Advanced Systems Security: Intel SGX

Trent Jaeger
Systems and Internet Infrastructure Security (SIIS) Lab
Computer Science and Engineering Department
Pennsylvania State University
Remaining Problems

• Deploying a custom OS is painful
  ‣ Building a special kernel is non-trivial

• And it may not be secure itself
  ‣ Still need a methodology to determine code correctness and tamperproofing

• What if you want to eliminate trust in the OS altogether?
Insight: Shadowing Memory

- VMMs need to manage physical to virtual mapping of memory
- This is done with a shadow page table
- Multi-shadowing give context-aware views of this memory
  - Use encryption instead
Memory Cloaking

- Not new idea
  - XOM, LT
- Encrypt the pages in memory
  - For each page, (IV, H) meta data
  - What should the “secure hash” be?
- OS can operate on encrypted pages
  - But can’t read them
Tasks of the Overshadow

- Mediate all application interaction with OS to ensure correct cloaking of memory
  - Context Identification
  - Secure Control Transfer
  - System Call Adaptation
  - Mapping Cloaked Resources
  - Managing Protection Metadata
Shim baby Shim

- The key to Overshadow is the **Shim**
  - Manages transitions to and from VMM via a hypercall
- **Shim Memory** protects application
  - CTC protects control registers
- **Uncloaked Shim**
  - Neutral ground
  - Trampoline!
Loading Applications

- The Shim uses a **Loader** program
- Sets up the cloaked memory with a hypercall
- The loader / shim must be trusted
  - Metadata on the CTC checks for compromise
  - Here is the **meat** of the problem
    - Is it even used?
- Propagate shims to spawned applications
Its not that easy…

- Lot of OS interfaces that must be handled
- Faults / Interrupts
- System Calls
  - Pass control to the VMM
  - The shim catches this and stores registers
    - Clear the registers to prevent side channels
Complex Syscalls

- Some syscalls are easy
  - No side effects
  - Nice, getpid, sync

- Others, less so…
  - Pipe, r/w (kernel sees zero data, VMM needs to fix)
  - Clone – must keep cloaked cloaked
  - Fork
  - Signal Handling – cannot signal cloaked code arbitrarily, so VMM must signal shim
Performance

- Microbenchmarks
  - Not so hot 15-60%
  - Although a lot better than Proxos

- Application Benchmarks
  - SPEC isn’t so bad
  - High bandwidth hits some bottlenecks
  - Why?
Intel® Software Guard Extensions (SGX)

- Security critical code isolated in enclave
- Only CPU is trusted
  - Transparent memory encryption
  - 18 new instructions
- Enclaves cannot harm the system
  - Only unprivileged code (CPU ring3)
  - Memory protection
- Designed for Multi-Core systems
  - Multi-threaded execution of enclaves
  - Parallel execution of enclaves and untrusted code
  - Enclaves are interruptible
- Programming Reference available

[McKeen et al, Hoekstra et al., Anati et al., HASP’13]
SGX Enclaves

- Enclaves are isolated memory regions of code and data
- One part of physical memory (RAM) is reserved for enclaves
  - It is called Enclave Page Cache (EPC)
  - EPC memory is encrypted in the main memory (RAM)
  - Trusted hardware consists of the CPU-Die only
  - EPC is managed by OS/VMM

RAM: Random Access Memory
OS: Operating System
VMM: Virtual Machine Monitor (also known as Hypervisor)
SGX Memory Access Control

- Access control in two direction
  - From enclaves to “outside“
    - Isolating malicious enclaves
    - Enclaves needs some means to communicate with the outside world, e.g., their “host applications”
  - From “outside“ to enclaves
    - Enclave memory must be protected from
      - Applications
      - Privileged software (OS/VMM)
      - Other enclaves

OS: Operating System
VMM: Virtual Machine Monitor (also known as Hypervisor)
SGX MAC from enclaves to “outside”

- From enclaves to “outside”
  - All memory access has to conform to segmentation and paging policies by the OS/VMM
    - Enclaves cannot manipulate those policies, only unprivileged instructions inside an enclave
  - Code fetches from inside an enclave to a linear address outside that enclave will result in a #GP(0) exception

MAC: Memory Access Control
#GP(0): General Protection Fault
SGX MAC “outside” to enclaves

- From “outside” to enclaves
  - Non-enclave accesses to EPC memory results in abort page semantics
  - Direct jumps from outside to any linear address that maps to an enclave do not enable enclave mode and result in a about page semantics and undefined behavior
  - Hardware detects and prevents enclave accesses using logical-to-linear address translations which are different than the original direct EA used to allocate the page. Detection of modified translation results in #GP(0)

MAC: Memory Access Control
EA: Enclave Access
#GP(0): General Protection Fault
What if we only want to run one high-integrity user-process?

