

A Case for Unified ABI Shielding in Intel SGX Runtimes

Jo Van Bulck, Fritz Alder, Frank Piessens

SysTEX'22 Workshop, Lausanne, Switzerland, March 1, 2022

🏠 imec-DistriNet, KU Leuven ✉ jo.vanbulck@cs.kuleuven.be 🐦 [jovanbulck](https://twitter.com/jovanbulck)

01 INTEL
OPEN
SOURCE
.org

PROJECTS



COMMUNITY

ABOUT

Intel®
Software
Guard
Extensions

INTEL® SOFTWARE GUARD EXTENSIONS SDK FOR LINUX*

Open Enclave SDK

Build Trusted Execution Environment based applications with an open source SDK that provides access to hardware technologies as well as all platforms from Intel to ARM.



LSDS

Large-Scale Data & Systems Group

SGX-LKL: Linux Binaries in SGX Enclaves



GRAMINE



Gramine - a Library OS for Unmodified Applications

Open-Source community project driven by a core team of contributors.
Previously Graphene

Fortanix
EDP

ENCLAVE DEVELOPMENT PLATFORM

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

01 INTEL
OPEN
SOURCE
.org

PROJECTS



COMMUNITY

ABOUT

Intel®
Software
Guard
Extensions

INTEL® SOFTWARE GUARD EXTENSIONS SDK FOR LINUX*

Open E

Build Trusted
with an open
technologies.



What do these projects have
in common?

System software for trusted execution?



LSDS

Large-Scale Data & Systems Group

SGX-LKL: Linux Binaries in SGX Enclaves



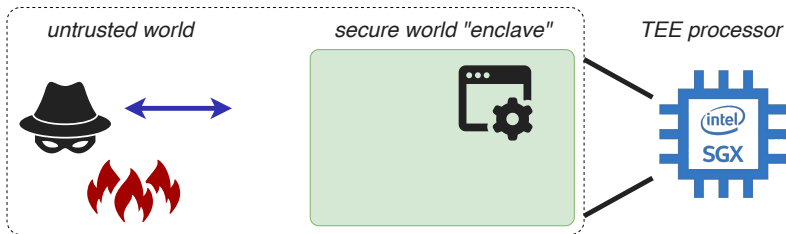
GRAMINE

Fortanix
EDP

ENCLAVE DEVELOPMENT PLATFORM

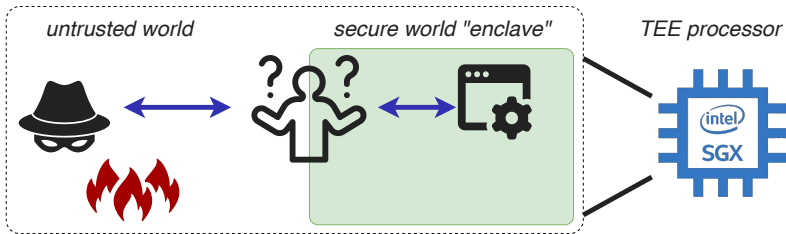
The Fortanix EDP is the preferred way for
writing Intel® SGX applications from
scratch.

Why isolation is not enough: Enclave shielding runtimes



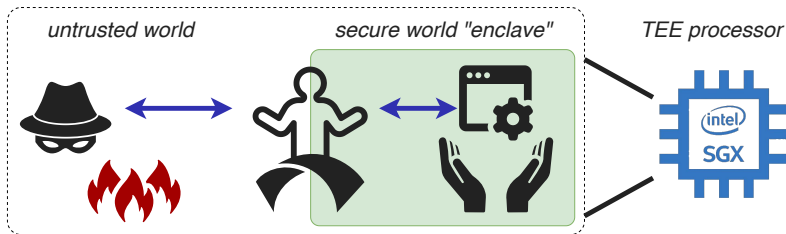
- TEE promise: enclave == “secure oasis” in a **hostile environment**

