The Guard’s Dilemma
Efficient Code-Reuse Attacks Against Intel SGX

Andrea Biondo\textsuperscript{1}, Mauro Conti\textsuperscript{1}, Lucas Davi\textsuperscript{2}, Tommaso Frassetto\textsuperscript{3},
Ahmad-Reza Sadeghi\textsuperscript{3}

\textsuperscript{1}University of Padua, Italy
\textsuperscript{2}University of Duisburg-Essen, Germany
\textsuperscript{3}TU Darmstadt, Germany
Intel SGX (Software Guard eXtensions)
Intel SGX (Software Guard eXtensions)
SGX provides strong isolation.
(that’s what it says on the box!)

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

Yanxiong Xu
The University of Texas at Austin
yanx@cs.utexas.edu

Jiexi Lei
Microsoft Research
jilei@microsoft.com

Marcus Peinado
Microsoft Research
marcuspe@microsoft.com

CacheZoom: How SGX Amplifies The Power of Cache Attacks

Alomud Moghimi, Gorka Irazoqui, and Thomas Eisenbarth

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

Telling Your Secrets Without Page Faults: Stealthy Page Table-Based Attacks on Enclaved Execution
Jo Van Bulcke, imec-DistriNet, KU Leuven; Nico Weichert and Rüdiger Kapitza, IBR DS, TU Braunschweig; Frank Piessens and Raoul Strackx, imec-DistriNet, KU Leuven

Software Grand Exposure: SGX Cache Attacks Are Practical
Ferdinand Brasser1, Uts Müller2, Alexandra Dreianenreko3, Kari Kostaria4, Sriljan Cipkun2, and
Aboud-Reza Saleghi3

Inferring Fine-grained Control Flow Inside SGX Enclaves with Branch Shadowing
Sangho Lee, Ming-Wei Shih, Parun Gera, Taeos Kim, and Hyesoon Kim, Georgia Institute of Technology; Marcus Peinado, Microsoft Research

FORESHADOW: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution
Jo Van Bulcke, imec-DistriNet, KU Leuven; Marina Minkin, Technion; Ofr Weisse, Daniel Genkin, and Baris Kasikci, University of Michigan; Frank Piessens, imec-DistriNet, KU Leuven; Mark Silberstein, Technion; Thomas F. Wenisch, University of Michigan; Yuval Yarom, University of Adelaide and Data61; Raoul Strackx, imec-DistriNet, KU Leuven

SGXPECTRE Attacks: Stealing Intel Secrets from SGX Enclaves via Speculative Execution
Guoxing Chen, Soundhan Chen, Yuan Xiao, Yining Zhang, Zhiqing Liu, Ten H. Lai, Department of Computer Science and Engineering
Just like normal programs, SGX code can have bugs.
Control-Flow Attacks

Code Injection

A → E
B → C
D → F

Control-Flow Hijacking

Shellcode
Control-Flow Attacks

Code Injection

Code Reuse
(e.g., Return-Oriented Programming)

Control-Flow Hijacking

Write ⊕ eXecute

Gadget
Control-Flow Attacks

Code Injection

Control-Flow Hijacking

Write \oplus eXecute

Code Reuse
(e.g., Return-Oriented Programming)

Control Flow Integrity

Gadget
Related work

**Dark-ROP**
[Lee et al., USENIX Security 2017]
- Remote attestation + loader = no access to enclave code
- ROP still feasible by finding gadgets through oracles

**SGX-Shield**
[Seo et al., NDSS 2017]
- Fine-grained enclave randomization, $W \oplus X$
- Software Fault Isolation, Control Flow Integrity
- State-of-the-art hardening scheme
The SGX SDK

Source

Compiler

SGX SDK

Enclave

Function 0

Function 1

Function 2

Function 3

Trusted Runtime System (tRTS)

Untrusted Runtime System (uRTS)

App Code

App-to-Enclave function call (ECALL)
The SGX SDK

Source

SGX SDK

Compiler

App

Enclave

Function 0

Function 1

Function 2

Function 3

Trusted Runtime System (tRTS)

Untrusted Runtime System (uRTS)

App Code

Enclave-to-App function call (OCALL)
The SGX SDK

Source

Compiler

SGX SDK

App

Enclave

Exception

Function 0

Function 1

Function 2

Exc. Handler

Trusted Runtime System (tRTS)

Untrusted Runtime System (uRTS)

App Code

OS Kernel

AEX

Signal
The Guard’s Dilemma

- Novel SGX code-reuse attack
- Dispatches ROP gadgets
- Uses only existing tRTS functionality

Why?
Motivation

- Widespread SDK usage
- Easier exploitation
- Existing hardening does not cover tRTS
The Basic Idea

App

Enclave

Trusted Runtime System (tRTS)

Restore State

State

Function 0

Function 1

Function 2

Function 3

Counterfeit state

15
The ORET Primitive

- OCALL
- Save context in OCALL frame
- Exit enclave
- Execute untrusted function
- Re-enter enclave
- Restore context (do_oret)

Partial register control
Fake OCALL frame

Control-Flow Hijacking
Stack control
The CONT Primitive

1. Exception
2. Exit enclave to OS handler
3. Re-enter, save context, exit
4. Resume enclave
5. Call exception handlers
6. Restore exception context (continue_execution)

Fake exception context

Full register control

Control-Flow Hijacking

1° argument control

(see paper)
The ORET+CONT Loop

ORET
- rip, rsp → rip, rdi, ...

CONT
- rip, rdi → rip, rsp, *

ROP Gadget
Attack Overview

1. Payload Preparation
   - Find gadgets
   - Design gadget chain

   - $n$ fake exception infos
   - 1 fake stack (ROP, OCALL)

3. Attack Execution
   - Launch first CONT

Diagram:
- CONT
- Gadget
- ORET
- Fake exc. info 1 → Gadget 1
- Fake exc. info 2 → Gadget 2
- Fake exc. info 3 → Gadget 3
- Fake stack
Example Attack

Trusted Runtime System (tRTS)

ORET: rip, stack → rdi + rip + ...

CONT: rdi → rip + all registers
SGX-Shield [Seo et al., NDSS 2017]

Source

SGX-Shield Toolchain

SGX-Shield Runtime

SGX SDK

App

Enclave

Rand. Unit 0

Rand. Unit 1

Rand. Unit 2

Rand. Unit 3

Trusted Runtime System (tRTS)

Untrusted Runtime System (uRTS)

App Code

Fine-grained code randomization

tRTS is not randomized
Attacking SGX-Shield

- Fine-grained code randomization
- Reusing tRTS code (not randomized)
- Coarse-grained Control Flow Integrity
- Return edges are not properly instrumented → ORET is possible

Other mitigations (SFI, W⊕X) assume CFI
SGX-Shield Exploit

1. Hijack return edge
2. Write shellcode to WX memory
3. Jump to shellcode
4. Gather enclave keys
5. Copy keys to attacker’s memory

Stage 1: ORET+CONT
Stage 2: Shellcode

Untrusted memory
Mitigations

SDK Hardening
- Secret canaries in contexts
- Mangling context data

External Hardening
- Randomization of SDK code
- Stronger CFI
Lessons Learned

• SGX presents significant hardening challenges
  • Strong attacker

• The SDK can increase an enclave’s attack surface
  • Powerful code-reuse primitives
  • Low-level code hidden from sight
Conclusion

• We presented a novel code-reuse attack on Intel SGX
• Using «forgotten» code to bypass SoA hardening
• Underlines the need to consider implications of SDK usage

Questions?