Security Evaluation for Code Generation Tool of Intel® Software Guard Extensions SDK

Zhang, Shunda; Jin, Shuai; Yang, Debin
Agenda

- Problem Statement
- Background Introduction – SGX/SDK/Edger8r Tool
- Security Evaluation and Analysis
- Summary
Problem Statement

- Intel SGX provided a mechanism to better isolate user-level software from attackers.
- SGX SDK provided a code generation tool to generate proxy/bridge code to help user writing Enclave code.
- What kind of security issues will most likely to happen in such code generation tool in a TEE (Trusted Execution Environment)?
  - What is the security impact and mitigation?
  - How to find such vulnerabilities effectively?
Intel SGX (Software Guard Extensions)

CPU Package

- Cores
- Cache

System Memory

1. Security perimeter is the CPU die boundary.
2. Data and code are unencrypted inside CPU die.
3. Data and code outside CPU die are encrypted and integrity checked*, with access control that prevents access from unauthorized SW.
4. External memory reads and bus snoops see only encrypted data.

* The portion of the System Memory that’s protected is called EPC.
SGX: Protection vs. Software Attack

Application

1. App is built with trusted and untrusted parts
2. App runs & creates enclave which is placed in trusted memory
3. Trusted function is called; code running inside enclave sees data in clear; external access to data is denied
4. Function returns; enclave data remains in trusted memory
SGX Programming Environment

- **OS**
- **Enclave Code**
- **Enclave Data**

**User Process**

**Enclave**

**Callable Environment**
- **Caller Environment**
  - **ISV code. Calls proxy.**
  - **RTS SDK-generated edge routine. Marshals parameters. Unmarshals return values.**

**Callee Environment**
- **ISV code. Carries out execution.**
- **RTS SDK-generated edge routine. Unmarshals parameters. Marshals return values.**

**Proxy Function**
- **uRTS / tRTS**
- **See SE PAS for descriptions of SE instructions**
- **EENTER / EEXIT**
- **EEXIT / EENTER**

**Call Path**

**Return Path**

**Bridge Function**

*note: in some cases, execution transitions directly from SE instruction to bridge.*

*Other brands and names are the property of their respective owners.*
Code Generation Tool in SGX SDK

- `sgx_edger8r` tool ships as part of the Intel® Software Guard Extensions Evaluation SDK
- Generates code by reading a user-provided EDL file
- These edge routines provide the interface between the untrusted application and the enclave

**Enclave Trusted Computing Base (TCB)**

Intel SGX Application

![Diagram of Enclave Trusted Computing Base (TCB)]
Security Evaluation of SGX_Edger8r

- Key points may impact security
  - Sensitive data transfer cross trust boundary
  - Un-validated Pointers
  - Race condition (TOCTOU)

- *Others – Side Channel Attack
Data across trust boundary

- TEE has trust boundaries
- Data across trust boundary should be taken care of
  - Buffer length
    - Miss-math
    - Length not check
  - Clear out secret/sensitive data after use
    - Example: Padding issue
Buffer length miss-math/overflow

- Buffer calculation error in corner cases
- Buffer length limitation check
Suggestion for Validation

- Test target is generated code, not only the tool’s code
- Fuzz test
struct test{
    int a;
    double b;
    short c;
}; /* 24 bytes in memory, but actually only 14 bytes are valid, others are padding */

struct test2{
    char a[2];
    short b;
}; /* => no padding */

struct test3{
    char a[3];
    short b;
}; /* => 6 bytes in memory, but only 5 bytes are valid */

struct test4{
    char a[3];
}; /* => no padding */

struct test5{
    struct test4 a;
    short b;
}; /* => 6 bytes in memory, but only 5 bytes are valid */

struct test6{
    int a;
    short b;
    long c;
}; /* => 16 bytes in memory, but only 14 bytes are valid */

