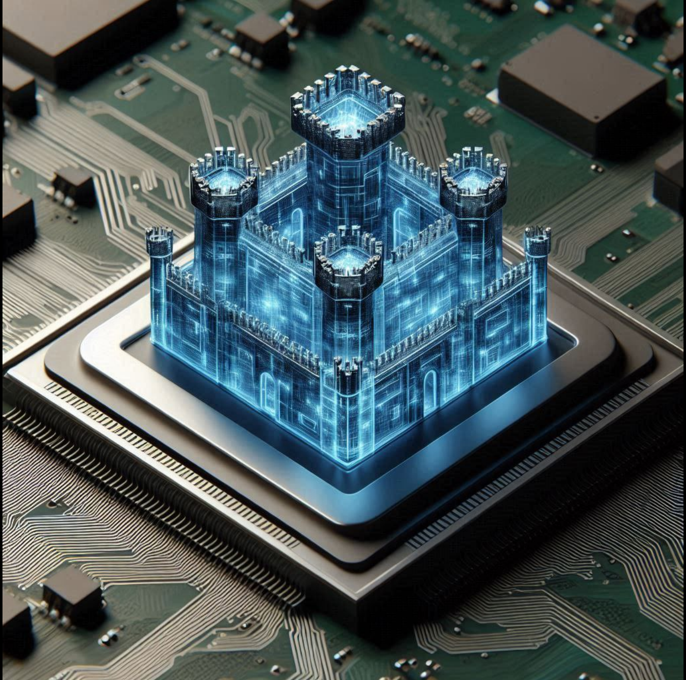
Pandora: Principled Symbolic Validation of Intel SGX Enclave Runtimes

Fritz Alder[1], Lesly-Ann Daniel[1], David Oswald[2], Frank Piessens[1], Jo Van Bulck[1]

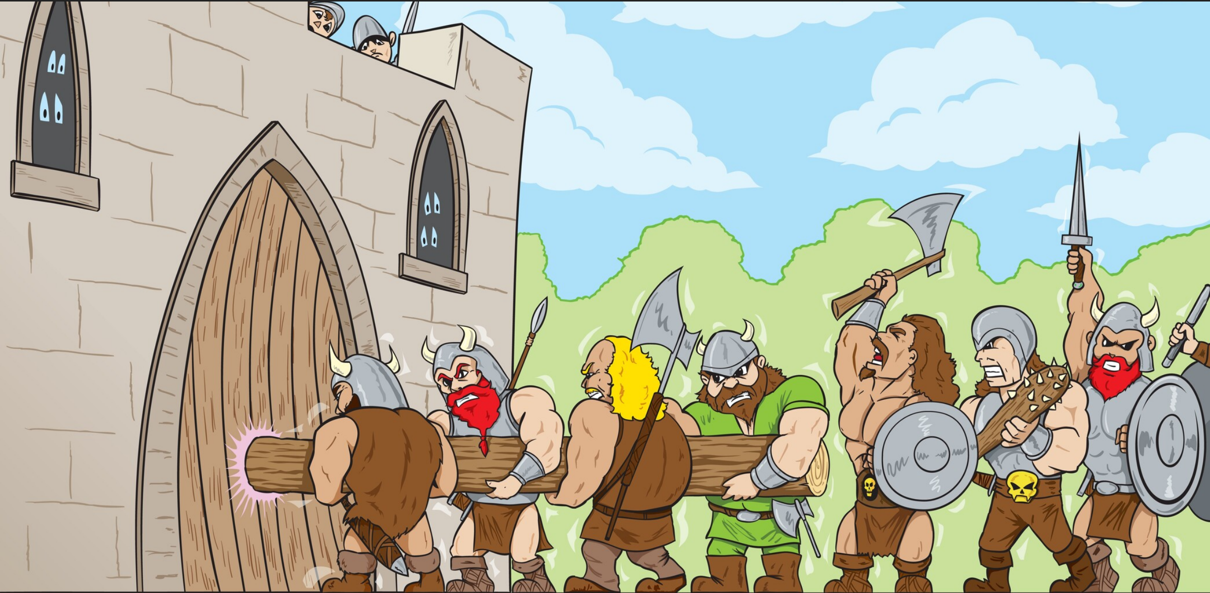
45th IEEE Symposium on Security and Privacy (S&P) – May 22, 2024

