Certified Confidential Computing: Principled Symbolic Validation for Enclave Shielding Runtimes

Jo Van Bulck

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

Confidential Computing Consortium Technical Advisory Council – August 22, 2024 (online)

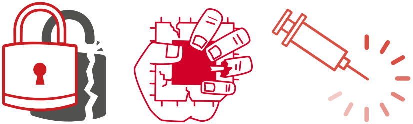
DistriNet, KU Leuven, Belgium   jo.vanbulck@cs.kuleuven.be   @jovanbulck   vanbulck.net

About DistriNet Enclave Research

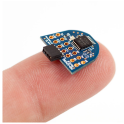
https://distrinet.cs.kuleuven.be/

- Trust across the **system stack:** App > compiler > OS > CPU > μ-arch
- Integrated **attack-defense** perspective and **open-source** prototypes

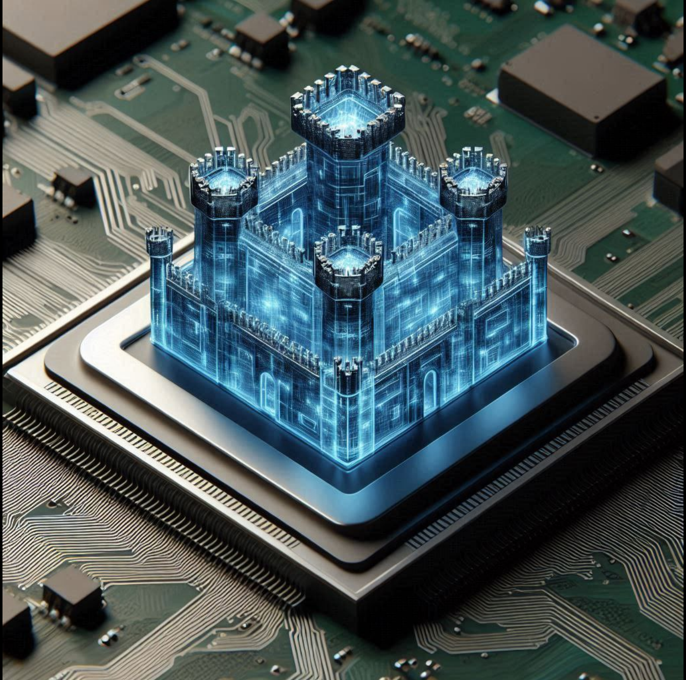
Side-channel analysis
(SGX-Step, AEX-Notify)



Transient-execution attacks
(Intel x86 SGX)

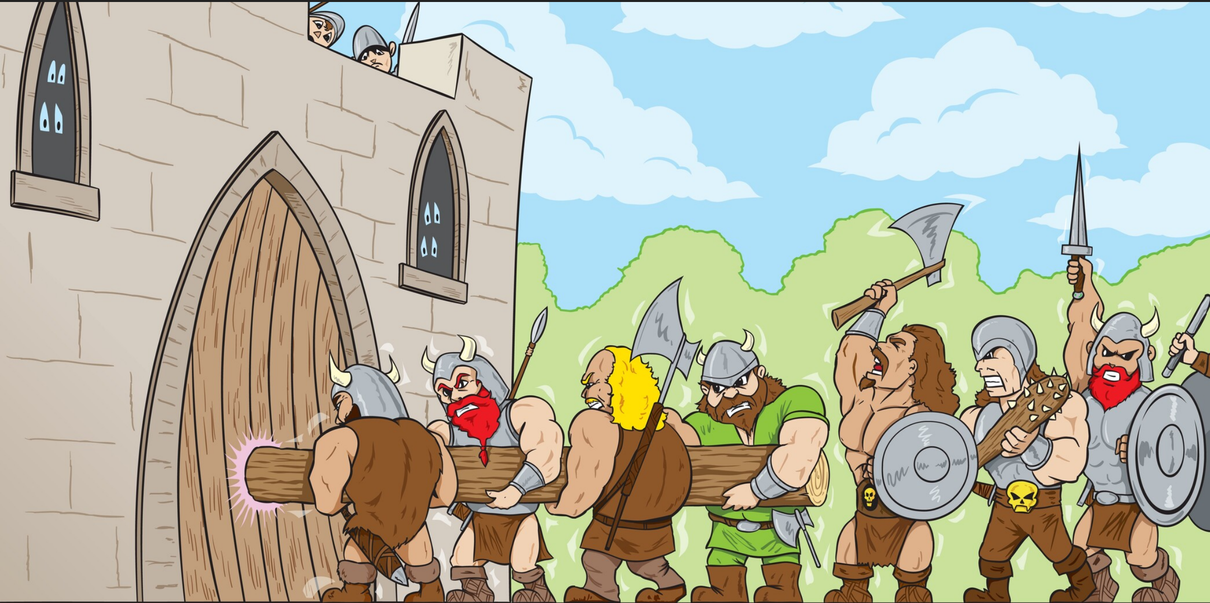


Embedded trust
(TI MSP430)



DALL·E 3

Besieging the SGX Fortress: Software Interface Attacks



Improper sanitization of MXCSR and RFLAGS — Moderate
GHSA-5gfr-m6mx-p5w4 published on Jul 17, 2023 by radhikaj

Intel Processor Stale Data Read from Legacy xAPIC — Moderate
GHSA-v3vm-9h66-wm76 published on Aug 13, 2022 by radhikaj

Intel Processor MMIO Stale Data Vulnerabilities — Moderate
GHSA-wm9w-8857-8fgj published on Jun 14, 2022 by radhikaj

Open Enclave SDK Elevation of Privilege Vulnerability — Moderate
GHSA-mj87-466f-jq42 published on Jul 13, 2021 by radhikaj

Socket syscalls can leak enclave memory contents — Moderate
GHSA-525h-wxcc-f66m published on Oct 12, 2020 by radhikaj

