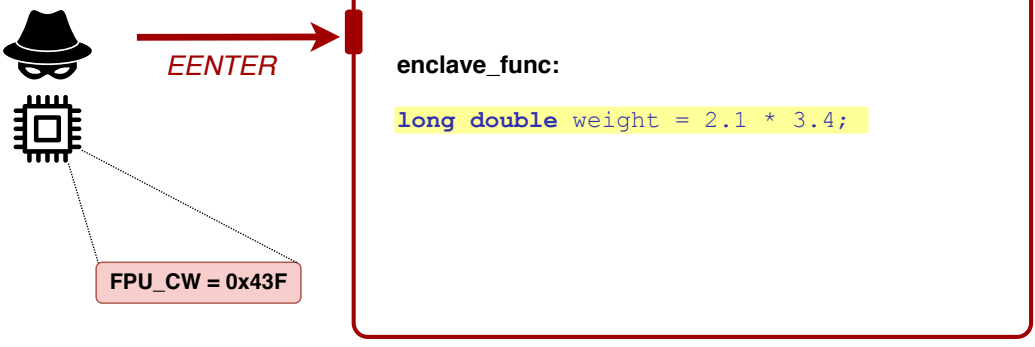
FPU settings are preserved across calls



Controlling FPU precision and rounding modes

CVE-2020-0561

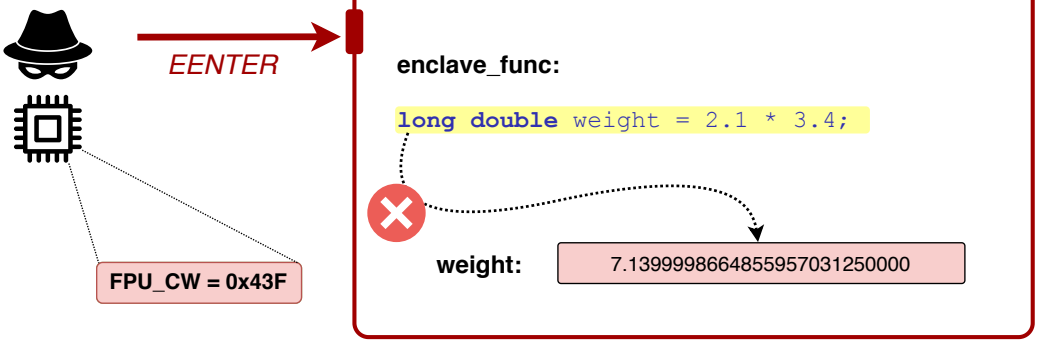
Corrupt precision and rounding mode...



Controlling FPU precision and rounding modes

CVE-2020-0561

Corrupt precision and rounding mode...



Controlling FPU precision and rounding modes

CVE-2020-0561

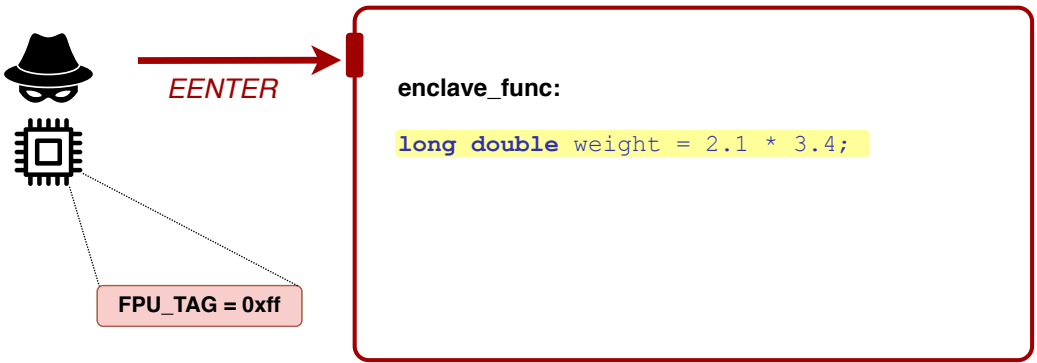
	SGX-SDK*	OpenEnclave	Graphene	SGX-LKL	Rust-EDP	Go-TEE	Enarx
Exploit	★	○	○	★	★	★	○
Patch	`xrstor`	`ldmxcsr/cw`	`fxrstor`	–	`ldmxcsr/cw`	`xrstor`	`xrstor`

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

Fill data registers to fault calculations

CVE-2020-15107

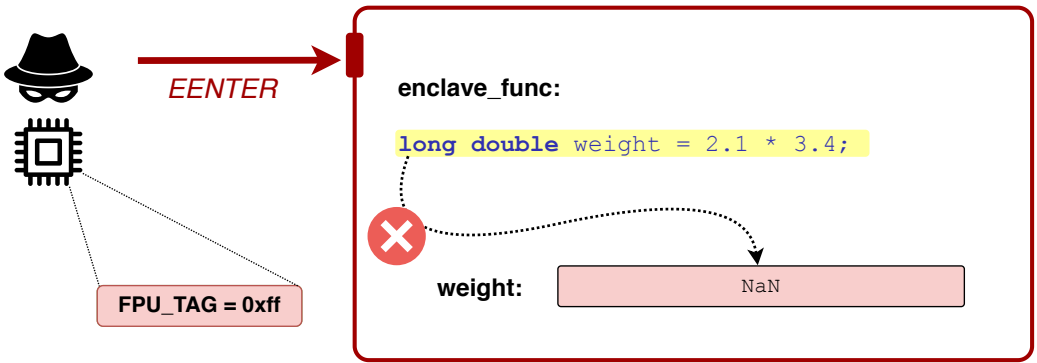
⚠ Mark data registers as in-use before entering the enclave