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



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

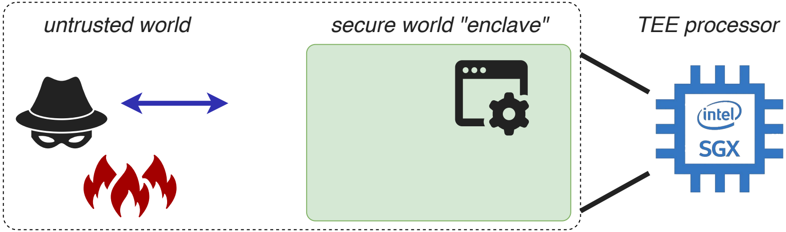
x87 FPU operations in enclaves are vulnerable to ABI poisoning — Low
GHSA-7wjx-wcwg-w999 published on Jul 14, 2020 by CodeMonkeyLeet

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

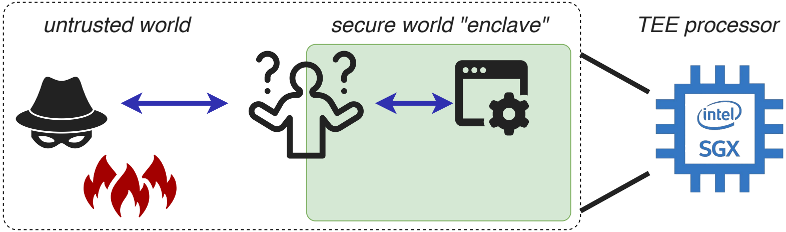
Enclave heap memory disclosure vulnerability — Moderate
GHSA-mg2p-657r-46cj published on Oct 8, 2019 by CodeMonkeyLeet

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

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

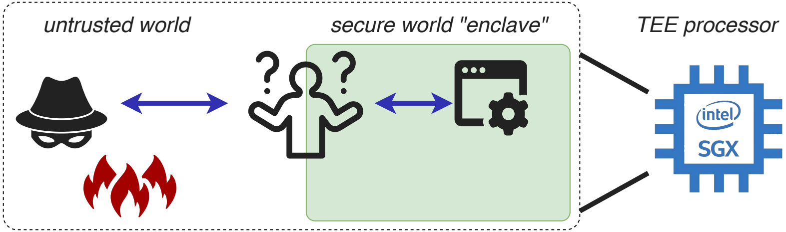


Context: Writing “Secure” Enclave Software is Hard. . .



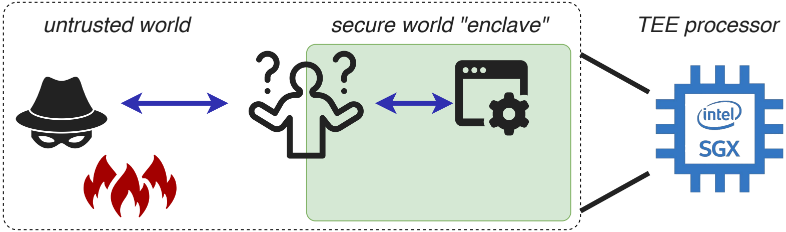
- API level: Sanitize pointer arguments in shared address space

