FOSDEM 2022

SGX Enclave Exploit Analysis and Considerations for Defensive SGX Programming

Zhang, Shunda; Jiang, Hongyan; Sun, Junli; Yang, Debin

EXECUTIVE SUMMARY

THE EXPLOITS IN THE PRESENTATION ARE KNOWN ISSUES AND HAVE ALREADY BEEN MITIGATED IN SGX

AGENDA

- Purpose
- Threats to SGX Enclave
- Defensive Programing for SGX
- Security impact analysis of specific attacks
  - Heap overflow
  - Stack overflow
  - Attack based on NULL PTR dereference
  - 3rd-party CVE
  - Cross Boundary Attacks

PURPOSE

01

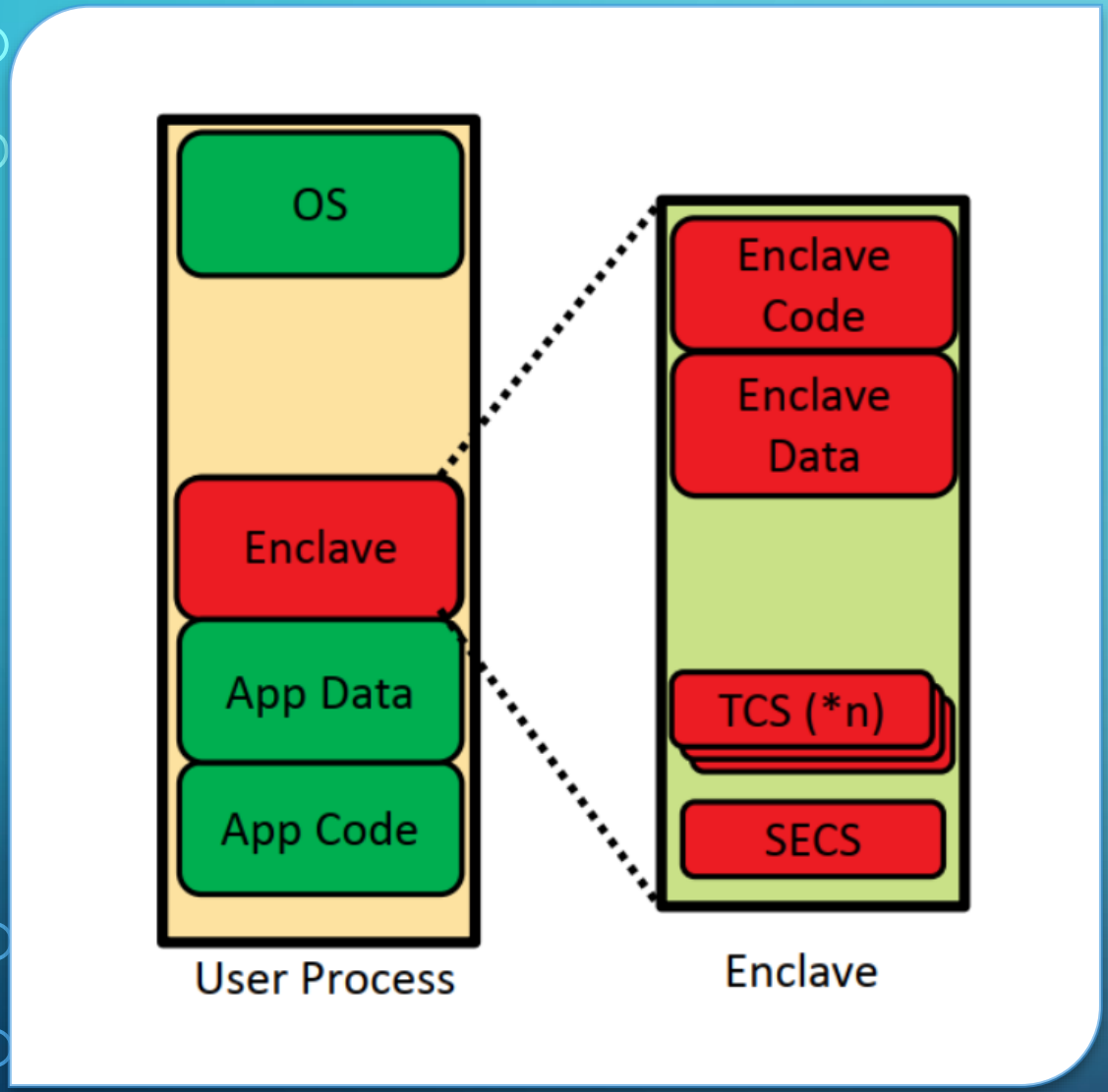
Explain the Security Properties of Intel® SGX Technology

