



# Opening Pandora's Box: Uncovering Pervasive Bugs in Intel SGX

## Runtimes with Truthful Symbolic Execution

**Jo Van Bulck**

*Joined work with Fritz Alder, Lesly-Ann Daniel, Frank Piessens, David Oswald*

Intel IPAS Tech Sharing – June 25, 2024 (online)

🏠 imec-DistriNet, KU Leuven, Belgium 📩 jo.vanbulck@cs.kuleuven.be 🐦 @jovanbulck



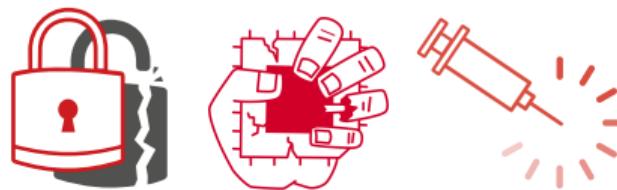
UNIVERSITY OF  
BIRMINGHAM



- Postdoctoral researcher @DistriNet, KU Leuven, Belgium  
→ PhD “Microarchitectural Side-Channel Attacks for Privileged Software Adversaries”
- Trust across the **system stack**: App > compiler > OS > CPU >  $\mu$ -arch
- Integrated **attack-defense** perspective and **open-source** prototypes



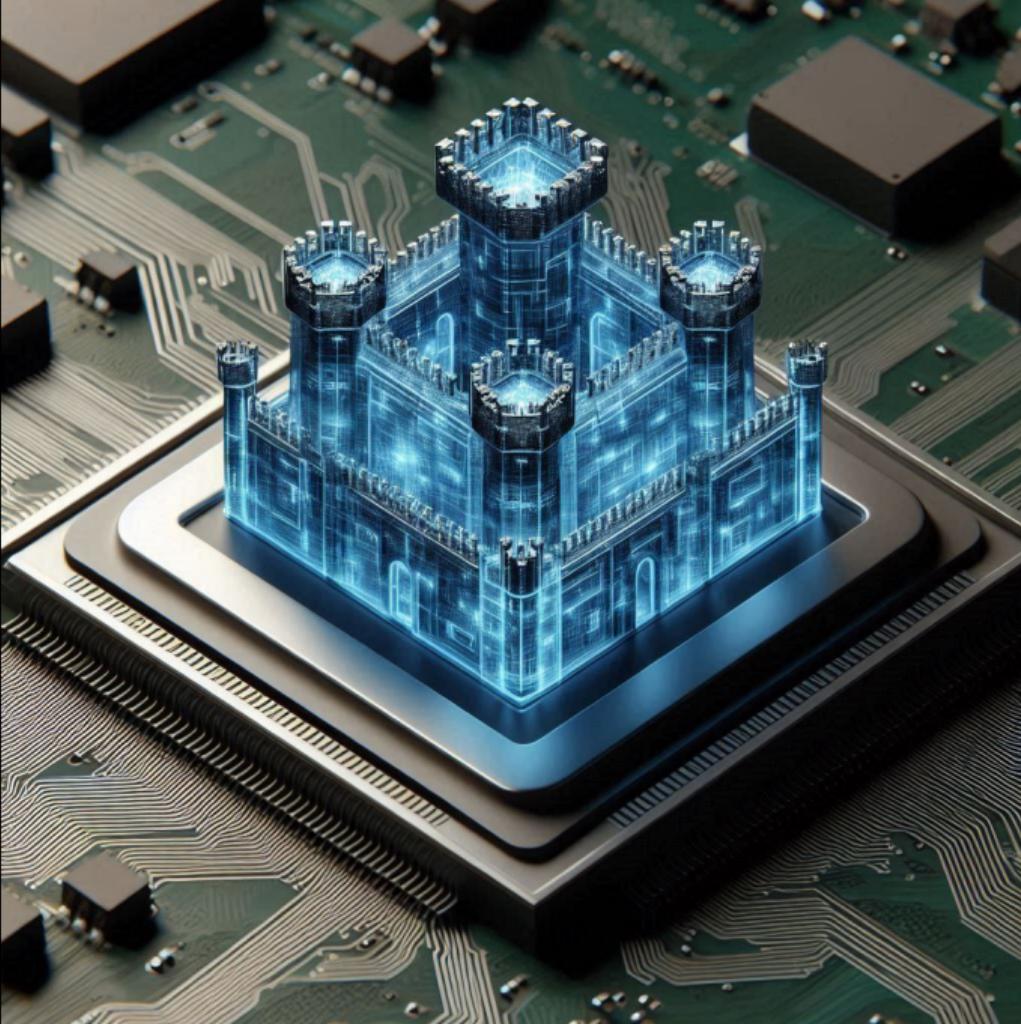
Side-channel analysis  
(*SGX-Step, AEX-Notify*)



Transient-execution attacks  
(*Intel x86 SGX*)

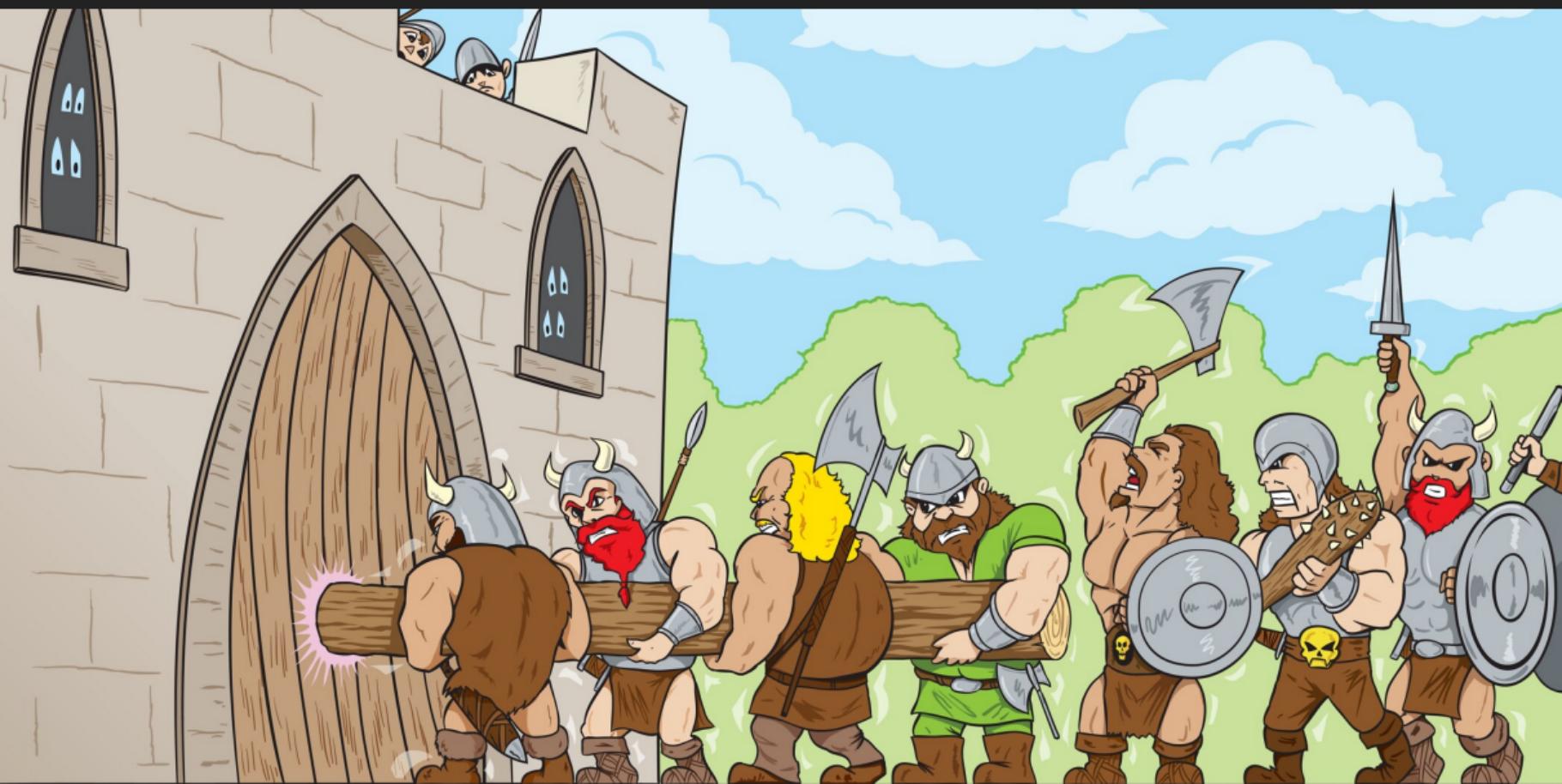


Embedded trust  
(*TI MSP430*)



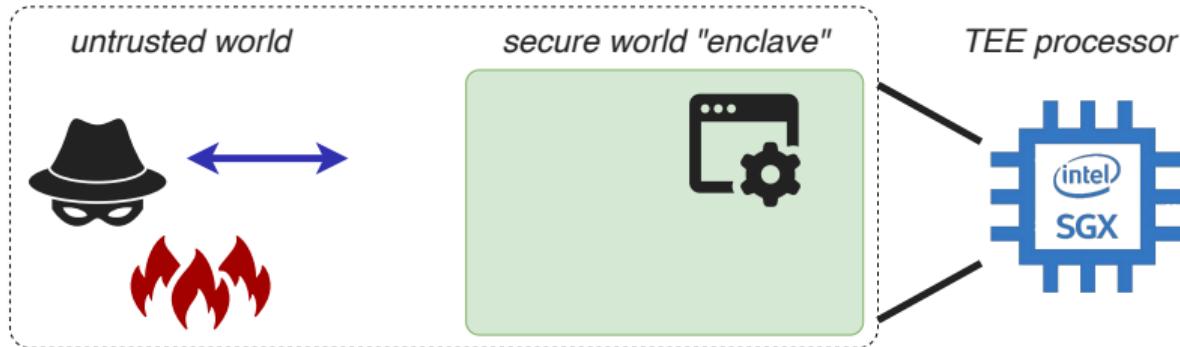
DALL-E 3

# Besieging the SGX Fortress: Software Interface Attacks

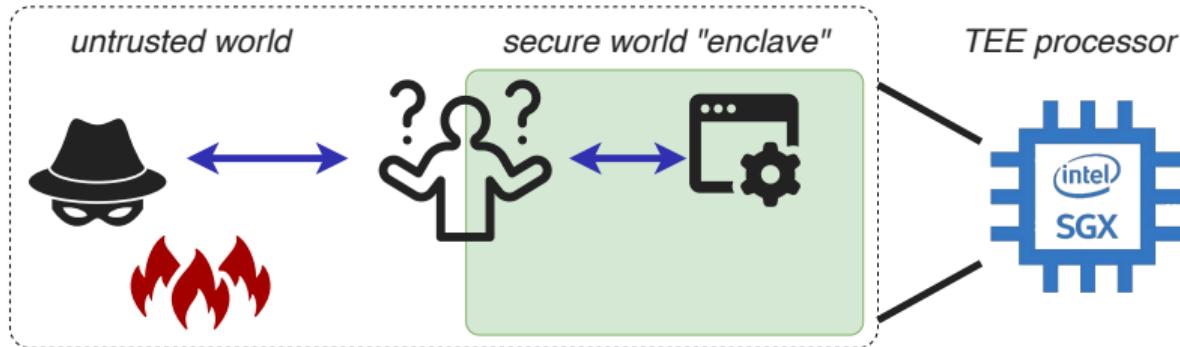


 <b>Improper sanitization of MXCSR and RFLAGS</b>	Moderate
GHSA-5gfr-m6mx-p5w4 published on Jul 17, 2023 by radhikaj	
 <b>Intel Processor Stale Data Read from Legacy xAPIC</b>	Moderate
GHSA-v3vm-9h66-wm76 published on Aug 13, 2022 by radhikaj	
 <b>Intel Processor MMIO Stale Data Vulnerabilities</b>	Moderate
GHSA-wm9w-8857-8fgj published on Jun 14, 2022 by radhikaj	
 <b>Open Enclave SDK Elevation of Privilege Vulnerability</b>	Moderate
GHSA-mj87-466f-jq42 published on Jul 13, 2021 by radhikaj	
 <b>Socket syscalls can leak enclave memory contents</b>	Moderate
GHSA-525h-wxcc-f66m published on Oct 12, 2020 by radhikaj	
 <b>x87 FPU operations in enclaves are vulnerable to ABI poisoning</b>	Low
GHSA-7wjx-wcwg-w999 published on Jul 14, 2020 by CodeMonkeyLeet	
 <b>Intel SGX Load Value Injection (LVI) vulnerability</b>	Moderate
GHSA-8934-g2pr-x6cg published on Mar 12, 2020 by radhikaj	
 <b>Enclave heap memory disclosure vulnerability</b>	Moderate
GHSA-mg2p-657r-46cj published on Oct 8, 2019 by CodeMonkeyLeet	