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



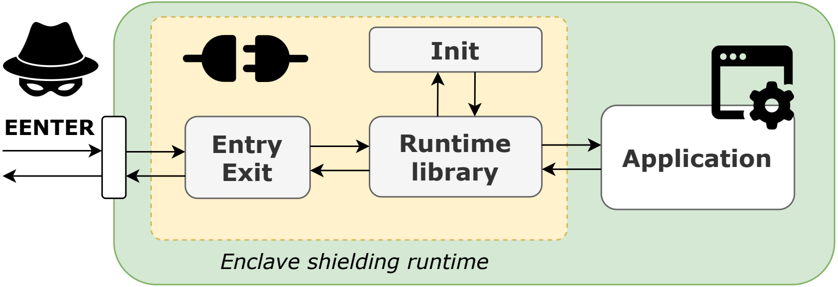
- API level: Sanitize pointer arguments in shared address space
- ABI level: Sanitize low-level CPU configuration registers

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

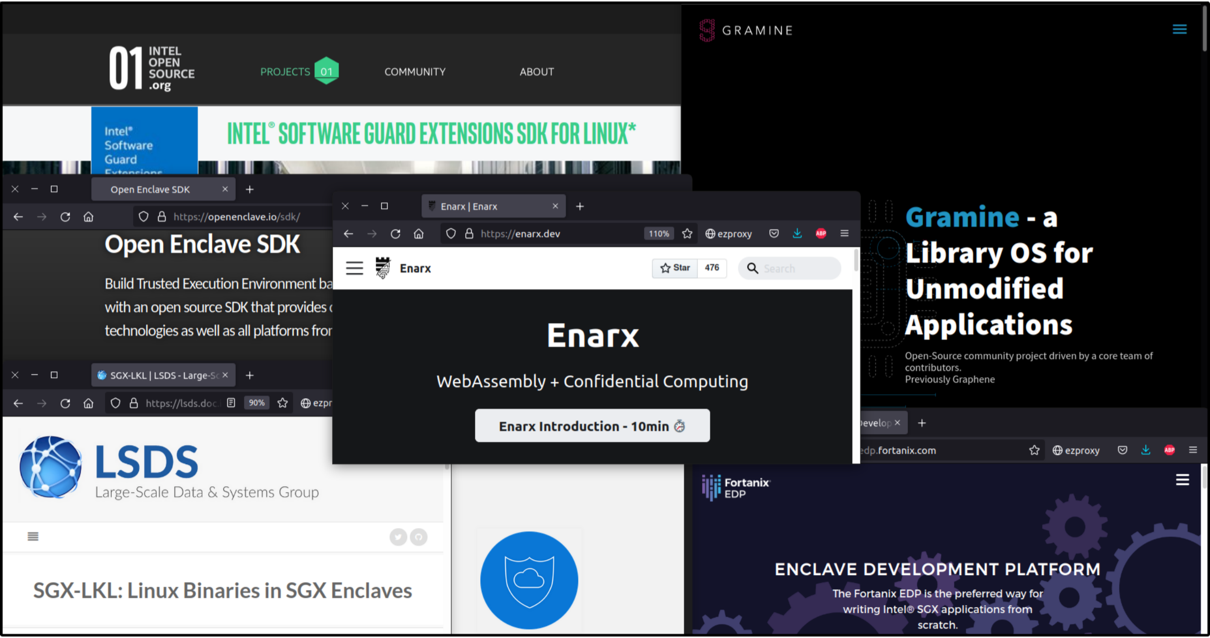


- API level: Sanitize pointer arguments in shared address space
- ABI level: Sanitize low-level CPU configuration registers
- μ-arch level: Spectre/LVI → `lfence`; ÆPIC/MMIO stale data → `verw`; cacheline GPU leak → avoid `dword0/1`...