02

Help Enclave Developers write more secure code within an enclave

03

Analyze a sampling of enclave defense-in-depth strategies and protections against known attacks



SGX ENCLAVE SECURITY PROPERTIES

- The security features of SGX include physical memory isolation, enclave measurement, software attestation, and data sealing
- Enclave provides confidentiality, integrity and controlled entry points

SGX ENCLAVE SECURITY AND CHALLENGE

- Intel(R) SGX Technology enables applications to execute code in a trusted environment - an enclave
- Code running within the enclave must be written securely
- Poorly written code may be subject to attack by various methods
- Developers must also be aware of potential side-channel attacks on code

THREATS TO SGX ENCLAVE

Threats to SGX Enclave

Attack Purpose: Steal Secret Information

- Code injection based on stack/heap overflow
- Execution flow control
- 3d-party vulnerabilities
- Cross boundary information leakage

DEFENSE-IN-DEPTH IN SGX

The defense-in-depth in SGX

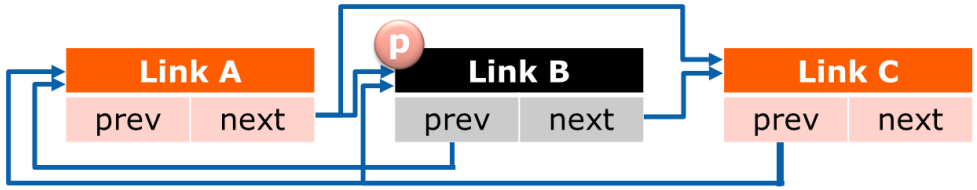
- Stack cookies
- Non-executable stack/heap
- Protection on exception handlers
- GCC virtual table verification
- Control flow hardening
- Safe-unlink
- Heap cookies
- Additional boundary checks inside enclave

Provided custom library/functions to ensure that these features worked inside an enclave

THE HEAP

```
struct link* unlink(struct link* p)
{
     p->prev->next = p->next;
     p->next->prev = p->prev;

     return p;
}
```




```
struct link* safe-unlink(struct link* p)
Add checks for:

p->prev->next == p && p->next->prev == p
```

- Extracting a link from a doubly-linked list (unlinking)

NEW TYPE OF HEAP INJECTION IN SGX

Attack Example

- An outside memory is injected to free chunk chain
- Dlmalloc allocates the outside memory to the trust code and the secret is exploited
- Although the magic number is incorrectly changed, the dlmalloc code cannot know it until the overflowed buffer be freed. In that time, attack has finished