## Context: Writing “Secure” Enclave Software is Hard...

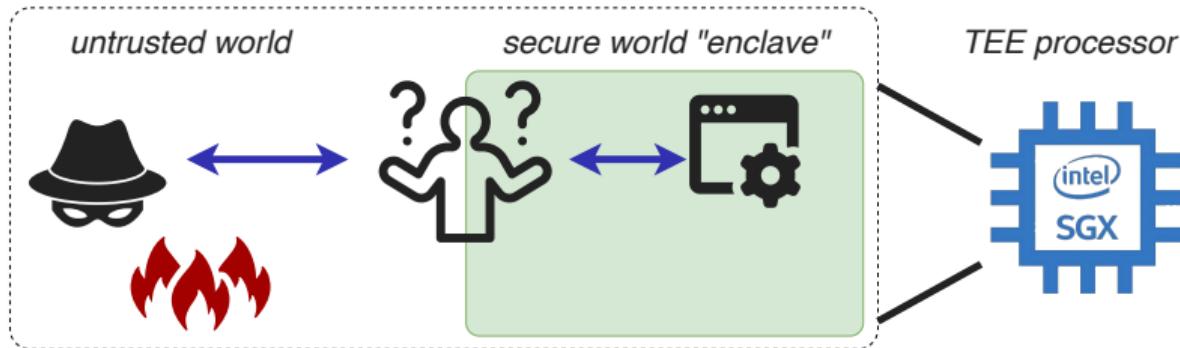


## Context: Writing “Secure” Enclave Software is Hard...



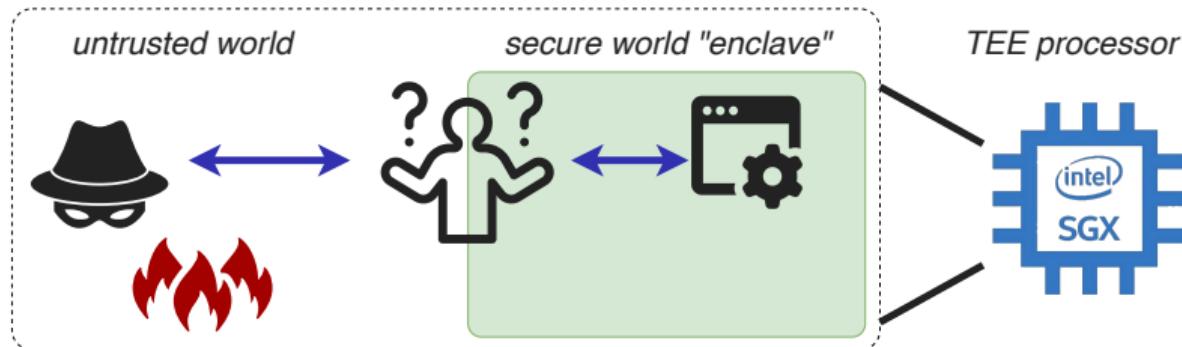
- **API level:** Sanitize pointer arguments in shared address space

## Context: Writing “Secure” Enclave Software is Hard...



- **API level**: Sanitize pointer arguments in shared address space
- **ABI level**: Sanitize low-level CPU configuration registers

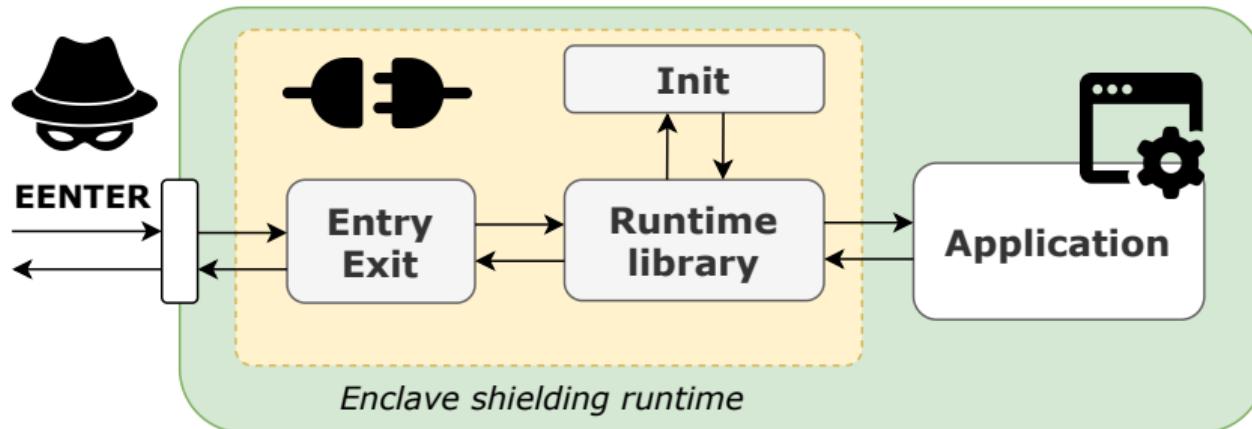
## Context: Writing “Secure” Enclave Software is Hard...



- **API level**: Sanitize pointer arguments in shared address space
- **ABI level**: Sanitize low-level CPU configuration registers
- **$\mu$ -arch level**: Spectre/LVI → `lfence`;  $\text{\AA}$ EPIC/MMIO stale data → `verw`; cacheline GPU leak → avoid `dword0/1...`



## Solution: Enclave Shielding Runtimes



**Key idea:** Transparent input sanitization on enclave entry/exit



PROJECTS 01

COMMUNITY

ABOUT

Intel®  
Software  
Guard  
Extensions

## INTEL® SOFTWARE GUARD EXTENSIONS SDK FOR LINUX\*

Open Enclave SDK

### Open Enclave SDK

Build Trusted Execution Environment based on Intel® SGX with an open source SDK that provides a simple API for developers to use Intel® SGX technologies as well as all platforms from

Enarx | Enarx

Enarx

GRAMINE



SGX-LKL | LSDS - Large-Scale Data & Systems Group



# LSDS

Large-Scale Data & Systems Group

SGX-LKL: Linux Binaries in SGX Enclaves



Fortanix  
EDP

### ENCLAVE DEVELOPMENT PLATFORM

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

## Gramine - a Library OS for Unmodified Applications

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



# Intel Software Guard Extensions (SGX)



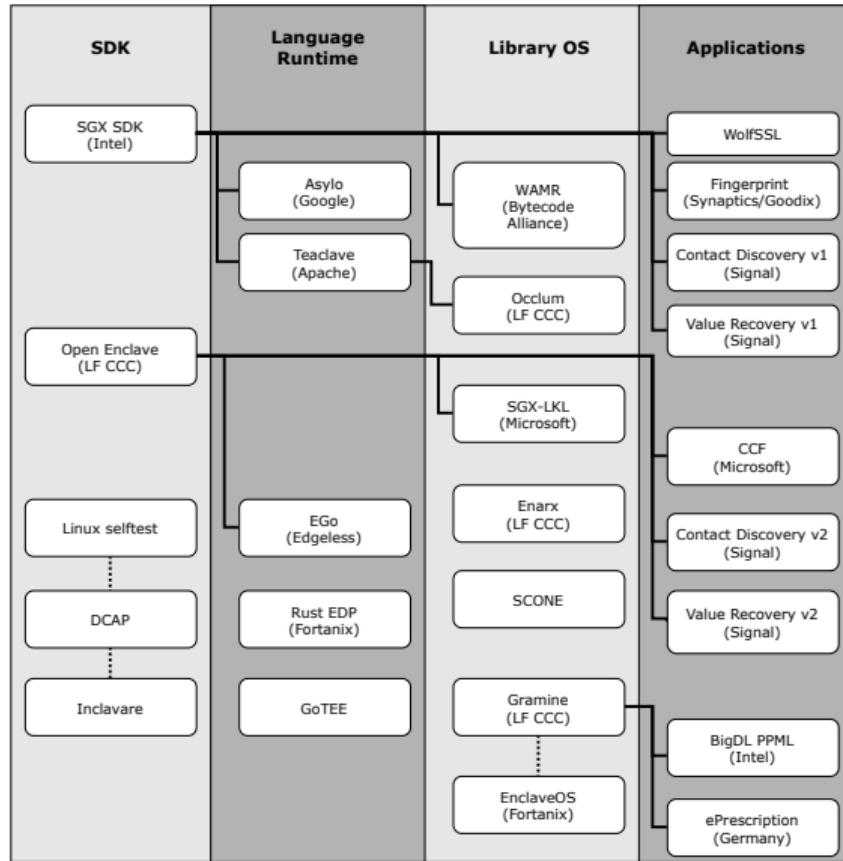
Enarx (Red Hat)



Asylo (Google)