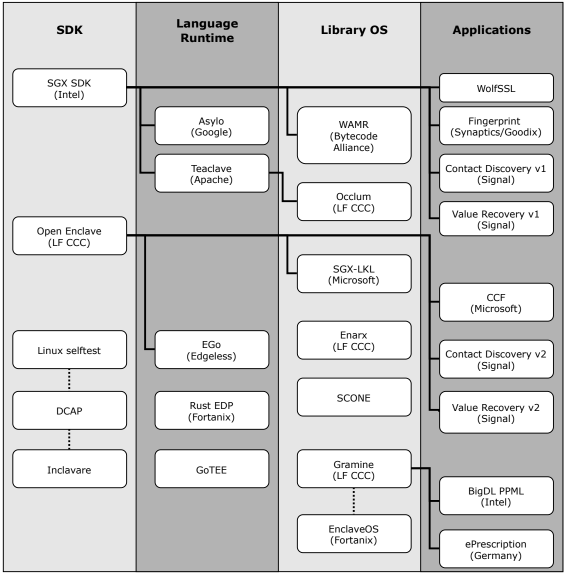
Solution: Enclave Shielding Runtimes



Key idea: Transparent input sanitization on enclave entry/exit

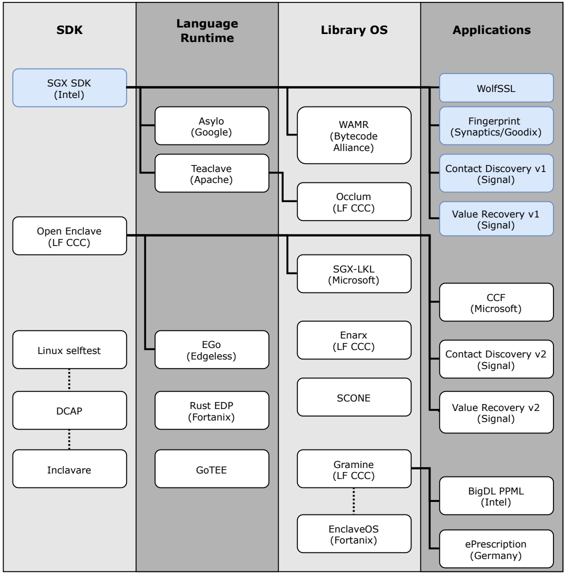

01 INTEL OPEN SOURCE .org
PROJECTS
COMMUNITY
ABOUT
INTEL® SOFTWARE GUARD EXTENSIONS SDK FOR LINUX*
Intel® Software Guard
Open Enclave SDK
https://openenclave.io/sdk/
Open Enclave SDK
Build Trusted Execution Environment ba
with an open source SDK that provides
technologies as well as all platforms fror
SGX-LKL | LSDS - Large-Sc
https://lsds.doc
LSDS
Large-Scale Data & Systems Group
SGX-LKL: Linux Binaries in SGX Enclaves
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
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



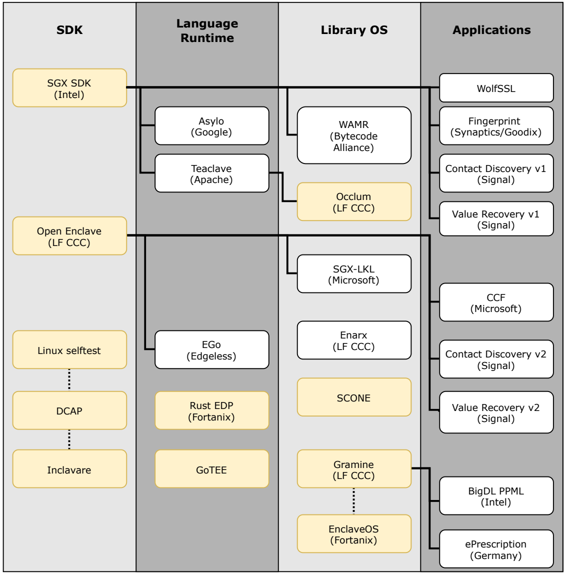
- **Ecosystem:** Diverse programming paradigms & abstractions

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



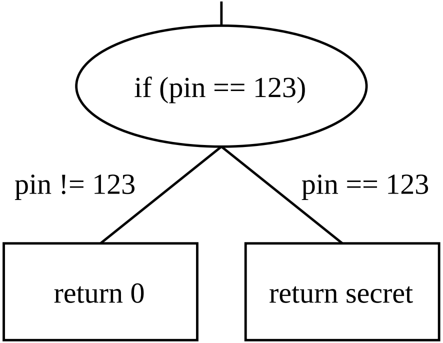
- **Ecosystem:** Diverse programming paradigms & abstractions
- **Prior work:** Selected applications on Intel SDK (e.g., `NULL` pointers)
- **Pandora:** Runtime-agnostic & truthful symbolic execution
    1. Exact attested memory binary
    2. Vulnerability detection plugins

