OpenSGX: An Open Platform for SGX Research

Peng Xu

April 29, 2019

Table of Contents

Introduction

1. Why TEE is necessary?
2. How to adopt the traditional software development into TEE model?
3. Access to the SGX platform is (currently) limited [2016]

Introduction

1. Why TEE is necessary?
2. How to adopt the traditional software development into TEE model?
3. Access to the SGX platform is (currently) limited [2016]
4. openSGX
   - 4.1 Emulates Intel SGX
   - 4.2 Implements on Instruction-level
   - 4.3 Extends open-sourced QEMU emulator
   - 4.4 Includes emulator, emulated OS layer, enclave program loader/package, user library, debugging supporting and performance monitoring

Background

1. Intel SGX
   - 1.1 SGX memory protection : enclave model
   - 1.2 Enclave Page Cache (EPC)
   - 1.3 Instruction Set Architecture (ISA)
   - 1.4 User-level and privileged instruction

Background

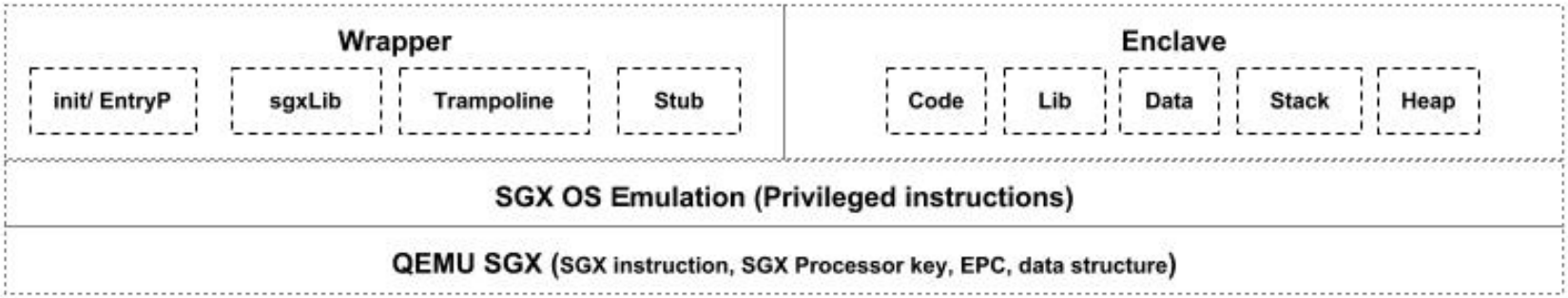
1. Intel SGX
   - 1.1 SGX memory protection : enclave model
   - 1.2 Enclave Page Cache (EPC)
   - 1.3 Instruction Set Architecture (ISA)
   - 1.4 User-level and privileged instruction
2. OpenSGX Specification
   - 2.1 Faithfully implements the Intel SGX specification
   - 2.2 Extends to support OS, debugging and monitoring
   - 2.3 Not implement all Intel SGX's instruction

Background

1. Intel SGX
   - 1.1 SGX memory protection : enclave model
   - 1.2 Enclave Page Cache (EPC)
   - 1.3 Instruction Set Architecture (ISA)
   - 1.4 User-level and privileged instruction
2. OpenSGX Specification
   - 2.1 Faithfully implements the Intel SGX specification
   - 2.2 Extends to support OS, debugging and monitoring
   - 2.3 Not implement all Intel SGX's instruction
3. Threats
   - 3.1 Assumes an adversary can control all software components
   - 3.2 Considers the thwart attacks mounted by system software
   - 3.3 Denial-of-service is our of scope

System Overview - Components

1. Emulated Intel SGX hardware
2. OS Emulation
3. Enclave program loader
4. OpenSGX user library (sgxlib)
5. Debugger support
6. Performance monitoring



Design - Hardware Emulation

1. Instruction Set Architecture (ISA): User and Super privilege
2. Enclave Page Cache (EPC)
   - 2.1 A contiguous memory region
   - 2.2 Sgx_init() allocates and initializes the system-wide EPC pages
3. EPC access protection
   - 3.1 An enclave accesses only its own EPC pages
   - 3.2 Enclave Page Cache Map (EPCM): maintains these access permissions
   - 3.3 Enclave and non-enclave accesses
4. EPC encryption
5. Data structure and SGX