Fill data registers to fault calculations

CVE-2020-15107

⚠ Mark data registers as in-use before entering the enclave



Summary: ABI-level FPU attack surface today

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

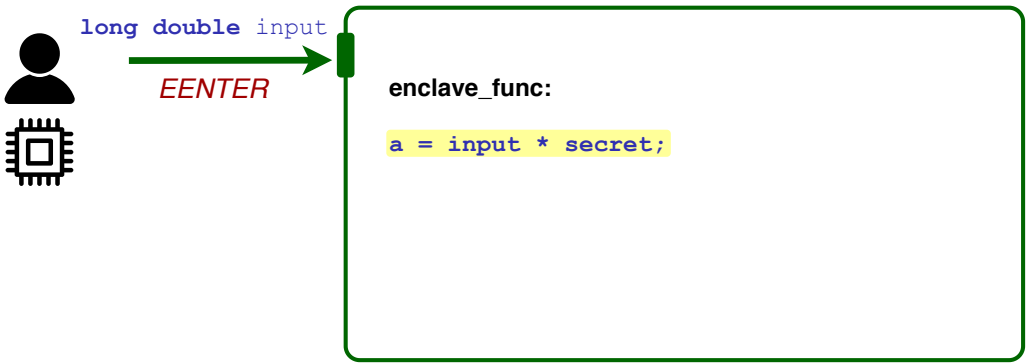
* Includes derived runtimes such as Baidu's Rust-SGX and Google's Asylo.

Patched:

CVE-2020-0561 Improper initialization in the Intel SGX SDK
CVE-2020-15107 x87 FPU operations in OpenEnclave were vulnerable to tampering

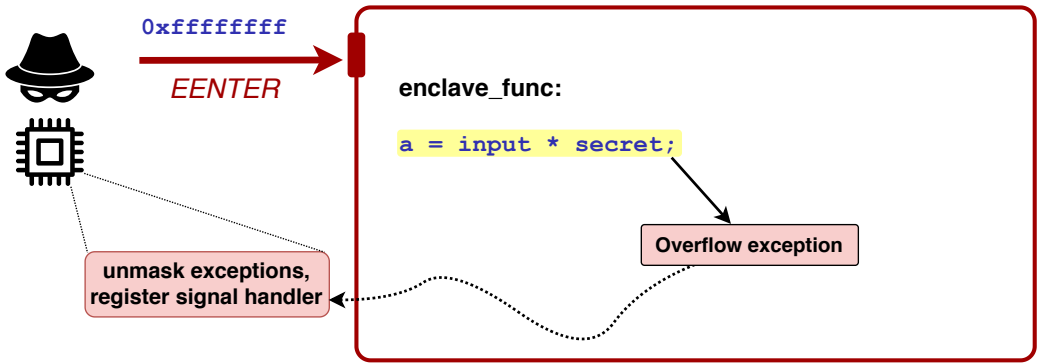
Case study 1: Floating-point exceptions as a side channel

Can we use overflows as a side channel to deduce secrets?



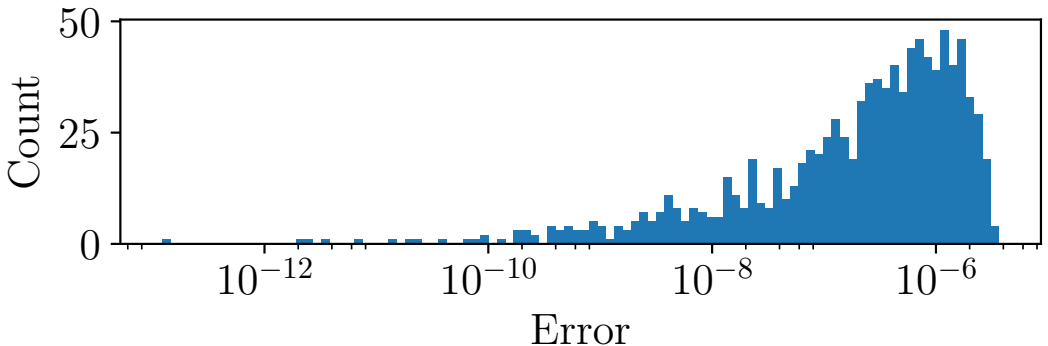
Case study 1: Floating-point exceptions as a side channel

Can we use overflows as a side channel to deduce secrets?