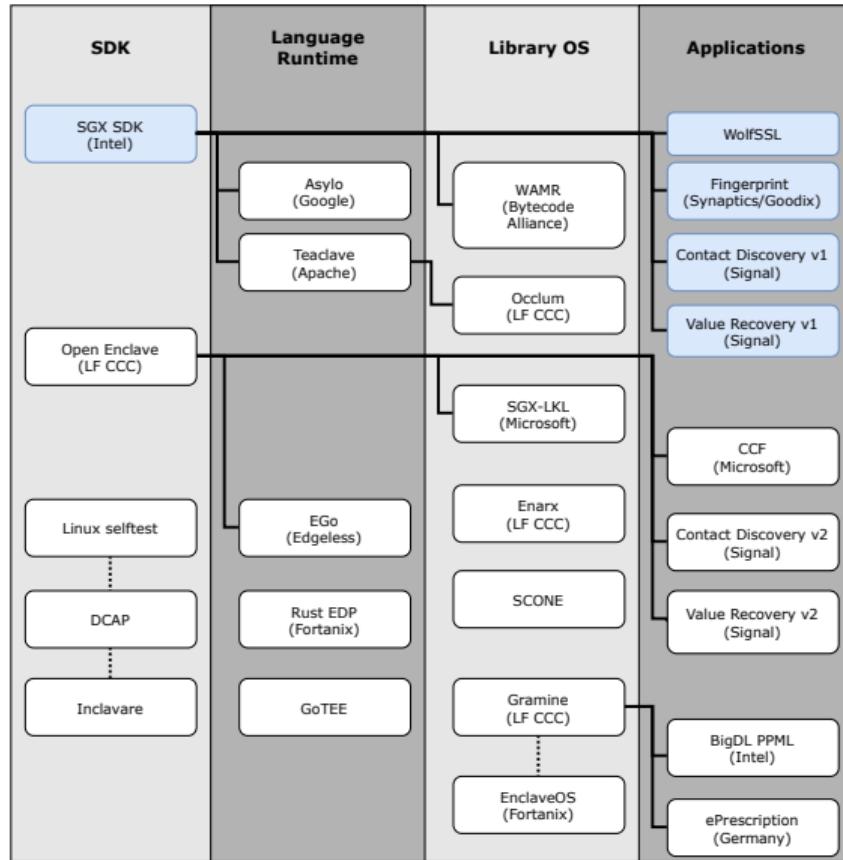


# Challenge: Diverse Intel SGX Software Ecosystem



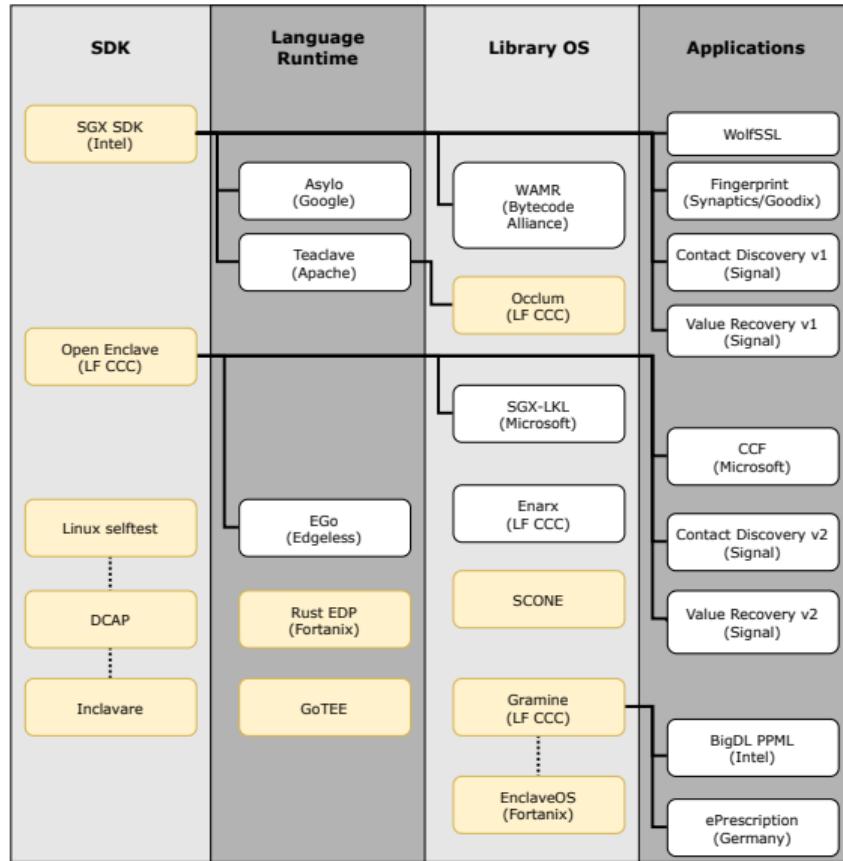
- **Ecosystem:** Diverse programming paradigms & abstractions

# Challenge: Diverse Intel SGX Software Ecosystem



- **Ecosystem:** Diverse programming paradigms & abstractions
- **Prior work:** Selected applications on Intel SDK (e.g., NULL pointers)

# Challenge: Diverse Intel SGX Software Ecosystem



- **Ecosystem:** Diverse programming paradigms & abstractions
- **Prior work:** Selected applications on Intel SDK (e.g., NULL pointers)
- **Pandora:** Runtime-agnostic & truthful symbolic execution
  1. Exact attested memory binary
  2. Vulnerability detection plugins



## 1. Truthful Symbolic Execution

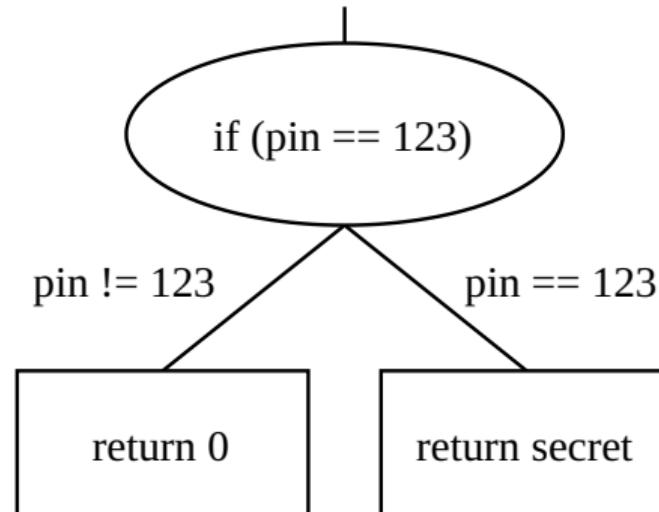
---

# Background: Symbolic Execution and angr

```
1 int ecall(int pin){  
2     if(pin == 123){  
3         return secret;  
4     } else {  
5         return 0;  
6     }  
7 }
```



<https://angr.io/>



- Symbolic execution uses a **constraint solver**
- Execution works on **instruction-level**, i.e., as close to the binary as possible

# TeeRex: Symbolic validation of [user\_check] pointers

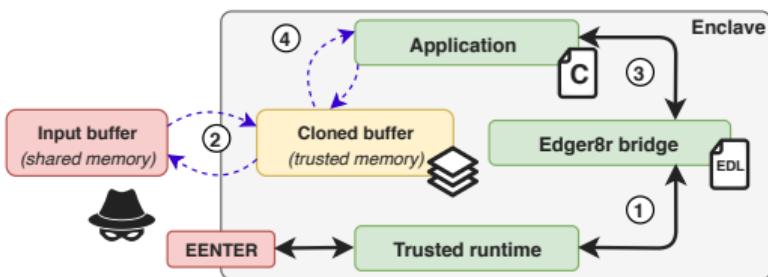
## TeeRex: Validate untrusted pointer dereferences

### ☺ Application-level [user\_check] pointers

- Missing or incorrect  
`sgx_is_outside_enclave()`
- TOCTOU: Double fetch ptr-to\_ptr
- NULL pointer dereference

### ☹ Ad-hoc, no “principled” criteria:

- Only user pointers: writes + NULL
- Ignore enclave trusted runtime
- Intel SDK-specific



# Guardian: Symbolic validation of “orderliness”

Our framework	Intel SGX SDK
EntryAddress	enclave_entry call
<b>Entry</b>	
EntrySanitisationDone	enter_enclave call
	sgx_ecall_function call
SBegin	ecall_function call
<b>Secure</b>	
OBegin	ocall_function call
<b>Ocall</b>	
OEnd	ocall_function return
<b>Secure</b>	
SEnd	ecall_function return
	sgx_ecall_function return
<b>Exit</b>	
ExitAddress	enclave_entry exit

**Guardian:** Restrict application access to unprotected memory

- ☺ “Principled” state machine: [Entry](#), [secure](#), [ocall](#), [exit](#)
- ☹ Coarse-grained access control; ignore “privileged” trusted runtime; ABI ad-hoc blocklist; Intel SDK-specific

□ Antonino et al. “Guardian: Symbolic validation of orderliness in SGX enclaves”, CCSW 2021.

TABLE 1. COMPARISON OF SYMBOLIC-EXECUTION TOOLS FOR SGX.

Tool	App	Runtime			Binary Dump	Reentry	Plugins				
		SDK	Entry	Init			Ptr	ABI	ÆPIC	Jmp	Open
TEEREX [37]	●	Intel	○	○	●	○	○	○	○	●	○
Guardian [38]	●	Intel	●	○	●	○	○	●	●	○	●
COIN [39]	●	Intel	○	○	○	○	●	○	○	○	●
Pandora	●	any	●	●	●	●	●	●	●	●	●

Features can be fully (●), partially (○), or not (○) supported. Columns 4–7 denote whether the tool executes the runtime *entry* and *initialization* phases; can handle *binaries* without additional specification; and uses the exact memory layout (*dump*).

# Pandora: Objectives for practical SGX binary validation

- **Engineering:** High-quality open-source (eventually!) **cmdline** tool (CI)



# Pandora: Objectives for practical SGX binary validation

- **Engineering:** High-quality open-source (eventually!) `cmdline` tool (CI)
- **Binary** level ↔ Possible leakage introduced by `compiler`



# Pandora: Objectives for practical SGX binary validation



- **Engineering:** High-quality open-source (eventually!) **cmdline** tool (CI)
- **Binary** level ↔ Possible leakage introduced by **compiler**
- **Extensible** symbolic validation infrastructure:
  1. Abstract **enclave runtime**: Intel SDK, OpenEnclave, Gramine, etc.
  2. Validation **plugins**: angr breakpoints → check ptrs, regs, etc.

# Pandora: Objectives for practical SGX binary validation



- **Engineering:** High-quality open-source (eventually!) **cmdline** tool (CI)
- **Binary** level ↔ Possible leakage introduced by **compiler**
- **Extensible** symbolic validation infrastructure:
  1. Abstract **enclave runtime**: Intel SDK, OpenEnclave, Gramine, etc.
  2. Validation **plugins**: angr breakpoints → check ptrs, regs, etc.
- “**Principled**” validation criteria: (Reasoning about) guarantees on the absence of vulnerabilities ↔ fuzzing approaches...

Usage: pandora.py [OPTIONS] BINARY\_PATH

Pandora: Principled vulnerability detection for SGX binaries.

Arguments

\* binary\_path FILE Path to the binary to open [default: None] [required]

Options

--config-file -c FILE	Path to optional config file [default: None]
--log-level -l [debug info warning error critical]	The log level for pandora [default: info]
--angr-log-level -L [debug info warning error critical]	The log level for angr [default: critical]
--action -a TEXT	Adds an action bound to a specific event via the format <b>event</b> = <b>action</b> . Possible values for the <u>event</u> key are: ↳ <b>error</b> -- Upon termination with error states. ↳ <b>explorer</b> -- For each symbolic execution step. ↳ <b>start</b> -- Once before explorer starts symbolic execution. ↳ <b>abi</b> -- For events reported by the 'abi' plugin (see below). ↳ <b>pointers</b> -- For events reported by the 'pointers' plugin (see below). Possible values for the <u>action</u> key are: ↳ <b>exit</b> -- Terminates the program. ↳ <b>shell</b> -- Spawns an interactive shell. ↳ <b>break</b> -- Stops and waits for user input before proceeding. ↳ <b>none</b> -- Do nothing (default). [default: None]
--help	Show this message and exit.

