CCS’17 Tutorial:
SGX Shielding Frameworks and Development Tools

Chia-Che Tsai
Stony Brook University / UC Berkeley
This presentation contains the general insights and opinions of Intel Corporation (“Intel”). The information in this presentation is provided for information only and is not to be relied upon for any other purpose than educational. Use at your own risk! Intel makes no representations or warranties regarding the accuracy or completeness of the information in this presentation. Intel accepts no duty to update this presentation based on more current information. Intel is not liable for any damages, direct or indirect, consequential or otherwise, that may arise, directly or indirectly, from the use or misuse of the information in this presentation.

Intel technologies’ features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at intel.com, or from the OEM or retailer.

No computer system can be absolutely secure.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel, the Intel Core, and the Intel logo are trademarks of Intel Corporation in the United States and other countries.

*Other names and brands may be claimed as the property of others.

© 2017 Intel Corporation.
Developing a SGX Application

• SDK model: build your own SGX applications

• Porting an existing application
  • Limitation 1: needs a signed, static image
  • Limitation 2: virtualized ISA (no CPUID/RDTSC)
  • Limitation 3: no trusted OS services

• Requires defenses against untrusted OSes
Choose Porting Strategy

• How much OS functionality is needed?
  • Little (e.g., crypto functions) ➔ SDK
  • Medium (e.g., microservices) ➔ Shielding layers
  • Heavy (e.g., language runtimes) ➔ Library OSes

• Always ensure a secure enclave interface

• Performance is a critical factor
Topics

• Porting challenges and OS attack vectors
• Library OS: Graphene-SGX
• System interface shield layers: SCONE, Panoply
• Dynamic page management on SGX2
• Exit-less enclaves with Eleos
For Each Framework

• What are the target applications?
• What are the key concepts?
• What to expect? How to use?
• Where to obtain the software?
SGX Porting Challenges

• Satisfying enclave requirements
• Defending against untrusted OS services
• Improving performance factors
SGX Application Requirements

“Enclave”

Sensitive Data  Signed App

Untrusted App

SGX instructions (ECREATE,EINIT)

Untrusted OS

Initial image, security measurement

Completely isolated from OS
SGX Application Requirements

1. Static initial image
2. No system calls
3. Check for untrusted inputs

Most Linux applications:
(1) Dynamic linked
(2) Built-in syscall usage
Porting a Legacy Application

1. Statically linking all binaries
2. Bypassing instructions (CPUID/RDTSC)
3. Exiting enclave for system calls

Security Challenge!
SGX Porting Challenges

- Satisfying enclave requirements
- Defending against untrusted OS services
- Improving performance factors
Attack Vectors from Untrusted OS

Apache Web Server

failed to correctly check syscall results

read()

Exit enclave

Iago Attacks
[Checkoway, ASPLOS 13]

Manipulate results to attack enclave

Untrusted Host OS
Iago Attacks In A Nutshell

- Semantic attacks by manipulating syscall results
- Application-specific
- Bugs that do not exist on a trusted OS
Iago Attack Example: SSL Random Generator Seed

```c
int ssl_rand_seed(...) {
  ...
  if (pRandSeed->nSrc == SSL_RSSRC_BUILTIN) {
    struct {
      time_t t;
      pid_t pid;
    } my_seed;
    l = sizeof(my_seed);
    my_seed.t = time(NULL);
    my_seed.pid = getpid();
    RAND_seed((unsigned char *)&my_seed, l);
  }
}
```
SGX Shielding Frameworks

• Several work address the problem of SGX porting
  • (1) Defenses against lago attacks
  • (2) Performance optimization
  • (3) Compatibility features (e.g., cross compilers)

• Two approaches:
  • (1) Library OSes
  • (2) Shielding layers
Key Factors

• Shielding mechanisms (especially Iago attacks)
• Attack surface
• Trusted computing base (TCB)
• OS functionality
Library OSes

- OS components in enclave
- Define small enclave interface with security in mind
- Example: Haven [OSDI’14]
  Graphene-SGX
Shielding Layers

- Shielding each API
- Avoid library OS overheads
- Small TCB
- Example: SCONE, Panoply
## Comparison

