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

Concurrent

Input Manipulation

Order

Nested
Background: Enclave Definition Language (EDL)

List of ECALL

List of OCALL

Data-flow Direction

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

<table>
<thead>
<tr>
<th>Information Leak</th>
<th>Memory Vulnerabilities</th>
<th>Control-flow Hijack</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stack information leak</td>
<td>Use after free</td>
<td>Ineffectual Condition</td>
</tr>
<tr>
<td>Heap information leak</td>
<td>Double free</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Stack overflow</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Heap overflow</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Null pointer dereference</td>
<td></td>
</tr>
</tbody>
</table>
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.

```c
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
    }
    ...
}
```
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.

```c
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(memo0.mutex);
        sqlite3GlobalConfig.m.xFree(db);
        sqlite3_mutex_leave(memo0.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
}
```
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.
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.
## Evaluation

<table>
<thead>
<tr>
<th>Project</th>
<th>Description</th>
<th>Enclave LoC</th>
<th># of Bugs</th>
</tr>
</thead>
<tbody>
<tr>
<td>mbedtls-SGX [11]</td>
<td>Crypto and SSL/TLS support for embedded systems.</td>
<td>59,228</td>
<td>11</td>
</tr>
<tr>
<td>SGX-Tor [17]</td>
<td>Tor anonymity network.</td>
<td>316,962</td>
<td>9</td>
</tr>
<tr>
<td>Bolos-enclave [22]</td>
<td>Trusted environment for blockchain applications.</td>
<td>8,463</td>
<td>6</td>
</tr>
<tr>
<td>Intel-SGX-SSL [15]</td>
<td>SSL cryptographic library from Intel.</td>
<td>6,508</td>
<td>5</td>
</tr>
<tr>
<td>SGX_SQLite3 [43]</td>
<td>Secure SQLite query.</td>
<td>118,997</td>
<td>4</td>
</tr>
<tr>
<td>SGX-Migration [34]</td>
<td>Live migration VMs.</td>
<td>2,829</td>
<td>3</td>
</tr>
<tr>
<td>SGX-Reencrypt [19]</td>
<td>Symmetric reencryption.</td>
<td>1,772</td>
<td>2</td>
</tr>
<tr>
<td>SGXCryptoFile [32]</td>
<td>Encrypting and decrypting HLS chunks.</td>
<td>157</td>
<td>2</td>
</tr>
</tbody>
</table>
# Evaluation Continued...

<table>
<thead>
<tr>
<th>Project</th>
<th>Heap info leak</th>
<th>Stack info leak</th>
<th>Ineffectual condition</th>
<th>UAF</th>
<th>Double-free</th>
<th>Stack overflow</th>
<th>Heap overflow</th>
<th>Null ptr deref</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>mbedtls-SGX</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>11</td>
</tr>
<tr>
<td>SGX-Tor</td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td>1</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>TaLoS</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>Bolos-enclave</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>Intel-SGX-SSL</td>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>SGX_SQLite</td>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>SGX-Migration</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>SGX-Wallet</td>
<td>1</td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>SGX-Reencrypt</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>SGXCryptoFile</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>3</td>
<td>5</td>
<td>9</td>
<td>8</td>
<td>4</td>
<td>7</td>
<td>3</td>
<td>13</td>
<td>52</td>
</tr>
</tbody>
</table>
## Evaluation Continued ...

<table>
<thead>
<tr>
<th>Policy</th>
<th>Input Manipulation</th>
<th>Call Permutation</th>
<th>Concurrent Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heap info leak</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stack info leak</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ineffectual condition</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Use after free</td>
<td></td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>Double free</td>
<td></td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Stack overflow</td>
<td></td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>Heap overflow</td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Null ptr deref</td>
<td>(13)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>41</td>
<td>6</td>
<td>5</td>
</tr>
</tbody>
</table>
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 concurrency, order, input, and nested 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