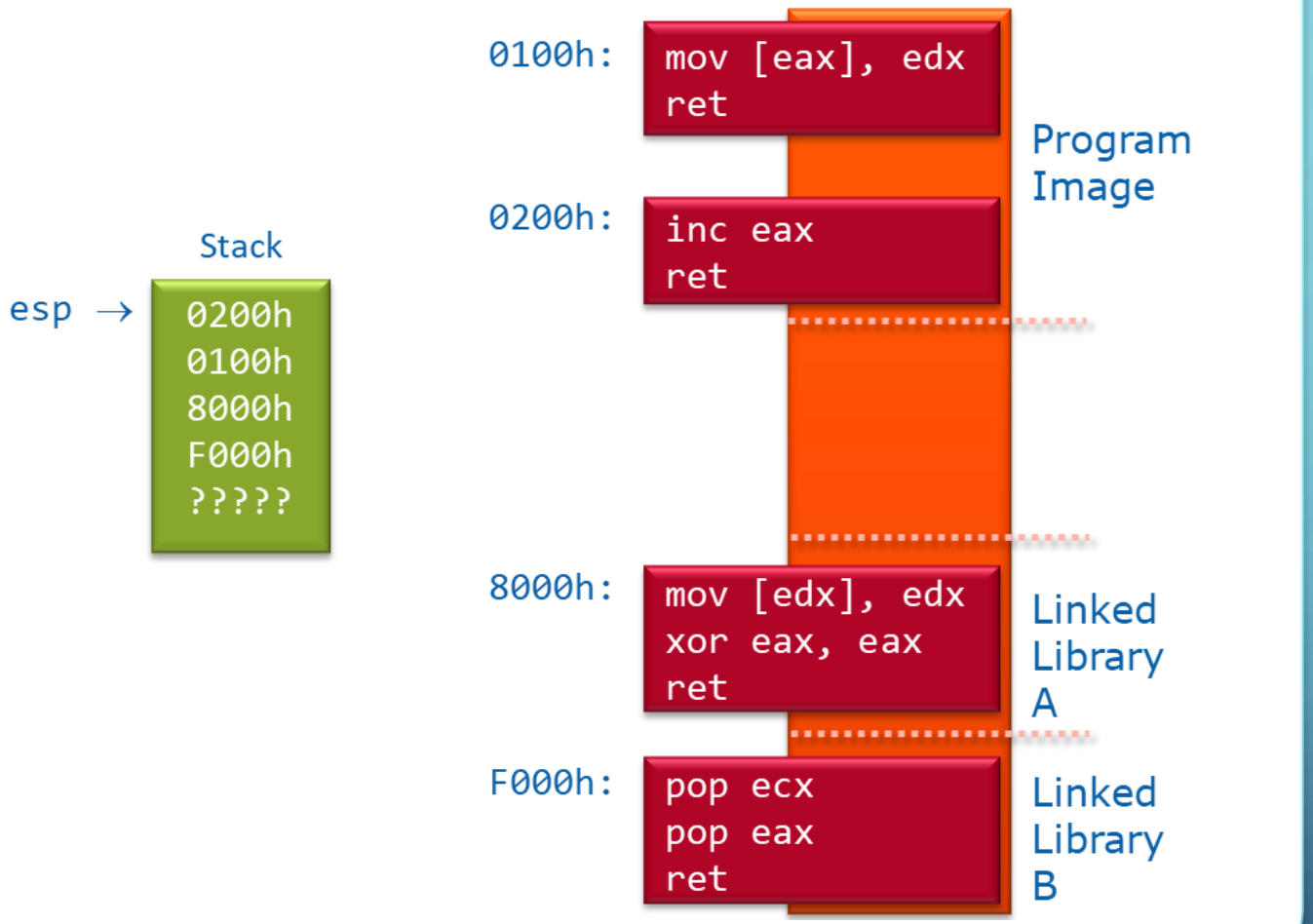
Mitigation in SGX SDK

- Check memory boundary before returning the allocated buffer to verify that memory is from the enclave heap

PROTECTION AGAINST HEAP OVERFLOW

- Background
  - Ported Dlmalloc to trusted library
  - Random magic number and safe unlink for heap overflow protection, but the checks in buffer free
- Major impacts
  - In enclave environment, the detection and protection in free() is late, attacker can copy out secret before the checks in free() in some cases
  - Enclave secret can be copied out in some cases
- Solution to the attack
  - Mitigation in SGX SDK (boundary checks in unlinking/malloc/free)