Exploration options

--num-steps -n INTEGER	Number of steps to execute in symbolic execution. 0 or negative allows to run to completion. [default: 100]
--num-threads -t INTEGER	Number of threads to use in parallel when stepping through the symbolic execution. 0 disables parallelization. [default: 0]

Validation plugins

--plugins -p [abi pointers]	Define the plugins to activate, separated by a comma. Possible values for the <u>plugin</u> key are: [default: all] ↳ <b>abi</b> -- Validates CPU register sanitizations. ↳ <b>pointers</b> -- Validates attacker-tainted pointer dereferences.
--report -r [html log]	Define the format for all plugin reports. [default: html]



## 2. Runtime-Agnostic Enclave Loading

---

# Challenge: Intel SGX Memory Layout



Angr is designed to load normal OS binaries

↔ No uniform **SGX enclave binary format!**

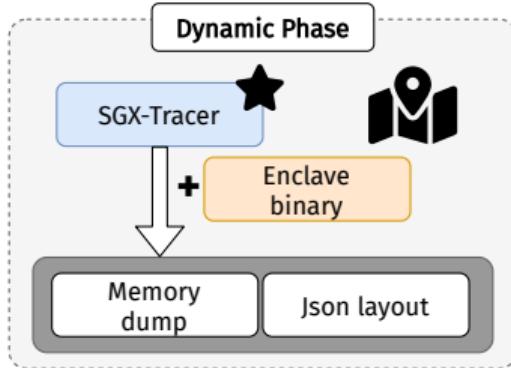
- Untrusted **runtime loader** parses ELF binary embedded metadata to create enclave image with **TCS, SSA, Stack, Heap**, etc.
- MRENCLAVE attestation independent of load address → partial **relocation** in enclave

↔ No syscalls; untrusted interaction through **enclu** (ecall/ocall/...)

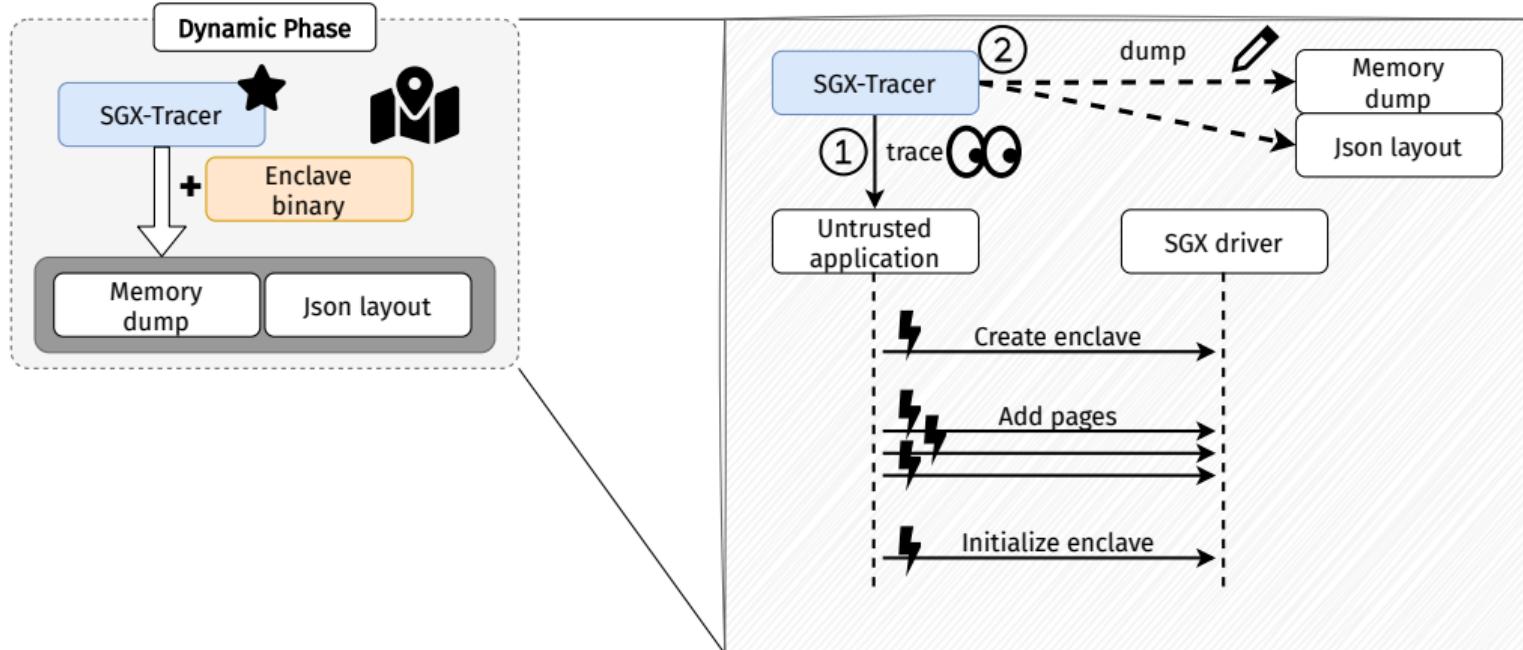
# Pandora: Runtime-Agnostic Enclave Loading



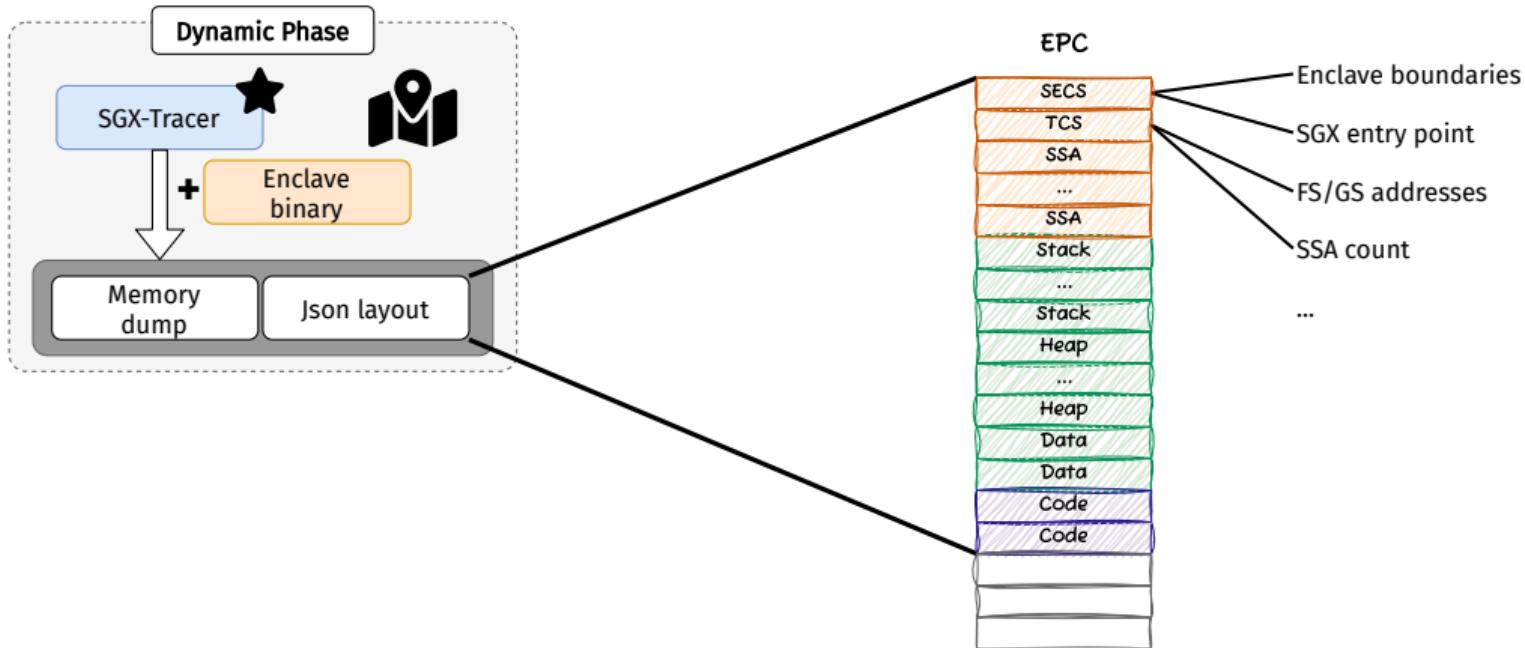
# Pandora: Runtime-Agnostic Enclave Loading



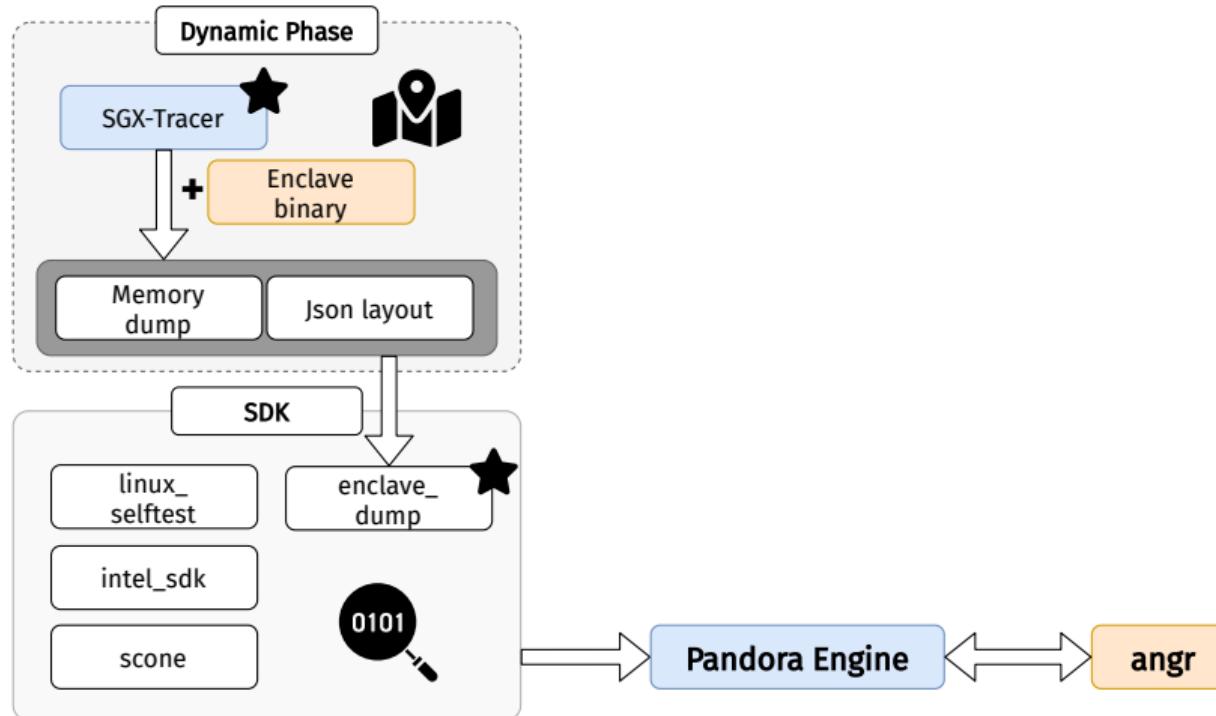
# Pandora: Runtime-Agnostic Enclave Loading