<table>
<thead>
<tr>
<th></th>
<th>Graphene-SGX</th>
<th>SCONE</th>
<th>Panoply</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Approach</strong></td>
<td>Library OS</td>
<td>Shielding Layers</td>
<td></td>
</tr>
<tr>
<td><strong>Enclave interface</strong></td>
<td>Fixed interfaces (regardless of libOS functionality)</td>
<td>Equals the system API needed by the application</td>
<td></td>
</tr>
</tbody>
</table>
## Trusted Computing Base

<table>
<thead>
<tr>
<th>LibOS/Shielding Layer</th>
<th>Graphene-SGX</th>
<th>SCONE</th>
<th>Panoply</th>
</tr>
</thead>
<tbody>
<tr>
<td>Libc option</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GLIBC (1.1 MLoC)</td>
<td>53 kLoC</td>
<td>97 kLoC</td>
<td>10kLoC</td>
</tr>
<tr>
<td>MUSL (88 kLoC)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No Libc in enclave</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**The choice of Libc is the highest-order bits**
SGX Porting Challenges

- Satisfying enclave requirements
- Defending against untrusted OS services
- Improving performance factors
Performance Factors

• Enclave creation time
  • Correlated with enclave memory size (1GB requires ~3s)

• Memory access overheads
  • LLC misses up to 10X
  • EPC paging: 128MB shared among all enclaves
    40,000 cycles for page-out and page-in

• Enclave exits
  • 7,000~8,000 cycles for exit and re-enter
Performance improvement

- Enclave creation time: EDMM on SGX2
  - Dynamically adding pages at run time
- Reduce explicit & implicit exits: Eleos
  - Completely exit-less enclaves
  - Pinning EPC pages with software-based paging
Topics

• Porting challenges and OS attack vectors
• Library OS: Graphene-SGX
• System interface shields: SCONE, Panoply
• EDMM on SGX2
• Exit-less enclaves with Eleos
Graphene-SGX: A LibOS for Unmodified Applications

- Servers, Command-line, Runtimes: Apache, NGINX, GCC, R, Python, OpenJDK, etc
- Multi-process APIs: fork, exec, IPC, etc
- Not perfect, but a quick, practical porting option
The Graphene LibOS Project [Eurosys14]

• Open library OS for reusing Linux applications (github.com/oscarlab/graphene)
  • Inspired by Drawbridge [ASPLOS11] and Haven [OSDI14]
  • Under active development

Easy to port to new OS/platform

Unmodified Application

Process

145 system calls (still growing)

LibOS

LibOS
Applications in Graphene-SGX

1. Static initial image
2. No system calls
3. Check for untrusted inputs

$ SGX=1 ./pal_loader httpd [args]

Graphene Loader

Untrusted OS
Applications in Graphene-SGX

1. Static initial image ✓
2. No system calls
3. Check for untrusted inputs
Applications in Graphene-SGX

1. Static initial image ✓
2. No system calls ✓
3. Check for untrusted inputs

- Application
- Libraries
- Modified GLIBC

- Linux system calls
- Graphene LibOS
- Manifest

- Enclave Interface (28 calls)

- System calls redirected into library OS

- Fixed interface to check

- Untrusted OS
Checking Enclave Interface

• Reduce enclave interface to 28 calls

• Design defense for each call
  • Define explicit semantics
    ➔ knowing exactly what/how to check
  • Crypto techniques

• Examples:
  • Accessing integrity-sensitive files (binaries / configs)
  • Process creation (see paper)
Ex: File Integrity Check

- Ask for exact file content
- Verify by checksums
Checking All 28 Enclave Calls

<table>
<thead>
<tr>
<th>Examples</th>
<th>#</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) Reading a file</td>
<td></td>
<td>Fully Checked</td>
<td>(1) File checksums</td>
</tr>
<tr>
<td>(2) Inter-proc coordination</td>
<td></td>
<td></td>
<td>(2) CPU attestation + crypto: inter-proc TLS connection</td>
</tr>
<tr>
<td></td>
<td>18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Yielding a thread</td>
<td>6</td>
<td>Benign</td>
<td>Do not take any input</td>
</tr>
<tr>
<td>(1) Polling</td>
<td></td>
<td>Unchecked</td>
<td>May cause DoS; Future work</td>
</tr>
<tr>
<td>(2) File attributes</td>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Apache (5 Procs w/ IPC Semaphore)