Design - OS Emulation

1. Bootstrapping: sys_init()
2. Enclave initialization
   - 2.1 SGX Instruction: ECREATE, EADD, EEXTEND and EINIT
   - 2.2 sys_create_enclave()
3. EPC page translation: virtual address to physical address
4. Dynamic EPC page allocation
   - 4.1 EAUG, EACCEPT
   - 4.2 sys_add_epc()
5. Performance monitor
   - 5.1 Custom identifier (Keid)
   - 5.2 sys_stat_enclave()
6. System call emulation
   - 6.1 Emulates system call as a function call
   - 6.2 Context switching
   - 6.3 State Save Area (SSA)
   - 6.4 sgx_add_epc()

Design - Loader

1. Compilation
   - 1.1 Toolchain: opensgx generates an enclave program
   - 1.2 Structure of enclave program: code, data sections, linked libs
2. Loader
   - 2.1 Determines the memory layout of code, data, stack and heap sections
   - 2.2 Obtains the information about code and data as well as base address
   - 2.3 Forwards the memory layout information to the OS emulation

Design - User library

1. User-level library - sgxlib
2. Host APIs and ENCLAVE APIs

Type	API	Description
HOST	`void sgx_init(void)`	Perform system initialization
HOST	`void sgx_enter(tcs_t tcs, void (*aep)())`	`EENTER` wrapper
HOST	`void sgx_resume(tcs_t tcs, void (*aep)())`	`ERESUME` wrapper
HOST	`int sgx_host_read(void *buf, int len)`	Read from enclave
HOST	`int sgx_host_write(void *buf, int len)`	Write to enclave
HOST	`void launch_quoting_enclave(void)`	Launch quoting enclave
ENCL	`void sgx_exit(void *addr)`	`EEXIT` wrapper
ENCL	`void sgx_remote(const struct sockaddr *target_addr, socklen_t addrlen)`	Request remote attestation
ENCL	`void sgx_getkey(keyrequest_t keyreq, void *key)`	`EGETKEY` wrapper
ENCL	`void sgx_getreport(targetinfo_t info, reportdata_t data, report_t *report)`	`EREPORT` wrapper
ENCL	`int sgx_enclave_read(void *buf, int len)`	Read from host
ENCL	`int sgx_enclave_write(void *buf, int len)`	Write to host
ENCL	`void *sgx_memcpy(void *dest, const void *src, size_t size)`	Memory copy
ENCL	`void *sgx_memmove(void *dest, const void *src, size_t size)`	Memory copy
ENCL	`void sgx_memset(void *ptr, int value, size_t num)`	Memory set to the specified valu
ENCL	`int sgx_memcmp(const void *ptr1, const void *ptr2, size_t num)`	Memory comparison
ENCL	`size_t sgx_strlen(const char *string)`	Get string length
ENCL	`int sgx_strcmp(const char *p1, const char *p2)`	String comparison
ENCL	`int sgx_printf(const char *format, ...)`	Write formatted data to standard

Design - User library

1. User-level library - sgxlib
2. Custom in enclave library
3. Shared code and data memory trampoline and stub
4. Enclave - Host communication: Pipe-liked mechanism
5. Dynamic memory allocation
6. Defense against malicious host apps and OS
7. Remote attestation

Design - Defense against malicious host apps and OS

1. Memory-related operations (MEM)
2. Network and I/O services (IO/NET)
3. Non-determinism and resources (DBG, TIME, RAND)

Debugging

1. Debugging hardware
2. Debugging enclaves: gdb-stub
3. New gdb commands: info epc, info epcm, info secs and list enclaves

Design - Performance Monitoring

1. Enclave descriptor: stores TCS and usage statistics
2. Number of context switches
3. Entires/exits of the OS emulation layer
4. Number of TLB flushes
5. Number of dynamically allocated EPC

Case studies - Shielding Tor Nodes

1. Attacks on Tor
   - 1.1 Manipulating Tor Components
   - 1.2 Manipulating routing
2. Benefits of applying TEE
   - 2.1 Attestation of software components
   - 2.2 Protection against tampered OS
   - 2.3 contains all critical operations that use private data structures in Tor-enclave
   - 2.4 expose an RPC interface to Tor-non-enclave

Secure I/O Path

1. Allows secure communication between the CPU/memory and devices
2. Emulates the encrypted communication channel with message authentication

Performance Profiling

1. Environment setup
2. The number of EPC pages used
3. Additional CPU cycles
4. Context switch overhead
5. The number of RPC calls

Questions?