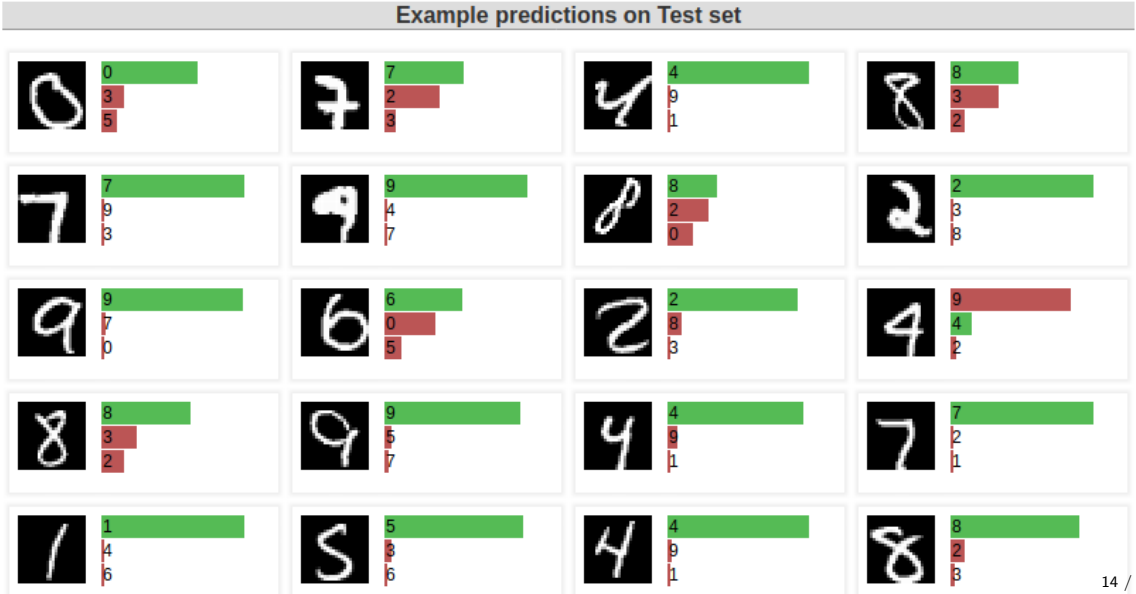


Case study 1: Floating-point exceptions as a side channel

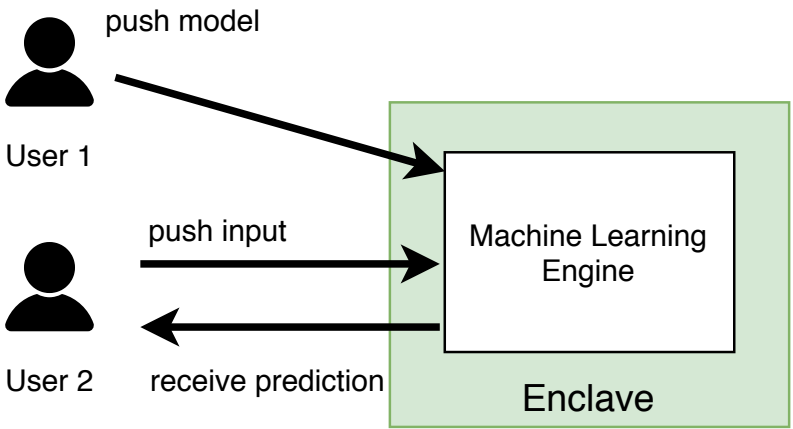
⇄ Binary search with deterministic # of steps retrieves secret



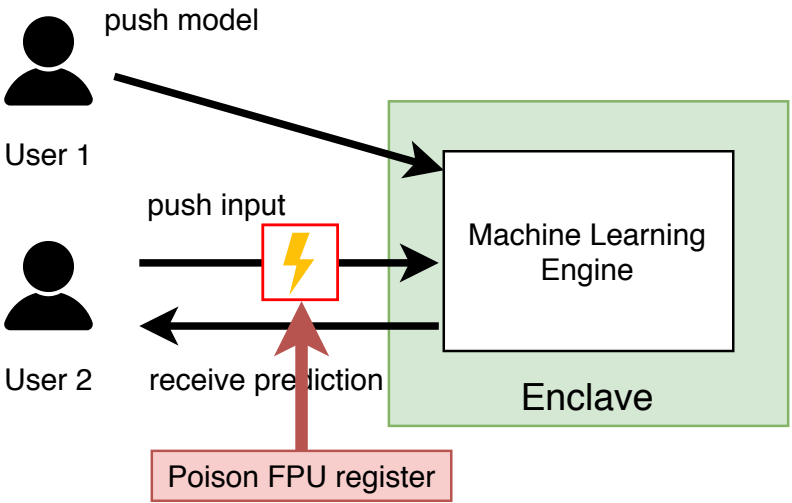
Case study 2: MNIST – ML handwriting recognition



Case study 2: MNIST – ML as an SGX Service



Case study 2: MNIST – ML as an SGX Service



Case study 2: MNIST – Predictions of 100 digits

Extended precision		Predicted digit count									
Rounding mode	Correct	0	1	2	3	4	5	6	7	8	9
Any mode	100%	9	14	8	10	14	8	9	14	3	11

x87 Extended precision: Default predictions

Case study 2: MNIST – Predictions of 100 digits

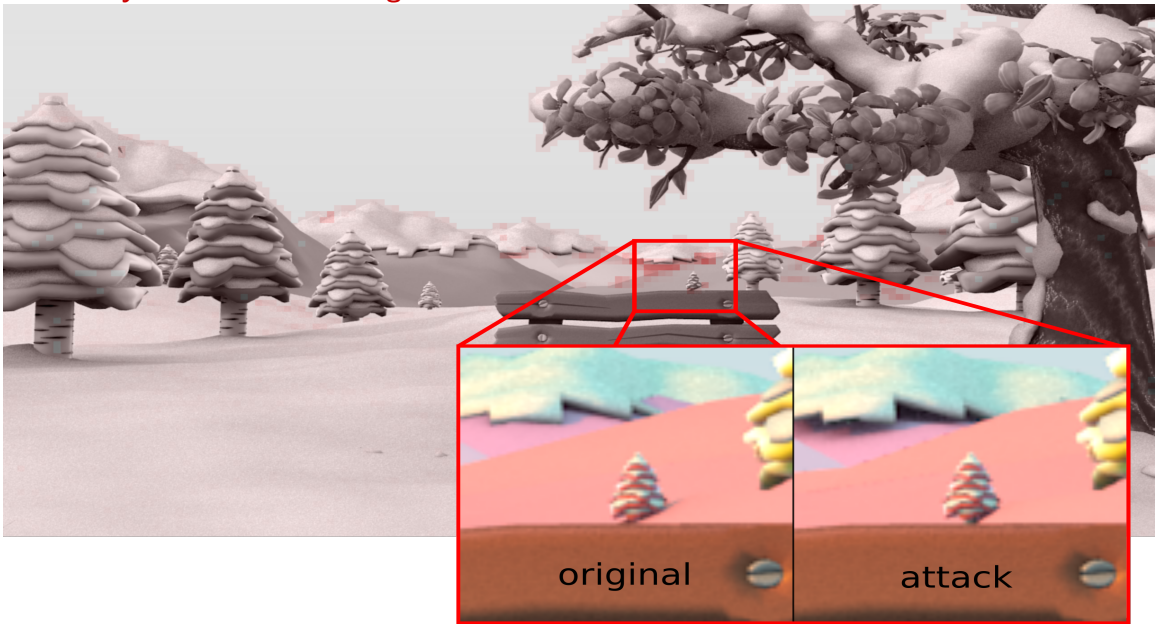
Extended precision		Predicted digit count									
Rounding mode	Correct	0	1	2	3	4	5	6	7	8	9
Any mode	100%	9	14	8	10	14	8	9	14	3	11