ROP


Stack
esp →
0200h
0100h
8000h
F000h
?????
0100h:
mov [eax], edx
ret
0200h:
inc eax
ret
Program Image
8000h:
mov [edx], edx
xor eax, eax
ret
Linked Library A
F000h:
pop ecx
pop eax
ret
Linked Library B

BUFFER OVERFLOW PREVENTION

Background

- Buffer overflows can use parts of your code as ROP gadgets

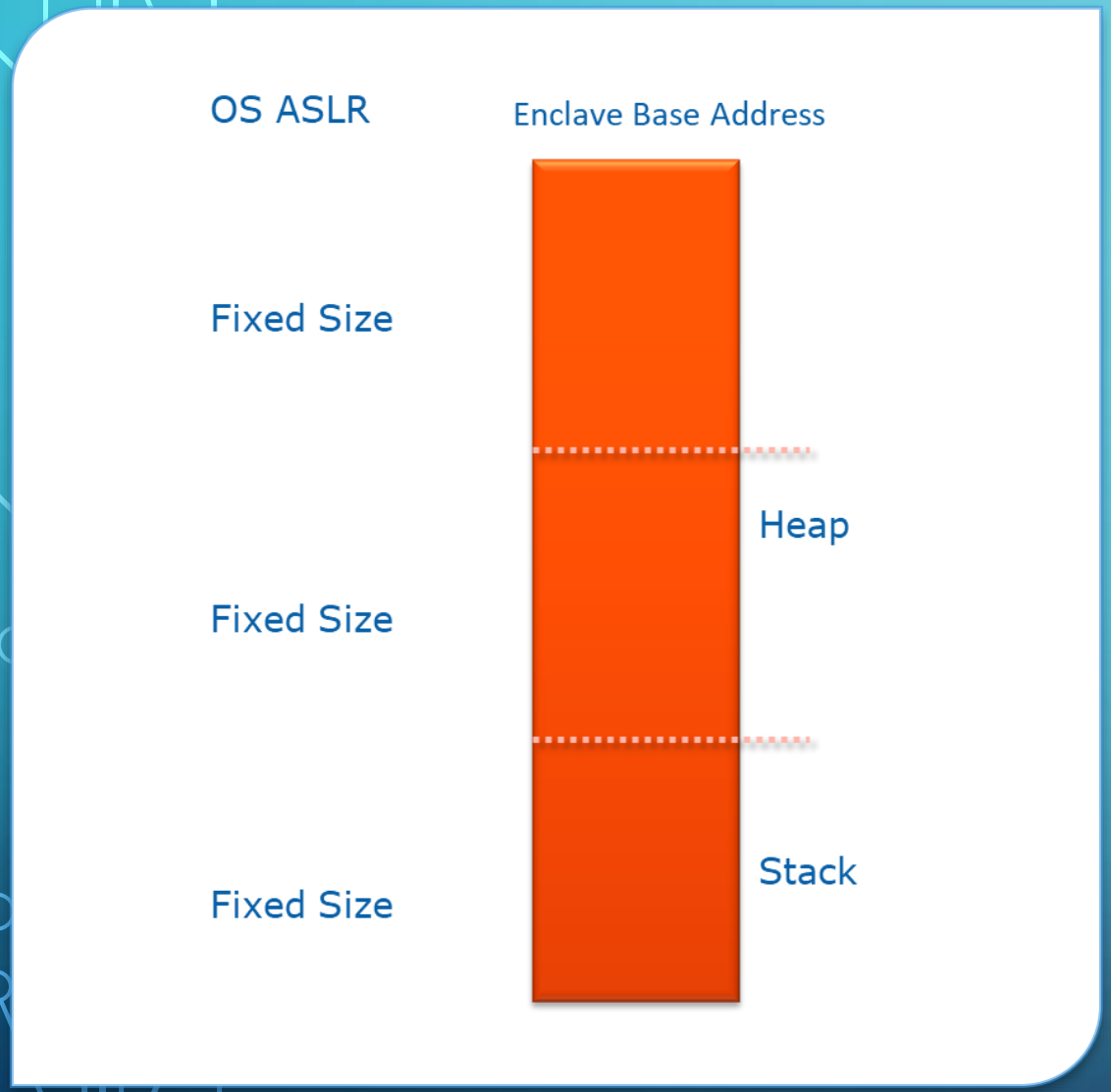
Major impacts

- If ISV Enclave code has an overflow bug, attacker can do ROP and may leverage the gadgets to jump to memcpy to copy data outside enclave, or switch the esp into untrusted part
- Stack Cookies catch stack overflows prior to return from a procedure

Solution to the attack

- The best defense in SGX is to prevent overflows, ensure Buffer Overflow Prevention
  - Make sure stack cookies is enabled
  - Make sure ASLR is enabled

ENCLAVE MEMORY



LIMITED ASLR PROTECTION AGAINST BUFFER OVERFLOW

Background

- ASLR is not supported in trust part, enclave base address changed, but not fully randomized and memory layout is fixed

Major impacts

- ASLR in Enclave cannot fully protect against memory location prediction for enclave
- Attacker may predict address location of Enclave, that increases the success rate of buffer overflow attack

Solution to the attack

- Depends on Enclave Developer

Suggestion to Developer

- Do not rely too much on ASLR, defense-in-depth for buffer overflow is important
- Could refer to 3rd party randomization enhancement projects if needed
  - e.g. Sgx-shield: https://github.com/jaebaek/SGX-Shield

