Memory Corruption Attacks Against Intel SGX Shielded Software

Lucas Davi
Secure Software Systems
University of Duisburg-Essen, Germany
Motivation

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

Passwords  Intellectual Property  Medical records
Moving to Cloud Computing

Customer Data
New Security Technology: Intel Software Guard Extensions (SGX)
[McKee et al., Hoekstra et al., Anati et al., HASP’13]
Overview on Intel SGX

APP
- App Code
- App Data

Operating System

APP
- App Code
- App Data

Hardware

CPU
Overview on Intel SGX
Overview on Intel SGX

APP
- App Code
- App Data
- Enclave

Malware
- App Code
- App Data

APP
- App Code
- App Data
- Enclave

Operating System

Hardware

CPU

SGX

Bug
Entry to Enclave code is only allowed at pre-defined entry points.
Academic Research on Side-Channel Attacks Against SGX

Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems

Yuanyong Xu
The University of Texas at Austin
yuanyong@utexas.edu

Weikang Cai
Microsoft Research
wcai@microsoft.com

Marcos Peinado
Microsoft Research
marcospe@microsoft.com

CacheZoom: How SGX Amplifies
The Power of Cache Attacks

Ahmad Mohimini, Gorka Irazoqui, and Thomas Eisenbarth

Telling Your Secrets Without Page Faults: Stealthy Page Table-Based Attacks on Enclaved Execution

Jo Van Bulcke, imec-DistriNet, KU Leuven; Nico Weichbrodt and Rüdiger Kapitza, IBR DS, TU Braunschweig; Frank Piessens and Raoul Strackx, imec-DistriNet, KU Leuven

Boffins show Intel's SGX can leak crypto keys
Software Guard Extensions are supposed to hide data. But the 'Prime+Probe attack' fixes that

Ferdinand Brasser¹, Ute Müller², Alexandra Braunstein², Karl Kosianer³, Sriraj Capkun², and Ahmad-Reza Sadeghi³

Inferring Fine-grained Control Flow Inside SGX Enclaves with Branch Shadowing

Sangho Lee, Ming-Wei Shih, Pranav Gera, Tatsuya Kim, and Hyunsoo Kim,
Georgia Institute of Technology; Marcus Pehnado, Microsoft Research

SGXPECTRE Attacks: Stealing Intel Secrets from SGX Enclaves via Speculative Execution

Guoxing Chen, Sanchuan Chen, Yuan Xiao, Yingjian Zhang, Zhigang Liu, Ten H. Lai
Department of Computer Science and Engineering
Academic Research on Side-Channel Attacks Against SGX

See next talk

Hardware-Assisted Security: From Trust Anchor to Meltdown Trust
Ahmad-Reza Sadeghi
What about Memory Corruption Attacks?
Memory Corruption Attack Classification

Code-Injection Attack

Code-Reuse Attack

Adversary

DEP

inject malicious code

corrupt code pointer

Data flow

Program flow
Return-Oriented Programming
Return-Oriented Programming Attack

**Program Stack**
- Return Address 1: 0x80102030
- Return Address 2: 0xAABBCCDD
- Return Address 3: 0x80102030

**Program Code**
- Sequence 1:
  - x86_ins
  - ret
- Sequence 2:
  - pop eax
  - pop ebx
  - ret
- Sequence 3:
  - x86_ins
  - ret

**Corrupt Control Structures**
- ESP
- EAX:
- EBX:
Return-Oriented Programming Attack

Program Stack

Return Address 3
0xAABBCCDD
0x80102030

Return Address 2
Return Address 1

ESP

Program Code

Sequence 1
x86_ins
ret

Sequence 2
pop eax
pop ebx
ret

Sequence 3
x86_ins
ret

EAX:
EBX:
Return-Oriented Programming Attack

Program Stack
- Return Address 3: 0xAABBCCDD
- Return Address 2: 0x80102030
- Return Address 1

Program Code
- Sequence 1:
  - x86_ins
  - ret
- Sequence 2:
  - pop eax
  - pop ebx
  - ret
- Sequence 3:
  - x86_ins
  - ret

