PRIDWEN
Universally Hardening SGX Programs via Load-Time Synthesis

Fan Sang*,1, Ming-Wei Shih*,3, Sangho Lee4, Xiaokuan Zhang1, Michael Steiner2, Mona Vij2, Taesoo Kim1

1Georgia Institute of Technology, 2Intel Labs, 3Microsoft, 4Microsoft Research

*Authors contributed equally to this work.
Prevalence of Cloud Computing Today

- PC
- Mobile
- Internet of Things
- Database
- Network

- Amazon Web Services
- Microsoft Azure
- Google Cloud Platform
Concerns with Cloud: Data Security

Data in transit

User

Data in use

Cloud Provider

Data at rest
Existing Security Solutions

User

TLS/SSL

Cloud Provider

Intel SGX (*)

Disk Encryption

*Intel Software Guard Extension (SGX)
• **Enclave**: Isolated memory region
  • Strict memory access control
  • Memory encryption
• **Remote attestation**
  • Allows for attesting code/data inside a remote enclave

Intel SGX 101

- Privileged software (e.g., OS, hypervisor)
- SGX-capable CPU
- Physical attacks (e.g., memory snooping)
- External memory access
- Enclave:
  - Isolated memory region
  - Strict memory access control
  - Memory encryption
  - Remote attestation
  - Allows for attesting code/data inside a remote enclave

- Decrypt
- Encrypt
- Access Deny
Achilles’ Heel of SGX: Side-Channel Attacks

Cloud providers as attackers (with root privilege)
• Side-channel inference with low-noise, high-resolution
Side-Channel Attacks Against SGX

• Shared resources as side channels
  • Page table \([SP’15, Security’17]\)
  • Cache \([WOOT’17, ATC’17, CHES’17]\)
  • Branch predictor \([Security’17, ASPLOS’18]\)
  • TLB \([CCS’17, Security’18]\)
  • CPU pipelines \([Security’18, EuroSP’19, SP’20, SP’21]\)

• Allow the attacker to infer fine-grained information inside the enclave
  \(\rightarrow\) Break the security guarantees of SGX

Question: How to address the side-channel attacks against SGX?
## Side Channel Mitigation Schemes

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Mitigation Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGX-Shield [NDSS’17]</td>
<td>Fine-grained ASLR</td>
</tr>
<tr>
<td>Varys [ATC’18]</td>
<td>High-frequent interrupt-based attacks</td>
</tr>
<tr>
<td>T-SGX [NDSS’17]</td>
<td>Page-fault attacks</td>
</tr>
<tr>
<td>Cloak [Security’17]</td>
<td>Cache attacks</td>
</tr>
<tr>
<td>HyperRace [DSC’19]</td>
<td>Hyperthread-based attacks</td>
</tr>
<tr>
<td>Retpoline &amp; Qspectre [2018]</td>
<td>Spectre attacks</td>
</tr>
</tbody>
</table>

**Similar design choices**
- Require no hardware modification
- Minimum manual efforts (*instrumentation-based*)
Deployment of a Mitigation Scheme

**Local**

- c/c++ Program
- Mitigation scheme
  - Customized compiler
  - Protected binary

**Remote**

- Enclave
- Protected code/data
  - SGX-capable CPU

Deployment of a Mitigation Scheme
Each Scheme Targets Limited Types of Attacks

Local

Program → Mitigation scheme → Protected binary

Remote

Enclave

Protected code/data

Problem: Multiple side channels can co-exist

Solution: Compose multiple mitigation schemes
Composing Multiple Mitigation Schemes

Local

Compiler

Protected binary

Mitigation schemes

Remote

Enclave

Protected code/data

SGX-capable CPU

That’s it?
Problems with Naïve Scheme Composition

Static Enforcement

Local

Compiler

Protected binary

Undeployable

Remote

No TSX support

Enclave

Enclave

HT disabled

Compiler

Incompatible

HT-based attacks

Cache attacks

Page-fault attacks

Require TSX support

Target attacks

Incompatible

Redundant

Enclave

Enclave

Enclave

12
When Can We Make the Best Decisions?

As Close to the Final Execution as Possible!
Local Scheme Enforcement

Local

Compiler

Protected binary

Remote

Enclave

Protected code/data

SGX-capable CPU

Program

c/C++

HT-based attacks

Cache attacks

Page-fault attacks

Target attacks

1011

0100
Post-Deployment Scheme Enforcement

Local

Remote

Requirements
• Detection of hardware configurations
• Selective enforcement of schemes
• Validation of enforcement

No TSX support
HT Disabled
PRIDWEN Overview

Challenges in SGX

- Target binary is dynamically generated
- Only the static part (loader) is attestable