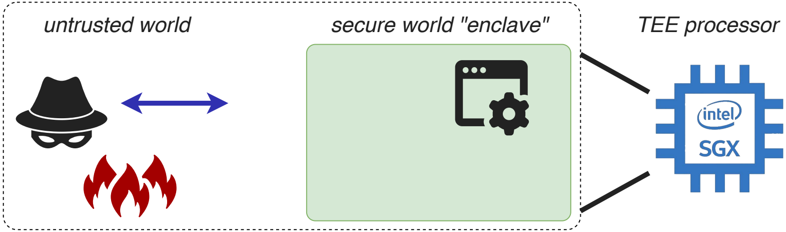
x87 FPU operations in enclaves are vulnerable to ABI poisoning — Low
GHSA-7wjx-wcwg-w999 published on Jul 14, 2020 by CodeMonkeyLeet

Intel SGX Load Value Injection (LVI) vulnerability — Moderate
GHSA-8934-g2pr-x6cg published on Mar 12, 2020 by radhikaj

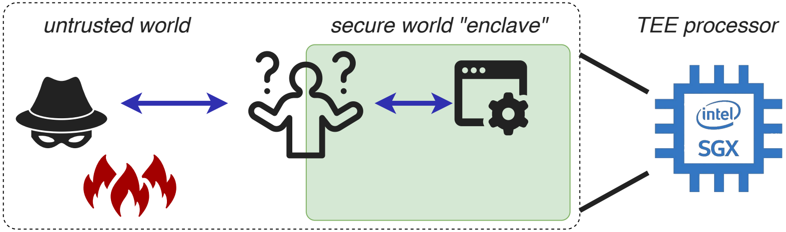
Enclave heap memory disclosure vulnerability — Moderate
GHSA-mg2p-657r-46cj published on Oct 8, 2019 by CodeMonkeyLeet

https://github.com/openenclave/openenclave/security

Context: Writing "Secure" Enclave Software is Hard. . .

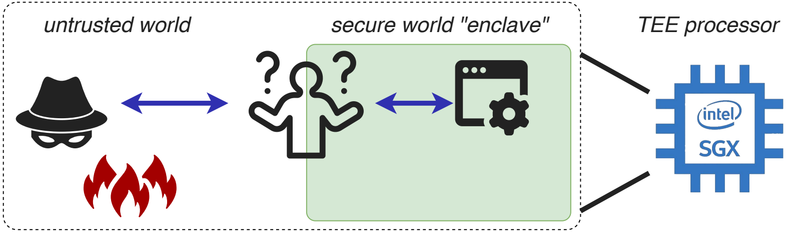


Context: Writing “Secure” Enclave Software is Hard...



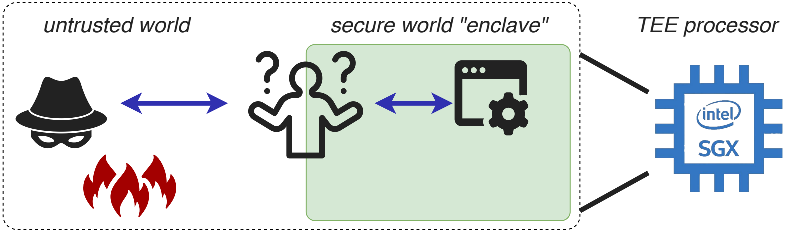
- API level: Sanitize pointer arguments in shared address space

Context: Writing "Secure" Enclave Software is Hard...



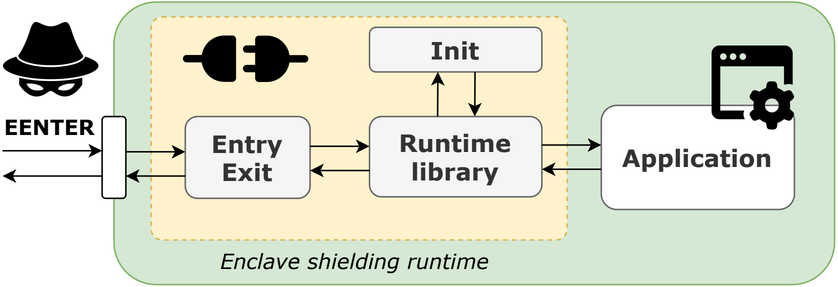
- API level: Sanitize pointer arguments in shared address space
- ABI level: Sanitize low-level CPU configuration registers