ESP →
EAX: 0x80102030
EBX: 0xAABBCCDD
First Run-Time Attacks and Defenses
Targeting Intel SGX
Related Work

**Dark ROP** [USENIX Sec. 2017]
- Analyzes the threat of memory corruption vulnerabilities in the context of SGX
- Presents ROP attack against (unknown) encrypted enclave binaries
- Based on probing attacks
- Requires kernel privileges and ability to repeatedly crash the enclave

**SGX-Shield** [NDSS 2017]
- Enforces fine-grained memory randomization of SGX enclave
- Software-based data execution prevention (DEP)
- Proposes control-flow integrity for return instructions
Can we bypass memory randomization in SGX?
The Guard's Dilemma: Efficient Code-Reuse Attacks Against Intel SGX

Andrea Biondo, Mauro Conti, Lucas Davi, Tommaso Frassetto, Ahmad-Reza Sadeghi

USENIX Security Symposium 2018
Our main observation is that the Intel SGX SDK includes dangerous return-oriented programming gadgets which are essential for app-enclave communication
ECALL: Call into an enclave

- **App Code**
- **App Data**
- **Untrusted Runtime System (uRTS)**
- **Trusted Runtime System (tRTS)**
- **Enclave Code**
  - Function 0
  - Function 1
  - Function 2
  - Function 3
- **Enclave Stack**
OCALL: Enclave Call to the Host Application

Enclave Code
- Function 0
- Function 1
- Function 2
- Function 3

Enclave Stack
- OCALL Frame
- Register State

Untrusted Runtime System (uRTS)

Trusted Runtime System (tRTS)

App Code

App Data

OCALL Frame

Register State
AEX: Asynchronous Enclave Exit (Exception)

APP
- App Code
- App Data

Enclave
- Enclave Code
  - Function 0
  - Function 1
  - Function 2
  - Function 3
- Enclave Stack
  - Exception information structure
    - Register State

Untrusted Runtime System (uRTS)

Trusted Runtime System (tRTS)

Operating System
Restoring State is Critical

- When OCALL returns, the register state is restored by the tRTS function `asm_oret()`
- If an attacker manages to inject a fake ocall frame, she controls the subsequent state

- After handling the exception, the register state is restored by the tRTS function `continue_execution()`
- If an attacker manages to inject a fake exception structure, she controls the subsequent state
Restoring State is Critical

When OCALL returns, the register state is restored by the tRTS function `asm_oret()`

If an attacker manages to inject a fake OCALL frame, she controls the subsequent state

After handling the exception, the register state is restored by the tRTS function `continue_execution()`

If an attacker manages to inject a fake exception structure, she controls the subsequent state
Basic Attack Idea

APP
- App Code
- App Data

Untrusted Runtime System (uRTS)

Trustd Runtime System (tRTS)

Enclave Code
- Function 0
- Function 1
- Function 2
- Function 3

Enclave Stack
- Counterfeit State
- Mal. Register State

Counterfeit State Information
Two Attack Primitives

- **Primitive to exploit OCALL mechanism**
- **It is based on injecting fake OCALL frames**
- **Prerequisites: stack control**

- **ORET Primitive**
  - rbx
  - rsi
  - rdi
  - rpb
  - r12
  - r14
  - r15

- **CONT Primitive**
  - rsp
  - rip
  - all_other_regs
  - rdi
  - all_other_regs

- **Primitive to exploit asynchronous exception handling in SGX**
- **Based on injecting fake exception structures**
- **Prerequisites: function pointer overwrite and control of rdi register**
Chaining the Two Primitives

**ORET Primitive**
- rip, rsp
- rip, rdi, rsi, rbp, rbx, r12-r15

**CONT Primitive**
- rip, rdi
- rip, rsp, all_other_regs

ROP Gadget
Attack Workflow for Stealing SGX-Protected Keys

APP

Untrusted Runtime System (uRTS)

Trusted Runtime System (tRTS)