x87 Extended precision: Default predictions

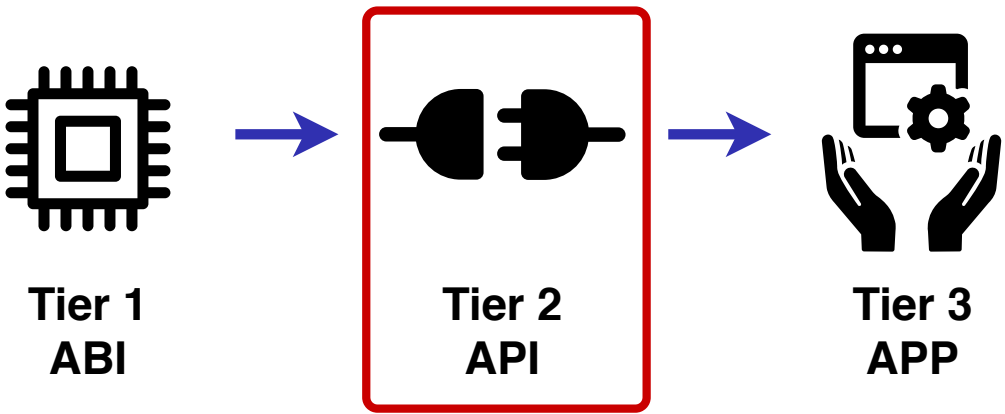
Single precision		Predicted digit count									
Rounding mode	Correct	0	1	2	3	4	5	6	7	8	9
Rounding down	8%	0	0	100	0	0	0	0	0	0	0

x87 Single precision: Attacked predictions

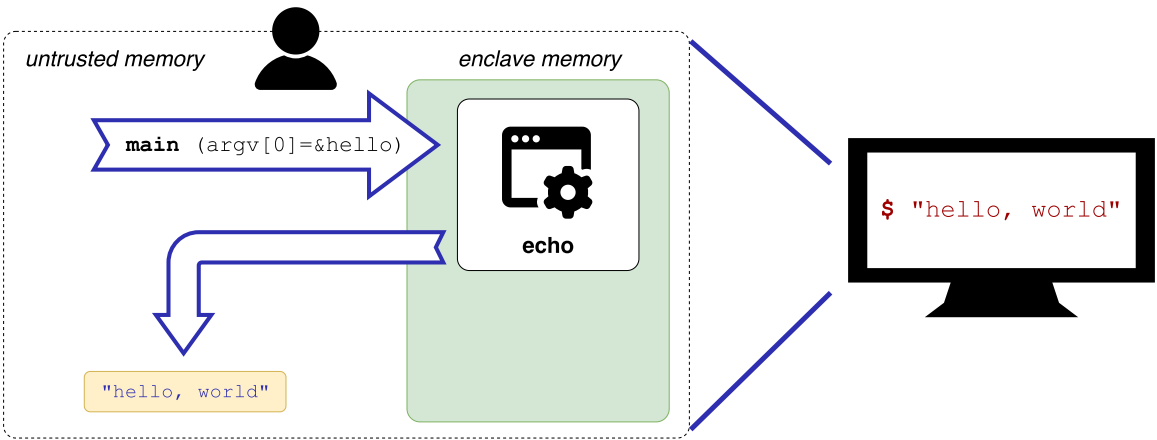
Case study 3: SPEC 2017. Image difference in Blender