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

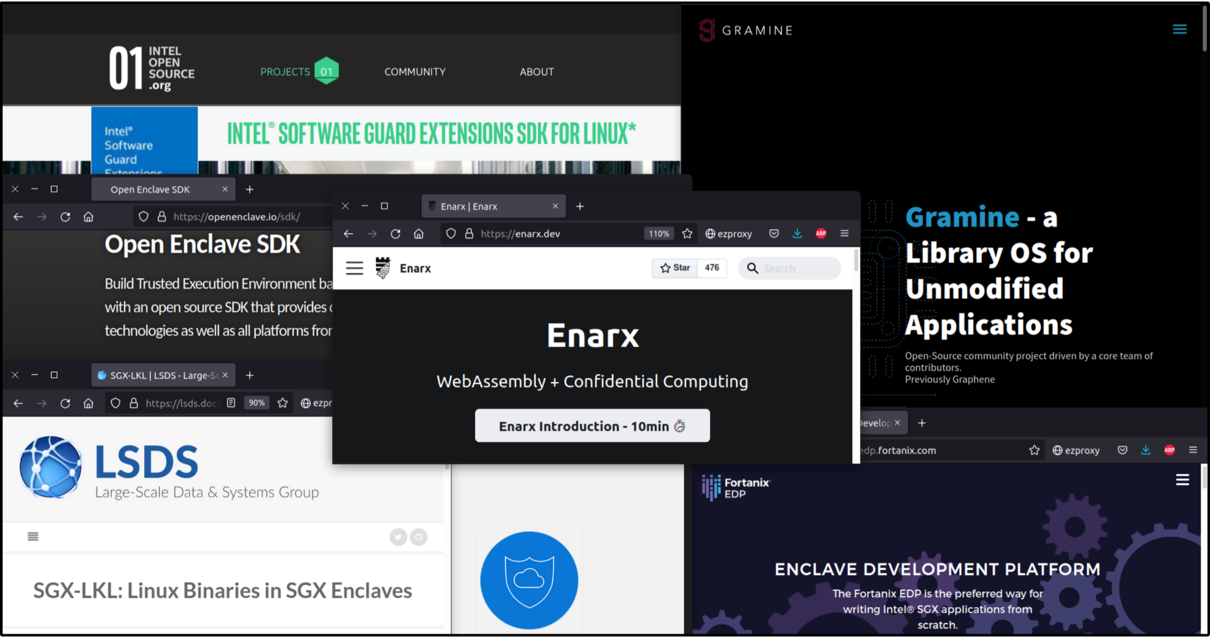


- API level: Sanitize pointer arguments in shared address space
- ABI level: Sanitize low-level CPU configuration registers
- μ-arch level: Spectre/LVI → `lfence`; ÆPIC/MMIO stale data → `verw`; cacheline GPU leak → avoid `dword0/1`...

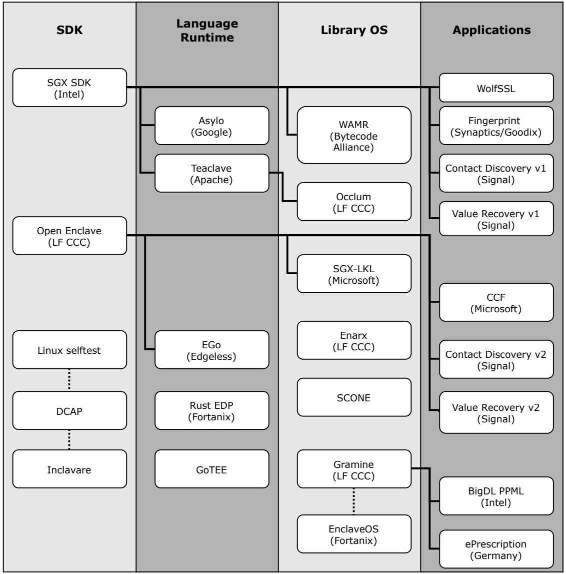
Solution: Enclave Shielding Runtimes



Key idea: Transparent input sanitization on enclave entry/exit

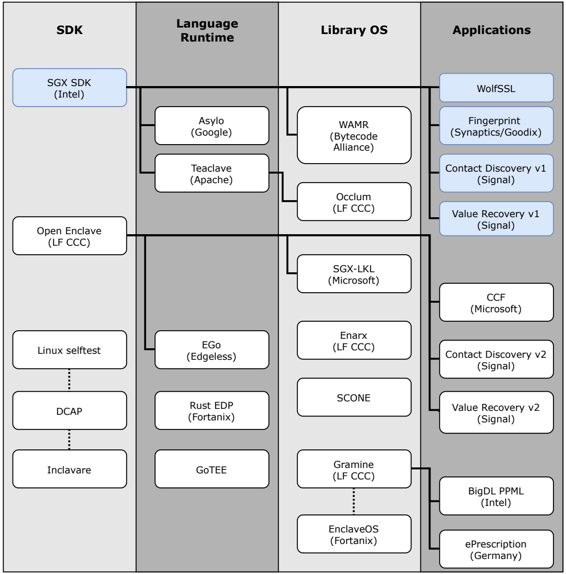

01 INTEL OPEN SOURCE .org
PROJECTS
COMMUNITY
ABOUT
Intel® Software Guard Extensions
INTEL® SOFTWARE GUARD EXTENSIONS SDK FOR LINUX*
Open Enclave SDK
https://openenclave.io/sdk/
Open Enclave SDK
Build Trusted Execution Environment ba
with an open source SDK that provides
technologies as well as all platforms fror
Enarx | Enarx
https://enarx.dev
Enarx
Star
476
Search
Enarx
WebAssembly + Confidential Computing
Enarx Introduction - 10min
GRAMINE
Gramine - a Library OS for Unmodified Applications
Open-Source community project driven by a core team of contributors.
Previously Graphene
SGX-LKL | LSDS - Large-S
https://lsds.doc
LSDS
Large-Scale Data & Systems Group
SGX-LKL: Linux Binaries in SGX Enclaves
edp.fortanix.com
Fortanix EDP
ENCLAVE DEVELOPMENT PLATFORM
The Fortanix EDP is the preferred way for writing Intel® SGX applications from scratch.

Challenge: Diverse Intel SGX Software Ecosystem



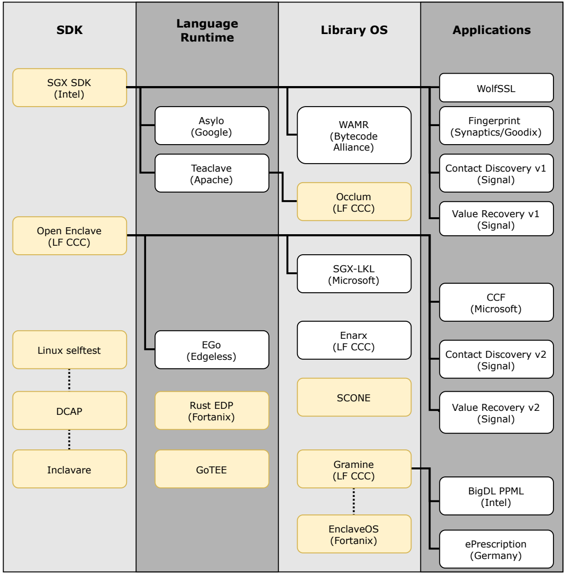
- **Ecosystem:** Diverse programming paradigms & abstractions

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



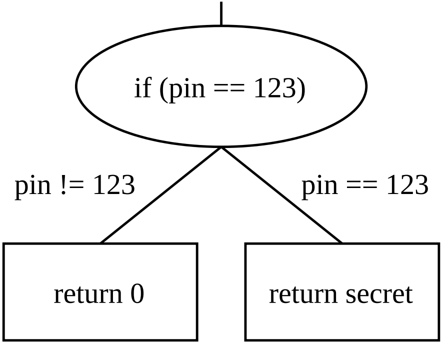
- **Ecosystem:** Diverse programming paradigms & abstractions
- **Prior work:** Selected applications on Intel SDK (e.g., `NULL` pointers)
- **Pandora:** Runtime-agnostic & truthful symbolic execution
  1. Exact attested memory binary
  2. Vulnerability detection plugins

1. Truthful Symbolic Execution

Background: Symbolic Execution and angr

```
int ecall(int pin){
    if(pin == 123){
        return secret;
    } else {
        return 0;
    }
}
```

https://angr.io/



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

TABLE 1. Comparison of symbolic-execution tools for SGX.

