463.6.1 Trusted Computing

Computer Security II
CS463/ECE424
Presented by Hyun Bin Lee
University of Illinois
• MP1 Inference Algorithm Contest Presentations
• Trusted Platform Module (TPM)
• Introduction to Secure Enclaves (Intel SGX)
• MP3
History of Tamper Resistance

- History of Tamper Resistance goes back centuries

Examples:
- Weight down code books on naval ships
- Codes and, more recently, the keys for wartime cipher machines have been printed in water soluble ink
- IBM 3848 and the VISA security module (1980s)

Motivation of Secure Hardware:
- Prevent powerful adversaries from getting secret
463.6.1.1
Trusted Platform Module (TPM)

Based on Slides by Stallling and Brown
Trusted Computing

• Trusted Computing is the term developed by the Trusted Computing Group (TCG)

• Trusted Computing allows “a piece of data to dictate what Operating System and Application must be used to open it”
Trusted Platform Module (TPM)

• hardware module at heart of *hardware/software* approach to Trusted Computing

• uses a TPM chip

• has three basic services:
  – authenticated boot, certification, encryption
Secure/Authenticated Boot Service

Boot Loader \rightarrow OS Kernel \rightarrow Kernel module \rightarrow ...
Q: How can we verify that our boot loader is not tampered?
Secure/Authenticated Boot Service

A: Hashing!

SHA-1(Boot Loader) \neq SHA-1(Boot Loader)
Q: How can we verify Boot Loader and OS Kernel are both not tampered?
Secure/Authenticated Boot Service

\[ \text{SHA-1(Boot Loader)} \quad || \quad \text{SHA-1(OS Kernel)} \]
Secure/Authenticated Boot Service

Secure boot

Secure/Authenticated Boot Service

Figure: Intel SGX Explained; Victor Costan and Srini Devadas
once a configuration is achieved and logged, the TPM can certify configuration to others (Attestation)

challenge value in certificate assures timeliness

provides a hierarchical certification approach
Encryption Service

- encrypts data so that it can only be decrypted by a machine with a certain configuration
- TPM maintains a master secret key unique to machine
- can extend scheme upward
Example: Windows BitLocker Drive Encryption

- Windows 10 Operating System uses TPM on multiple components including its drive encryption
- BitLocker relies on the TPM to allow the use of a key only when startup occurs in an **expected** way
- Volume remains confidential when:
  - Different OS booted from USB device
  - Hard disk is lost or stolen when powered off
The concept of Trusted Computing is developed and promoted by the Trusted Computing Group.

- Root of trust
- Anti-competitive effect
463.6.1.2
Introduction to Secure Enclaves

Based on Slides from
1. Intel SGX (Reference Number: 332680-002) presented at ISCA 2015
2. Intel’s SGX In-depth Architecture by Syed Kamran Haider with Hamza Omar, Masab Ahmad, Chenglu Jin, and Marten van Dijk
Why Aren’t Compute Devices Trustworthy?

Protected Mode (rings) protects OS from apps ...

App

Privileged Code
Why Aren’t Compute Devices Trustworthy?

Protected Mode (rings) protects OS from apps ...

... and apps from each other ...
Why Aren’t Compute Devices Trustworthy?

Protected Mode (rings) protects OS from apps ...

App

x

x

Malicious App

Bad Code

... and apps from each other ...

... UNTIL a malicious app exploits a flaw to gain full privileges and then tampers with the OS or other apps

Privileged Code

attack

Apps not protected from privileged code attacks
Why Aren’t Compute Devices Trustworthy?

Protected Mode (rings) protects OS from apps ...

App

Malicious App

Info

Bad Code

Bad Code

Privileged Code

... and apps from each other ...

... UNTIL a malicious app exploits a flaw to gain full privileges and then tampers with the OS or other apps

Apps not protected from privileged code attacks
Reduced Attack Surface with the Intel SGX

Complexity of modern systems force programmers to inspect a large code base for vulnerability detections.

- Application code
- OS code
- Virtual Machine Manager Code
Reduced Attack Surface with the Intel SGX (cont.)

Application gains ability to defend its own secrets
- Smallest attack surface (App + processor)
- Malware that subverts OS/VMM, BIOS, Drivers etc. cannot steal app secrets

Familiar deployment model
- Platform integration not a bottleneck to deployment of trusted apps

Intel becomes the root of trust in this security model.
SGX enabled processors offer two crucial properties.

- **Isolation**: Each enclave’s environment is isolated from the untrusted software outside the enclave, as well as from other enclaves.
- **Attestation**: A software attestation scheme that allows a remote party to authenticate the software running inside an enclave.

We will focus on the isolation property for the remainder of the lecture.
How Secure Enclaves Work

- Application is built with **trusted** and **untrusted** parts
- **Trusted** and **untrusted** parts are explicitly separated by app developers

![Diagram showing trusted and untrusted parts of an application](image)
How Secure Enclaves Work

- App runs & creates **enclave** which is placed in **trusted memory**
How Secure Enclaves Work

- Trusted function (ECALL) is called
- Code running inside enclave sees data in clear
- External access to data is denied
How Secure Enclaves Work

- Function returns; enclave data **remains** in trusted memory
Enclaves and Objects

• An Enclave is like a class.
• It can maintain its own state like Objects do with private variables.
• ECALLs is like a method, and OCALL is like a return.
• Enclave’s property of isolation is like Object Oriented Programming’s principle of encapsulation.
Intel provides a Software Development Kit (SDK) for C/C++ programmers


