Automatic Enforcement of Expressive Security Policies using Enclaves

Anitha Gollamudi  Stephen Chong
Harvard University

OOPSLA’16
Application-level Security
Application-level Security

```plaintext
x = secret_info
//compute with x
...
x = public_info
output x
```
Example Programs

Encryption

```plaintext
x = key
//encrypt with x
encrypt(message, x)
x = 0
output x
```

Signature

```plaintext
x = key
//sign with x
sign(message, x)
x = 0
output x
```
Application-level Security

x = secret_info
//compute with x
...

x = public_info
output x
Application-level Security

\[ x = \text{secret}_\text{info} \]

//compute with x

...

\[ x = \text{public}_\text{info} \]

output x
Application-level Security

x = secret_info
//compute with x
...
x = public_info
output x

Language-based Security
Application-level Security

x = secret_info
//compute with x
...
x = public_info
output x

Language-based Security

Real World Scenario: Application running a real machine
Application-level Security

x = secret_info
//compute with x
...

x = public_info
output x

Language-based Security
Application-level Security

```plaintext
x = secret_info
//compute with x
...
x = public_info
output x
```

Operating System

Language-based Security
Application-level Security

\[ x = \text{secret}_\text{info} \]
\[ //\text{compute with } x \]
\[ ... \]
\[ x = \text{public}_\text{info} \]
\[ \text{output } x \]

Language-based Security
Application-level Security

```python
x = secret_info
//compute with x
...
```

Language-based Security

Operating System
Application-level Security

x = secret_info
//compute with x
...

Language-based Security

Operating System
Application-level Security

Question: How to enforce application security guarantees against low-level attackers?
Application-level Security

Our Solution

- Extend the Language-based Security with hardware protection mechanisms (Intel SGX, ARM TrustZone)
- Enforce security against low-level attackers

```plaintext
x = secret_info
//compute with x
...
x = public_info
output x
```
Hardware Protection Mechanisms

- Intel SGX enables applications to build enclaves: protected memory containers
- Isolated execution
- Restricted access
- ARM TrustZone

```plaintext
x = secret_info
//compute with x
...
x = public_info
output x
```
Overview of Enclaves

Program

```
x = secret_info
//compute with x
...
```

```
x = secret_info
//compute with x
...
```

```
x = public_info
output x
```

```
x = public_info
output x
```

Execution

<table>
<thead>
<tr>
<th>MEMORY</th>
</tr>
</thead>
<tbody>
<tr>
<td>CODE</td>
</tr>
<tr>
<td>x = secret_info //compute with x</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>x = public_info output x</td>
</tr>
<tr>
<td>kill</td>
</tr>
</tbody>
</table>

```
x = secret_info
//compute with x
...
```

```
x = secret_info
//compute with x
...
```

```
x = public_info
output x
```

```
x = public_info
output x
```

```
x = secret_info
//compute with x
...
```

```
x = secret_info
//compute with x
...
```

```
x = public_info
output x
```

```
x = public_info
output x
```
Overview of Enclaves

Program

```plaintext
enclave {
  x = secret_info
  //compute with x
  ...
}

x = public_info
output x
```

Execution

| MEMORY |
|------------------|------------------|
| CODE             | DATA             |
| x = secret_info  | secret_info      |
| //compute with x |                  |
| x = public_info  | public_info      |
| output x         |                  |
| kill             |                  |
| kill             |                  |
Overview of Enclaves

<table>
<thead>
<tr>
<th>MEMORY</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CODE</strong></td>
</tr>
<tr>
<td>$x = secret_info$</td>
</tr>
<tr>
<td>//compute with $x$</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>$x = public_info$</td>
</tr>
<tr>
<td>//compute with $x$</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

Program

```
enclave {
    x = secret_info
    //compute with x
    ...
}
```

$x = public_info$
output x
kill

Execution
Overview of Enclaves

```c
enclave {
    x = secret_info //compute with x ...
}

x = public_info output x

kill
```