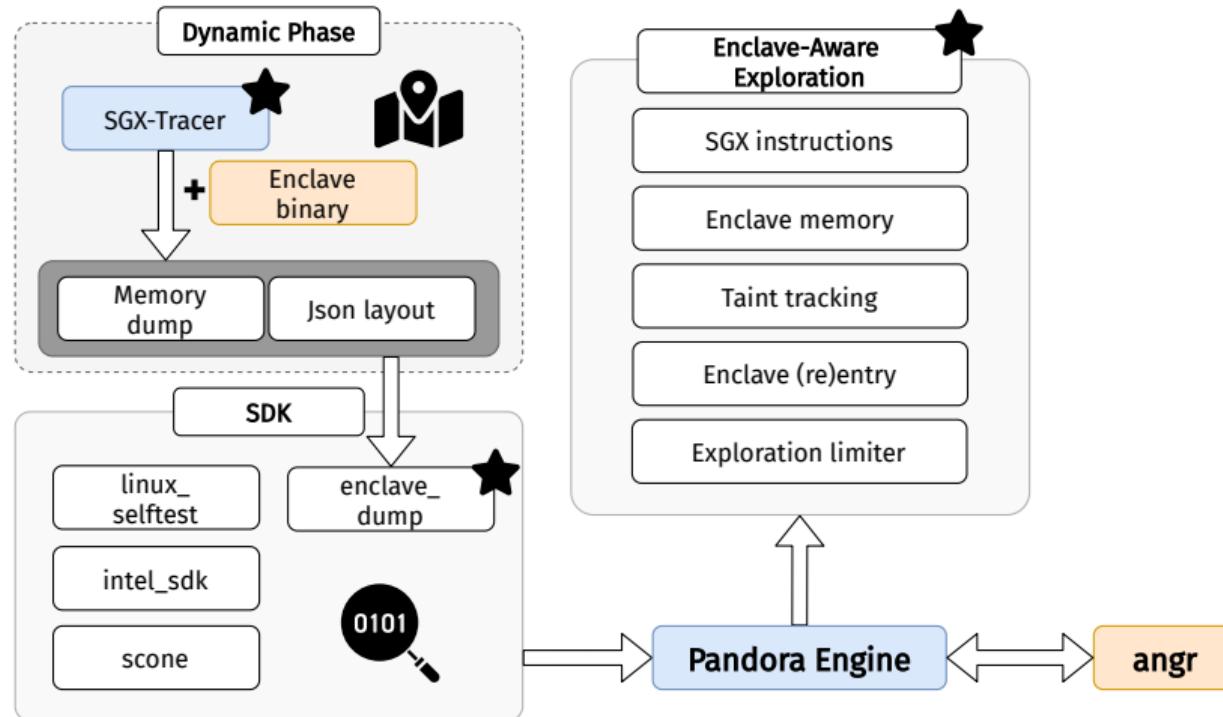
# Pandora: Runtime-Agnostic Enclave Loading



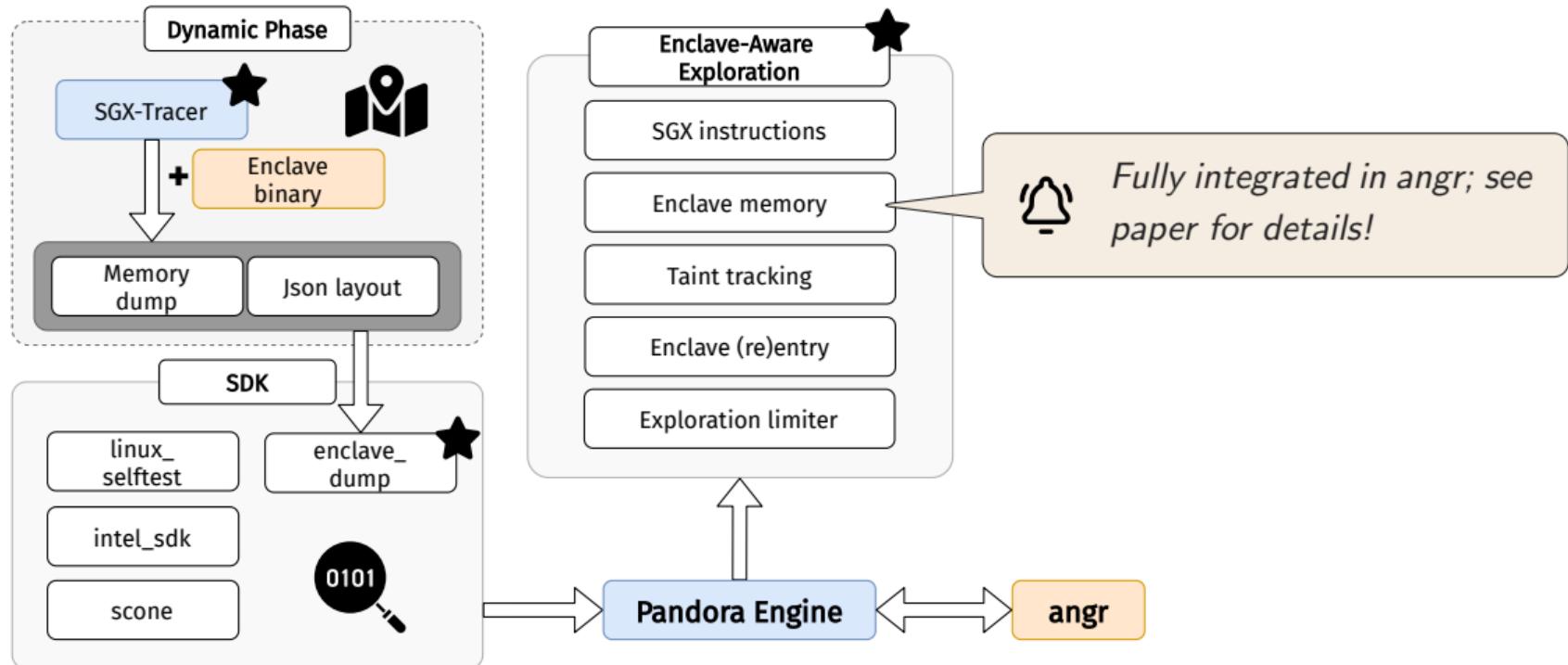
# Pandora: Runtime-Agnostic Enclave Loading



# Pandora: Enclave-Aware Symbolic Exploration



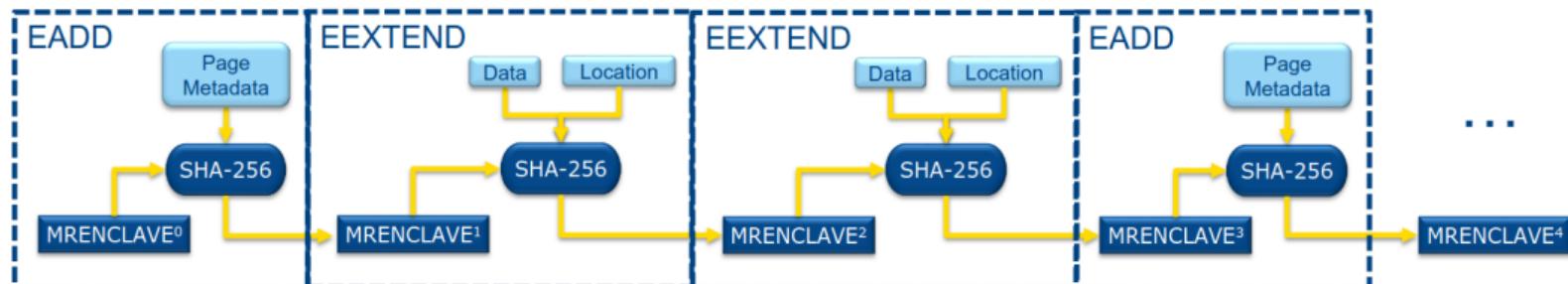
# Pandora: Enclave-Aware Symbolic Exploration



## Excuse: Towards a Unified SGX Binary Format?

Enclave identity (MRENCLAVE) is a 256-bit digest of the log that represents the enclave

- Provided during attestation to remote platform



Fortanix SGX stream (SGXS) format



sha256sum \*.sgxs == MRENCLAVE

- <https://github.com/fortanix/rust-sgx/tree/master/intel-sgx/sgxs-tools>.
- <https://rwc.iacr.org/2016/Slides/Sealing%20and%20Attestation%20in%20SGX.pdf>.
- Van Bulck et al. "A Case for Unified ABI Shielding in Intel SGX Runtimes", SysTEX 2022.

```
jo@gropius:~/Documents/t2w-continues/binaries$ sgxs-info summary gramine_v1.2.dump.sgxs
 0- ffff (unmapped)
 10000-fa7ffff Reg  rwx (empty) meas=none
fa7a000-fabcff Reg  r-x (data) meas=all
fabd000-fabeff Reg  rw- (data) meas=all
fabf000-fabffff Reg  rw- (empty) meas=all
fac0000-fac0fff Reg  rw- (data) meas=all
fac1000-ffd4fff Reg  rw- (empty) meas=all
ffd5000-ffd8fff Reg  rw- (data) meas=all
ffd9000-ffd9fff Tcs --- (data) meas=all [oentry=0xfa8a186, ossa=0xffdd000, nssa=2]
ffda000-ffdafff Tcs --- (data) meas=all [oentry=0xfa8a186, ossa=0xffe5000, nssa=2]
ffdb000-ffdbfff Tcs --- (data) meas=all [oentry=0xfa8a186, ossa=0xffffd000, nssa=2]
ffdc000-ffdcfff Tcs --- (data) meas=all [oentry=0xfa8a186, ossa=0xffff5000, nssa=2]
ffdd000-ffffcff Reg  rw- (empty) meas=all
ffffd000-ffffffff Reg  r-- (data) meas=all
```

```
jo@gropius:~/Documents/t2w-continues/binaries$ sgxs-info summary intel_2.1.101.42529.dump.sgxs
```

```
 0- 17fff Reg  r-x (data) meas=all
 18000-216fff (unmapped)
217000-219fff Reg  rw- (data) meas=all
21a000-318fff Reg  rw- (empty) meas=all
319000-319fff Reg  rw- (data) meas=all
31a000-31a0ff Reg  rw- (empty) meas=all
31b000-41a0ff Reg  rw- (empty) meas=none
41b000-42a0ff (unmapped)
42b000-46a0ff Reg  rw- [0xcc]* meas=all
46b000-47a0ff (unmapped)
47b000-47b0ff Tcs --- (data) meas=all [oentry=0x4b32, ossa=0x47c000, nssa=2]
47c000-47d0ff Reg  rw- (empty) meas=all
47e000-48d0ff (unmapped)
48e000-48efff Reg  rw- (empty) meas=all
48f000-7fffff (unmapped)
```

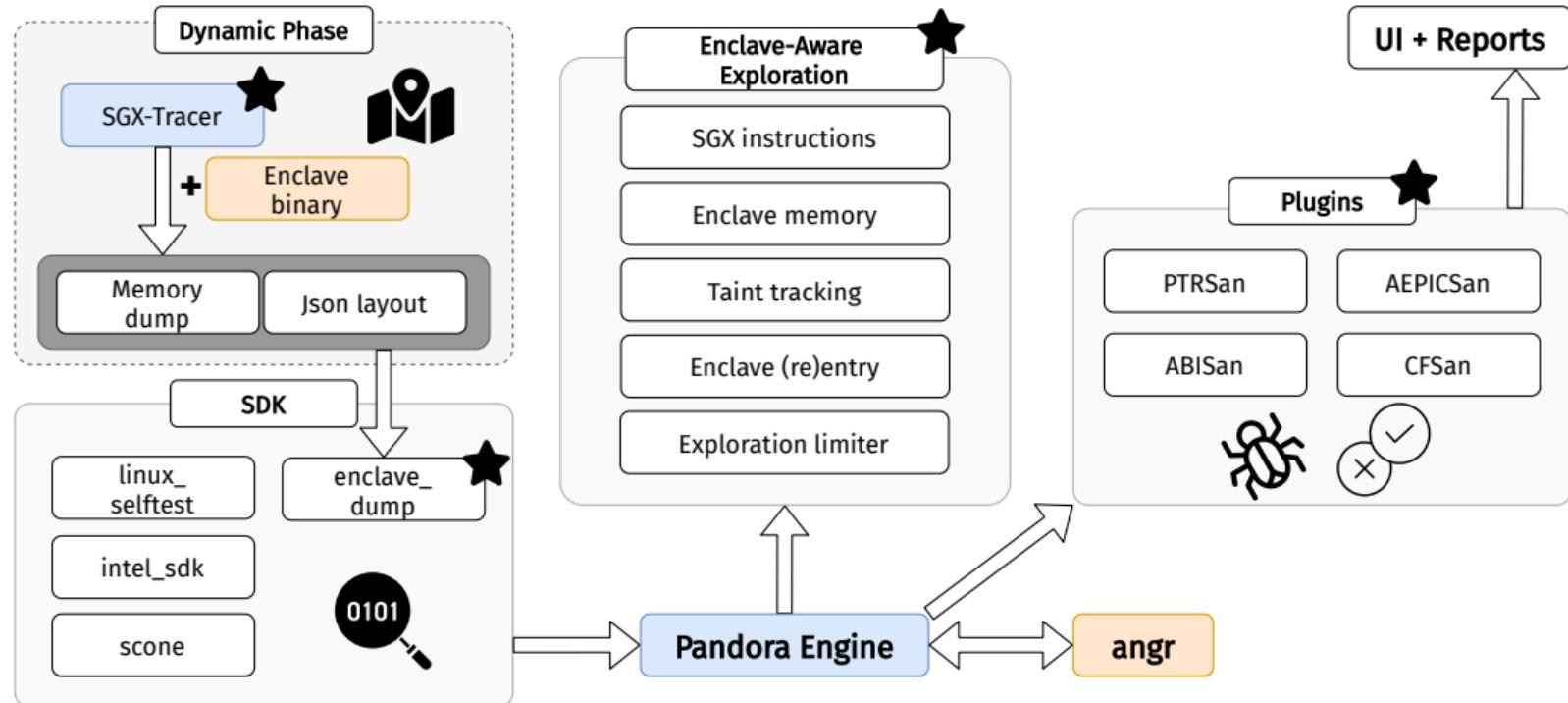
```
jo@gropius:~/Documents/t2w-continues/binaries$ 
```



