COIN Attacks: On Insecurity of Enclave Untrusted Interfaces in SGX

Mustakimur Khandaker[✦], Yueqiang Cheng[↕], Zhi Wang[✦], Tao Wei[↕]

Florida State University[✦], Baidu Security[↕]

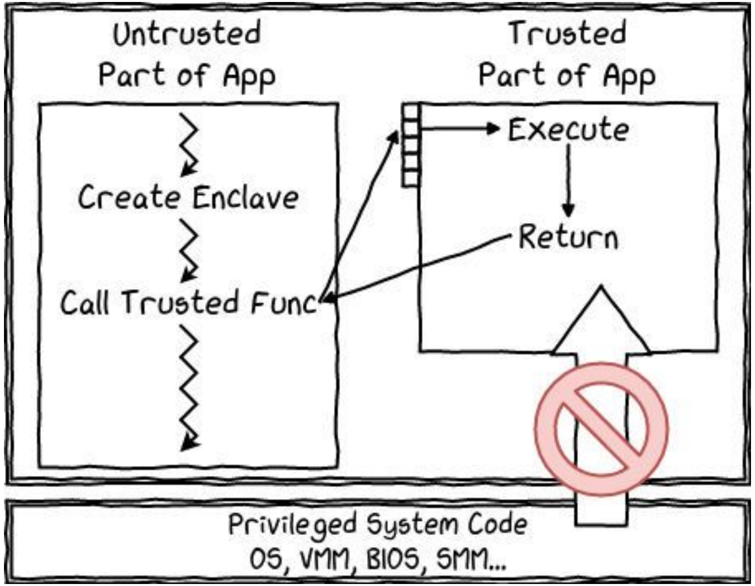
Background: Intel SGX

Intel Software Guard Extension:

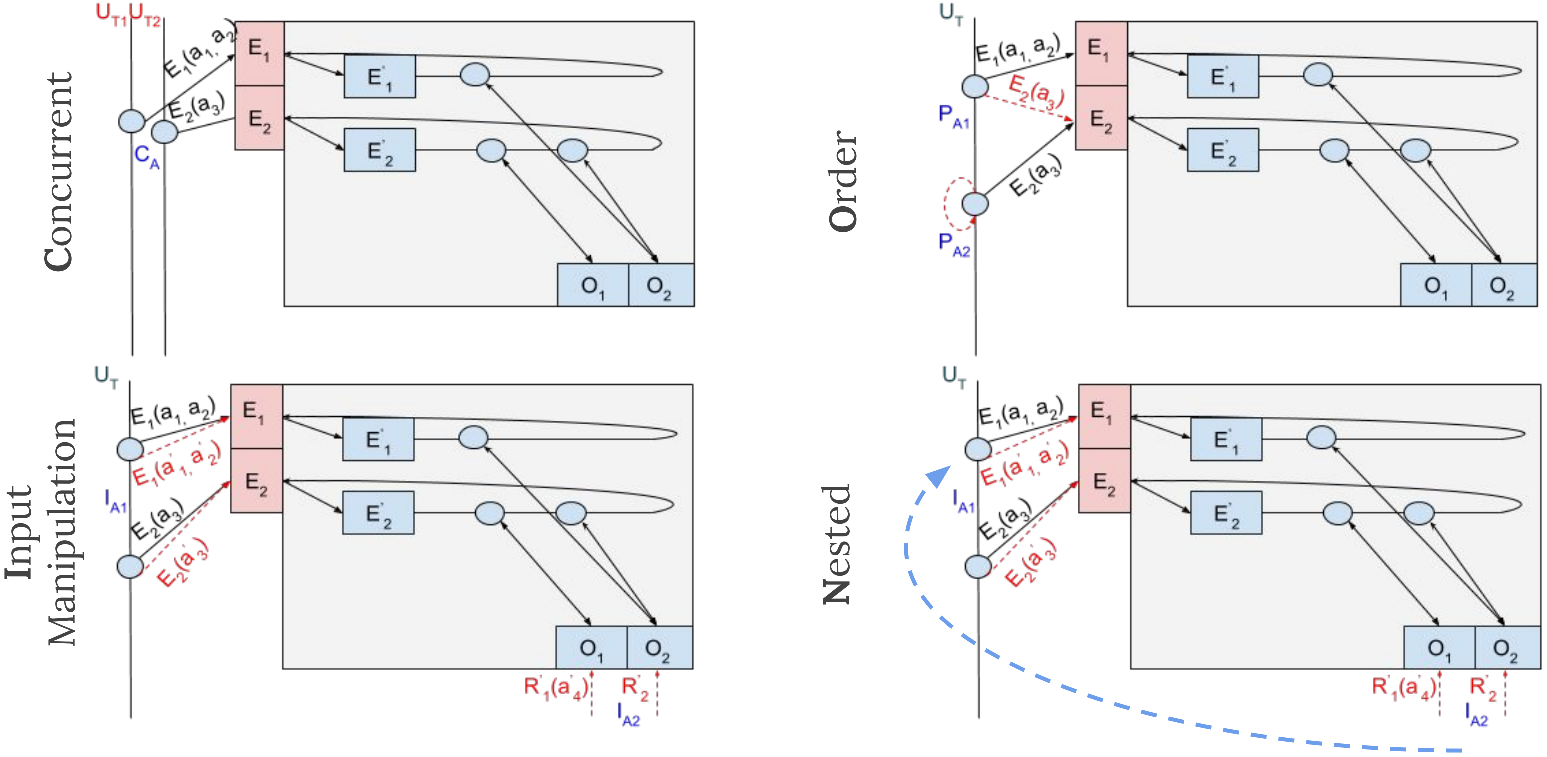
A hardware-support for Trusted Execution Environment (TEE).