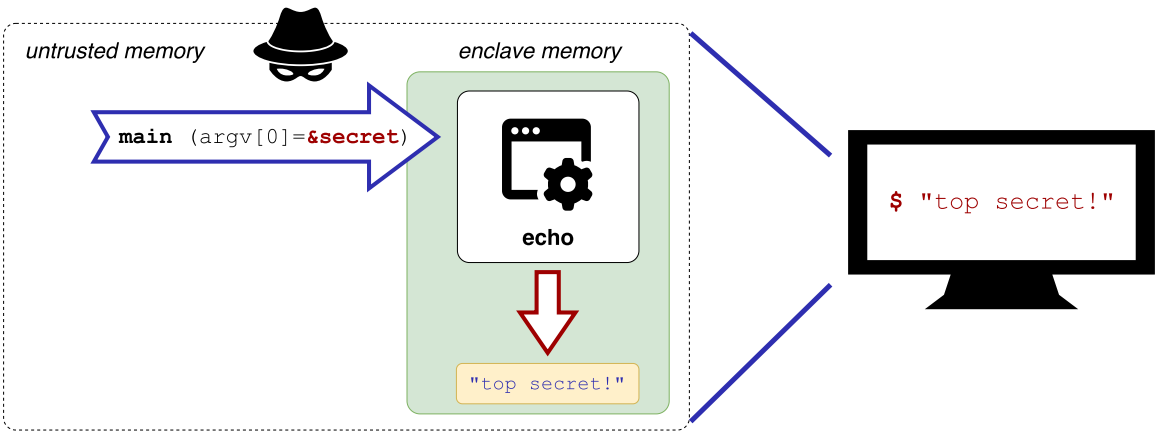
Tier 2: Sanitizing the enclave API



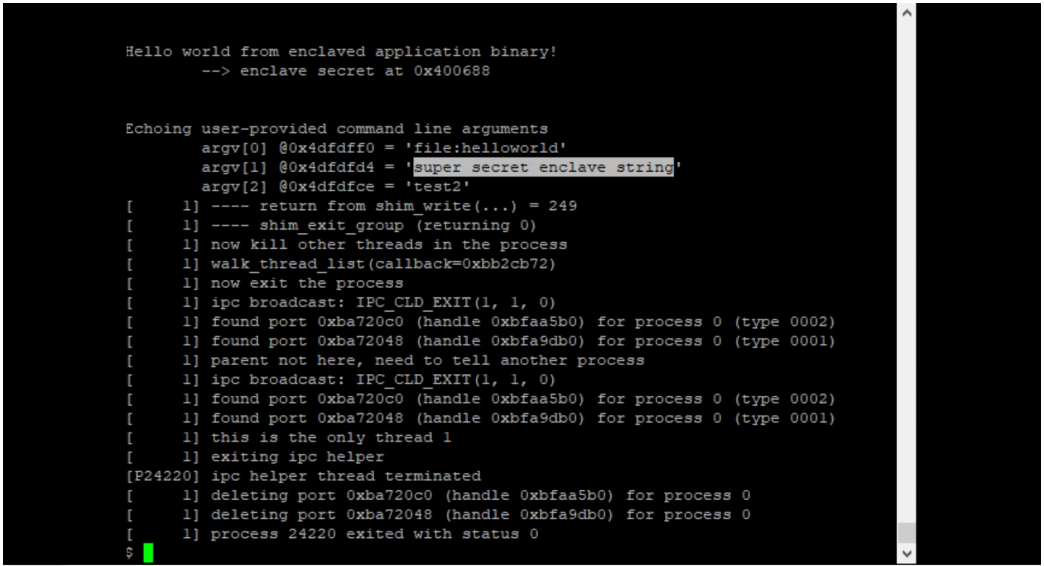
Validating pointer arguments: Confused deputy attacks



Validating pointer arguments: Confused deputy attacks



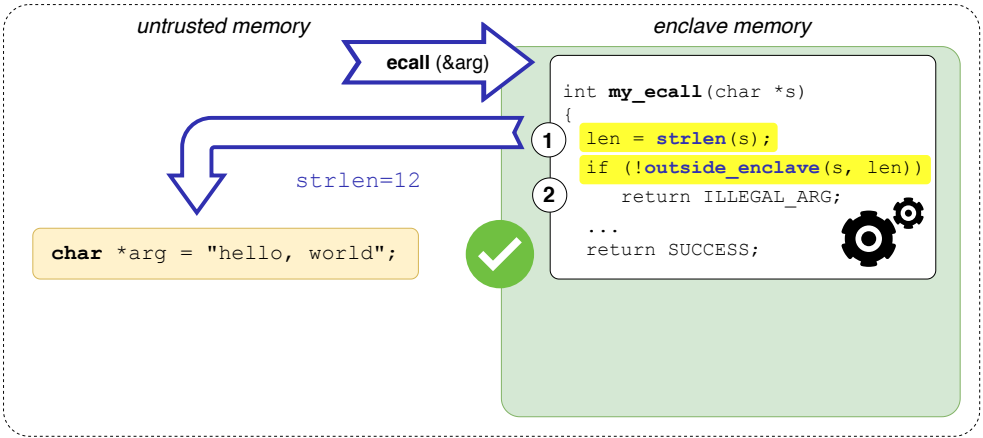
Validating pointer arguments: Confused deputy attacks



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

CVE-2018-3626

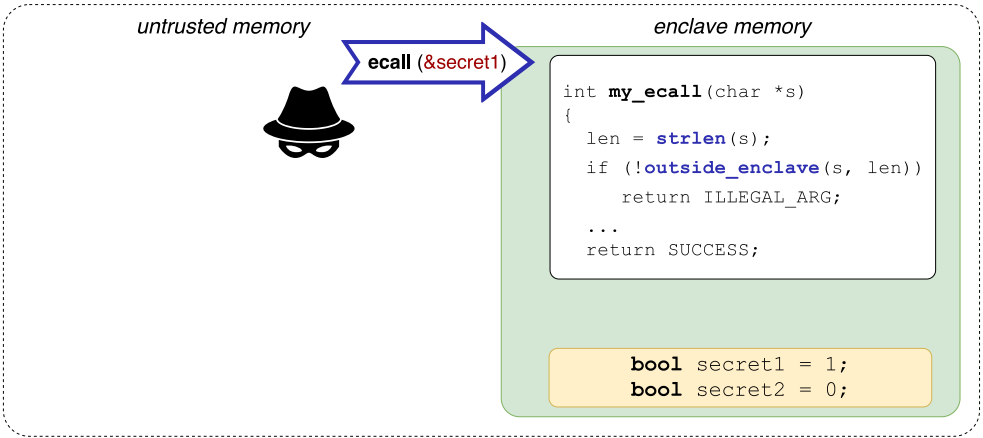
Idea: 2-stage approach ensures string arguments fall *entirely* outside enclave



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

CVE-2018-3626

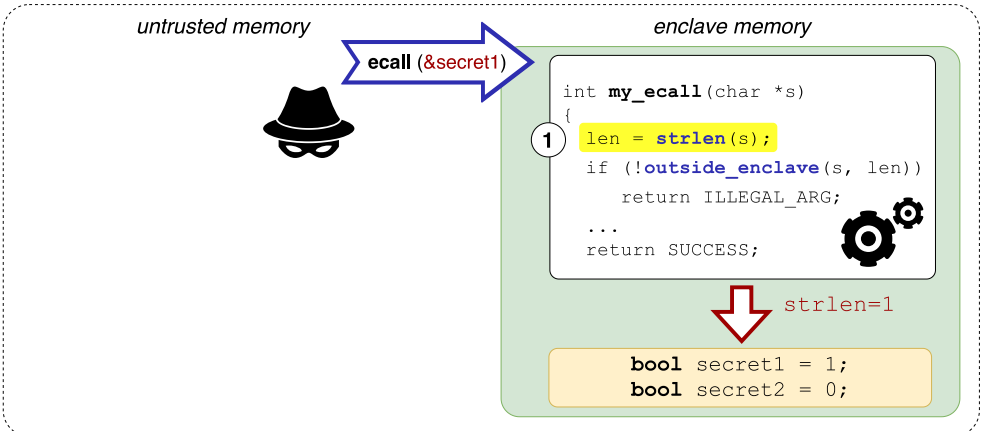
. . . but what if we try passing an illegal, in-enclave pointer anyway?