1. Truthful Symbolic Execution

Background: Symbolic Execution and angr

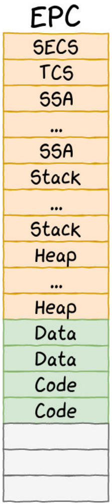
```
int ecall(int pin){
    if(pin == 123){
        return secret;
    } else {
        return 0;
    }
}
```

https://angr.io/



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

Challenge: Intel SGX Memory Layout



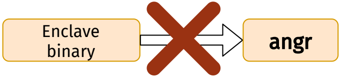
Angr is designed to load normal OS binaries

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

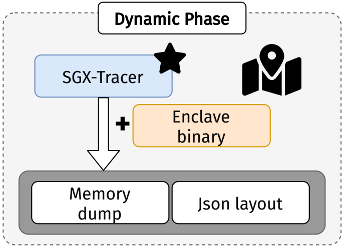
- Untrusted runtime loader parses ELF binary embedded metadata to create enclave image with TCS, SSA, Stack, Heap, etc.
- `MRENCLAVE` attestation independent of load address → partial relocation in enclave

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

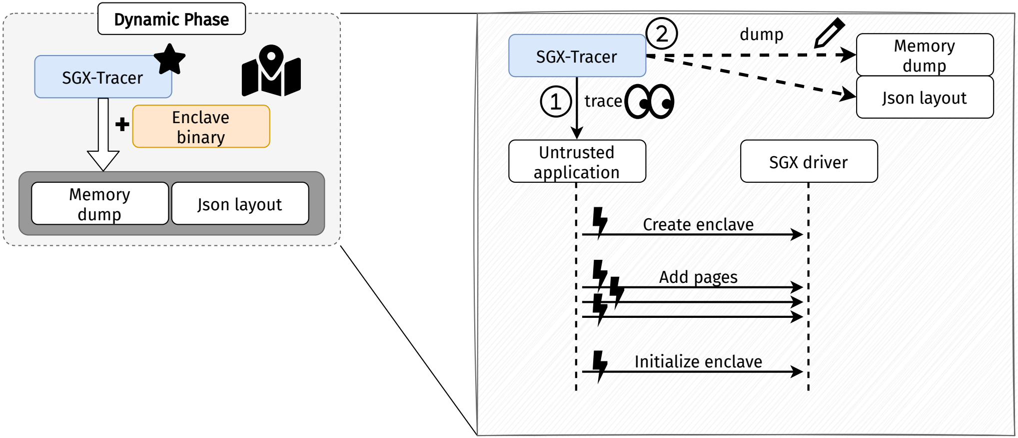
Pandora: Runtime-Agnostic Enclave Loading



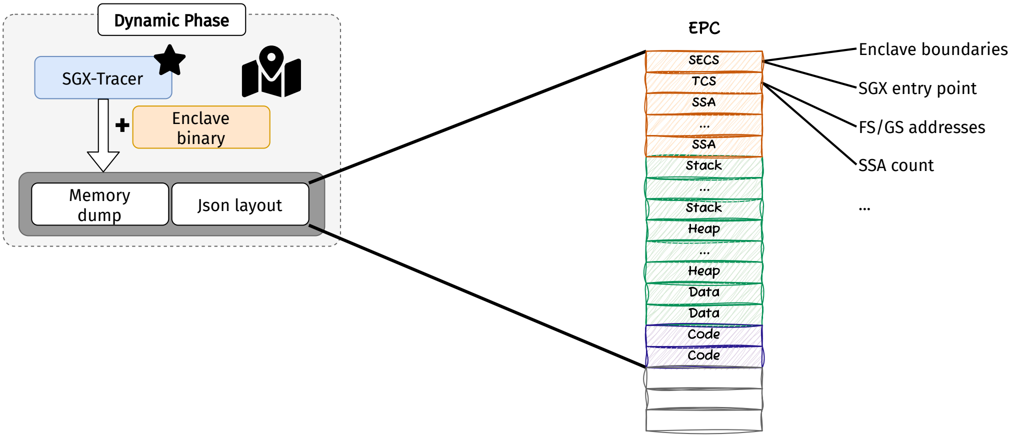
Pandora: Runtime-Agnostic Enclave Loading