A TEE is an isolated execution environment (enclave) provides:

→ isolated execution.
→ integrity of enclave.
→ confidentiality of enclave data.



COIN Attacks: A Comprehensive Threat Model



Background: Enclave Definition Language (EDL)

```
enclave {
  include "../ocall_types.h"
  from "sgx_tstdc.edl" import *;

  trusted {
    public void ecall_opendb([in, string] const char *dbname);
    public void ecall_execute_sql([in, string] const char *sql);
    public void ecall_closedb(void);
  };

  untrusted {
    int ocall_stat([in, string] const char *path,
             [in, out, size=size] struct stat *buf, size_t size);
    int ocall_ftruncate(int fd, off_t length);
    int ocall_getpid(void);
    char* ocall_getenv([in, string] const char *name);
  };
};
```

Data-flow Direction

List of ECALL

List of OCALL

Variable Type

Extensible Framework for COIN Attacks

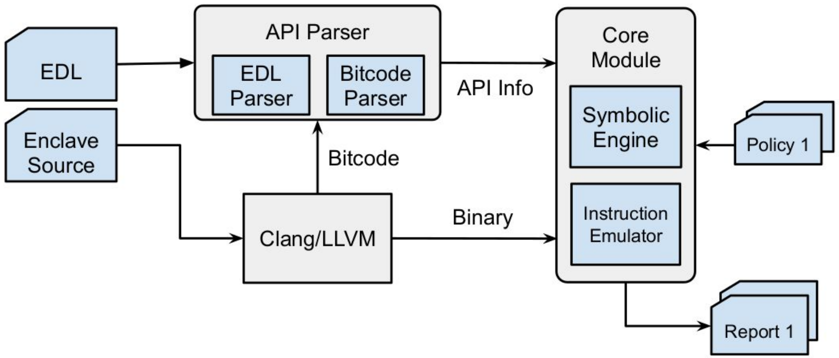


Fig: Overview of the enclave analysis framework

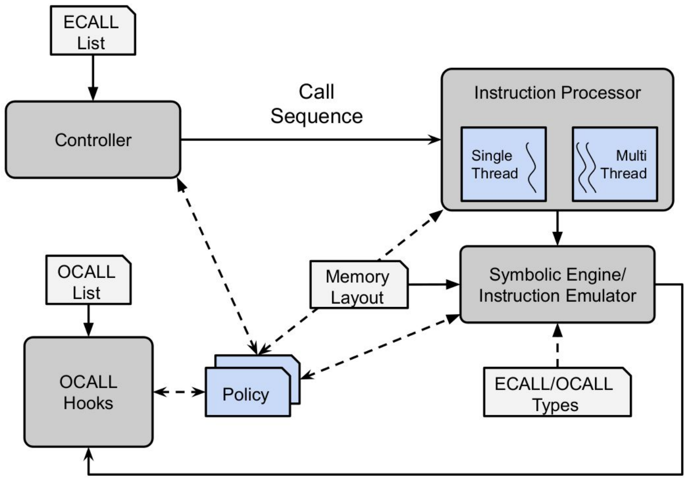


Fig: Core module architecture

Framework Continued ...

```
[EMULATION] attempted sequence: ('ecall_create', 'ecall_use', 'ecall_destroy', 'ecall_create', 'ecall_destroy', 'ecall_use')
[UAF-REPORT] Potential Use-after-free (UAF) at 0xd2e: mov ecx, dword ptr [rax]
Try to use memory at 0x30000064 - 0x30000067
Allocated memory range is 0x30000064 - 0x30000070
Allocated memory at 0xcc6 and Freed at 0xd7a

Recent 200 emulated instructions:
0xaace: mov rax, qword ptr [rbp - 8]
0xaad2: mov rdi, rax
0xaad5: call 0x12fa0
0x12fa0: push rsi
0x12fa1: mov rdx, rdi
...
0xd18: mov dword ptr [rbp - 4], edi
0xd1b: cmp qword ptr [rip + 0x2256fd], 0
0xd23: jne 0xd27
0xd27: mov rax, qword ptr [rip + 0x2256f2]
0xd2e: mov ecx, dword ptr [rax]
Seed information:
0x30000000 [ 0x55 ] 0x30000001 [ 0x41 ] 0x30000002 [ 0x46 ] 0x30000003 [ 0xff ]
```

Fig: Sample report for use-after-free

Implemented Policies

Information Leak	Memory Vulnerabilities	Control-flow Hijack
Stack information leak	Use after free	Ineffectual Condition
Heap information leak	Double free	
	Stack overflow	
	Heap overflow	
	Null pointer dereference	

https://github.com/bl4ck5un/mbedtls-SGX/issues/16

Policy: Heap Information Leak

1. The core module triggers an event to notify the policy module about an infinite loop it encounters.
2. The policy then checks whether the loop condition is symbolic.
3. If the loop condition is symbolic, the policy extracts the loop body and analyzes whether it contains an OCALL or not.
4. If there is an OCALL, the policy uses the definition of the OCALL to identify memory buffers in the parameters.
5. The policy reports a potential heap memory leak if a pointer points to the enclave heap and can be modified in every iteration of the loop.

```
int
mbedtls_ssl_flush_output(mbedtls_ssl_context *ssl){
  ...
  while(ssl->out_left > 0){  // size_t type
    buf = ssl->out_hdr + mbedtls_ssl_hdr_len(ssl) +