Detection of hardware configurations

- cpuid, syscall are not allowed
- OS is untrusted

Selective enforcement of schemes

Validation of enforcement

Protected code/data

User Binary

Synthesizer

Validator

Prober

Pass Manager

TSGX  Varys  ASLR  CoTest
PRIDWEN Overview

Challenges in SGX

Detection of hardware configurations

User Binary

Synthesizer

Validator

Prober

Pass Manager

Selective enforcement of schemes

Validation of enforcement

Protected code/data

PRIDWEN Loader
Selective Enforcement of Schemes

• Approach: *Load-time synthesis*
  • Take the intermediate representation (IR) of a program as input
  • Support compilation and instrumentation of the IR
  • Provide APIs for implementing schemes as instrumentation passes
Use IR as Input

• Advantages over native binary
  • Friendly for code analysis and instrumentations
  • Platform independent

• IR selection: WebAssembly (WASM)
  • Lightweight (small instruction set), small TCB
  • Supports multiple high-level languages (e.g., C/C++, Rust)
  • Straightforward compilation

<table>
<thead>
<tr>
<th></th>
<th>Line of Code</th>
<th>Binary Size (MiB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRIDWEN backend</td>
<td>8,166</td>
<td>1.26</td>
</tr>
<tr>
<td>LLVM x86 backend</td>
<td>80,449</td>
<td>1,026.00</td>
</tr>
</tbody>
</table>
C to WebAssembly

C Language

```c
int foo(int x) {
    if (x != 0) {
        return x * x;
    }
    return 0;
}
```

WASM IR

```wasm
(func (;0;) (param i32)
  result i32)
local.get 0
local.get 0
i32.mul
local.set 1
local.get 0
if (result i32)
  local.get 1
else
  i32.const 0
end)
```

Supported by opensource compiler

Require no source-code modifications

Supported by PRIDWEN
Flexible Instrumentation

**IR-level**
(func (;0;) (param i32)
  result i32)
  local.get 0
  local.get 0
  i32.mul
  local.set 1
  local.get 0
  if (result i32)
    local.get 1
  else
    i32.const 0
end)

**Native-level**
push rbp
mov rbp, rsp
sub rsp, 0x10
mov eax, edi
imul eax, edi
test edi, edi
je $1 ; else
jmp $2
$1: xor eax, eax
$2: mov rsp, rbp
pop rbp
ret
PRIDWEN Instrumentation APIs

• IR-level
  onFunctionStart(CompilerContext *ctx)
onFunctionEnd(CompilerContext *ctx)
onControlStart(CompilerContext *ctx)
onControlEnd(CompilerContext *ctx)
onInstrStart(CompilerContext *ctx)
onInstrEnd(CompilerContext *ctx)

• Native-level
  onMachineInstrStart(CompilerContext *ctx, MachineInstr *mi)
onMachineInstrEnd(CompilerContext *ctx, MachineInstr *mi)
PRIDWEN In Action

WASM Binary

Enclave

PRIDWEN Loader

Synthesizer → Validator → Protected code/data

Prober → Pass Manager

TSX is not available
HT is enabled
• The performance of synthesizing small and large programs
  • Small program (~50 kB): 50 - 60 ms
  • Large program (~500 kB): < 500 ms

Paid only once
Evaluation: Baseline Runtime Performance

- Relative runtime performance of PRIDWEN-synthesized applications compared to native versions
  - Lighttpd: 1.5x
  - SQLite: 1.3x
  - libjpeg: 1.4x
  - Recent study shows in-browser WASM JITs on SPEC: 1.45x – 1.55x
Evaluation: Overhead of Mitigation Schemes

- The performance of libjpeg and SQLite
  - HW-assisted: \(1.9x\)
  - SW-only: \(3.4x\)

Comparable to the original implementations
• SGX side-channel attacks can co-exist
• Existing model for deploying mitigation schemes is limited
• We propose PRIDWEN to achieve scheme composition
  • Detect hardware configurations
  • Adaptively enforce mitigation schemes with an in-enclave loader
  • Extensible framework to support more schemes

https://github.com/sslab-gatech/Pridwen
Q&A
Thank You!

Fan Sang†,1, Ming-Wei Shih3, Sangho Lee4, Xiaokuan Zhang1, Michael Steiner2, Mona Vij2, Taesoo Kim1

†Georgia Institute of Technology, 2Intel Labs, 3Microsoft, 4Microsoft Research

†fsang@gatech.edu

Systems Software & Security Lab

Georgia Tech College of Computing
School of Cybersecurity and Privacy