Language Support for Secure Software Development with Enclaves

Aditya Oak  
TU Darmstadt

Amir M. Ahmadian  
KTH Royal Institute of Technology

Musard Balliu  
KTH Royal Institute of Technology

Guido Salvaneschi  
University of St.Gallen
Trusted Execution Environments (TEEs)

- Recent Privacy Enhancing Technology
- ‘Enclaves’ protected by hardware
- Enclaves are opaque to the OS
- Private computations on cloud hosts
Programming with TEEs

Application

Code outside enclave

data in

result

Enclave
Problems with using TEEs

• Manual application partitioning

• Program the enclave – non-enclave interface

• Application partitioning may not preserve security guarantees

Compiler can do these !!!
• Language-level abstractions
• Automatic application partitioning
• A static type system for information flow control
• Robustness guarantees against active attackers
JE Programming Model: Language-level abstractions

Annotations:
@Enclave
@Gateway
@Secret

Methods:
endorse
declassify

```java
// PasswordChecker.java

class PasswordChecker {
    static String password = ...;

    @Gateway
    public static boolean checkPassword(String guess) {
        String guessE = endorse(guess);
        boolean result = guessE.equals(password);
        return declassify(result);
    }
}

// Main.java

class Main {
    public static void main(String[] args) {
        String guess = ... // read guess from stdin
        PasswordChecker.checkPassword(guess);
    }
}
```
JE: Program partitioning and attacker Model

JE code → JE compiler

JE compiler → Enclave

Enclave → Outside the enclave

Outside the enclave → Non-enclave attacks
JE: Attacker models

- **Stronger Attackers**
  - They can modify data- and code-memory

- **Robustness**
  - Controlling the release of secret data (declassification)

1. Data-memory attacker (HAA)
2. Data- and code-memory attacker (HRAA)
J_E: Attacker models

- HAA Attacker
  - In control of data-memory
  - Change data outside of enclave
    - Parameters of gateways

```java
@Enclave
class PasswordChecker {
    @Secret static String password;

    @Gateway
    public static boolean checkPassword(String guess) {
        boolean result = guess.equals(password);
        return declassify(result);
    }
}
```

```java
class Main {
    public static void main(String[] args) {
        String guess = ... // read guess from stdin
        PasswordChecker.checkPassword(guess);
    }
}
```
Robustness under HAA

program $S[\mathcal{\cdot}]$ is robust w.r.t HAA attacker $A$ if for all $\sigma_1, \sigma_2, \tilde{a}_1, \tilde{a}_2$:

$$N \vdash_\delta \langle S[\tilde{a}_1], \sigma_1 \rangle \approx_A N \vdash_\delta \langle S[\tilde{a}_1], \sigma_2 \rangle \Rightarrow$$

$$N \vdash_\delta \langle S[\tilde{a}_2], \sigma_1 \rangle \approx_A N \vdash_\delta \langle S[\tilde{a}_2], \sigma_2 \rangle$$

- We use holes $\mathcal{\cdot}$ outside of the enclave to capture the effects of this attacker
- $S[\mathcal{\cdot}]$ is the program
- Attacker’s code $a$ will be placed in these holes
  - It can only affect data-memory
Robustness under HAA

- Parameters of gateways are under attacker’s control
- They are untrusted
- Untrusted values should not affect declassification

```java
@Enclave
class PasswordChecker {
    @Secret static String password;

    @Gateway
    public static boolean checkPassword(String guess) {
        boolean result = guess.equals(password);
        return declassify(result);
    }
}
```

```java
class Main {
    public static void main(String[] args) {
        String guess = ... // read guess from stdin
        PasswordChecker.checkPassword(guess);
    }
}
```
Enforcing Robustness under HAA

- Type system ensures that:
  - Only trusted values can be declassified
  - Declassification can only happen under trusted context

\[
\frac{\Gamma, \Pi \vdash_\delta e : (\ell, d) \quad \ell \sqsubseteq \langle S, T \rangle \quad pc_\ell \sqsubseteq \langle P, T \rangle \quad \delta(x) = E}{\text{T-DECLASSIFY} \quad pc, \Gamma, \Pi \vdash_\delta x := \text{declassify}(e) : \Gamma[x \mapsto \ell \cap \langle P, T \rangle], \Pi[x \mapsto T]}
\]
Theorem 1

If $pc, \Gamma, \Pi \vdash_\delta S[\bullet]: \Gamma', \Pi'$ then $S[\bullet]$ satisfies robustness under HAA.