NULL PTR DEREFERENCE

Background

- 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

Major impacts

- Control page table to make NULL pointer points to arbitrary data to inject into enclave
- Enclave start address overwritten
- NULL pointer remap, may cause stack/heap overflow, code injection, variable overwritten and impact logic

Solution to the attack

- SDK has added mitigation that allow enclave include 0 address inside enclave (hardware prevented inside address remapping)
- Suggest Enclave Developer to add appropriate checks

NULL PTR DEREFERENCE

Attack Example

- Bad (EC)DHE parameters cause a client crash (CVE-2017-3730) is a NULL pointer dereference issue and will impact trusted code.
- OpenSSL Security Advisory: https://www.openssl.org/news/secadv/20170126.txt
- CVE-2017-3730 is only Moderate in Open-SSL security advisory, but can be a severe issue for SGX

Suggestion to Developer

- Check NULL or abnormal parameters: if (NULL == ptr) abort(); //error handling

3RD-PARTY CVE

Background

- Enclave can be linked with libraries, such as OpenSSL/IPP, which will contain CVE

Major impacts

- This 3rd-party CVE can cause be leveraged by attackers

Solution to the attack

- Upgrade the 3rd-party libraries to the latest secure version
- Use tools like BDBA (Black Duck Binary Analysis) to help find CVE in 3rd-party libraries

Cross Boundary Information Leakage

ALIGNMENT AND PADDING INTRODUCTION

```
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
```