SGX – Create Enclave

1. Create App
2. Create app certificate (includes HASH(App) and Client PK)
3. Upload App to Loader
### SGX – Create Enclave

1. Create App
2. Create app certificate (includes HASH(App) and Client PK)
3. Upload App to Loader
4. Create enclave
5. Allocate enclave pages

**Hardware**

**User space**

**Operating system**

**Loader**

**Enclave**

**SGX driver**

**Client**

SK/PK

---


Slide Nr. 20, Lecture Embedded System Security, SS 2014

Systems and Internet Infrastructure Security (SIIS) Laboratory

Trusted Execution Environments / Intel SGX
What if we only want to run one high-integrity user-process?

Trusted Execution Environments / Intel SGX

1. Create App
2. Create app certificate (includes HASH(App) and Client PK)
3. Upload App to Loader
4. Create enclave
5. Allocate enclave pages
6. Load & Measure App
7. Validate certificate and enclave integrity

SGX – Create Enclave

Hardware
What if we only want to run one high-integrity user-process?

**SGX – Create Enclave**

1. Create App
2. Create app certificate (includes HASH(App) and Client PK)
3. Upload App to Loader
4. Create enclave
5. Allocate enclave pages
6. Load & Measure App
7. Validate certificate and enclave integrity
8. Generate enclave K key
9. Protect enclave

---

Slide Nr. 22, Lecture Embedded System Security, SS 2014

Trusted Execution Environments / Intel SGX
A Problem

- My computer is running a process
- It makes a request to your computer
  - Asks for some secret data to process
  - Provides an input you depend on
- How do you know it is executing correctly?

- Example
  - ATM machine is uploading a transaction to the bank
  - How does the bank know that this ATM is running correctly, so the transaction can be considered legal?
Question You Might Ask?

• Who owns the remote computer?
  ‣ Does this tell you whether the computer has malware?

• Is the computer protected from ever running malware?
  ‣ How would we know this?

• What is actually running on the computer?
  ‣ How can get this information securely?

• Would any of these things enable you to determine whether to supply your personal information to the remote computer?
What would you do?

• Proof by authority (Certificates)
  ‣ Validate the source of messages from the remote system
  ‣ Tells you who and what (maybe), but how

• Constrain the system (Secure Boot)
  ‣ Remote system boots using only trusted software
  ‣ Is only running if secure

• Inspect the runtime state (Authenticated Boot)
  ‣ Remote system produces record of software run
  ‣ You validate whether you trust the software
Secure Boot

• Why not just boot from a floppy (DVD now)?
Secure Boot

• Check each stage in the boot process
  ‣ Is code that you are going to load acceptable?
  ‣ If not, terminate the boot process

• Must establish a **Root-of-Trust**
  ‣ A component trusted to speak for the correctness of others
  ‣ Assumed to be correct because errors are **undetectable**
AEGIS

- AEGIS architecture (1997)
  - ROM checks the BIOS
  - BIOS checks expansion ROMs and boot block
  - Boot Loader checks the OS

- What is the root of trust?
- What can it verify?
- How do we know it booted securely?
Authenticated Boot

- Secure boot enforces requirements and uses special hardware to ensure a specific system is booted
  - Implied verification (Good because it is)
- By contrast, we can measure each stage and have a verifier authenticate the correctness of the stage
  - Verifier must know how to verify correctness
  - Behavior is uncertain until verification
- What is root-of-trust for authenticated boot?
Secure v Authenticated Boot

- Odd implications of each

- **Secure boot** enables you to tell if your machine is secure
  - But remote parties cannot tell

- **Authenticated boot** enables remote parties to tell if your machine is secure
  - But you cannot tell by using it yourself
Trusted Computing

• The Trusted Platform Module (TPM) brought authenticated boot into the mainstream