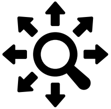
		Runtime						Plugins				
Tool	App	SDK	Entry	Init	Binary	Dump	Reentry	Ptr	ABI	ÆPIC	Jmp	Open
TeeRex [37]	●	Intel	○	○	●	○	◐	◐	○	○	◐	○
Guardian [38]	●	Intel	●	○	◐	○	○	◐	◐	○	◐	●
COIN [39]	●	Intel	○	○	○	○	○	◐	○	○	○	●
Pandora	●	*any*	●	●	●	●	●	●	●	●	●	●

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

Pandora: Objectives for practical SGX binary validation

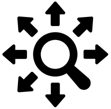
- **Engineering:** High-quality open-source cmdline tool (CI)
  `https://github.com/pandora-tee`

Pandora: Objectives for practical SGX binary validation



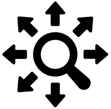
- **Engineering:** High-quality open-source cmdline tool (CI)
  `https://github.com/pandora-tee`
- **Binary** level ↔ Possible leakage introduced by compiler

Pandora: Objectives for practical SGX binary validation



- **Engineering:** High-quality open-source cmdline tool (CI)
  `https://github.com/pandora-tee`
- **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 cmdline tool (CI)
  `https://github.com/pandora-tee`

- **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 event=action. Possible values for the event
│                                                               key are:
│                                                               ↪ error    -- Upon termination with error states.
│                                                               ↪ explorer -- For each symbolic execution step.
│                                                               ↪ start    -- Once before explorer starts symbolic execution.
│                                                               ↪ abi      -- For events reported by the 'abi' plugin (see below).
│                                                               ↪ pointers -- For events reported by the 'pointers' plugin (see below).
│                                                               Possible values for the action key are:
│                                                               ↪ exit  -- Terminates the program.
│                                                               ↪ shell -- Spawns an interactive shell.
│                                                               ↪ break -- Stops and waits for user input before proceeding.
│                                                               ↪ none  -- 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 plugin key are: [default: all]
│                                    ↪ abi      -- Validates CPU register sanitizations.
│                                    ↪ pointers -- Validates attacker-tainted pointer dereferences.
│ --report   -r      [html|log]      Define the format for all plugin reports. [default: html]
╰──────────────────────────────────────────────────────────────────────────────────────────────────────╯
```