Why isolation is not enough: Enclave shielding runtimes



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

Why isolation is not enough: Enclave shielding runtimes




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

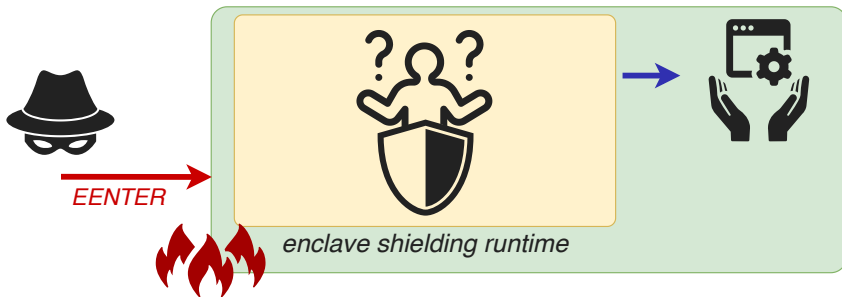


Shielding runtime == secure bridge on enclave entry/exit




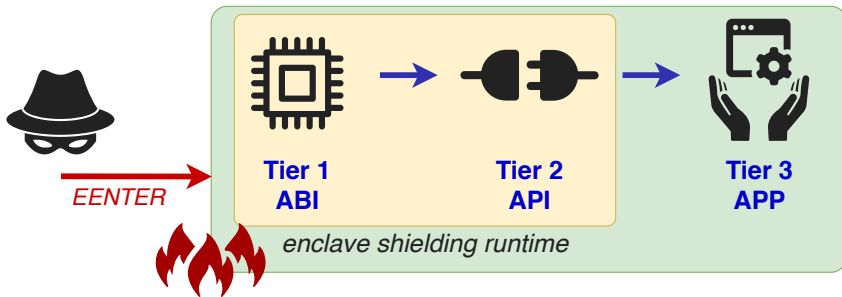
The big picture: Enclave shielding responsibilities

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



The big picture: Enclave shielding responsibilities

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



ABI vs. API sanitization responsibilities

Application Binary Interface

- Expectations by **compiler**
- Low-level **CPU** state (registers)
- Hand-written **assembly** stub

Application Programming Interface

- Expectations by **application** writer
- High-level **program** state (pointers)
- Automated **abstractions** (e.g.,
edger8r DSL, EDP type system)

ABI vs. API sanitization responsibilities

Application Binary Interface

- Expectations by **compiler**
- Low-level **CPU** state (registers)
- Hand-written **assembly** stub



(Needlessly) **duplicated**
effort across runtimes!

Application Programming Interface

- Expectations by **application** writer
- High-level **program** state (pointers)
- Automated **abstractions** (e.g.,
edger8r DSL, EDP type system)

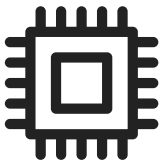


Depending on specific runtime
and programming model...

The background of the slide is a dense, colorful mosaic of many small, stylized human avatars. Each avatar has a unique combination of skin tone (ranging from light to dark brown), hair color (including blonde, brown, black, and red), and clothing (various colors and styles like shirts, sweaters, and suits). The overall effect is a vibrant, multi-colored pattern representing a diverse group of people.

Towards unified shielding in Intel SGX runtimes?

- We celebrate **application and programming language** diversity
- ... but unification of shared insights at the **ABI level!**



**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)



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!



Non-trivial for x86 ISA!

SHELDON COOPER
presents
-FUN- WITH FLAGS

● REC

$$\begin{aligned}P &= P_{nc} + P_{nc} \\ \nabla^2 \phi &= 0 \\ P_{nc} &= (x_{nc}) / V \\ P_{nc} &= 2P(2) \\ &= P_{nc}\end{aligned}$$



x86 string instructions: Direction Flag (DF) operation



- 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

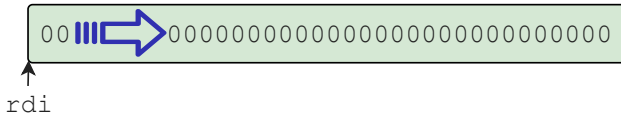


- **x86 rep string instructions** to speed up streamed memory operations
- Default operate **left-to-right**

```
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

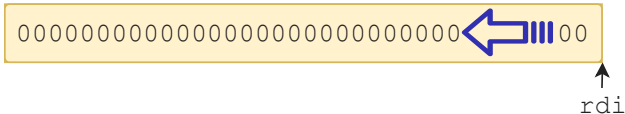


- x86 *rep* string instructions to speed up streamed memory operations
- Default operate **left-to-right**, unless software sets *RFLAGS.DF=1*