<table>
<thead>
<tr>
<th>MEMORY</th>
</tr>
</thead>
<tbody>
<tr>
<td>CODE</td>
</tr>
</tbody>
</table>
| x = secret_info //compute with x ...
| DATA   |
| secret_info |
| x = public_info output x
| public_info |
| kill    |

Program

Execution

Enclave Memory
Overview of Enclaves

enclave {
    x = secret_info
    //compute with x
    ...
}

x = public_info
output x

kill

Program

<table>
<thead>
<tr>
<th>MEMORY</th>
</tr>
</thead>
<tbody>
<tr>
<td>CODE</td>
</tr>
<tr>
<td>--------</td>
</tr>
<tr>
<td>x = secret_info</td>
</tr>
<tr>
<td>//compute with x</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CODE</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>x = public_info</td>
<td>public_info</td>
</tr>
<tr>
<td>//compute with x</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

kill (1)
Overview of Enclaves

Program

```plaintext
enclave {
    x = secret_info
    //compute with x
    ...
}

x = public_info
output x
```

Execution

```
kill
```

```
x = public_info
output x
```

```
kill (1)
```

```
public_info
```

```
```

<table>
<thead>
<tr>
<th>MEMORY</th>
</tr>
</thead>
<tbody>
<tr>
<td>CODE</td>
</tr>
<tr>
<td>--------</td>
</tr>
<tr>
<td>x = public_info</td>
</tr>
<tr>
<td>kill (1)</td>
</tr>
</tbody>
</table>

```
Overview of Enclaves

enclave {
  x = secret_info
  //compute with x
  ...
}

x = public_info
output x

Why are enclaves useful for enforcing security?
Overview of Enclaves

```
enclave {
    x = secret_info
    //compute with x
    ...
}

x = public_info
output x

kill
```

<table>
<thead>
<tr>
<th>MEMORY</th>
<th>CODE</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>x = secret_info</td>
<td>secret_info</td>
</tr>
<tr>
<td></td>
<td>//compute with x</td>
<td></td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td></td>
<td>x = public_info</td>
<td>public_info</td>
</tr>
<tr>
<td></td>
<td>output x</td>
<td></td>
</tr>
<tr>
<td></td>
<td>kill</td>
<td></td>
</tr>
</tbody>
</table>

Program

Execution
Overview of Enclaves

<table>
<thead>
<tr>
<th>MEMORY</th>
</tr>
</thead>
<tbody>
<tr>
<td>CODE</td>
</tr>
<tr>
<td>x = secret_info //compute with x</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>x = public_info //compute with x</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>x = public_info</td>
</tr>
<tr>
<td>output x</td>
</tr>
<tr>
<td>output x</td>
</tr>
<tr>
<td>kill</td>
</tr>
<tr>
<td>kill</td>
</tr>
</tbody>
</table>

Program

Execution
Overview of Enclaves

enclave {
  x = secret_info
  //compute with x
  ...
}

Program

x = public_info
output x

Execution

kill

MEMORY

<table>
<thead>
<tr>
<th>CODE</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCESS DENIED!</td>
<td></td>
</tr>
</tbody>
</table>

x = public_info
output x

Enclave Memory

kill

public_info
Overview of Enclaves

```c
enclave {
    x = secret_info
    //compute with x
    ...
}

x = public_info
output x

kill
```

However, enclaves by themselves are insufficient!
Enclaves are Insufficient

```
enclave {
  x = secret_info
  //compute with x
  ...
  output x
}
x = public_info
output x
kill
```

### Program

<table>
<thead>
<tr>
<th>MEMORY</th>
</tr>
</thead>
<tbody>
<tr>
<td>CODE</td>
</tr>
<tr>
<td>x = secret_info</td>
</tr>
<tr>
<td>//compute with x</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>output x</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>output x</td>
</tr>
<tr>
<td>x = public_info</td>
</tr>
<tr>
<td>output x</td>
</tr>
<tr>
<td>kill</td>
</tr>
</tbody>
</table>

### Execution
Enclaves are Insufficient