“It is a collection of APIs, libraries, documentation, sample source code, and tools that allows developers to create and debug Intel SGX enabled applications.”
• SGX allows a subset of C/C++ library functions to be used inside the Enclave
• List of allowed/disallowed library functions are defined at the Intel SGX Developer Reference

https://download.01.org/intel-sgx/linux-2.4/docs/Intel_SGX_Developer_Reference_Linux_2.4_Open_Source.pdf
Example: Hardening an Application with SGX

1. Identify Sensitive Data Objects and Code
Hardening an Application with SGX

1. Identify Sensitive Data Objects and Code
2. Use SGX Tools to create an Enclave Module (Shared Object)
1. Identify Sensitive Data Objects and Code
2. Use SGX Tools to create an Enclave Module (Shared Object)
3. Move Sensitive Data Objects and Code to the Enclave
   - Change only affects Application Modules which host sensitive data and code.
   - Application now consists of an Untrusted Component and a Trusted Component.
4. Identify Entry Points into Trusted Code
4. Identify Entry Points into Trusted Code

5. Use Tools to Create Glue Code
   - Untrusted component can now call into trusted component (ECall)
Hardening an Application with SGX

6. Need an OS Call from the Enclave?
   • Use Tools to Create Glue Code for Enclave calls to the Application (OCall)

7. Use SGX Libraries in the Application and the Enclave
   • Call loader API to Load the Enclave
SGX does not defend against software side-channel adversary!

**Software Side-Channel Adversary**: An adversary who can gather statistics from the CPU regarding execution and may be able to use them to deduce characteristics of the software being executed (side-channel analysis)

1. Gather power statistics
2. Gather performance statistics including platform cache misses
3. Gather branch statistics via timing
4. Gather information on pages accessed via page tables
Data scientists often use interpreted languages (e.g. R, python and etc.) to perform data analysis on sensitive data

R provides convenient libraries for the analysis

How would one know if the R functions are vulnerable against side channel attacks?
For high-level languages, we must consider not only source code but also interpreter code as well.
High-level Languages Workflow

- **R Source**
  - is interpreted in

- **R Library Functions**
  - which are compiled as

- **Object Code**
  - which, when run, provide the

- **Computation Result**
Input: x and y are private arrays

(a) C source code snippet of:

```c
if (x1 == 0 || x2
    pa[i] = 0;
else if (x1 == NA_
    NA_LOGICAL)
    pa[i] = NA_LOGICAL
else
    pa[i] = 1;
```

Cycle differences are evident when evaluations are repeated for large iterations.
463.6.1.3
MP3
• In MP2, we learned how to do a Private Set Intersection with homomorphic encryption.
• For this machine problem, we will learn how to run logistic regression, a machine learning algorithm, on a secure enclave.
Question: Can data owners collaborate and perform a multi-party computation without explicitly sharing their data?

• Each group will have one of three disjoint datasets (Iris plant dataset with 30 samples each)
• Four features were measured from each sample, the length and width of sepals and petals.
• Three groups will send their datasets to a server that runs an enclave application. This enclave application receives datasets and perform a machine learning algorithm (logistic regression) on the datasets.
1. Makes a TLS connection with a server
2. Sends a dataset (dataset_{i}.csv) to the server
3. Receives the model from the server as a reply
4. Tests the model on a test set (testset.csv)
Server

1. Listens to TLS connections
2. Receives datasets from three clients
3. Trains the model **inside the Enclave**
4. Sends the model result back to clients
Wolfssl

- Embedded, lightweight SSL library for IoTs, embedded systems, or real-time operating systems
- Provides support for Intel SGX
- We will use their SGX Linux client & server example as a template for our MP

https://github.com/wolfSSL/wolfssl-examples
The provided template has following limitations.

1. Single threaded

2. No Filesystem

3. Untrusted Code Must Load Private Key/Certificate Chain

   For this MP, protection of key/cert chain is out of scope.
wolfssl template

• Composed of two folders
• mp3/untrusted
  - App.c
  - server-tls.c
  - client-tls.c
• mp3/trusted
  - Wolfssl_Enclave.c
  - Wolfssl_Enclave.edl
Logistic Regression

- Features (90 × 4 matrix): $X$
- Labels (90 × 1 matrix): $Y$
- Weights (4 × 1 matrix): $W$

- Model: $z = \sum_{i=0}^{3} w_i x_i$
Training the model

0. Initialize Weights: $W_0 = \overrightarrow{0}$

1. Get probability for each row in $X$

Sigmoid($z$):

$$P(class = 1) = \frac{1}{1 + e^{-z}}$$

Predictions ($90 \times 1$ matrix): $S(Z)$
Training the model (cont.)

2. Compute the Gradient

\[ G = X^T (S(Z) - Y) \]

3. Update Weight

\[ W_{i+1} = W_i - 0.01 (G/90) \]

4. Repeat procedure 1, 2 and 3 for 500 iterations
Testing your model

Test your model by classifying each sample in test set (testset.csv) with the model

\[ z = w_0 x_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 \]

\[ \text{Sigmoid}(z) = P(\text{class} = 1) = \frac{1}{1 + e^{-z}} \]

Classify sample to be Virginica (1) if \( P \geq 0.5 \)

Not Virginica (0) otherwise.

Measure accuracy by comparing your classification results with the ground truth labels.
• Send an email to the course staff (lee559@illinois.edu) once you formed a group for the MP so that we can designate SGX resources for the group.
• Please include [CS463 MP3] in your email title.
• Find two other groups with different datasets for Checkpoint 2 early!
Discussion Questions

• Should we accept Intel as a root of trust?

• What are some use cases for Trusted Computing in addition to disk encryption (e.g. Bitlocker)?