### 3. Pluggable Vulnerability Detection

---

# Pandora: Plugin-Based Vulnerability Detection



# Pandora: Principled Symbolic Validation?



1. Extend angr with **enclave-aware breakpoints**
2. Validate **software invariants** during symbolic exploration!
3. Aggregate violations in human-readable rich **HTML reports**

# Pandora: Principled Symbolic Validation?

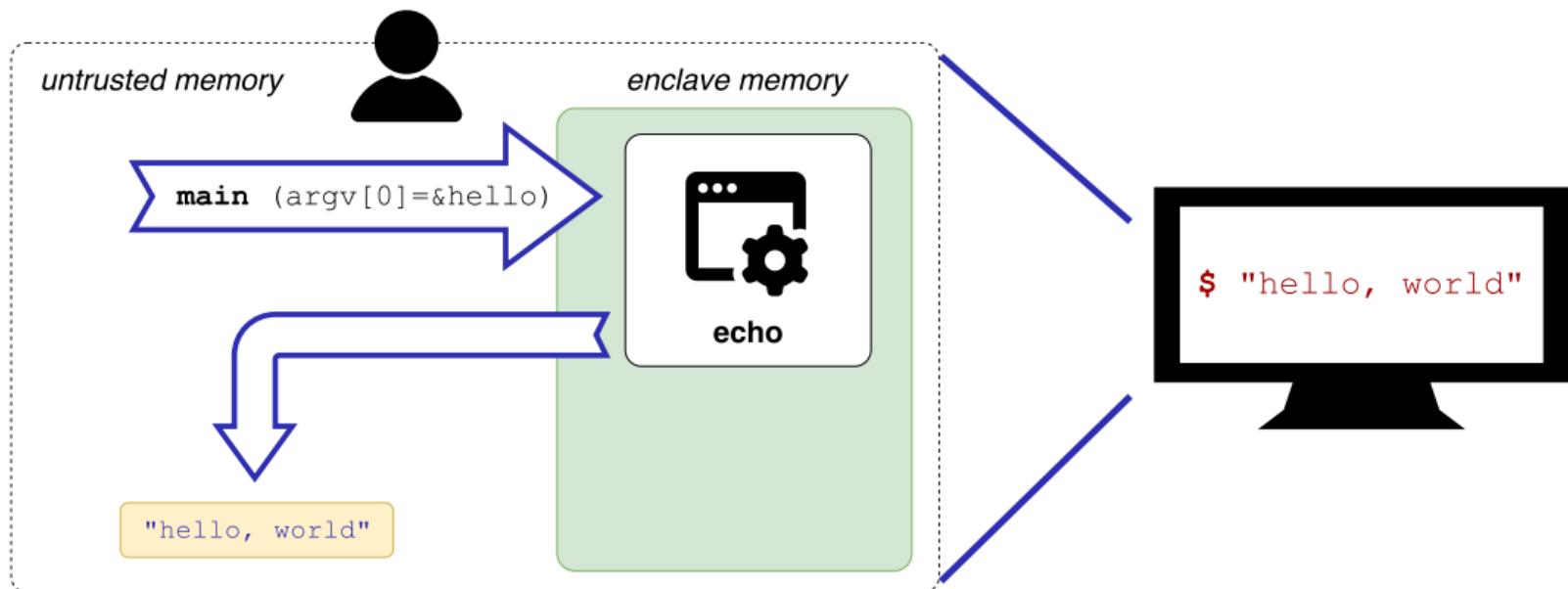


1. Extend angr with **enclave-aware breakpoints**
2. Validate **software invariants** during symbolic exploration!
3. Aggregate violations in human-readable rich **HTML reports**

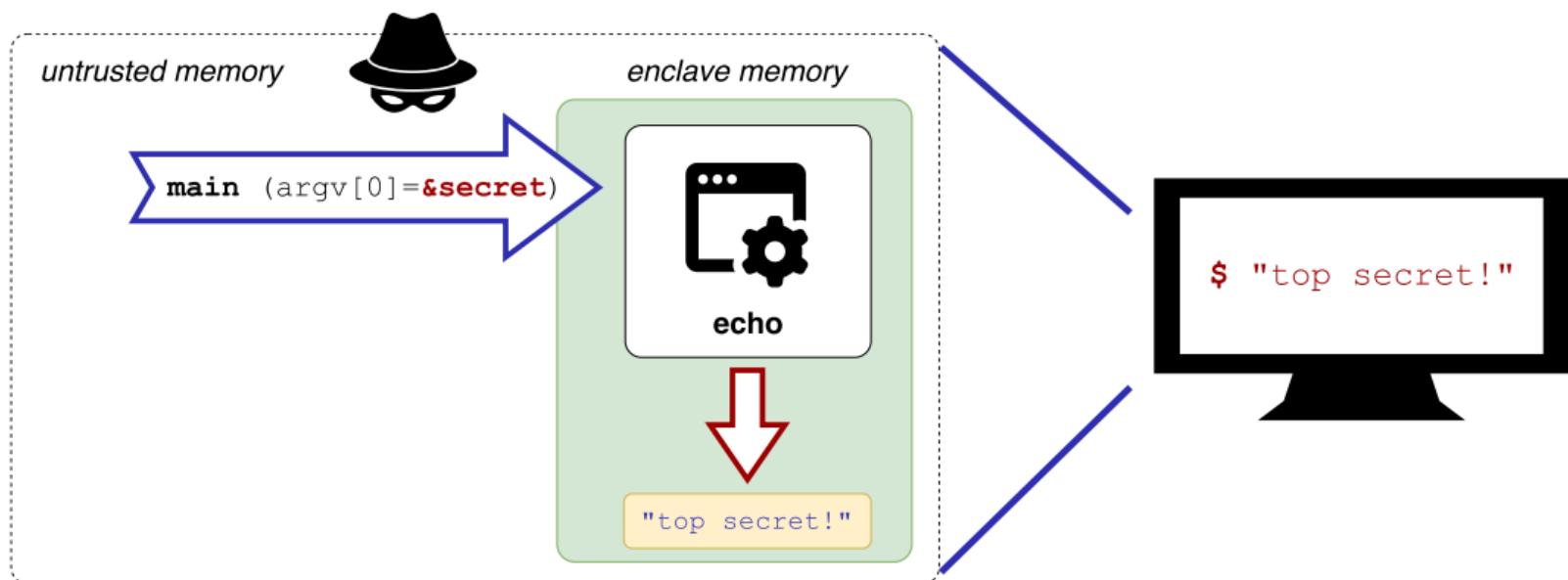
**Challenge:** Understanding attacks + specifying adequate invariants:

- **ABI:** No *attacker-tainted CPU control register* reads
- **API:** No *attacker-tainted addresses* (partially) inside the enclave
- **MMIO/ÆPIC:** All *attacker-tainted addresses* aligned or preceded by `verw`
- **Control flow:** No (arbitrary) *attacker-tainted jumps* in enclave memory

# API Vulnerabilities: Confused-Deputy Attacks



# API Vulnerabilities: Confused-Deputy Attacks



## Excuse: Secure Enclave Pointer Usage is Hard...

---

```
1 struct encl_args { uint64_t value; uint64_t addr; };
2
3 static void do_encl_op_get_from_addr(struct encl_args *op)
4 {
5     /* 1. Base pointer check */
6     if (!sgx_is_outside_enclave(op, sizeof(struct encl_args)))
7         return;
8     /* 2. Prevent time-of-check time-of-use */
9     volatile void* ptr = (void*) op->addr;
10    /* 3. Nested pointer check */
11    if (!sgx_is_outside_enclave((void*) ptr, 8))
12        return;
13    memcpy(&op->value, (void*) ptr, 8);
14 }
```

---

## Excuse: Secure Enclave Pointer Usage is Hard...

---

```
1 struct encl_args {uint64_t value; uint64_t addr;};
2
3 static void do_encl_op_get_from_addr(struct encl_args *op)
4 {
5     /* 1. Base pointer check */
6     if (!sgx_is_outside_enclave(op, sizeof(struct encl_args)))
7         return;
8     /* 2. Prevent time-of-check time-of-use */
9     volatile void* ptr = (void*) op->addr;
10    /* 3. Nested pointer check */
11    if (!sgx_is_outside_enclave((void*) ptr, 8))
12        return;
13    memcpy(&op->value, (void*) ptr, 8);
14 }
```

## Excuse: Secure Enclave Pointer Usage is Hard...

---

```
1 struct encl_args {uint64_t value; uint64_t addr;};
2
3 static void do_encl_op_get_from_addr(struct encl_args *op)
4 {
5     /* 1. Base pointer check */
6     if (!sgx_is_outside_enclave(op, sizeof(struct encl_args)))
7         return;
8     /* 2. Prevent time-of-check time-of-use */
9     volatile void* ptr = (void*) op->addr;
10    /* 3. Nested pointer check */
11    if (!sgx_is_outside_enclave((void*) ptr, 8))
12        return;
13    memcpy(&op->value, (void*) ptr, 8);
14 }
```

## Excuse: Secure Enclave Pointer Usage is Hard...

---

```
1 struct encl_args {uint64_t value; uint64_t addr;};
2
3 static void do_encl_op_get_from_addr(struct encl_args *op)
4 {
5     /* 1. Base pointer check */
6     if (!sgx_is_outside_enclave(op, sizeof(struct encl_args)))
7         return;
8     /* 2. Prevent time-of-check time-of-use */
9     volatile void* ptr = (void*) op->addr;
10    /* 3. Nested pointer check */
11    if (!sgx_is_outside_enclave((void*) ptr, 8))
12        return;
13    memcpy(&op->value, (void*) ptr, 8);
14 }
```

## Summary: API-level attack surface

Vulnerability \ Runtime	SGX-SDK	OpenEnclave	Graphene	SGX-LKL	Rust-EDP	Asylo	Keystone	Sancus
#4 Missing pointer range check	○	★	★	★	○	●	○	★
#5 Null-terminated string handling	★	★	○	○	○	○	○	○
Tier2 (API) #6 Integer overflow in range check	○	○	●	○	●	○	●	●
#7 Incorrect pointer range check	○	○	●	○	○	●	○	●
#8 Double fetch untrusted pointer	○	○	●	○	○	○	○	○
#9 Ocall return value not checked	○	★	★	★	○	●	★	○



**Critical oversights** in production and research code

→ across TEEs and programming languages (incl. safe langs like Rust)

## Intel software mitigation advisory: AEPICTM stale data injection

Although this MCU mitigates potential exposure of data after an LP exits an enclave, enclave data could also be exposed when an enclave reads data from outside its own linear memory range (ELRANGE). This may occur when a malicious OS/VMM maps the xAPIC into an enclave-accessible page outside of ELRANGE. If the enclave unintentionally accesses the xAPIC in an attempt to read memory, it may receive stale enclave data instead of the data that it had attempted to read. The enclave may then unintentionally perform an operation that could allow an attacker to infer this data.