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

CVE-2018-3626

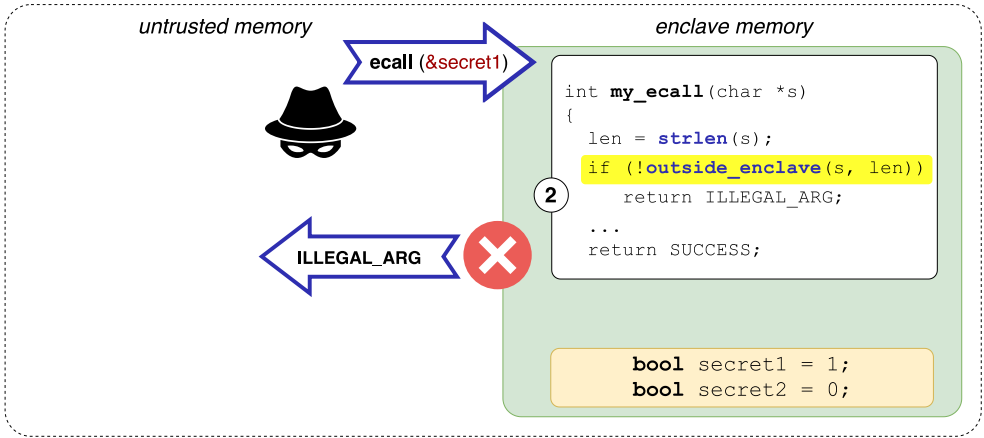
Enclave **first** computes length of secret, in-enclave buffer!



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

CVE-2018-3626

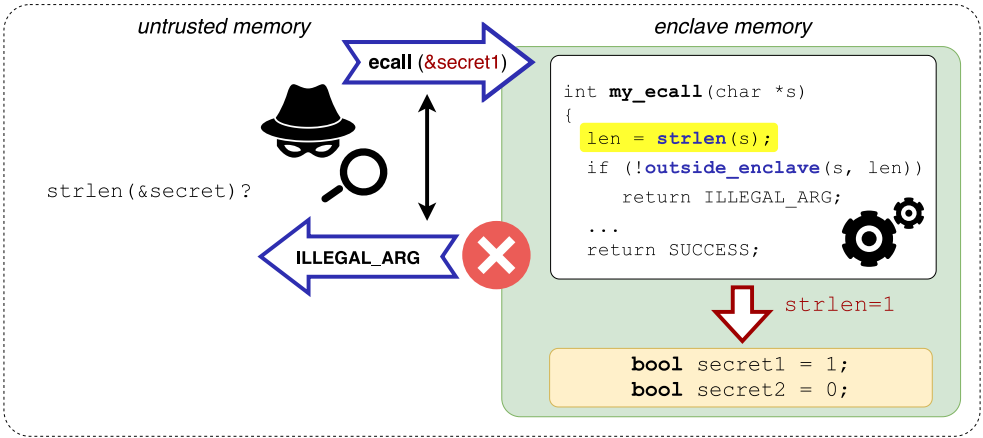
. . . and only **afterwards verifies** whether *entire string* falls outside enclave



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

CVE-2018-3626

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



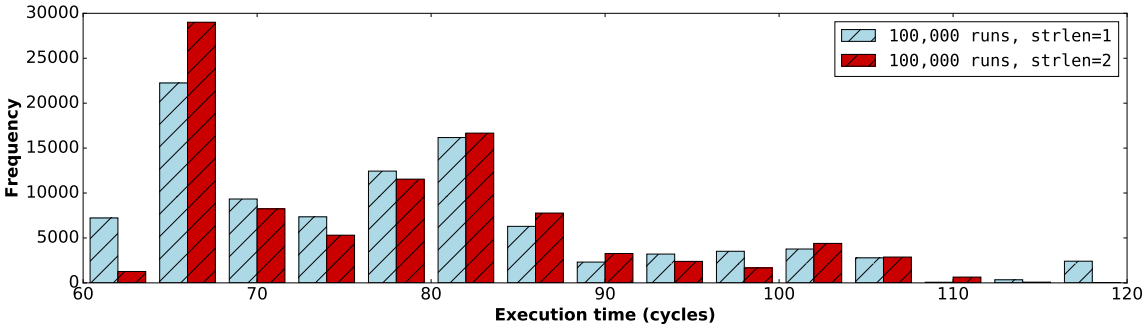
Challenge: Building a precise null byte oracle

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.

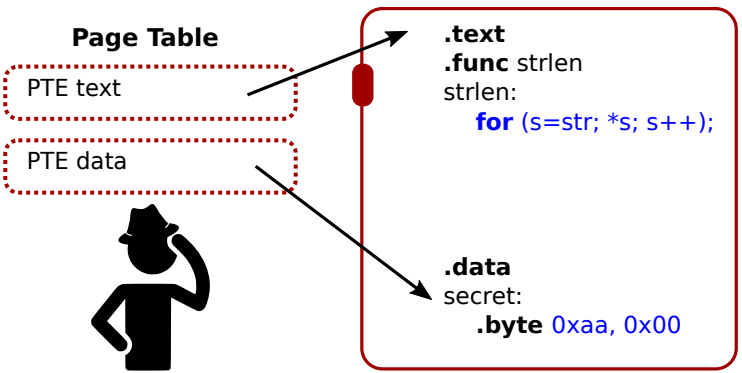
An attacker with access to the platform can see what pages are being executed or accessed. This side-channel vulnerability can be mitigated by aligning specific code and data blocks to exist entirely within a single page.

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?