```c
enclave {
  x = secret_info
  //compute with x
  ...
  output x
}
```

```
x = secret_info
//compute with x
...
output x
```

```
x = public_info
output x
```

Program

```
x = public_info
output x
kill
```

Execution

```
x = secret_info
//compute with x
...
output x
```

```
secret_info
```

```
x = public_info
output x
```

```
public_info
```

```
kill
```
Enclaves are Insufficient

Extend the language-based security mechanisms with enclaves!
IMPE

- An expressive formal calculus
- Extends a standard imperative calculus with
  - Enclaves
  - First-class functions
  - Security policies
    - Express application-specific security requirements
Security Policies

- Confidentiality levels
  - form a linear order
Security Policies

- Confidentiality levels
  - form a linear order

Erased from the system
Security Policies

- Confidentiality levels
  - form a linear order

- Erasure policy: change of confidentiality level during the program execution
  - e.g. secret_info: secret \rightarrow public

Erased from the system
Security Policies

- Confidentiality levels
  - form a linear order
- Erasure policy: change of confidentiality level during the program execution
  - e.g. secret_info:

```plaintext
enclave {
  x = secret_info
  //compute with x
  ...
}
```
```
x = public_info
```
```
set(end)
output x
kill
```
Security Policies

- Confidentiality levels
  - form a linear order
- Erasure policy: change of confidentiality level during the program execution
  - e.g. secret_info:

Security policies are enforced w.r.t. a threat model

```
enclave {
  x = secret_info
  //compute with x
  ...
}
```
```
x = public_info
```
```
set(end)
```
```
output x
```
```
kil
```
<table>
<thead>
<tr>
<th>Attacker</th>
<th>Observe Output</th>
<th>Modify Non-Enclave Memory</th>
<th>Modify Enclave Memory</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Passive</td>
<td>✔️</td>
<td>❌</td>
<td>❌</td>
<td>Network monitoring</td>
</tr>
<tr>
<td>Non-Enclave Active</td>
<td>✔️</td>
<td>✔️</td>
<td>❌</td>
<td>Malware</td>
</tr>
<tr>
<td>Attacker</td>
<td>Observe Output</td>
<td>Modify Non-Enclave Memory</td>
<td>Modify Enclave Memory</td>
<td>Example</td>
</tr>
<tr>
<td>-------------------</td>
<td>----------------</td>
<td>---------------------------</td>
<td>-----------------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>Passive</td>
<td>✔️</td>
<td>❌</td>
<td>❌</td>
<td>Network monitoring</td>
</tr>
<tr>
<td>Non-Enclave Active</td>
<td>✔️</td>
<td>✔️</td>
<td>❌</td>
<td>Malware</td>
</tr>
<tr>
<td>Enclave Active</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>Vulnerabilities in enclave code</td>
</tr>
<tr>
<td>Attacker</td>
<td>Observe Output</td>
<td>Modify Non-Enclave Memory</td>
<td>Modify Enclave Memory</td>
<td>Example</td>
</tr>
<tr>
<td>------------------</td>
<td>----------------</td>
<td>---------------------------</td>
<td>-----------------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>Passive</td>
<td>✔️</td>
<td>✗</td>
<td>✗</td>
<td>Network monitoring</td>
</tr>
<tr>
<td>Non-Enclave Active</td>
<td>✔️</td>
<td>✔️</td>
<td>✗</td>
<td>Malware</td>
</tr>
<tr>
<td>Enclave Active</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>Vulnerabilities in enclave code</td>
</tr>
</tbody>
</table>

When is a program secure against these attackers?
Security

- Formally defined as a non-interference property
  - Public outputs are not influenced by private inputs
  - Parameterized by the kind of attacker
Security

- Formally defined as a non-interference property
  - Public outputs are not influenced by private inputs
  - Parameterized by the kind of attacker

Security against weaker attacker $\iff$ security against powerful attacker
enclave {
    x = secret_info
    //compute with x
    ...
}
x = public_info
set(end)
output x
kill

Security Against Passive Attacker

<table>
<thead>
<tr>
<th>MEMORY</th>
</tr>
</thead>
<tbody>
<tr>
<td>CODE</td>
</tr>
<tr>
<td>DATA</td>
</tr>
</tbody>
</table>

- secret_info
- public_info
enclave {
    x = secret_info
    //compute with x
    ...
}

x = public_info
set(end)
output x
kill

Security Against Non-enclave Active Attacker