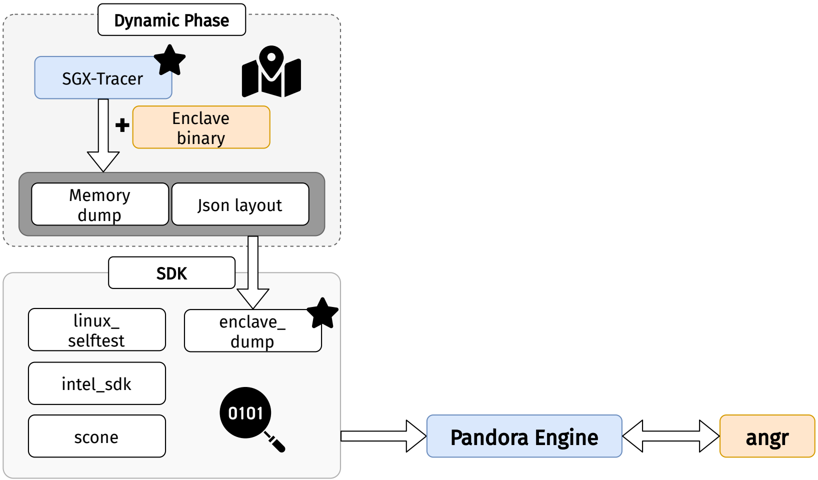
Pandora: Runtime-Agnostic Enclave Loading



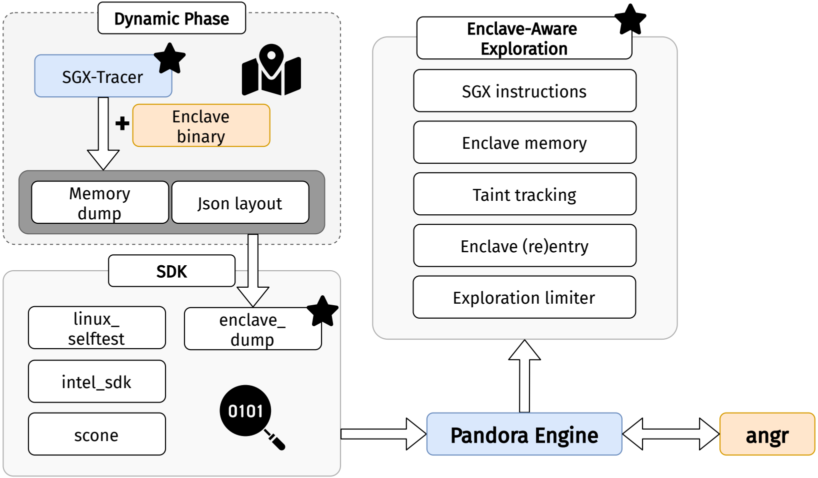
Pandora: Runtime-Agnostic Enclave Loading