                        ssl->out_msglen - ssl->out_left;

    //an indirect call to OCALL
    ret = ssl->f_send(ssl->p_bio,
                             buf, ssl->out_left);

    if(ret <= 0)             // ret > ssl->out_left
        return(ret);

    ssl->out_left -= ret;   // integer overflow
  }
  ...
}
```

https://github.com/yerzhan7/SGX_SQLite/issues/2
https://github.com/intel/linux-sgx/issues/456

Policy: Use-after-free

In SGX, access to freed memory can cause an enclave to crash, use unexpected values, or even execute arbitrary code.

1. If a free function is called, the policy requests the core module to pause the associated thread until other threads have completed N instructions.
2. If a memory dereference event is triggered, the policy validates the respected memory against the memory status and raises an alert if the memory has been freed.

```
sqlite3* db; // database object

int sqlite3SafetyCheckOk(sqlite3 *db){
  u32 magic;
  if( db==0 ){
   return;
  }
  magic = db->magic; // use
}
void sqlite3_close(sqlite3 *db){
  if( sqlite3GlobalConfig.bMemstat){
    sqlite3_mutex_enter(mem0.mutex);
    sqlite3GlobalConfig.m.xFree(db);
    sqlite3_mutex_leave(mem0.mutex);
  }
}
void ecall_opendb(const char *dbname){
  rc = sqlite3_open(dbname, &db);
}
void ecall_execute_sql(const char *sql){
  rc = sqlite3_exec(db, sql, callback, 0, &zErrMsg);
}
void ecall_closedb(){
  sqlite3_close(db);
  // forget to set db = 0
}
```

https://github.com/kudelskisecurity/sgx-reencrypt/issues/1

Policy: Ineffectual Condition

A conditional check in the enclave becomes ineffectual if the attacker can control its outcome. Therefore, an ineffectual condition would allow attackers to bypass validation, avoid authentication, etc.

1. An ineffectual condition is identified if both sides of the condition contain symbolic variables or if one side contains symbolic variables and the other side is a constant.
2. It further checks whether the conditional check is followed by an error code generator basic block with an unconditional control transfer.

```
int
reencrypt(client_id *clid, uint8_t *request,
          size_t requestlen, uint8_t *response,
                         size_t *responselen) {
  ...
   // keyin originates from the unsafe ECALL param clid
  if((ret = check_policy(&keyin, &keyout, *clid,
                         keyIDin, keyIDout))
                              != REENCRYPT_OK) {
     ...
  }
  // OCALL to get (unsafe) timestamp
  if(ret = unsafe_timestamp(&timestamp)
                             != REENCRYPT_OK){
     ...
  }
  // both sides of the conditional statement
  // contain symbolic variables
  if (timestamp > keyin->expiration_date ||
    timestamp > keyout->expiration_date) {
    ret = REENCRYPT_KEY_EXPIRED;
    goto err;
  }

  if ((ret = decrypt(&m, &mlen,c,clen,
                  keyin)) != REENCRYPT_OK) {
     ...
  }
}
```

Policy: Null Pointer Dereference

Check if a dereferenced pointer is null?