Graphene-SGX:
Impact by enclave exits & checking OS inputs

Graphene:
little impact (~5%) on top throughput

Average Response Time (S)

Throughput (k.req/S)

Linux

30% loss
Graphene-SGX: Memory-intensive workloads are expensive
Graphene-SGX Features

• Current features
  • Use GLIBC by default; can use MUSL if acceptable
  • A wide range of servers, command-lines, language runtimes tested
  • Static binary support
  • Limitations: cannot support shared memory
Demo: GCC on Graphene-SGX

• Multi-process: gcc → cc1 → collect2 → ld

• Turn on DEBUG=1

• Attack: Try to modify the GCC binary
Demo: GCC on Graphene-SGX
GSC: Graphene Secure Container

• Docker images ➞ enclaves
  • Dockerfiles ➞ manifests

• Graphene-SGX runs in container
  • Mutual isolation between OS and application
GSC: Graphene Secure Container

- Docker Image
  - Application
  - Libraries

Conversion
- GSC Engine (GSCE)
  - GSC Image
    - Application
    - Libraries
    - Graphene-SGX
    - BootStrapper

Docker Container
- Enclave
  - Application
  - Libraries
  - Graphene-SGX
  - BootStrapper

OS
VMM
Hardware
Demo: Graphene-SGX Container
Availability

• Open-source at http://github.com/oscarlab/graphene

• Currently under GPLv3, switching to LGPL soon

• Contact:
  • chiache@cs.stonybrook.edu
  • porter@cs.unc.edu
  • https://graphene-libraryos.slack.com (contact me for invitation)
SCONE: A Lightweight Layer for SGX

• An enhanced C library with file and network shields
• Strictly requires no library OS
• Optimized syscall performance for enclaves
SCONE Architecture

Application

Libraries

- Network shield
- File system shield
- M:N threading
- SCONE C library (based on MUSL)
- Asynchronous system calls

SCONE module

Intel SGX driver

Container (cgroups)

Inside enclave
(trusted)

Host OS kernel
(untrusted)
SCONE Architecture

- Network and FS shields: encrypting and authenticating network and file contents
- MUSL: small TCB (88KLoC)
- Asynchronous system calls: avoid enclave exits
- SCONE module (optional): improve performance

Application

Libraries

Network shield

File system shield

M:N threading

SCONE C library (based on MUSL)

Asynchronous system calls

SCONE module

Intel SGX driver

Container (cgroups)
System Call Overheads

- `pwrite()` with 32 byte buffer
- 4 cores with hyper threading
Asynchronous System Calls

The graph shows the system call frequency (1000s/second) across different thread counts. The x-axis represents the number of threads, ranging from 1 to 8. The y-axis represents the system call frequency in thousands per second, ranging from 1 to 100,000.

- **async with 1 thread** achieves 80% performance.
- **native** performance remains stable across varying thread counts.
- **sync** shows a linear increase with the number of threads.

The chart indicates that asynchronous calls, especially with a single thread, can improve performance significantly compared to synchronous calls.
Apache Throughput

Latency (seconds) vs. Throughput (requests / second)

- Sync
- Async
- Native
Memcached Throughput

- YCSB workload A (50/50)
- Data fits into EPC

*inline encryption has less overhead than TLS*

*proxy*

*async*

*glibc + stunnel*

Latency (ms)

Throughput (operations / second)
SCONE Language Support

- Cross compiler for several languages
  - C and C++
  - GO
  - Rust
  - Python
  - PHP
  - Java (partial support, still work in progress)
Demo: SCONE Cross Compiler

```
sergey@beast:~/workspace/scone$
```

SCONE Hello World DEMO
SCONE Features

• Current SCONE features
  • Support static and dynamic linking
  • Unmodified binaries must be position independent (built with \texttt{--fPIC})
  • Compatible with MUSL
  • No multi-processing (fork / execve)
SCONE Docker Integration