Intel will provide an updated Intel SGX Software Development Kit (SDK) that helps mitigate potential exposure under this scenario. The updated SDK reads data from outside the enclave's ELRANGE at a size and alignment of 8 bytes. It also provides new programming interfaces that can be used by developers to ensure that enclave application code reads data from outside the enclave's ELRANGE at a minimum alignment of 8 bytes. Some enclave developers may choose to update their Intel® SGX software once the updated SDK is available.

# Intel software mitigation advisory: MMIO stale data injection

## Implications for Software Running in Intel SGX Enclave Mode

Because the Intel SGX security model does not trust the OS, a malicious OS could map MMIO memory into the untrusted memory space of an application that uses one or more Intel SGX enclaves. This could include the region of untrusted memory used for parameter passing to/from ECALLs and OCALLs, or any external buffers that an enclave might use to communicate with its application. If the malicious OS does such mapping, then when the enclave writes to this memory, it could propagate the stale data in its fill buffers into the uncore, where it could later be extracted by malicious software.

The mitigations below assume that Intel HT Technology is disabled to ensure that once the fill buffers are overwritten, a sibling thread cannot repopulate them. The mitigation depends on how the enclave is accessing the non-enclave memory regions.

- Enclaves that only write to memory outside the enclave in the context of ECALLs and OCALLs:

For enclaves using the Intel SGX SDK and Edger8r tool included with the SDK to create and manage the ECALL and OCALL interface, Intel has released an update to the SDK and Edger8r tool that will prevent fill buffer data exposure through the code generated by the Edger8r tool. Similarly, the Intel SGX SDK will include updates that will prevent fill buffer data exposure through the code used by enclaves that use the switchless mode supported by the Intel SDK.

- Enclaves that write to memory outside the enclave using code that isn't associated with ECALLs or OCALLs:

This includes enclaves that use the Intel SGX SDK but which specify the [user\_check] attribute in their enclave definition language (EDL)<sup>12</sup>. For these enclaves, all writes to untrusted memory must either be preceded by the VERW instruction and followed by the MFENCE; LFENCE instruction sequence or must be in multiples of 8 bytes, aligned to an 8-byte boundary.

Intel will work with the developers of other SGX SDKs and runtimes to help ensure that they have similar mitigations.

## Idea: Towards “principled” pointer validation?



**Invariant #1:** Any attacker-tainted pointer should always lie outside the enclave's protected ELRANGE.

- Symbolic **taint tracking**: Initial register state + untrusted memory reads
- 😊 **Constraint solver**: Auto validate sanitization (e.g., `sgx_is_outside_enclave`)
- 😢 **Too strong**: E.g., array indices `ecall_table[%rdx]` → false-positive heuristics

# Report PointerSanitizationPlugin

Plugin description: Validates attacker-tainted pointer dereferences.

Analyzed 'pandora\_selftest\_enclave\_sanitization3.elf', with 'Linux selftest enclave' enclave runtime. Ran for 0:00:12.758955 on 2023-08-03\_19-16-58.



Enclave info: Address range is [0x0, 0xbfff]



Summary: Found 1 unique WARNING issue; 2 unique CRITICAL issues.

## Report summary

Severity	Reported issues
WARNING	<ul style="list-style-type: none"><li><i>Attacker tainted read inside enclave at 0x2476</i></li></ul>
CRITICAL	<ul style="list-style-type: none"><li><i>Unconstrained read at 0x22c3</i></li><li><i>Unconstrained read at 0x20be</i></li></ul>

## Issues reported at 0x22c3

1

do\_encl\_op\_get\_from\_unmeasured

CRITICAL

Unconstrained read

### Unconstrained read

CRITICAL

RIP=0x22c3

#### Plugin extra info

Key	Value
Address	<BV64 0x3000 + ((attacker_mem_66_32{UNINITIALIZED} .. 0x1) << 0x3)>
Attacker tainted	True
Length	8
Pointer range	[0x3008, 0xffffffff800003008]
Pointer can wrap address space	False
Pointer can lie in enclave	True
Extra info	Read address may lie inside or outside enclave

#### Execution state info

Disassembly



CPU registers



#### Backtrace

Basic block trace (most recent first)



# SHELDON COOPER

PRESENTS

## FUN WITH FLAGS

REC



## 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;  
4
```



---

```
1  lea rdi, buf  
2  mov al, 0x0  
3  mov ecx, 100  
4  rep stos [rdi], al  
5
```

---

## x86 string instructions: Direction Flag (DF) operation



- [x86 rep string instructions](#) to speed up streamed memory operations
- Default operate **left-to-right**

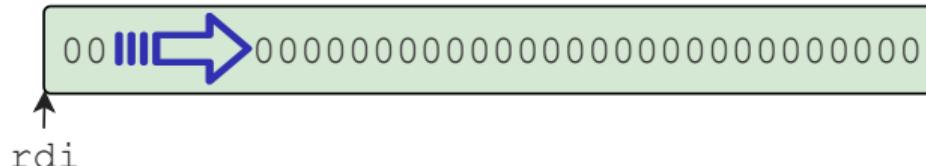
```
1  /* memset(buf, 0x0, 100) */  
2  for (int i=0; i < 100; i++)  
3      buf[i] = 0x0;  
4
```



---

```
1  lea rdi, buf  
2  mov al, 0x0  
3  mov ecx, 100  
4  rep stos [rdi], al  
5
```

---



## x86 string instructions: Direction Flag (DF) operation

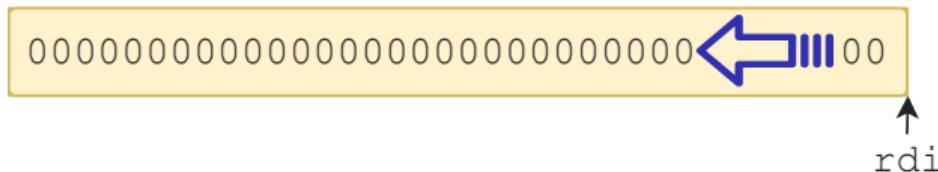


- **x86 rep string instructions** to speed up streamed memory operations
  - Default operate **left-to-right**, unless software sets *RFLAGS.DF=1*

```
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

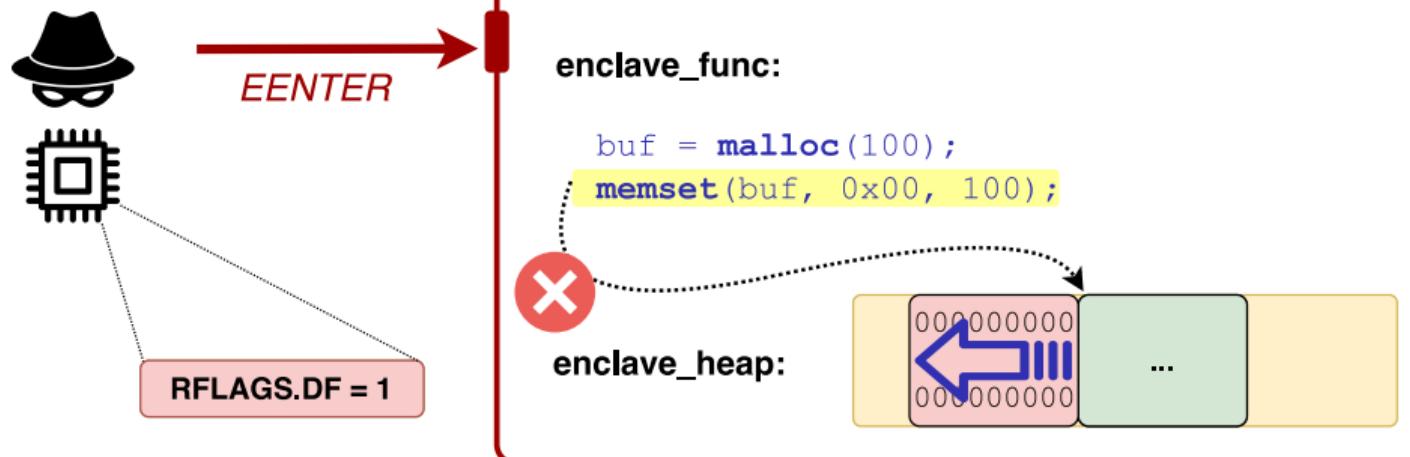


<sup>8</sup> The direction flag DF in the %rFLAGS register must be clear (set to “forward” direction) on function entry and return. Other user flags have no specified role in the standard calling sequence and are *not* preserved across calls.

# SGX-DF: Inverting enclaved string memory operations

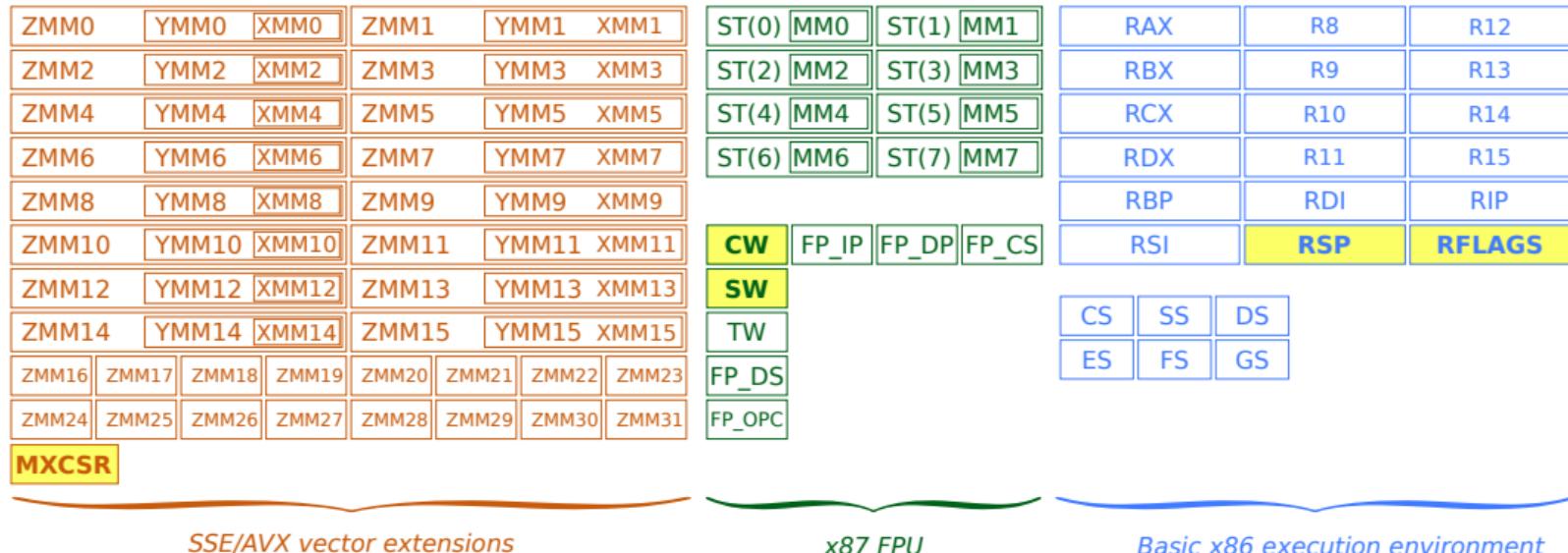


Enclave heap **memory corruption**: right-to-left...