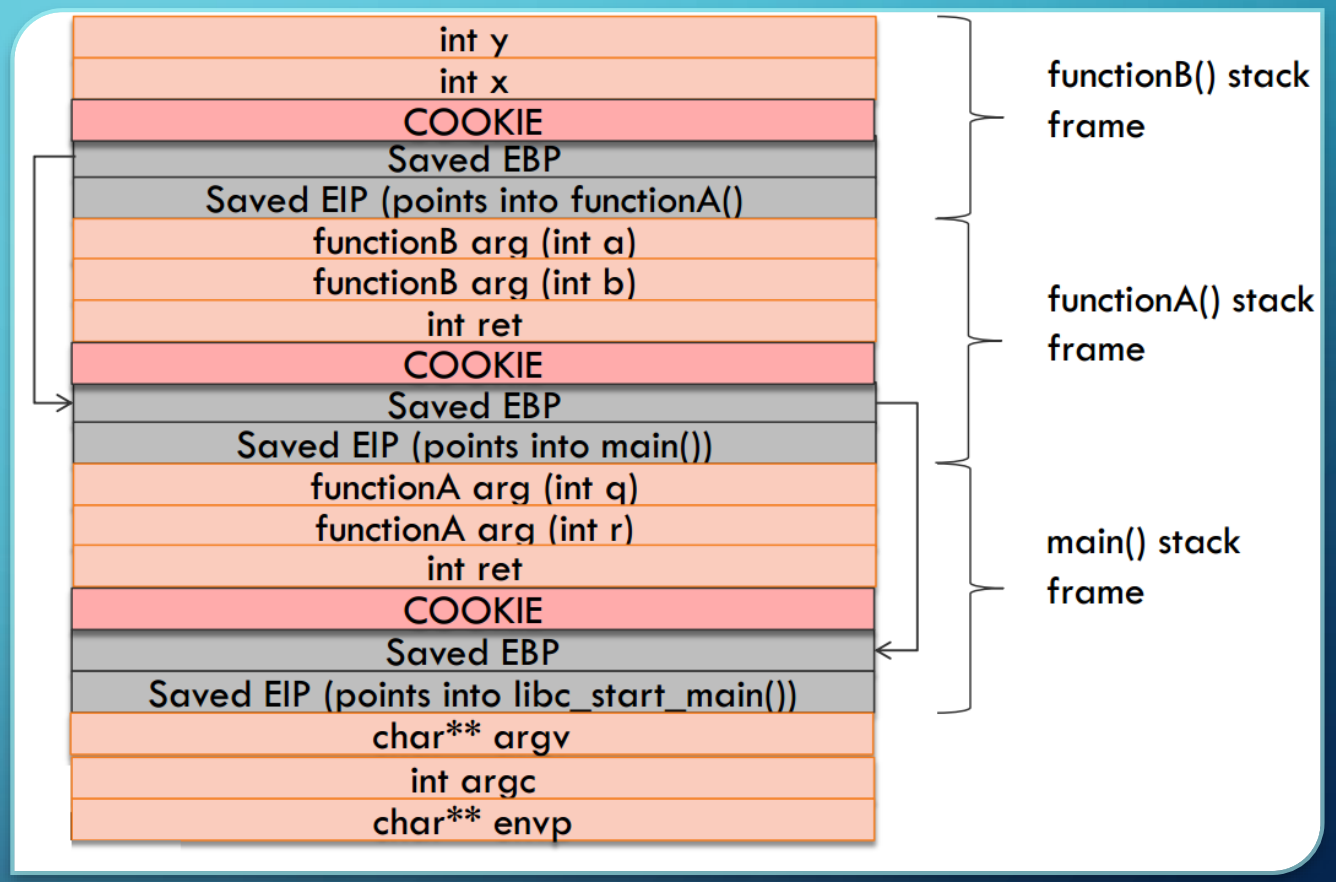
```
int main(int argc, char* argv[])
{
    printf("sizeof int is %d\n", sizeof(int));
    printf("sizeof short is %d\n", sizeof(short));
    printf("sizeof double is %d\n", sizeof(double));
    printf("sizeof long is %d\n", sizeof(long));

    printf("sizeof test is %d\n", sizeof(struct test));
    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 test is 24
sizeof test2 is 4
sizeof test3 is 6
sizeof test4 is 3
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

RACE CONDITION – TOCTOU

- Race condition
  - Pointers across boundary can be replaced
  - Check should before use: github.com/01org/linux-sgx/pull/135
- Mitigation
  - Stress-test in multi-thread environment
  - Specific code review

SUMMARY

SGX provided a mechanism to better isolate user-level software from attackers, but it does not mean 100% secure

If ISV Enclave has traditional vulnerabilities like buffer overflow in their code, it may still make trouble

Enclave developers should pay attention to these attacks other than directly crypto-analysis to Enclave

Q&A