```
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
```



SGX-DF: Inverting enclaved string memory operations

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



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



EENTER

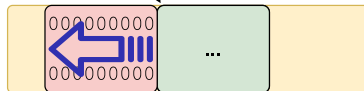
RFLAGS.DF = 1

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](#)

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



Summary: Intel SGX ABI vulnerability landscape

	SGX-SDK	OE	EDP	Gramine	Enarx	GoTEE	SGX-LKL	OpenSGX
Entry flags [3]	●	●	●	●	—	—	●	—
Entry stack [3]	○	○	○	●	—	—	●	—
Exit registers [3]	○	○	○	○	—	—	●	—
Entry FPU [1]	●	●	●	○	○	●	●	—
Exception stack [2]	●	●	○	○	●	—	●	—



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

Summary: Intel SGX ABI vulnerability landscape

	SGX-SDK	OE	EDP	Gramine	Enarx	GoTEE	SGX-LKL	OpenSGX
Entry flags [3]	●	●	●	●	—	—	●	—
Entry stack [3]	○	○	○	●	—	—	●	—
Exit registers [3]	○	○	○	○	—	—	●	—
Entry FPU [1]	●	●	●	○	○	●	●	—
Exception stack [2]	●	●	○	○	●	—	●	—
Production?	✓	✓	✓	✓	✓	✗	✗	✗



(Aspired) **production-quality** runtimes vs. research prototypes



KEEP CALM

AND

SHOW ME THE NUMBERS

Summary: Intel SGX ABI shielding layer metrics

	SGX-SDK	OE	EDP	Gramine	Enarx	GoTEE	SGX-LKL	OpenSGX
LoC ABI stub	301	277	248	427	169	239	103	49
LoC changed	243	589	187	1,840	844	65	47	0
Production?	✓	✓	✓	✓	✓	✗	✗	✗
Entry flags [3]	●	●	●	●	—	—	●	—
Entry stack [3]	○	○	○	●	—	—	●	—
Exit registers [3]	○	○	○	○	—	—	●	—
Entry FPU [1]	●	●	●	○	○	●	●	—
Exception stack [2]	●	●	○	○	●	—	●	—



Size: Non-trivial: > 100s lines of hand-written, vulnerable asm code

Summary: Intel SGX ABI shielding layer metrics

	SGX-SDK	OE	EDP	Gramine	Enarx	GoTEE	SGX-LKL	OpenSGX
LoC ABI stub	301	277	248	427	169	239	103	49
LoC changed	243	589	187	1,840	844	65	47	0
Production?	✓	✓	✓	✓	✓	✗	✗	✗
Entry flags [3]	●	●	●	●	—	—	●	—
Entry stack [3]	○	○	○	●	—	—	●	—
Exit registers [3]	○	○	○	○	—	—	●	—
Entry FPU [1]	●	●	●	○	○	●	●	—
Exception stack [2]	●	●	○	○	●	—	●	—



History: Maintaining ABI code is an *ongoing* and *living* effort!

Summary: Intel SGX ABI patch timelines

	SGX-SDK	OE	EDP	Gramine	Enarx
Initial commit	° 24/06/16	° 29/08/17	° 07/12/18	° 20/06/16	° 20/02/20
Direction flag [3]	▣ 17/10/19	▣ 09/10/19	07/12/18	01/05/19	20/03/20
Alignment-check flag [3]	▣ 12/11/19	▣ 09/10/19	▣ 21/10/19 10/02/20	▣ 19/11/19	★ 17/02/22
FPU extended state [1]	▣ 16/01/20	09/10/19 ▣ 14/07/20	▣ 10/02/20 ▣ 19/06/20	17/10/19	29/05/20
Exception stack [2]	▣ 13/07/21	▣ 13/07/21	N/A	01/04/19 31/01/20	▣ 22/10/21



Security: Already known, not communicated, open gap

Summary: Intel SGX ABI patch timelines

	SGX-SDK	OE	EDP	Gramine	Enarx