```
Pandora completed after taking 18 steps.
Pandora completed gracefully, no errored states created.
Final stash sizes after step 18: active (0), stashed (0), pruned (0), unsat (0), deadended (0), unconstrained (0), eexited (6), uniques (1) errored (0)
```

SystemEvents summary: SystemEvents reported 2 unique INFO issues.

Severity	Reports by SystemEvents
INFO	'Runtime statistics of hit symbols by count' at 0x2086; 'Runtime statistics of hit symbols by time of occurrence' at 0x2086

ABISanitizationPlugin summary: ABISanitizationPlugin reported 1 unique INFO issue; 2 unique CRITICAL issues.

Severity	Reports by ABISanitizationPlugin
INFO	'API entry point' at 0x2092
CRITICAL	'Attacker-tainted read from D register' at 0x206b; '36 attacker-tainted entry registers; MCDT' at 0x2092

AepicPlugin summary: AepicPlugin reported 5 unique CRITICAL issues.

Severity	Reports by AepicPlugin
CRITICAL	'SBDR read from untrusted memory with length 8' at 0x201e; 'SBDR read from untrusted memory with length 8' at 0x2030; 'SBDR read from untrusted memory with length 8' at 0x2072; 'DRPW write to untrusted memory with length 1' at 0x200d

ControlFlowSanitizationPlugin summary: ControlFlowSanitizationPlugin reported 1 unique WARNING issue.

Severity	Reports by ControlFlowSanitizationPlugin
WARNING	'Symbolic jmp tainted target in enclave memory' at 0x2083

PointerSanitizationPlugin summary: PointerSanitizationPlugin reported 5 unique CRITICAL issues; 1 unique WARNING issue.

Severity	Reports by PointerSanitizationPlugin
CRITICAL	'Unconstrained write' at 0x200d; 'Unconstrained read' at 0x2030; 'Unconstrained read' at 0x200a; 'Unconstrained read' at 0x2072; 'Unconstrained read' at 0x201e
WARNING	'Attacker tainted read inside enclave' at 0x207b

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, 0xfffffff800003008]
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

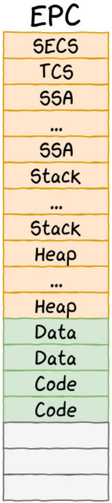
CPU registers

Backtrace

Basic block trace (most recent first)

2. Runtime-Agnostic Enclave Loading

Challenge: Intel SGX Memory Layout



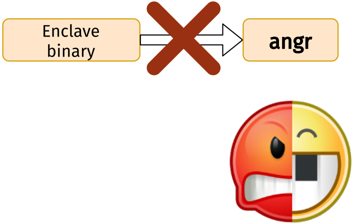
Angr is designed to load normal OS binaries

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