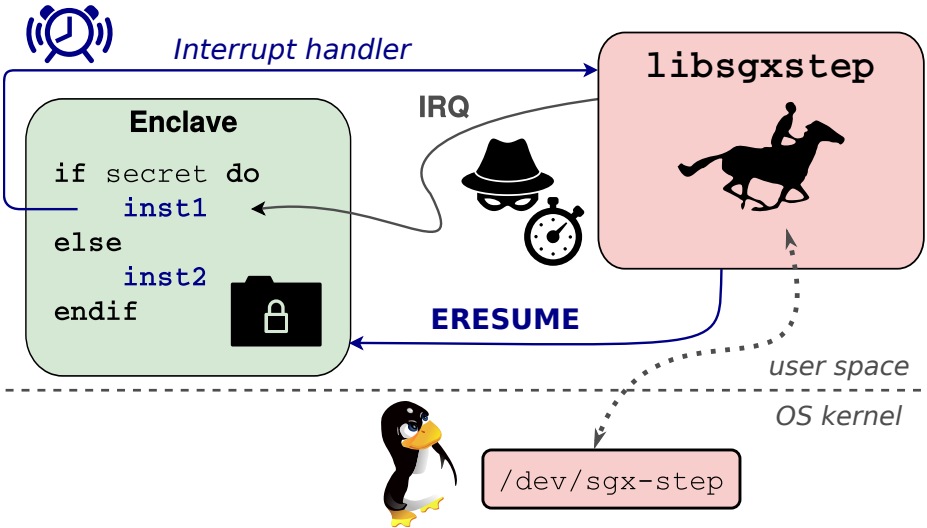
Temporal resolution: progress requires both code + data pages mapped in



Challenge: Counting `strlen()` loop iterations

What about leveraging interrupts?

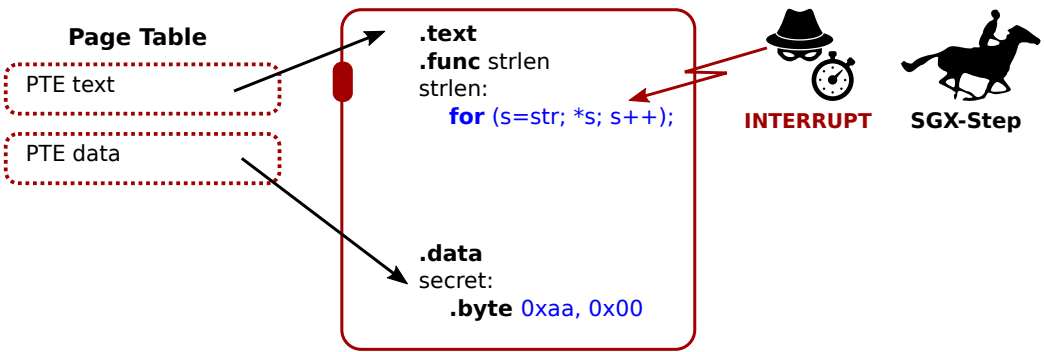
SGX-Step: Executing enclaves one instruction at a time



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

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

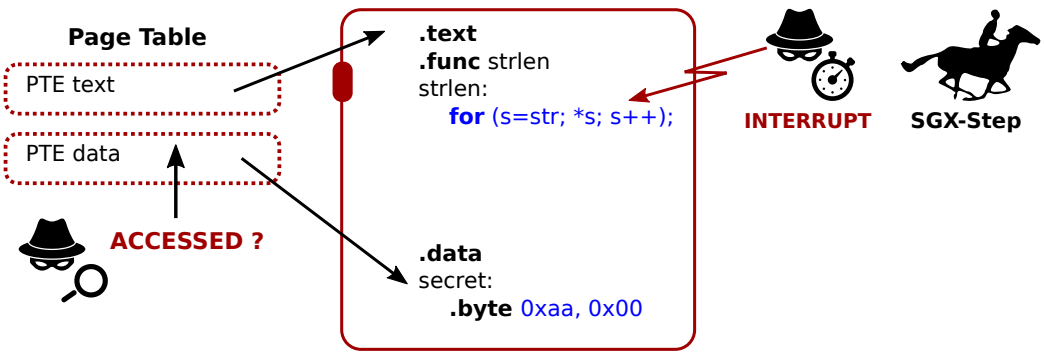
Execute *exactly* one enclave instruction → **timer interrupt**



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

SGX-Step

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

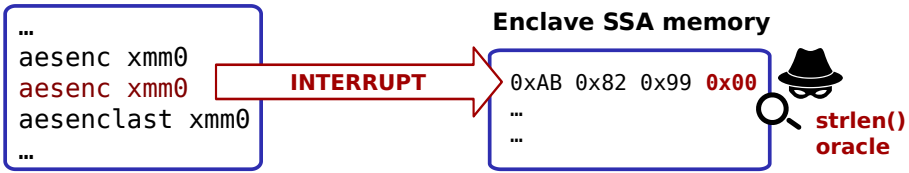
INTERRUPT

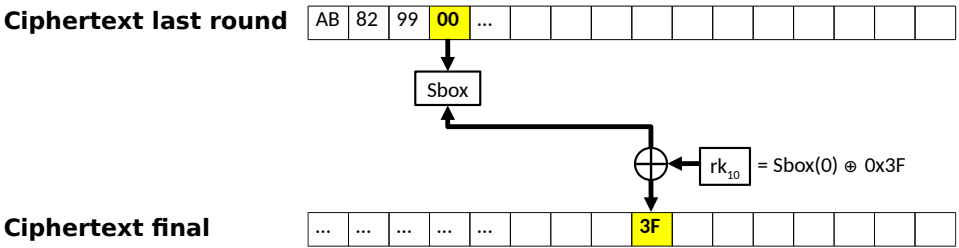
(store registers)