• SCONE supports (extended) Docker compose files
  • Transparent attestation of services
  • Transparent configurations

• Unmodified Docker Engine
  • Docker engine runs outside enclave
Availability

• Commercially available via SCONTAIN
• Acquire the software: www.scontain.com
• Contact: christof.fetzer@gmail.com
Panoply: POSIX API with Small TCB

- A POSIX library without Libc in enclave
- Placing applications and libraries into separate enclaves
- 10kLoC TCB in Panoply shim library

Panoply: Low-TCB Linux Applications with SGX Enclaves

Shweta Shinde
National University of Singapore
shweta24@comp.nus.edu.sg

Dat Le Tien
University of Oslo
dattl@ifi.uio.no

Shruti Tople
National University of Singapore
shruti90@comp.nus.edu.sg

Prateek Saxena
National University of Singapore
prateeks@comp.nus.edu.sg

Abstract—Intel SGX, a new security capability in emerging CPUs, allows user-level application code to execute in hardware-isolated enclaves. Enclave memory is isolated from all other software on the system, even from the privileged OS or hypervisor. While being a promising hardware-rooted building block, enclaves have severely limited capabilities, such as no native access to system calls and standard OS abstractions. These OS abstractions are used ubiquitously in real-world applications. has been a threat to privileged software layer, often targeting vulnerabilities in privileged code such as the OS. In this paper, we envision providing the benefits of privilege separation and isolation based on a strong line of defense against OS-resident malware. Such a defense is based on a new trusted computing primitive, which can isolate a sensitive user-level application from a compromised OS. Hardware support for this primitive...
Panoply Architecture

Panoply expels GLIBC outside of the enclave
Panoply Architecture

• Micron can be an application or a library
• Multi-enclave collaboration:
Panoply cross-compiler

(1) Compiler instrumentation
- Add calls to Panoply API
- Adding flow checks

(2) Creating enclaves
- Enclave-bound code
- Panoply Shim
- Intel SDK

Panoply application
Attacks on Multi-Enclave Applications

```c
session_t session;
certificate_credentials_t xcred;

/* Specify callback function*/
certificate_set_verify_function (...);

/* Initialize TLS session */
init (&session, TLS_CLIENT);
```

Diagram:
- Webserver Enclave
- SSL Library Enclave
- Set SSL Callback
- OS
Securing Multi-Enclave Applications

Enclave 1  \[\Rightarrow\]  Pair-wise Nonce  \[\Rightarrow\]  Enclave 2

Enclave Identity

Defenses
- Sender / Receiver Authentication
- Message Freshness
- Reliable Delivery

Attack
- Spoofing
- Replay
- Silent Drops

Call Ack
## Performance Overview

<table>
<thead>
<tr>
<th>App</th>
<th>Panoply</th>
<th>Empty enclave</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenSSL</td>
<td>3.16</td>
<td>2.79</td>
<td>13%</td>
</tr>
<tr>
<td>H2O</td>
<td>8.79</td>
<td>6.56</td>
<td>34%</td>
</tr>
<tr>
<td>FreeTDS</td>
<td>8.74</td>
<td>8.60</td>
<td>1%</td>
</tr>
<tr>
<td>Tor</td>
<td>6.72</td>
<td>4.54</td>
<td>48%</td>
</tr>
</tbody>
</table>
Panoply Features

• Currently support 254 POSIX API
• 91 guarantee to preserve API semantics
• Multi-process: fork and exec
Availability

• Open-source at https://shwetasshinde24.github.io/Panoply/

• Apache 2.0 License

• Contact: shweta24@comp.nus.edu.sg
EDMM: Enclave Dynamic Memory Mgmt

• Current SGX: fixed enclave memory and threads

• SGX2: adding pages at run time
  • Reduce initial enclave memory size
  • Dynamic thread creation
  • Dynamic page protection (for dynamic loading / JIT)

• Supported in future Graphene-SGX
Current SGX Limitations

- For integrity, each enclave has a static memory layout
  - Signed by users
  - Initialized at loading time
- Reserved heap for malloc()
- \# TCS = \# Threads
EDMM on SGX2