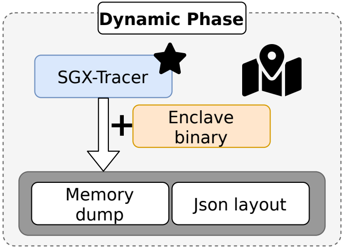
- Untrusted runtime loader parses ELF binary embedded metadata to create enclave image with TCS, SSA, Stack, Heap, etc.
- `MRENCLAVE` attestation independent of load address → partial relocation in enclave

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

Pandora: Runtime-Agnostic Enclave Loading

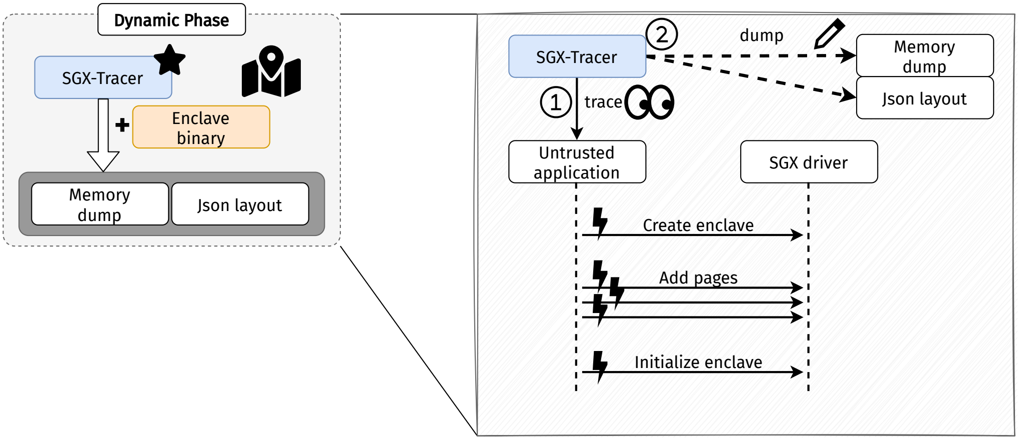


Pandora: Runtime-Agnostic Enclave Loading

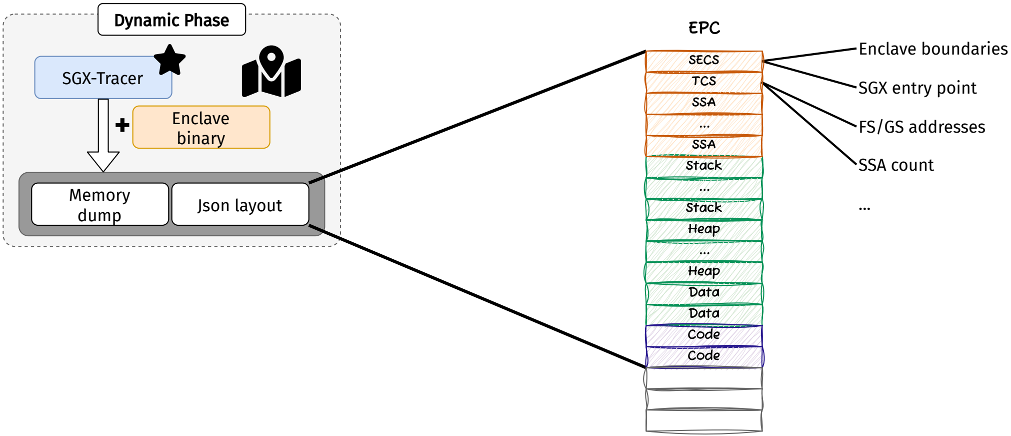


`ptrace`

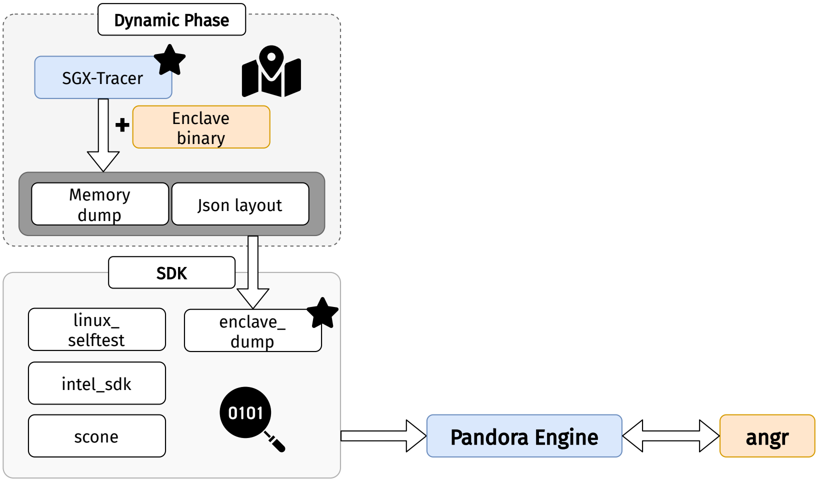
Pandora: Runtime-Agnostic Enclave Loading



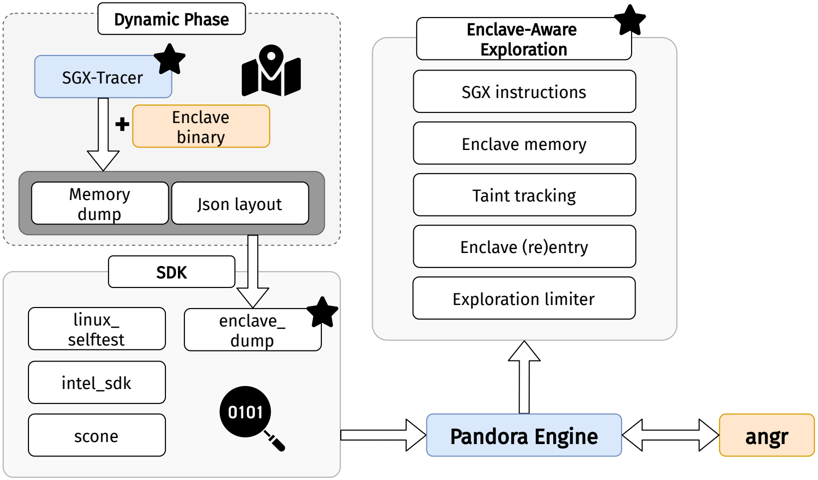
Pandora: Runtime-Agnostic Enclave Loading



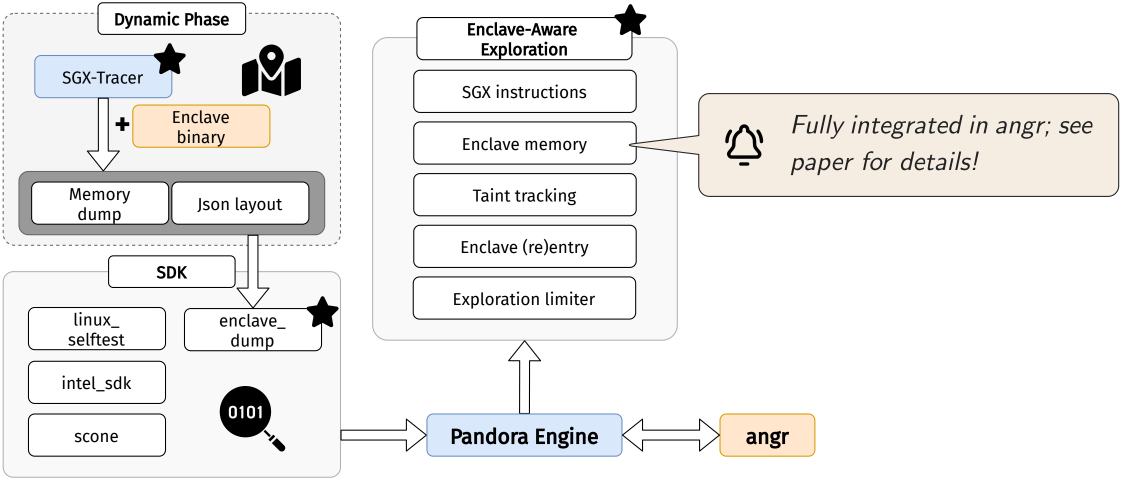
Pandora: Runtime-Agnostic Enclave Loading



Pandora: Enclave-Aware Symbolic Exploration



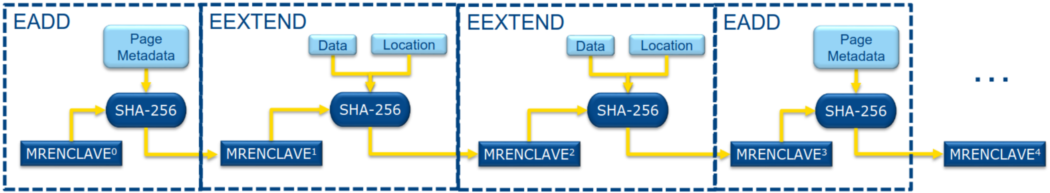
Pandora: Enclave-Aware Symbolic Exploration



Excurse: 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

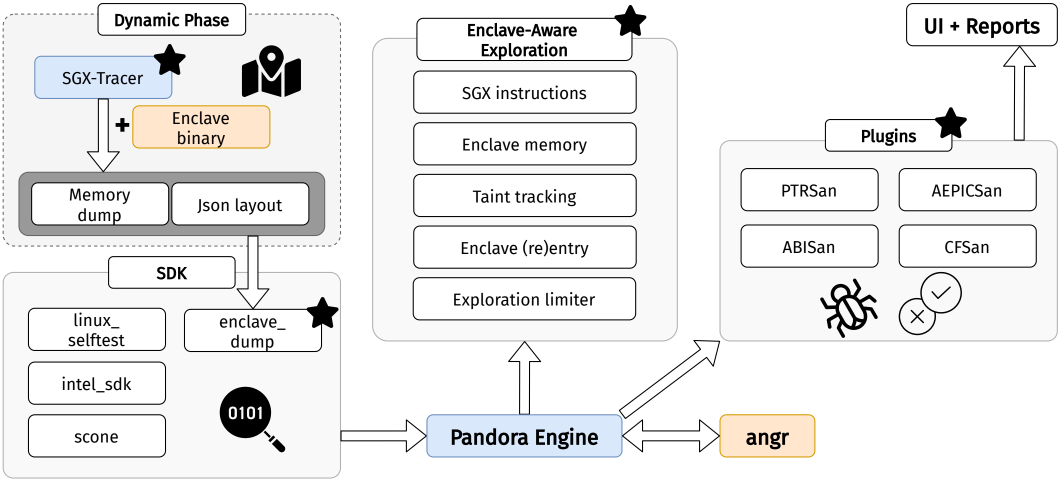
https://rwc.iacr.org/2016/Slides/Sealing%20and%20Attestation%20in%20SGX.pdf

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-fa79fff Reg  rwx (empty) meas=none
fa7a000-fabcfff Reg  r-x  (data) meas=all
fabd000-fabefff 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=0xffed000, nssa=2]
ffdc000-ffdcfff Tcs  ---  (data) meas=all [oentry=0xfa8a186, ossa=0xfff5000, nssa=2]
ffdd000-fffcfff Reg  rw- (empty) meas=all
fffd000-fffffff 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
31a000-31afff Reg  rw- (empty) meas=all
31b000-41afff Reg  rw- (empty) meas=none
41b000-42afff (unmapped)
42b000-46afff Reg  rw- [0xcc]* meas=all
46b000-47afff (unmapped)
47b000-47bfff Tcs  ---  (data) meas=all [oentry=0x4b32, ossa=0x47c000, nssa=2]
47c000-47dfff Reg  rw- (empty) meas=all
47e000-48dfff (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**

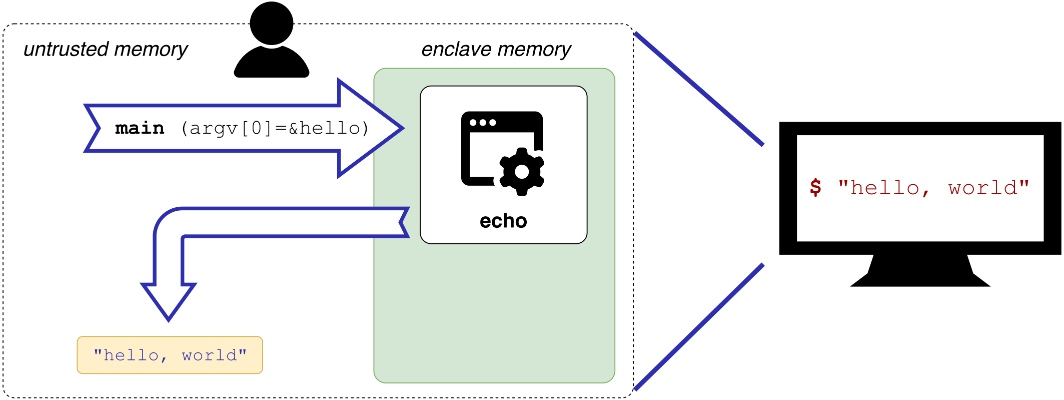
Pandora: Principled Symbolic Validation?

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

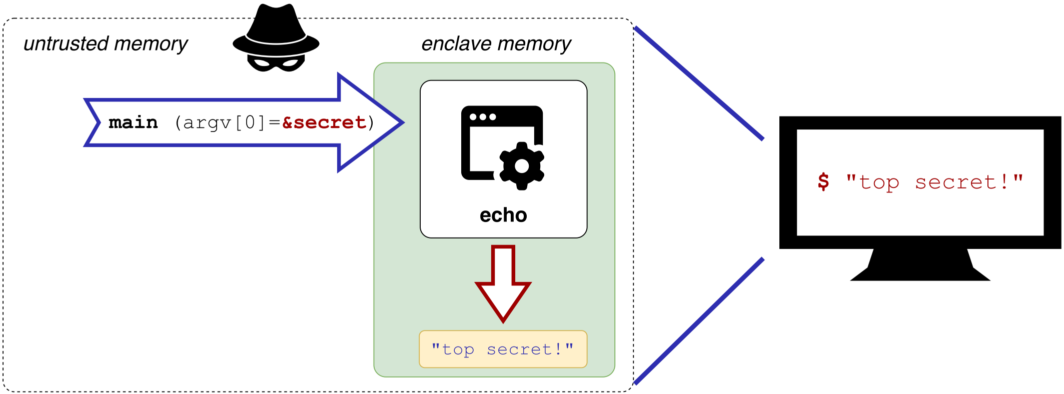
Challenge: Understanding attacks + specifying adequate invariants:

- 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



Excurse: Secure Enclave Pointer Usage is Hard. . .