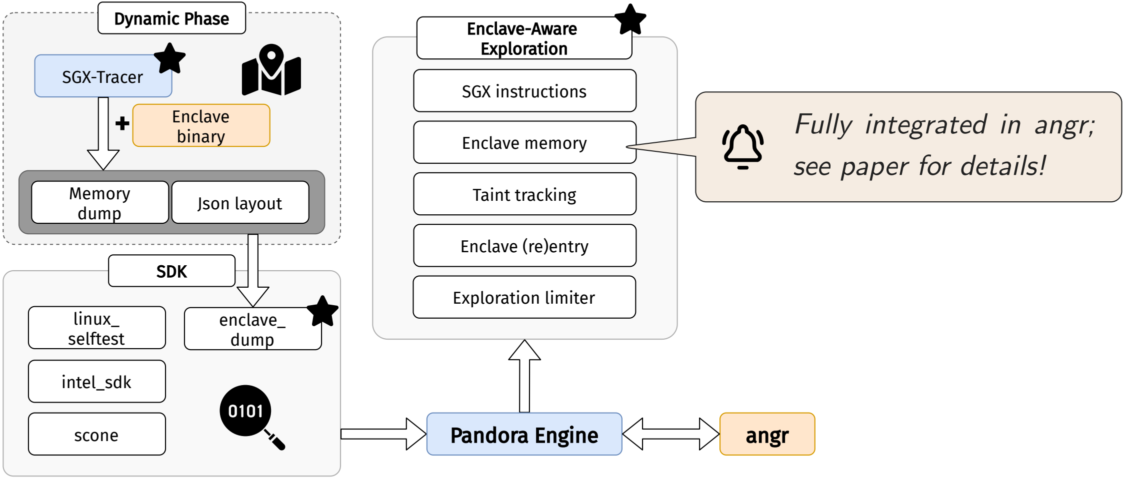
Pandora: Runtime-Agnostic Enclave Loading