```
<table>
<thead>
<tr>
<th>MEMORY</th>
</tr>
</thead>
<tbody>
<tr>
<td>CODE</td>
</tr>
<tr>
<td>x = secret_info</td>
</tr>
<tr>
<td>x = public_info</td>
</tr>
</tbody>
</table>
```

18
Security Against Non-enclave Active Attacker

```plaintext
enclave {
    x = secret_info
    //compute with x
    ...
}

x = public_info
set(end)
output x
kill
```

<table>
<thead>
<tr>
<th>MEMORY</th>
</tr>
</thead>
<tbody>
<tr>
<td>CODE</td>
</tr>
<tr>
<td>--------</td>
</tr>
</tbody>
</table>
| x = secret_info //compute with x ...
|         | secret_info |
| x = public_info
set(end)
output x
kill | public_info |
```
Security Against Non-enclave Active Attacker

enclave {
  x = secret_info
  //compute with x
...
  x = public_info
}
set(end)
output x
kill
Security Against Non-enclave Active Attacker

enclave {
  x = secret_info
  //compute with x
  ...
  x = public_info
}
set(end)
output x
kill

<table>
<thead>
<tr>
<th>MEMORY</th>
</tr>
</thead>
<tbody>
<tr>
<td>CODE</td>
</tr>
</tbody>
</table>
| x = secret_info
  //compute with x | secret_info |
| ... | ... |
| x = public_info | public_info |
| set(end) | output x |
| kill |         |
Enforcing Security

- Security Type System
  - secret information is placed only in enclaves
  - code that manipulates the secret information is placed in the same enclave
Enforcing Security

- **Theorem:** Well-typed IMPE programs are secure against
  - Passive attacker
  - Non-enclave active attacker
Enforcing Security

- **Theorem:** Well-typed IMP programs are secure against
  - Passive attacker
  - Non-enclave active attacker
  - Is the program secure for **enclave active attacker**?
enclave {
    x = secret_info
    //compute with x
    ...
    x = public_info
}

set(end)

output x

kill

Security Against Enclave Active Attacker

<table>
<thead>
<tr>
<th>MEMORY</th>
</tr>
</thead>
<tbody>
<tr>
<td>CODE</td>
</tr>
<tr>
<td>x = secret_info</td>
</tr>
<tr>
<td>x = public_info</td>
</tr>
<tr>
<td>set(end)</td>
</tr>
<tr>
<td>output x</td>
</tr>
<tr>
<td>kill</td>
</tr>
</tbody>
</table>
Security Against Enclave Active Attacker

enclave {
    x = secret_info
    //compute with x
    ...
    x = public_info
}
set(end)
output x

kill
Security Against Enclave Active Attacker

```
enclave {
    x = secret_info
    //compute with x
    ...
    x = public_info
}
set(end)
output x
kill
```

<table>
<thead>
<tr>
<th>MEMORY</th>
<th>CODE</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>x = secret_info</td>
<td>secret_info</td>
</tr>
<tr>
<td></td>
<td>//compute with x</td>
<td></td>
</tr>
<tr>
<td></td>
<td>x = public_info</td>
<td>public_info</td>
</tr>
<tr>
<td>set(end)</td>
<td>output x</td>
<td></td>
</tr>
<tr>
<td></td>
<td>kill</td>
<td></td>
</tr>
</tbody>
</table>
Security Against Enclave Active Attacker

enclave{
  x = secret_info
  //compute with x
  ...
  x = public_info
}
set(end)

output x

- Window of vulnerability

| MEMORY |
|---|---|
| CODE     | DATA    |
| x = secret_info //compute with x | secret_info |
| ...                    | ...                        |
| x = public_info        | public_info                |

set(end)
output x
kill

\[ end \]
\[ secret \]
enclave {
  x = secret_info
  //compute with x
  ...
  x = public_info
}

output x

kill

- Window of vulnerability
Security Against Enclave Active Attacker

- Window of vulnerability

```
enclave {
  x = secret_info
  //compute with x
  ...
  x = public_info
}
set(end)
```

```
enclave {
  output secret_info
}
```

```
kill
```