```
struct encl_args {uint64_t value; uint64_t addr;};

static void do_encl_op_get_from_addr(struct encl_args *op)
{
    /* 1. Base pointer check */
    if (!sgx_is_outside_enclave(op, sizeof(struct encl_args)))
        return;
    /* 2. Prevent time-of-check time-of-use */
    volatile void* ptr = (void*) op->addr;
    /* 3. Nested pointer check */
    if (!sgx_is_outside_enclave((void*) ptr, 8))
        return;
    memcpy(&op->value, (void*) ptr, 8);
}
```

Excurse: Secure Enclave Pointer Usage is Hard...

```
struct encl_args {uint64_t value; uint64_t addr;};

static void do_encl_op_get_from_addr(struct encl_args *op)
{
    /* 1. Base pointer check */
    if (!sgx_is_outside_enclave(op, sizeof(struct encl_args)))
        return;
    /* 2. Prevent time-of-check time-of-use */
    volatile void* ptr = (void*) op->addr;
    /* 3. Nested pointer check */
    if (!sgx_is_outside_enclave((void*) ptr, 8))
        return;
    memcpy(&op->value, (void*) ptr, 8);
}
```

Excurse: Secure Enclave Pointer Usage is Hard...

```c
struct encl_args {uint64_t value; uint64_t addr;};

static void do_encl_op_get_from_addr(struct encl_args *op)
{
    /* 1. Base pointer check */
    if (!sgx_is_outside_enclave(op, sizeof(struct encl_args)))
        return;
    /* 2. Prevent time-of-check time-of-use */
    volatile void* ptr = (void*) op->addr;
    /* 3. Nested pointer check */
    if (!sgx_is_outside_enclave((void*) ptr, 8))
        return;
    memcpy(&op->value, (void*) ptr, 8);
}
```

Excurse: Secure Enclave Pointer Usage is Hard. . .

```
struct encl_args {uint64_t value; uint64_t addr;};