- A well-typed program $pc, \Gamma, \Pi \vdash_\delta S[\bullet]: \Gamma', \Pi'$ is robust against HAA attacker.
JE: Attacker models

- **HRAA Attacker**
  - In control of data-memory and code-memory
  - Can change data and control flow outside of enclave
    - Parameters of gateways
    - Order of calling gateways
    - Frequency of calling gateways

```java
@Enclave
class FooClass {
    @Secret static int secret1, secret2;
    static boolean releaseTrigger = false;

    @Gateway
    public static void bar() {
        releaseTrigger = true;
    }

    @Gateway
    public static int foo() {
        int res = 0;
        if (releaseTrigger) {
            res = declassify(secret1);
        } else {
            res = declassify(secret2);
        }
        return res;
    }
}
```
Robustness under HRAA

- `foo;bar` or `bar;foo`
  - Can lead to different values
  - Attacker learns more by changing the order of gateway calls

```
@Enclave

class FooClass {

  @Secret static int secret1, secret2;
  static boolean releaseTrigger = false;

  @Gateway
  public static void bar() {
    releaseTrigger = true;
  }

  @Gateway
  public static int foo() {
    int res = 0;
    if (releaseTrigger) {
      res = declassify(secret1);
    } else {
      res = declassify(secret2);
    }
    return res;
  }
}
```
We define the program under HRAA attacker as a sequence of gateway calls:

\[ S'[\textbullet] ::= S'_1[\textbullet]; S'_1[\textbullet] | [\textbullet]; x := C.m(\overline{p}) \]

- \( S'[\textbullet] \) models attacker’s control over **code-memory**
- \( \textbullet\textbullet \) and attacker’s code \( a \) model attacker’s control over **data-memory**
Robustness under HRAA

Program $S[\hat{\bullet}]$ is robust w.r.t. HRAA attacker $A$ if for all $\sigma_1, \sigma_2, \hat{a}_1, \hat{a}_2$ and for all $S'[\hat{\bullet}]$:

$$N \vdash_\delta \langle S[\hat{a}_1], \sigma_1 \rangle \approx_A N \vdash_\delta \langle S[\hat{a}_1], \sigma_2 \rangle \Rightarrow$$

$$N \vdash_\delta \langle S'[\hat{a}_2], \sigma_1 \rangle \approx_A N \vdash_\delta \langle S'[\hat{a}_2], \sigma_2 \rangle$$

- $S'[\hat{\bullet}]$ is the attacker program
  - A list of gateway calls
We extend the type system to account for this attacker.

**Theorem 2**

If $\text{pc}, \Gamma, \Pi \vdash_\delta S[\hat{\tau}]: \Gamma', \Pi'$ with regard to $\Sigma$ and $G^D$, then $S[\hat{\tau}]$ satisfies robustness under $\text{HRAA}$. 
More Investigated Features

- Endorsement
- Flow Sensitive Variables
- Delayed Declassification
**J_E Implementation and Workflow**

1. **J_E Code**
2. **Static Security Analysis**
3. **Conversion to Jif**
4. **Jif Compiler**
   - **Enclave Code**
   - **Non-Enclave Code**
   - **Enclave Interface (Java RMI)**

**Phase 2**
- Intel SGX
- Regular
- JVM
Evaluation

• **Password Checker**: Password stored inside the enclave

• **Updatable-Password Checker**: Password inside the enclave, modifiable from outside

• **Medical Data Processing**: Decrypt and process data inside the enclave
A programming model for secure programming with enclaves

Abstractions for code placement and data security attributes

Type system to defend against strong realistic attacks

More in the paper

Language Support for Secure Software Development with Enclaves

Aditya Oak
TV Demme

Amir M. Ahmadian
KTH Royal Institute of Technology

Mouad Balia
KTH Royal Institute of Technology

Guido Salvaneschi
University of Siena

Abstract—Confidential computing is a promising technology for securing code and data across untrusted hosts, e.g., the cloud. Many hardware vendors offer different implementations of Trusted Execution Environments (TEEs). A TEE is a hardware-supported execution environment that allows performing confidential computations over sensitive data on untrusted hosts. Despite the rapid adoption of TEEs, many software applications still run on bare metal, which is a significant roadblock to the adoption of TEEs. This paper presents Secure Enclave Java (JE), a secure, enclaved Java for cloud deployments. JE includes a new runtime, a security type system, and a security type checker that verifies that a Java program is secure. The JE runtime executes Java programs in an isolated enclave running in the user space, and the security type checker enforces that programs are written in a secure manner. The JE runtime is built on top of the Java Virtual Machine (JVM) and the Secure Enclave Java (JE) implementation is based on Intel’s Software Guard Extensions (SGX) [1].}

I. INTRODUCTION

Confidential computing includes recent technologies that promise to increase the security and privacy of data. The main idea is to use hardware-enforced isolation to prevent access to sensitive data. However, current technologies do not provide an easy-to-use programming model for enclaves. In this paper, we introduce Secure Enclave Java (JE), a new programming model for secure programming with enclaves. JE is designed to be easy to use and to provide a secure programming model for enclaves. JE includes a new runtime, a security type system, and a security type checker that enforces that programs are written in a secure manner. The JE runtime is built on top of the JVM, and the security type checker enforces that programs are written in a secure manner.