Pandora: Enclave-Aware Symbolic Exploration

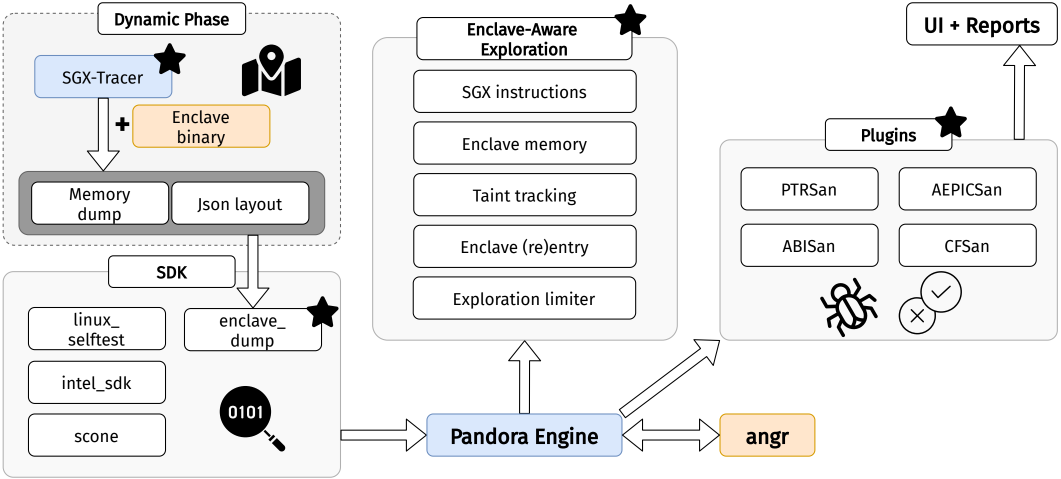


Pandora: Enclave-Aware Symbolic Exploration

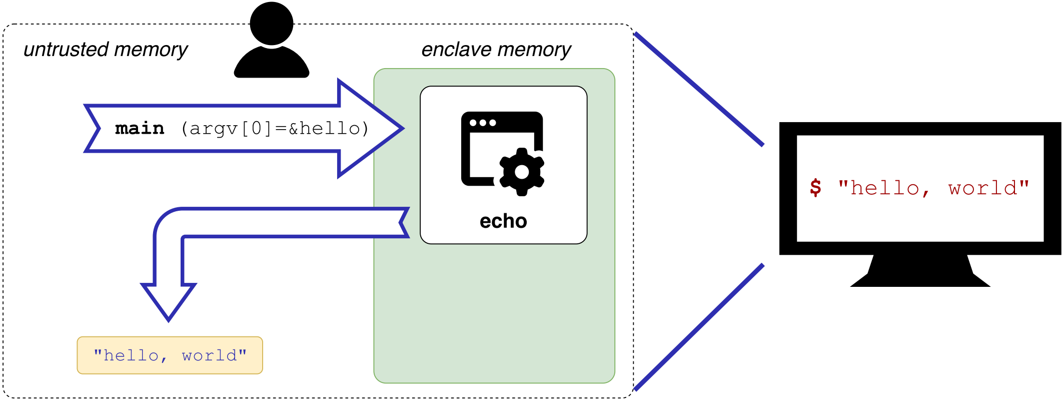


2. Pluggable Vulnerability Detection

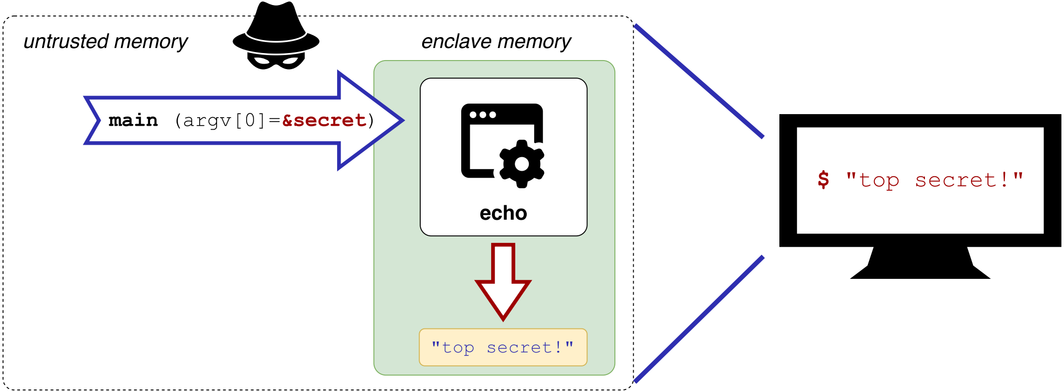
Pandora: Plugin-Based Vulnerability Detection



API Vulnerabilities: Confused-Deputy Attacks



API Vulnerabilities: Confused-Deputy Attacks



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...

```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);
}
```

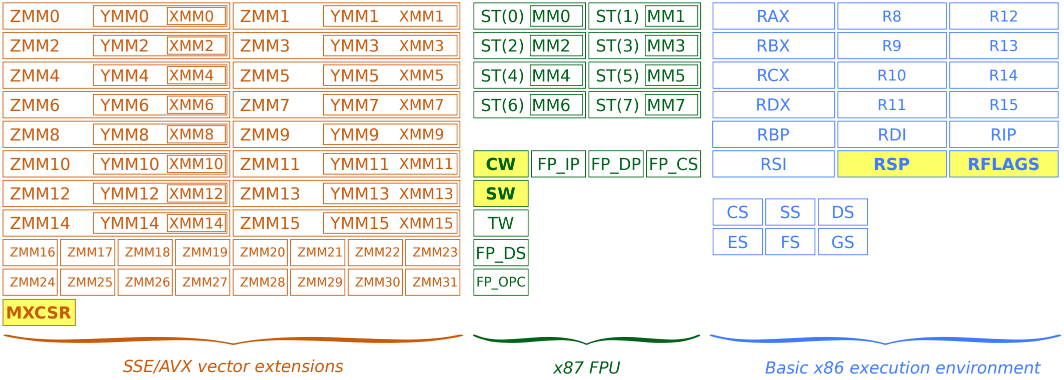
Excurse: Secure Enclave Pointer Usage is Hard...

```
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);
}
```

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);
}
```

ABI Vulnerabilities: x86 Control Register Poisoning



x86 user-space CPU control registers

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

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

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)

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!*