Initial commit	° 24/06/16	° 29/08/17	° 07/12/18	° 20/06/16	° 20/02/20
Direction flag [3]	📅 17/10/19	📅 09/10/19	07/12/18	01/05/19	20/03/20
Alignment-check flag [3]	📅 12/11/19	📅 09/10/19	📅 21/10/19	📅 19/11/19	★ 17/02/22
			10/02/20		
FPU extended state [1]	📅 16/01/20	09/10/19	📅 10/02/20	17/10/19	29/05/20
		📅 14/07/20	📅 19/06/20		
Exception stack [2]	📅 13/07/21	📅 13/07/21	N/A	01/04/19	📅 22/10/21
				31/01/20	



Deepened understanding: Importance of academic research!

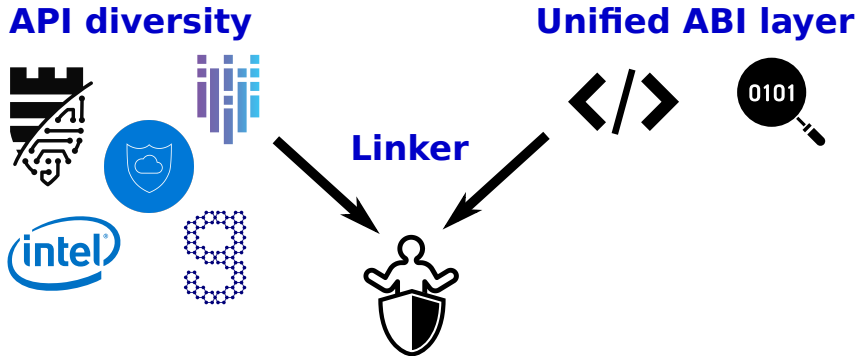
Summary: Intel SGX ABI patch timelines

	SGX-SDK	OE	EDP	Gramine	Enarx
Initial commit	° 24/06/16	° 29/08/17	° 07/12/18	° 20/06/16	° 20/02/20
Direction flag [3]	■ 17/10/19	■ 09/10/19	07/12/18	01/05/19	20/03/20
Alignment-check flag [3]	■ 12/11/19	■ 09/10/19	■ 21/10/19 10/02/20	■ 19/11/19	★ 17/02/22
FPU extended state [1]	■ 16/01/20	09/10/19 ■ 14/07/20	■ 10/02/20 ■ 19/06/20	17/10/19	29/05/20
Exception stack [2]	■ 13/07/21	■ 13/07/21	N/A	01/04/19 31/01/20	■ 22/10/21



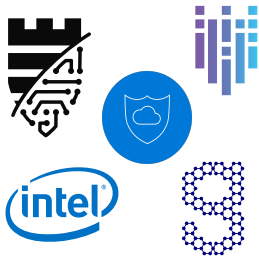
Systematization: Revealed *missing patch*, fixed in Enarx v0.2.1

Towards unified ABI shielding for Intel SGX runtimes

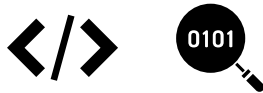


Towards unified ABI shielding for Intel SGX runtimes

API diversity



Unified ABI layer






Linker



Thank you! Questions?

 <https://github.com/jovanbulck/sgx-abi-data>

-  Fritz Alder, Jo Van Bulck, David Oswald, and Frank Piessens.
Faulty point unit: ABI poisoning attacks on Intel SGX.
In *36th Annual Computer Security Applications Conference (ACSAC)*, pages 415–427, December 2020.
-  Jinhua Cui, Jason Zhijingcheng Yu, Shweta Shinde, Prateek Saxena, and Zhiping Cai.
SmashEx: smashing SGX enclaves using exceptions.
In *28th ACM Conference on Computer and Communications Security (CCS)*, page 779–793, 2021.

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

A tale of two worlds: Assessing the vulnerability of enclave shielding runtimes.

In *26th ACM Conference on Computer and Communications Security (CCS)*, pages 1741–1758, November 2019.

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

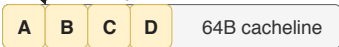


There's more! [Alignment Check \(AC\) flag](#) enables **exceptions for unaligned data accesses** → *intra-cacheline side-channel* 😊

enclave_func:

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

enclave_data:

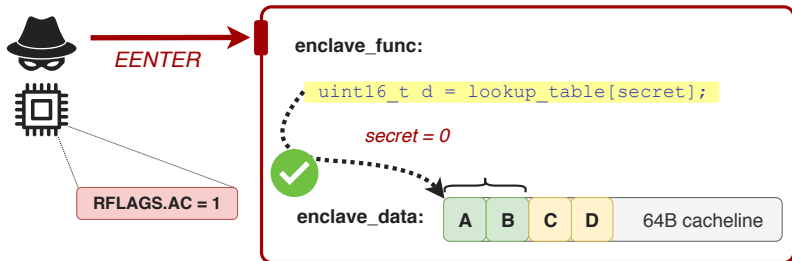


SGX-AC: Building an intra-cacheline side-channel



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

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



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



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

