TeeREx: Discovery and Exploitation of Memory Corruption Vulnerabilities in SGX Enclaves

Tobias Cloosters, Michael Rodler, Lucas Davi

Secure Software Systems
University of Duisburg-Essen, Germany
Find memory corruption vulnerabilities specific to SGX enclaves

Successfully exploited:
- Code from Intel, Baidu/Apache, WolfSSL
- Fingerprint Drivers
  - Synaptics (Lenovo/HP): CVE-2019-18619
  - Goodix (Dell): CVE-2020-11667
Motivation: Why SGX?

• How to reliably protect sensitive data and code from disclosure and modification?

Passwords  Intellectual Property  Medical records
System Model of SGX

- Host application
- Enclave
  - Critical point
  - Secure storage for secrets
  - Processing untrusted data and secrets
  - Trust Boundary
  - Untrusted data
  - Results
SGX – Application Layout

Application CFG

Memory

Secure Memory
SGX – Trusted Runtime

SGX Enclave

init
INT
ORET
EEXIT

EENTER

ECALL

Memory

Secure Memory
SGX – ECALL

*args

secure copy of *args

ECALL logic

copy output
Large attack surface

- Trust input data: exploitable
- Trust system calls: exploitable
- Use NULL-pointer: exploitable
- One corruptible byte: exploitable
- Trust pointers to enclave memory: exploitable
Symbolic Execution
Vulnerability Detection

• Controlled Jump
• Controlled Memory Access
• NULL-pointer Dereference
TeeRex Architecture

Preprocessor (Static Analysis)
- Identify ECALLs
- Symbolic Hooks for common Functions

TEE Rex Symbolic Execution
- Emulation of Special Instructions
- Enclave Loader
- Symbolic Explorer (ANGR)
- Vulnerability Detection
  - Controlled Jumps
  - Controlled Write
  - NULL dereference
- Pointer Tracking

Vulnerability Report
- Vulnerability Class
- Vuln. Instruction
- Controlled Pointer
- Symbolic Execution Trace

Exploit
Analyst
Exploits in Public Enclaves found with TEERX

<table>
<thead>
<tr>
<th>Project</th>
<th>Exploit</th>
<th>Fixed</th>
<th>Source Code</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel SGX GMP Example</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Linux amd64</td>
</tr>
<tr>
<td>Baidu Rust SGX SDK “tlsclient”</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Linux amd64</td>
</tr>
<tr>
<td>TaLoS</td>
<td>✓</td>
<td>Not planned</td>
<td>✓</td>
<td>Linux amd64</td>
</tr>
<tr>
<td>WolfSSL Example Enclave</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Linux amd64</td>
</tr>
<tr>
<td>Synaptics Fingerprint Driver</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>Windows amd64</td>
</tr>
<tr>
<td>Goodix Fingerprint Driver</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>Windows amd64</td>
</tr>
<tr>
<td>SignalApp Contact Discovery</td>
<td>✗</td>
<td>-</td>
<td>✓</td>
<td>Linux amd64</td>
</tr>
</tbody>
</table>

Exploit Source Code: [https://github.com/uni-due-syssec/teerex-exploits](https://github.com/uni-due-syssec/teerex-exploits)
Baidu/Apache Rust SDK: tlsclient
Pointers to overlapping memory

APP

Enclave

Enclave Code

if sgx_is_outside_enclave(ssl)
  return ERROR;

// use ssl session

ORET Primitive
Arbitrary code execution

SSL object is not strictly outside
vtable pointer in object (outside enclave)

Memory

Enclave Memory

SSL

vptr

SSL

SSL
void arbitrary_write(int* a, int b)
{
    *a = b;
}

void fixed_address(int b)
{
    global_mem = b;
}

void limited_value(int* a)
{
    OR: *a = 42;
    *a = global_mem;
}

void limited_size(int* a, char b)
{
    *(char*)a = b;
}

void no_user_input()
{
    // global_mem = 42;
}

int global_mem = 0;
int* global_addr;

void limited_address(int b)
{
    *global_addr = b;
}

// Controlled Address
void limited_address(int b)
{
    *global_addr = b;
}

// Controlled Value
void fixed_address(int b)
{
    global_mem = b;
}

void limited_size(int* a, char b)
{
    *global_addr = b;
}
Exploiting using Multiple Limited Primitives:

- `ecall_process` trusts `data`
  - In secure memory
  - Never leaves enclave
  - NULL checked

- Attacker can corrupt `data`
  - `ecall_vuln` writes a small constant to an unchecked address
  - Changing one byte moves `data` to unsecure memory

- Combined the attacker can execute arbitrary code

```java
void ecall_init() {
    data = new Obj();
}

void ecall_process(int arg) {
    if (data)
        data->foo(arg);
}

void ecall_vuln(Bar* p) {
    if (...)
        p->return_value = ERROR;
}
```

- Corrupted to point outside enclave memory
- Enclave loads code pointer from host-memory
- Store byte `ERROR = -5` at `return_value`
Conclusions

• Enclave boundary is a highly critical attack surface
• Current development practices do not consider such vulnerabilities
  • Increase awareness
  • Automatic analysis tools needed
• TeeRex can automatically detect vulnerabilities!
Tobias Cloosters, Michael Rodler, Lucas Davi
tobias.cloosters@uni-due.de, michael.rodler@uni-due.de, lucas.davi@uni-due.de

TEEReX: Discovery and Exploitation of Memory Corruption Vulnerabilities in SGX Enclaves

Exploit Source Code: https://github.com/uni-due-syssec/teerex-exploits