An attacker can exploit a window of vulnerability by accessing secret_info:

```
MEMORY

<table>
<thead>
<tr>
<th>CODE</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>x = secret_info</td>
<td>secret_info</td>
</tr>
<tr>
<td>//compute with x</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>x = public_info</td>
<td>public_info</td>
</tr>
</tbody>
</table>

set(end)
```
Security Against Enclave Active Attacker

- Window of vulnerability
- Smaller the window, better the security
Enforcing Security against Enclave Active Attackers

- **Theorem:** Well-typed IMP programs are secure against enclave active attacker

  - Only for attacks launched after the enclaves (containing the data to be erased) are killed
Recap
Recap

- Well-typed IMPE programs are:
  - Secure against passive attacker
  - Secure against non-enclave active attacker
  - Partially secure against enclave active attacker
Recap

- Well-typed IMPE programs are:
  - Secure against passive attacker
  - Secure against non-enclave active attacker
  - Partially secure against enclave active attacker

How to partition IMPE programs into enclaves
How to Partition Program into Enclaves?

Trivial solution: Place entire application inside an enclave
How to Partition Program into Enclaves?

Trivial solution: Place entire application inside an enclave

- Increases trusted computing base (TCB)
  - For non-enclave active attacker
    - enclave code is assumed to have no vulnerabilities
  - More enclave code = more assumptions
How to Partition Program into Enclaves?

Trivial solution: Place entire application inside an enclave

- Increases trusted computing base (TCB)
  - For non-enclave active attacker
    - enclave code is assumed to have no vulnerabilities
  - More enclave code = more assumptions
- Increases window of vulnerability
  - Can’t kill an enclave until the end
  - Data to be erased lives longer
How to Partition Program into Enclaves?

Using multiple enclaves:

- Fine-grained partitioning leads to smaller enclaves
  - Reduces the lifetime of data to be erased
- Tedious and error-prone
How to Partition Program into Enclaves?

Using multiple enclaves:

- Fine-grained partitioning leads to smaller enclaves
  - Reduces the lifetime of data to be erased
- Tedious and error-prone

We can automatically infer enclave placement!
Enclave Inference as Constraint Satisfaction
Enclave Inference as Constraint Satisfaction
Enclave Inference as Constraint Satisfaction

- Sensitive data should be in some enclave
- A killed enclave cannot be accessed

Program w/out Enclaves

constraints
Enclave Inference as Constraint Satisfaction

- Sensitive data should be in some enclave
- A killed enclave cannot be accessed

Program w/out Enclaves → Constraints → Solver
Enclave Inference as Constraint Satisfaction

- Sensitive data should be in some enclave
- A killed enclave cannot be accessed

Solution is a well-typed IMPE program
Enclave Inference as Constraint Satisfaction

- Sensitive data should be in some enclave
- A killed enclave cannot be accessed

Multiple Solutions!

Solution is a well-typed IMPE program
Enclave Inference as Constraint Optimization

- Sensitive data should be in some enclave
- A killed enclave cannot be accessed

Program w/out Enclaves → Constraints → Solver → IMPE Program
Enclave Inference as Constraint Optimization

- Sensitive data should be in some enclave
- A killed enclave cannot be accessed

Program w/out Enclaves -> constraints -> Solver -> IMPE Program

Objective Function
Example Objective Functions

1. Minimize the TCB
   - Reduce the number of statements inside enclaves
Example Objective Functions

1. Minimize the TCB
   ‣ Reduce the number of statements inside enclaves

2. Minimize the window of vulnerability
   ‣ Place code and data in as many different enclaves as possible
   ‣ Kill an enclave as soon as possible
Summary
Summary

1. Strong information-flow guarantees using hardware protection mechanisms (enclaves)

2. Automatically infer enclave placement in an application relieving the programmers’ burden