□ Van Bulck et al. "A Tale of Two Worlds: Assessing the Vulnerability of Enclave Shielding Runtimes", CCS 2019.

# ABI Vulnerabilities: x86 Control Register Poisoning



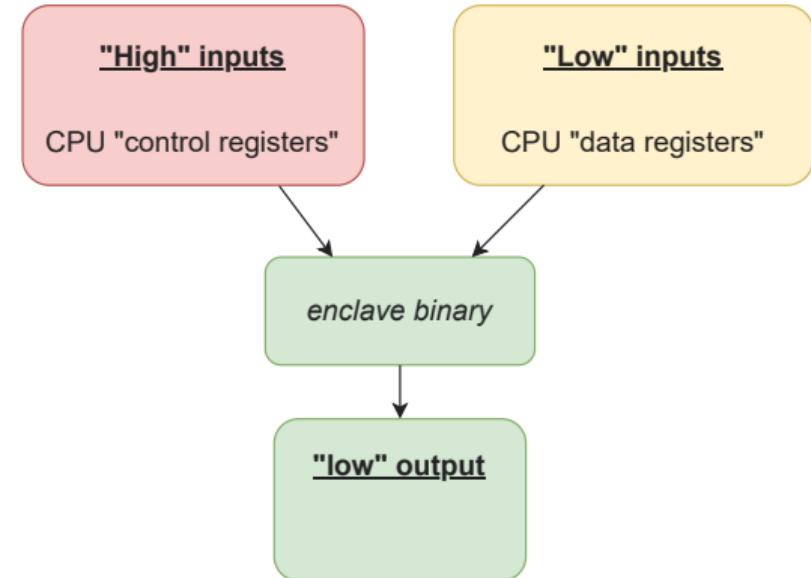
■ **x86 user-space CPU control registers**

# Idea: Towards “principled” ABI validation?



*“A computer has the noninterference property if and only if any sequence of low inputs will produce the same low outputs, regardless of what the high level inputs are.”*

— Wikipedia

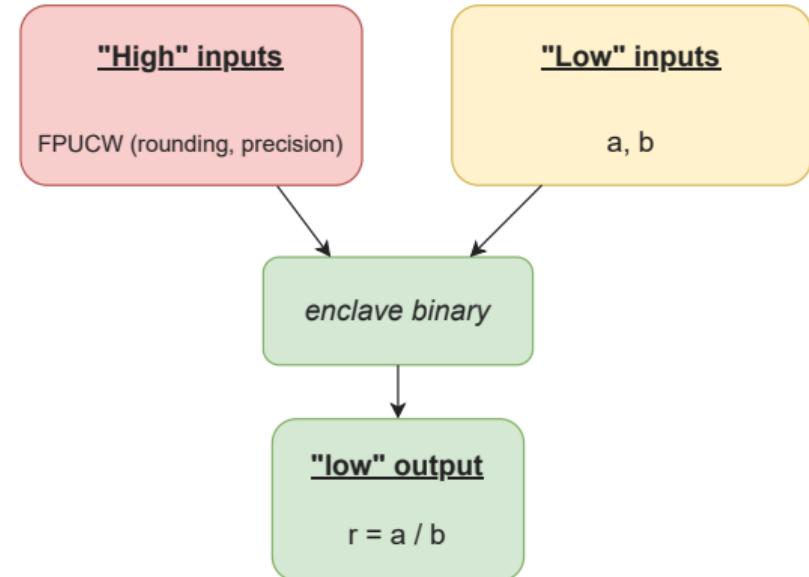


# Idea: Towards “principled” ABI validation?



*“A computer has the noninterference property if and only if any sequence of low inputs will produce the same low outputs, regardless of what the high level inputs are.”*

— Wikipedia



## Attacker-tainted read from FPTAG register

CRITICAL

RIP=0x2182

### Plugin extra info

Key	Value
reg_name	fptag
reg	<BV8 if 0x3c8 + (0x0 .. ftop_attacker_88_32 - 0x1) % 0x8[31:0] * 0x1 == 0x3cc then fptag_attacker_91_64[39:32] else (if 0x3c8 + (0x0 .. ftop_attacker_88_32 - 0x1) % 0x8[31:0] * 0x1 == 0x3cb then fptag_attacker_91_64[31:24] else (if 0x3c8 + (0x0 .. ftop_attacker_88_32 - 0x1) % 0x8[31:0] * 0x1 == 0x3cd then fptag_attacker_91_64[47:40] else (if 0x3c8 + (0x0 .. ftop_attacker_88_32 - 0x1) % 0x8[31:0] * 0x1 == 0x3ca then fptag_attacker_91_64[23:16] else (if 0x3c8 + (0x0 .. ftop_attacker_88_32 - 0x1) % 0x8[31:0] * 0x1 == 0x3ce then fptag_attacker_91_64[55:48] else (if 0x3c8 + (0x0 .. ftop_attacker_88_32 - 0x1) % 0x8[31:0] * 0x1 == 0x3c9 then fptag_attacker_91_64[15:8] else (if 0x3c8 + (0x0 .. ftop_attacker_88_32 - 0x1) % 0x8[31:0] * 0x1 == 0x3cf then fptag_attacker_91_64[63:56] else (if 0x3c8 + (0x0 .. ftop_attacker_88_32 - 0x1) % 0x8[31:0] * 0x1 == 0x3c8 then fptag_attacker_91_64[7:0] else 190))))))>

### Execution state info

#### Disassembly



```
2182 fld    dword ptr [val]
2188 fld    dword ptr [val]
218e fdivrp st(0x1)
2190 lea    rax, [rbx+encl_stack]
2197 xchg   rax, rsp
2199 push   rax
219a push   rcx
219b push   rbx
219c cld
219d call   encl_body
```

#### CPU registers



## Issues reported at 0x2070

13

encl\_body

CRITICAL

Attacker-tainted read from D register

### Attacker-tainted read from D register

CRITICAL

RIP=0x2070

#### Plugin extra info

Key	Value
reg_name	d
reg	<BV64 d_attacker_38_64>

#### Execution state info

##### Disassembly

```
2070 rep movsd dword ptr [rdi], dword ptr [rsi]
```

##### CPU registers

*	<b>rax</b>	: <BV64 rsp_attacker_8_64>
	<b>rcx</b>	: <BV64 0x9>
*	<b>rdx</b>	: <BV64 rdx_attacker_5_64>
	<b>rbx</b>	: <BV64 0x0>
	<b>rsp</b>	: <BV64 0x0fe0>
*	<b>rbp</b>	: <BV64 rbp_attacker_10_64>
	<b>rsi</b>	: <BV64 0x3000>
	<b>rdi</b>	: <BV64 0x8fb0>
*	<b>r8</b>	: <BV64 rdi_attacker_14_64>
*	<b>r9</b>	: <BV64 r9_attacker_18_64>
*	<b>r10</b>	: <BV64 r10_attacker_20_64>
*	<b>r11</b>	: <BV64 r11_attacker_22_64>
*	<b>r12</b>	: <BV64 r12_attacker_24_64>
*	<b>r13</b>	: <BV64 r13_attacker_26_64>
*	<b>r14</b>	: <BV64 r14_attacker_28_64>
*	<b>r15</b>	: <BV64 r15_attacker_30_64>
*	<b>d</b>	: <BV64 d_attacker_38_64>
	<b>rip</b>	: <BV64 0x2070>

## Pandora: ABISanitization plugin



*In principle* could detect unknown interferences from *non-data* registers

### ABISan plugin:

- ±100 LoC in Python, using angr `reg.read` breakpoint
- Requires annotation of x86 `data` registers

# Pandora: ABISanitization plugin



*In principle* could detect unknown interferences from *non-data* registers

## ABISan plugin:

- ±100 LoC in Python, using angr `reg_read` breakpoint
- Requires annotation of x86 `data` registers

## Limitations:

- angr only implements subset of `x86 semantics` (e.g., FPU rounding)
- No angr support for `exceptions` (e.g., SIGBUS/SIGFPE/...)

# Experimental Results: > 200 New Vulnerable Code Locations

Runtime	Version	Prod	Src	Plugin	Instances
EnclaveOS	3.28	✓	✗ <sup>†</sup>	ABISan	1
EnclaveOS	3.28	✓	✗ <sup>†</sup>	PTRSan	15
EnclaveOS	3.28	✓	✗ <sup>†</sup>	ÆPICSan	33
EnclaveOS	3.28	✓	✗ <sup>†</sup>	CFSan	2
GoTEE	b35f	✗	✓	PTRSan	31
GoTEE	b35f	✗	✓	ÆPICSan	18
GoTEE	b35f	✗	✓	CFSan	1
Gramine	1.4	✓	✓	ABISan	1
Intel SDK	2.15.1	✓	✓	PTRSan	2
Intel SDK	2.19	✓	✓	ÆPICSan	22
↪ Occlum	0.29.4	✓	✓	ÆPICSan	11
Open Enclave	0.19.0	✓	✓	ABISan	2
Rust EDP	1.71	✓	✓	ABISan	1

Runtime	Version	Prod	Src	Plugin	Instances
Linux selftest	5.18	✗	✓	ABISan	1
↪ DCAP	1.16	✓	✓	ABISan	1
↪ Inclavare	0.6.2	✗	✓	ABISan	1
Linux selftest	5.18	✗	✓	PTRSan	5
↪ DCAP	1.16	✓	✓	PTRSan	17
↪ Inclavare	0.6.2	✗	✓	PTRSan	2
Linux selftest	5.18	✗	✓	CFSan	1
↪ Inclavare	0.6.2	✗	✓	CFSan	1
SCONE	5.7 / 5.8	✓	✗	ABISan	2 / 1
SCONE	5.7 / 5.8	✓	✗	PTRSan	10 / 3
SCONE	5.7 / 5.8	✓	✗	ÆPICSan	11 / 3
SCONE	5.8	✓	✗	CFSan	1

# Conclusions and Outlook



[github.com/  
pandora-tee](https://github.com/pandora-tee)



**Truthful:** Runtime-agnostic enclave memory model  
→ *Exact attested memory layout (MRENCLAVE)*



**Extensible:** Validate vulnerability invariants via **plugins**  
→ *ABISan, PTRSan, ÆPICSan, CFSan*



**Evaluation:** > 200 instances; 7 CVEs; **11 SGX runtimes**  
→ *Including low-level initialization & relocation logic!*

# Conclusions and Outlook



[github.com/  
pandora-tee](https://github.com/pandora-tee)



**Truthful:** Runtime-agnostic enclave memory model  
→ *Exact attested memory layout (MRENCLAVE)*



**Extensible:** Validate vulnerability invariants via **plugins**  
→ *ABISan, PTRSan, ÆPICSan, CFSan*



**Evaluation:** > 200 instances; 7 CVEs; **11 SGX runtimes**  
→ *Including low-level initialization & relocation logic!*



*Thank you! Questions?*