Common Scenario:

- ➔ ECALL output param receives null pointer from unsafe application.
- ➔ API code declares a counter null pointer.
- ➔ Enclave code uses memcpy() to copy enclave data to the null pointer.

```
static sgx_status_t SGX_CDECL sgx_sgxEncryptFile(void* pms){
  ...
  // _tmp_encMessageOut could be NULL, results in
  // _in_encMessageOut to be NULL
  if (_tmp_encMessageOut != NULL && _len_encMessageOut != 0) {
    if ((_in_encMessageOut = (unsigned char*)
                 malloc(_len_encMessageOut)) == NULL) {
    }
    ...
  }
  sgxEncryptFile(_in_decMessageIn, _tmp_len,
                   _in_encMessageOut, _tmp_lenOut);

  if (_in_encMessageOut) {
    if (memcpy_s(_tmp_encMessageOut, _len_encMessageOut,
               _in_encMessageOut, _len_encMessageOut)) {
    }
  }
}

void sgxEncryptFile(unsigned char *decMessageIn, size_t len,
                    unsigned char *encMessageOut, size_t lenOut){
  uint8_t p_dst[lenOut];
  ...
  // encMessageOut should be checked for NULL
  memcpy(encMessageOut, p_dst, lenOut);
}
```

Evaluation

Project	Description	Enclave LoC	# of Bugs
mbedtls-SGX [11]	Crypto and SSL/TLS support for embedded systems.	59,228	11
SGX-Tor [17]	Tor anonymity network.	316,962	9
TaLoS [21]	Secure TLS termination.	183,958	7
Bolos-enclave [22]	Trusted environment for blockchain applications.	8,463	6
Intel-SGX-SSL [15]	SSL cryptographic library from Intel.	6,508	5
SGX_SQLite3 [43]	Secure SQLite query.	118,997	4
SGX-Migration [34]	Live migration VMs.	2,829	3
SGX-Wallet [1]	Trusted password-wallet.	252	3
SGX-Reencrypt [19]	Symmetric reencryption.	1,772	2
SGXCryptoFile [32]	Encrypting and decrypting HLS chunks.	157	2

Evaluation Continued ...

Project	Heap info leak	Stack info leak	Ineffectual condition	UAF	Double-free	Stack overflow	Heap overflow	Null ptr deref	Total
mbedtls-SGX	2	3				3	1	2	11
SGX-Tor			2			2	1	4	9
TaLoS	1	1	1	2		1		1	7
Bolos-enclave		1	1		1		1	2	6
Intel-SGX-SSL			3			1		1	5
SGX_SQLite				4					4
SGX-Migration				2	1				3
SGX-Wallet			1		2				3
SGX-Reencrypt			1					1	2
SGXCryptoFile								2	2
Total	3	5	9	8	4	7	3	13	52

Evaluation Continued ...

Policy	Input Manipulation	Call Permutation	Concurrent Calls
Heap info leak	3		
Stack info leak	5		
Ineffectual condition	9		
Use after free		5	3
Double free	1	1	2
Stack overflow	7		
Heap overflow	3		
Null ptr deref	13		
Total	41	6	5

Evaluation: Performance

Configuration:

Machine: Intel Core i7, 32 GB memory.
OS: Ubuntu 18.04 LTS Server.
SDK: Intel SGX SDK (v2.5).
Compiler: Clang/LLVM (v9.0).
Symbolic Engine: Triton.
Emulator: QEMU.

Limitation:

- Instruction not recognized e.g. endbr64 from Intel CET (updated QEMU).
- ISA too complex e.g. Intel AES-NI (complicated to handle by symbolic engine).
- Nested calls (future work).

Runtime:

- Allocated 30 hrs for each project.
- Small projects e.g. SGX-Wallet finished within 4 hrs.
- Multi-thread mode of emulation is 6.5x higher overhead than single-thread mode of emulation.

Conclusion

- We introduced the COIN attacks, a systematic analysis of the SGX interface attack surface. It consists of **c**oncurrency, **o**rder, **i**nput, and **n**ested call attacks.
- We proposed the design of an extensible framework targeting the COIN attacks.
- We implemented the design with 8 detection policies that cover many common vulnerabilities.
- We evaluated our system with 10 open-source SGX projects and found (and reported) 52 vulnerabilities, including a whole SGX memory leak.

https://github.com/mustakcsecuet/COIN-Attacks