Enclave

Counterfeit State Information

App Code

App Data

Enclave Code

Function 0
get_key
send_file
Function 3

Enclave Stack

Counterfeit State
Fake OCALL Frames
Except. Structures

Counterfeit State Information

riprsprdi all_other_regs

ORET Primitive
CONT Primitive

Function 0
Attack Workflow for Stealing SGX-Protected Keys

[Diagram showing the workflow with labeled components such as App Code, App Data, Enclave Code, Enclave Stack, ORET Primitive, CONT Primitive, Counterfeit State, and Fake OCALL Frames Except. Structures.]
Attack Workflow for Stealing SGX-Protected Keys

APP

App Code

App Data

Untrusted Runtime System (uRTS)

Enclave

Enclave Code

Function 0
get_key
send_file
Function 3

Enclave Stack

Counterfeit State
Fake OCALL Frames Except. Structures

Trusted Runtime System (tRTS)

ri
r
sp
rdi
all_other_regs

ORET Primitive
CONT Primitive
Attack Workflow for Stealing SGX-Protected Keys

APP

Enclave

Enclave Code

Function 0
get_key
send_file
Function 3

Trusted Runtime System (tRTS)

Untrusted Runtime System (uRTS)

Enclave Stack

Counterfeit State
Fake OCALL Frames
Except. Structures

ripl  rspi  rdil  all_other_regs
Attack Workflow for Stealing SGX-Protected Keys

App Code

App Data

Untrusted Runtime System (uRTS)

Trusted Runtime System (tRTS)

Enclave Code

Function 0
get_key
send_file
Function 3

Enclave Stack
Counterfeit State
Fake OCALL Frames
Except. Structures

Enclave Stack

riprsprdi all_other_regs

get_key
send_file
Function 3
Attack Workflow for Stealing SGX-Protected Keys

- **APP**
  - App Code
  - App Data

- **Enclave**
  - Enclave Code
    - Function 0
    - get_key
    - send_file
    - Function 3

- **Enclave Stack**
  - Counterfeit State
  - Fake OCALL Frames
  - Except. Structures

- **Trusted Runtime System (tRTS)**
  - ORET Primitive
  - CONT Primitive

- **Untrusted Runtime System (uRTS)**
  - rip
  - rsp
  - rdi
  - all_other_regs

- Function 0:
  - get_key
  - send_file

- Function 3
Attack Workflow for Stealing SGX-Protected Keys

**APP**
- **App Code**
- **App Data**

**Enclave**
- **Enclave Code**
  - Function 0
  - `get_key`
  - `send_file`
  - Function 3

**Enclave Stack**
- Counterfeit State
  - Fake OCALL Frames
  - Except. Structures

**Trusted Runtime System (tRTS)**
- ORET Primitive
- CONT Primitive

**Untrusted Runtime System (uRTS)**
- rip
- rsp
- rdi
- all_other_regs
However, this attack doesn’t work if SGX-Shield randomizes the SGX address space
Revisited Attack to Bypass SGX-Shield

![Diagram showing the flow of data between App Code, App Data, Enclave Code, and Trusted Runtime System (tRTS) vs. Untrusted Runtime System (uRTS). The diagram includes functions such as get_key and send_file within the Enclave, along with primitives like ORET and CONT, and exceptions like Memory Write and Counterfeit State.](image-url)
Revisited Attack to Bypass SGX-Shield

APP

Enclave Code
- Function 0
- get_key
- send_file
- Function 3

Enclave Stack
- Counterfeit State
- Fake OCALL Frames
- Except. Structures

Shellcode
Stealing Keys

Memory Write

ORET Primitive
CONT Primitive

trusted runtime system (tRTS)

untrusted runtime system (uRTS)

App Code

App Data
Possible Defenses

• Removing SDK from enclave memory?
  • Not feasible as OCALL, ECALL, AEX require the tRTS

• Randomizing SDK code?
  • Challenging, the tRTS is accessed through fixed entry points

• Control-Flow Integrity (CFI)
  • Seems promising as SGX code is not as huge as complex browser code
  • Many projects on CFI: Microsoft CFG, Intel CET, Google clang CFI compiler
Conclusion

• Memory corruption attacks in SGX can exploit dangerous gadgets located in linked Intel SDK libraries

• Not randomizing SKD libraries allows an attacker to instantiate a ROP attack

• Protecting SGX enclaves from memory corruption attacks is vital to preserve the security guarantees offered by Intel SGX