• Essentially, the TPM offers few primitives
  ‣ Measurement, cryptography, key generation, PRNG
  ‣ Controlled by physical presence of the machine
  ‣ BIOS is Core Root of Trust for Measurement (CRTM)

• Spec only discussed how to measure early boot phases and general userspace measurements
Trusted Computing

Trusted Computing Group Architecture

- **Execution Flow**
  - Application code
  - OS code
  - OS Loader code
  - CRTM code
  - TBB + Roots of Trust

- **Measurement Flow**
  - TCG-based Integrity Measurement Architecture
    - Defined by Grub (IBM Tokyo Research Lab)
    - Defined by TCG (Platform specific)
  - Platform Configuration Registers 0-23

- **TCG-based**
  - Measurement, cryptography, key generation, PRNG
  - Controlled by physical presence of the machine
  - BIOS is Core Root of Trust for Measurement (CRTM)

- Spec only discussed how to measure early boot phases and general userspace measurements.
• What would you measure in the following system to prove it is running correctly to remote verifiers?

Example: Web Server

- **Executables** (Program & Libraries)
  - apachectl, httpd, java, ...
  - mod_ssl.so, mod_auth.so, mod_cgi.so, ...
  - libc-2.3.2.so libjvm.so, libjava.so, ...

- **Configuration Files**
  - httpd.conf, html-pages,
  - httpd-startup, catalina.sh, servlet.jar

- **Unstructured Input**
  - HTTP-Requests
  - Management Data
Trusted Computing

- How would you measure code and static files to prove system is running correctly to remote verifiers?

![Diagram of Trusted Computing](image)

- System Properties
- Attested System
- Analysis
Linux Integrity Measurement

**Measurement list aggregation:**

- **Compute** 160bit-SHA1 over the contents of the data (measurement)
- **Adjust** Protected hw Platform Configuration Register (PCR) to maintain measurement list integrity value
- **Add** measurement to ordered measurement list
  - Executable content is recorded before it impacts the system
  - That is, before it can corrupt the system

```
PCR_{0} := 0

PCR_{k} = SHA1( PCR_{k} \| \text{new measurement})

\text{System-start} \rightarrow k \rightarrow k+1
```

Measurement List Integrity Value
IMA Implementation

- Place hooks throughout Linux kernel
  - Later added more general LIM hooks
- Extend **TPM PCR** at file load-time
  - PCR = SHA1(File || PCR)
- Extend kernel-stored **measurement list**
  - List of SHA1 hashes taken by kernel
  - Including those requested by user space applications
- Generate attestation using TPM hardware
  - \( S(K_{TPM}^{-}, PCR + \text{nonce}) \)
What if we only want to run one high-integrity user-process?

SGX – Remote Attestation

1. Verifier sends nonce
2. Generate Report = (HASH(Enclave1), ID-QuotingEnclave, nonce)

- Verifier
- Quoting Enclave
- Enclave
- User space
- Operating system

Trusted Execution Environments

Slide Nr. 33, Lecture Embedded System Security, SS 2014

Systems and Internet Infrastructure Security (SIIS) Laboratory
What if we only want to run one high-integrity user-process?

**SGX – Remote Attestation**

1. Verifier sends nonce
2. Generate Report = (HASH(Enclave1), ID-QuotingEnclave, nonce)
3. Pass Report to Quoting Enclave
SGX – Remote Attestation

1. Verifier sends nonce
2. Generate Report = (HASH(Enclave1), ID-QuotingEnclave, nonce)
3. Pass Report to Quoting Enclave
4. Quoting Enclave verifies Report
5. Signs Report with “Platform Key”
SGX – Remote Attestation

1. Verifier sends nonce
2. Generate Report = (HASH(Enclave1), ID-QuotingEnclave, nonce)
3. Pass Report to Quoting Enclave
4. Quoting Enclave verifies Report
5. Signs Report with “Platform Key”
6. Signed Report is send to verifier
Take Away

- Problem: Do not want to trust systems software
  ‣ Idea: Cloak memory from system software

- Overshadow – Virtualization-based implementation of cloaking

- Intel SGX
  ‣ Hardware-based memory cloaking
  ‣ Hardware-based attestation to prove properties

- VC3 – Application of Intel SGX for cloud computing