static void do_encl_op_get_from_addr(struct encl_args *op)
{
    /* 1. Base pointer check */
    if (!sgx_is_outside_enclave(op, sizeof(struct encl_args)))
        return;
    /* 2. Prevent time-of-check time-of-use */
    volatile void* ptr = (void*) op->addr;
    /* 3. Nested pointer check */
    if (!sgx_is_outside_enclave((void*) ptr, 8))
        return;
    memcpy(&op->value, (void*) ptr, 8);
}
```

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	○	★	★	★	○	●	★	○

Critical oversights in production and research code

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

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

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	• *Attacker tainted read inside enclave* at 0x2476
CRITICAL	• *Unconstrained read* at 0x22c3 • *Unconstrained read* at 0x20be

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, 0xfffffff800003008]
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

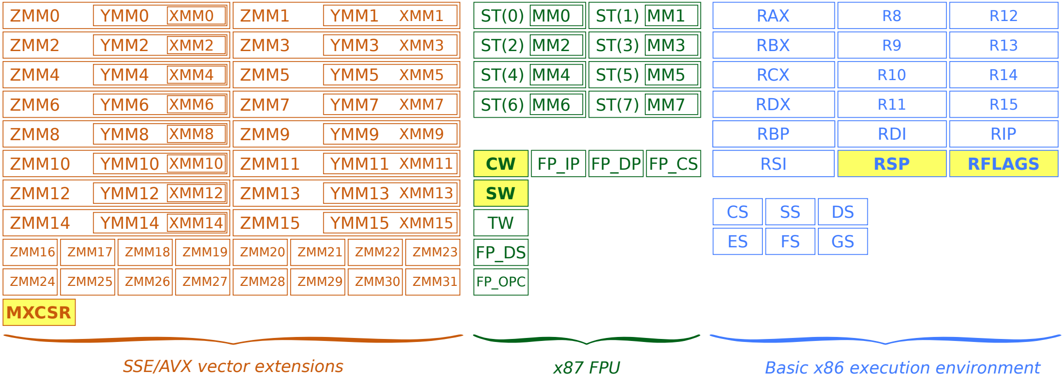
Disassembly

CPU registers

Backtrace

Basic block trace (most recent first)

ABI Vulnerabilities: x86 Control Register Poisoning

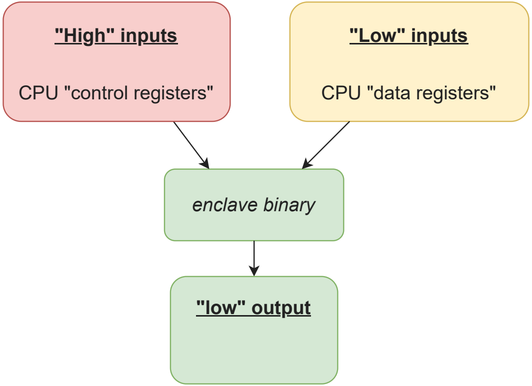


x86 user-space CPU control registers

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."

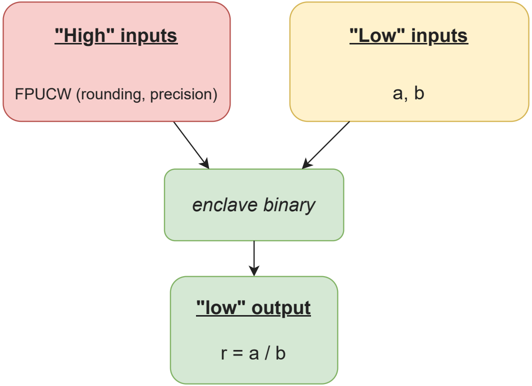
— Wikipedia



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



Alder et al. "Faulty Point Unit: ABI Poisoning Attacks on Intel SGX", ACSAC 2020.

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

```
*       rax     : <BV64 rsp_attacker_8_64>
        rcx     : <BV64 0x9>
*       rdx     : <BV64 rdx_attacker_5_64>
        rbx     : <BV64 0x0>
        rsp     : <BV64 0x8fe0>
*       rbp     : <BV64 rbp_attacker_10_64>
        rsi     : <BV64 0x3000>
        rdi     : <BV64 0x8fb0>
*       r8      : <BV64 rdi_attacker_14_64>
*       r9      : <BV64 r9_attacker_18_64>
*       r10     : <BV64 r10_attacker_20_64>
*       r11     : <BV64 r11_attacker_22_64>
*       r12     : <BV64 r12_attacker_24_64>
*       r13     : <BV64 r13_attacker_26_64>
*       r14     : <BV64 r14_attacker_28_64>
*       r15     : <BV64 r15_attacker_30_64>
*       d       : <BV64 d_attacker_38_64>
        rip     : <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	✔	✘ †	ABISan	1
EnclaveOS	3.28	✔	✘ †	PTRSan	15
EnclaveOS	3.28	✔	✘ †	ÆPICSan	33
EnclaveOS	3.28	✔	✘ †	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

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

- **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?

Intel software mitigation advisory: ÆPIC 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)[12]. 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.


SHELDON COOPER
presents
FUN WITH FLAGS
REC

x86 string instructions: Direction Flag (DF) operation

- x86 rep string instructions to speed up streamed memory operations

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

→

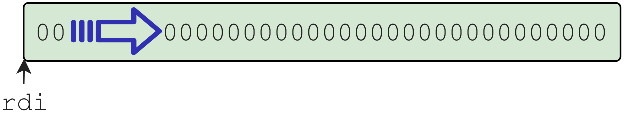
```asm
lea rdi, buf
mov al, 0x0
mov ecx, 100
rep stos [rdi], al
```

x86 string instructions: Direction Flag (DF) operation

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

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

```asm
lea rdi, buf
mov al, 0x0
mov ecx, 100
rep stos [rdi], al
```

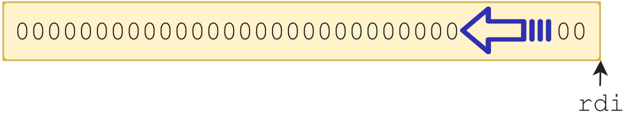


x86 string instructions: Direction Flag (DF) operation

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

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

```asm
lea rdi, buf+100
mov al, 0x0
mov ecx, 100
std ; set direction flag
rep stos [rdi], al
```



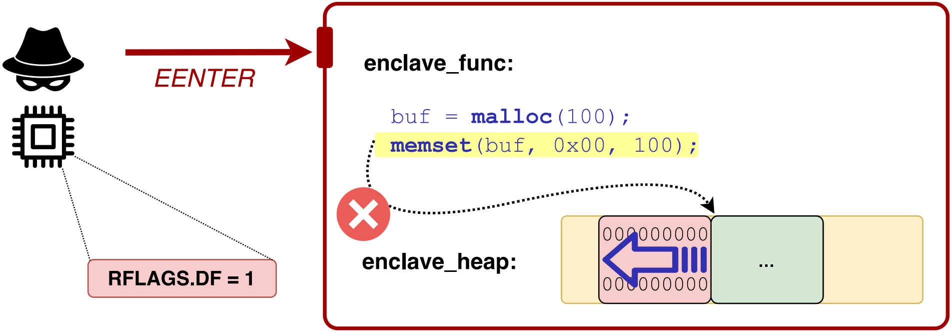
SGX-DF: Inverting enclaved string memory operations

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. . .



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