- Adding and protecting enclave pages at run time

- Page adding semantics:
  - Normal or TCS pages
  - Must be zeroed
  - “Approved” by enclave
EDMM Support in Graphene-SGX

- Compatibility and performance features
  - Largely reduce startup time
  - Dynamic thread creation
  - Protect pages after finishing dynamic loading
  - Support mprotect()
Demo: Graphene-SGX with EDMM
Availability

• SGX2 release date expected in 1~2 years

• EDMM support will be open-sourced as part of Graphene
  • [http://github.com/oscarlab/graphene](http://github.com/oscarlab/graphene)
Eleos: Exit-less Enclaves

• Avoids enclave exits and EPC paging
• Combined w/ SDK: Generating RPC-based interface
• Software-based paging: SUVM

Eleos: ExitLess OS Services for SGX Enclaves

Meni Orenbach, Pavel Lifshits, Marina Minkin, Mark Silberstein
Technion - Israel Institute of Technology

Abstract

Intel Software Guard eXtensions (SGX) enable secure and trusted execution of user code in an isolated enclave to protect against a powerful adversary. Unfortunately, running I/O-intensive, memory-demanding server applications in enclaves leads to significant performance degradation. Such applications put a substantial load on the in-enclave system call and secure paging mechanisms, which turn out to be the main reason for the application slowdown. In addition to the high direct cost of thousands-of-cycles long SGX management instructions, these mechanisms incur the high indirect cost of OS and/or a hypervisor, yet the code running in the enclave may access untrusted memory of the owner process.

While SGX provides the convenience of a standard x86 execution environment inside the enclave, there are important differences in the way enclaves manage their private memory and interact with the host OS.

First, because an enclave may only run in user mode, OS services, e.g., system calls, are not directly accessible. Instead, today’s SGX runtime forces the enclave to exit, that is, to securely transition from trusted to untrusted mode, and to re-enter the enclave after the privileged part of the system call has been completed.
Direct Enclave Costs

• Enclave enter / exit: 3,300 / 3,800 cycles vs System call: 250 cycles

• LLC misses: 5.6~9.5 X

• EPC paging: 40,000 cycles for evict and page-in
Indirect Cost: LLC Pollution

LLC pollution causes up to 2X slowdown

**KVS server with batched requests**

<table>
<thead>
<tr>
<th>Number of keys per request</th>
<th>Slowdown factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>32,768</td>
<td>0.5</td>
</tr>
<tr>
<td>65,546</td>
<td>1</td>
</tr>
<tr>
<td>131,072</td>
<td>1.5</td>
</tr>
<tr>
<td>262,144</td>
<td>2</td>
</tr>
<tr>
<td>524,288</td>
<td>2.5</td>
</tr>
</tbody>
</table>
Indirect Cost: TLB Pollution

KVS server with different collision resolution:
- Open addressing
- Separate chaining
(insensitive to TLB flushes)

TLB Flushes at every exits cause up to 6X slowdown
RPC-based Enclave Interfaces

Outside enclave (Untrusted)  Inside enclave (Trusted)

RPC Thread Pool

RPC Queue

Pass request

Enclave Software

untrusted_call()

Execute

Unlock

“Server”

Spinlock

“Client”
Eleos keeps EPC footprint static, to avoid fault-based exits
Demo: Memcached on Native SGX
Demo: Memcached with Eleos (RPC)
Demo:
Memcached with Eleos (RPC+SUVM)
Memcached Performance

PRC improves 23%, RPC+SUVM improves 51%
Availability

• Open-source available at: http://github.com/acsl-technion/eleos

• Contact: mark@ee.technion.ac.il
Acknowledgement

Assistance from the following individuals:

• Christof Fetzer (TU Dresden)
• Li Lei (Intel Labs)
• Meni Orenbach (Technion)
• Donald E. Porter (UNC at Chapel Hill / Fortanix)
• Shweta Shinde (Natl. Univ. of Singapore)
• Mark Silberstein (Technion)
• Mona Vij (Intel Labs)