Intel SGX
Emulation using QEMU

Prerit Jain
Soham Desai
Overview

• Problem Statement
• Proposed Solution & Design
• Difficulties Faced
• What’s Next? Future Work
Problem Statement: (A short Recap)

• SGX provides a set of new CPU instructions that can be used by applications to set aside private regions of code and data.

• We aimed to create an emulation platform for SGX using ‘QEMU’ the open source machine emulator.
Proposed Solution and Design

• Developed QEMU translation core for interpreting and translating new SGX instructions.

• Take advantage of the ‘user emulation’ feature of QEMU. Created a User space Library providing support for both User and Kernel Space SGX functionalities.

• Provide Access Control, Data Structures within QEMU

• Cryptographic functionality using Polar SSL Crypto Library
Overview of Modifications

Application Binary
  ➔ Target Specific Translation
    ➔ TCG Micro Ops
      ➔ Host Object Code
        ➔ CPU Execution Loop

Applicatio
n Binary
  ➔ Enclav
  e Code

SGX Library
  ➔ Target Specific Translation
    ➔ Translation Logic for SGX Instructions

QEMU
Tiny Code Generator

QEMU
Decoding Logic to Interpret New Instructions
Providing Access Control

• Dedicate Virtual Space for the Enclave Page Cache. (During Initialization of QEMU)
• Check all Load/Stores within QEMU translation. (During Translation phase)
• Prevent Access if address falls within the dedicated region if not in Enclave mode.
// Non-Enclave Mode Access
if(!env->cregs.CR_ENCLAVE_MODE) {
    if(checkEINIT(env) && enclave_Access) {
        if(checkRange(mem_addr) || (mem_addr == (uint64_t)epcm) || (mem_addr == (uint64_t)process_priv_key)
            || (mem_addr == (uint64_t)process_pub_key) || (checkEnclaveFunctions(mask(mem_addr, PAGE_SIZE))) {
            if(checkEnclaveState()) {
                setEnclaveState(false);
                updateEntry(mem_addr);
                return;
            }
        }
        sgx dbg(trace, "!Mode EINIT Range: Accessed %lx", mem_addr);
        raise_exception(env, EXCP0D_GPF);
    }
}

// Enclave Mode Access Check
if(env->cregs.CR_ENCLAVE_MODE) {
    if(((checkRange(mem_addr)) &&
        (!(mem_addr == env->cregs.CR_EL RANGE[0]) &&
        (mem_addr <= (env->cregs CR_EL RANGE[0] + env->cregs.CR_EL RANGE[1]))) || checkOtherEnclaveFunctions(curr_Eid, men_addr)) {
        sgx dbg(trace, "Mode EINIT Range: Accessed %lx", mem_addr);
        sgx dbg(trace, "Inside Enclave. Accessing Incorrect enclave memory");
        raise_exception(env, EXCP0D_GPF);
    }
}
```c
void enclu(enclu_cmd_t leaf, u64 rbx, u64 rcx, u64 rdx, out_regs_t* out_regs)
{
    /*
    sgx_dbg(trace,
        "leaf=%d, rbx=%"PRIx64", rcx=%"PRIx64", rdx=%"PRIx64")",
        leaf, rbx, rcx, rdx);
    */
    asm("movl %0, %eax\n"
         "movq %1, %rbx\n"
         "movq %2, %rcx\n"
         "movq %3, %rdx\n"
         ".byte 0x0F\n"
         ".byte 0x01\n"
         ".byte 0xd7\n"
         ":="
         ":="(u32)leaf),
    "b"(rbx),
    "c"(rcx),
    "d"(rdx));

    // Check whether function requires out_regs
    if(out_regs != NULL) {
        asm volatile ("": : : "memory"); // Compile time Barrier
        asm volatile ("movl %eax, %0\n"
                     "movq %rbx, %1\n"
                     "movq %rcx, %2\n"
                     "movq %rdx, %3\n"
                     ":=a"(out_regs->oeax),
                     ":=b"(out_regs->orbx),
                     ":=c"(out_regs->orcx),
                     ":=d"(out_regs->ordx);
    }
}
```
Difficulties Faced

• Understanding QEMU semantics. Its Internal Representation of x86 architecture, control flow, translation of guest to host operations.

• Collaboration with multiple contributors. (ensuring segregation of work and interoperability of modules created by different individuals)
Approaching Completion

• Exception Handling Mechanism:

• Currently working with a basic exception such as a Floating Point Exception (FPE) but need to take into account different asynchronous exits.
Some Statistics

• Total Lines of Code Added: 5000 +

• Total GIT Commits : 350 +

• Number of Contributors: 5
  Professors: Dr. Taesoo Kim, Dr. Dongsu Han (KAIST)
  Students: Seongmin(KAIST), Prerit Jain, Soham Desai

• What we Learned:
  Emulation using QEMU, Development of Console Application,
  shared and static libraries, kernel module, x86 Architecture, Unit
  Testing, Device Driver, GUI development using QT.
Future Work Possibilities

• Showcasing SGX functionalities for different applications and creating prototypes.

• Providing SGX Support for Applications built for different platforms like ARM, SPARC, etc. by using QEMU translation

• Extending Emulation support for remaining SGX instructions
Demo
Time for Q & A !!
Questions for us?
Back Up
Overview of Enclave Creation

1. Application hands over Enclave content to OS enclave creation service. (ENCLS Leaf Instructions)
   - Initial Setup, Reserving Memory, Basis Data Structures -> ECREATE Instruction
   - Committing pages from protected storage for code and data -> EADD Instruction
   - Finalize Measurement and complete creation process -> EINIT Instructions

1. Once the Enclave is created, the application can execute ENCLU leaf instructions.
   - Entering into the enclave, performing a context switch to the Enclave execution context -> EENTER
   - Restoring the context and exiting the enclave -> EEXIT

Thus for showcasing a Simple Application we need to emulate the following instructions ECREATE, EADD, EINIT, EENTER, EEXIT -> Our Primary target.
SGX and QEMU Architecture

**Intel SGX**
- 2 New Instructions
  - ENCLU (For User Space)
  - ENCLS (For Kernel)
- Each has multiple leaf instructions which together provide the complete SGX functionality

**QEMU**
- Interpreting the new Opcode
- And Leaf functions and providing the functionality expected from Hardware.

---

**Diagram**

- Application
- User Space
- ENCLU
- SGX Kernel Module
- Linux Kernel
- ENCLS
- QEMU X86 Emulation
- SGX Architectural Implementation