struct test7{
    char a[3];
    char b[8];
}; /* => no padding */

int main(int argc, char* argv[]){
    printf("sizeof int is %d\n", sizeof(int));
    printf("sizeof short is %d\n", sizeof(short));
    printf("sizeof double is %d\n", sizeof(double));
    printf("sizeof long is %d\n", sizeof(long));
    printf("sizeof test1 is %d\n", sizeof(struct test1));
    printf("sizeof test2 is %d\n", sizeof(struct test2));
    printf("sizeof test3 is %d\n", sizeof(struct test3));
    printf("sizeof test4 is %d\n", sizeof(struct test4));
    printf("sizeof test5 is %d\n", sizeof(struct test5));
    printf("sizeof test6 is %d\n", sizeof(struct test6));
    printf("sizeof test7 is %d\n", sizeof(struct test7));
    return 0;
}

output:
    sizeof int is 4
    sizeof short is 2
    sizeof double is 8
    sizeof long is 8
    sizeof test1 is 4
    sizeof test2 is 8
    sizeof test3 is 6
    sizeof test4 is 8
    sizeof test5 is 6
    sizeof test6 is 16
    sizeof test7 is 11
Padding Issue

- Information leak
  - Secret data, not cleared by other function calls
  - Security cookie/canary in stack
    - Security cookie is random but unchanged after each load
Mitigation

- Clear secret (memset_s) after use
- Initialize (memset_s) data structure
- Use pack(1)
- Compiler option /Zp1 (VS), -fpack-struct (gcc)
- Add static_assert for each structure they are the same size as designed and without padding
- Remind Enclave writer in developer guide
Security cookies

- Security cookies in stack and heap can be leaked by padding issue. It is not cleaned after use.
- Clang/LLVM plugin to clear security cookies
- ICC new feature to clear security cookies

```assembly
mov %fs:0x28,%r9
mov %r9,-0x8(%rbp)
...
mov %fs:0x28,%rcx
mov -0x8(%rbp),%rdx
movq $0x0,-0x8(%rbp) ;clear the stack guard on stack
cmp %rdx,%rcx
```
Un-validated Pointers

- Pointers need to be validated
  - NULL pointer de-reference
  - Potential memory overflow caused by un-validated pointer from malloc(0)
    - Should not assume malloc(0) will return NULL
NULL Pointer Dereference Introduction

• NULL Pointer Dereference
  ▪ A NULL pointer dereference occurs when the application dereferences a pointer that it expects to be valid, but is NULL, typically causing a crash or exit.
  ▪ NULL pointer dereference issues can occur through a number of flaws, including race conditions, and simple programming omissions.

• NULL Pointer Dereference in Linux Kernel
  ▪ In Ring 0 attacks, e.g., attack to Linux Kernel, if NULL pointer can be mapped to somewhere attacker can control, NULL pointer dereference vulnerability will cause more serious issue like arbitrary code execution.
NULL Pointer Dereference Examples in SGX

- NULL Pointer Dereference in Open-SSL
  - Bad (EC)DHE parameters cause a client crash (CVE-2017-3730) is actually a NULL pointer dereference issue and will impact trusted code.
  - OpenSSL Security Advisory: https://www.openssl.org/news/secadv/20170126.txt
Security Impact Analysis

- Security Analysis beyond DoS Attack
- Enclave Start Address Overwritten Attack
  - Attacker make enclave address starts at 0 and attack to overwrite the beginning portion of the enclave
- Address Remap Attack
  - Control page table to make NULL pointer points to arbitrary data to inject into enclave
- Finally no security issue because of other checks in flowing code in OpenSSL
Race Condition – TOCTOU

- Race condition
  - Pointers across boundary can be replaced
  - Check should before use
    - https://github.com/01org/linux-sgx/pull/135

- Stress-test in multi-thread environment
- Specific code review
Summary – Security Test Points

- Key points of code generation tool in TEE SDK
  - Take SGX as an example:
    - Data across trusted boundary
    - Pointer Check
    - Race Condition
Summary – Security Impact

- Code generation tool in TEE SDK is very import to security
  - Information leakage across boundary
  - Potential overflow and arbitrary code execution
Summary – Validation

- Code scan and code review
  - Klocwork
- Target to generated code instead of the code of tools
  - How to generated various code? Ensure Coverage?
  - Cover the basic types of code, and then complex code
- Fuzz test