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

```
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!

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

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
https://github.com/fritzalder/faulty-point-unit/

Ramming Enclave Gates: A Systematic Vulnerability Assessment of TEE Shielding Runtimes

Jo Van Bulck [1], Fritz Alder [1], David Oswald [2]

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

rC3 — Remote Chaos Experience, December 2020

References I

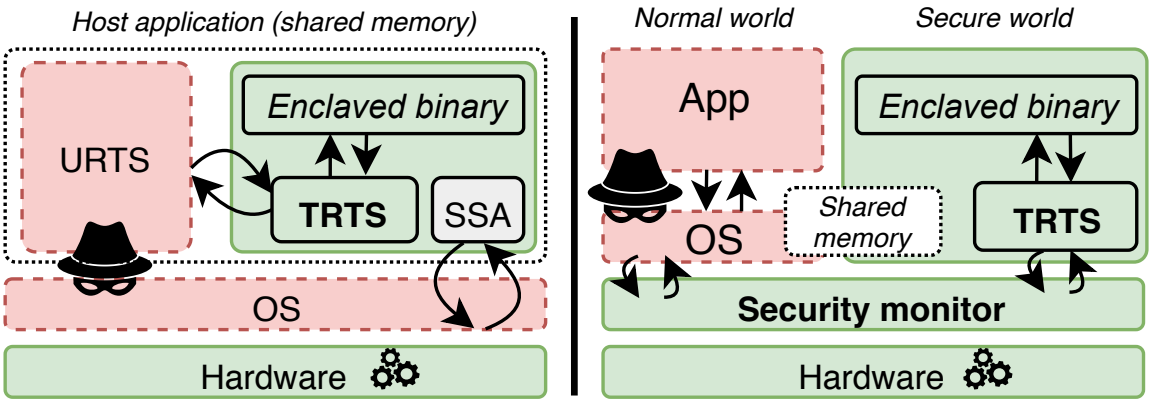
S. Checkoway and H. Shacham.
Iago Attacks: Why the System Call API is a Bad Untrusted RPC Interface.
In *International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS, pp. 253–264, 2013.

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

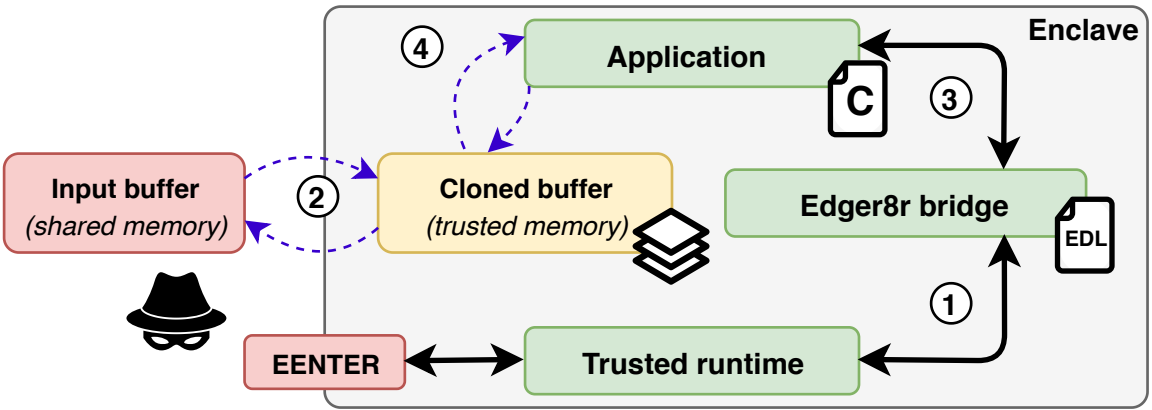
J. Van Bulck, F. Piessens, and R. Strackx.
SGX-Step: A practical attack framework for precise enclave execution control.
In *SysTEX*, pp. 4:1–4:6, 2017.

J. Van Bulck, 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.

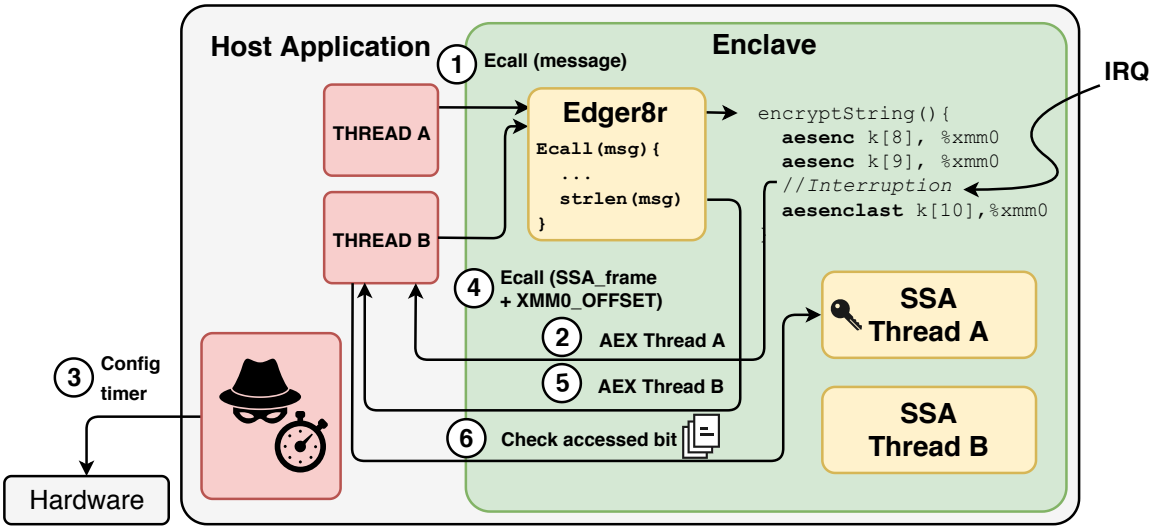
TEE design: Single-address-space vs. world-shared memory approaches



edger8r: Input/output buffer cloning



Intel SGX `strlen` oracle attack



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

SGX-Step: Executing enclaves one instruction at a time