Towards Formal Verification of State Continuity for Enclave Programs
Outline

- Background
- Motivation
- Our approach
- Case study - Sawtooth
- Conclusion
Outline

- **Background**
  - Motivation
  - Our approach
  - Case study - Sawtooth
  - Conclusion
Operating system protects user data by

- Process isolation
- Access control (privileged access to devices)

Assumption
OS is trusted
Is it possible to protect the user data when OS is compromised?
Intel SGX

- Trusted hardware solution to protect confidentiality and integrity of the runtime code and data.
- App is divided into trusted and untrusted code section.
- Hardware encrypted trusted code run inside the protected memory regions (enclaves).
State Continuity

- Classic definition: protected module must resume from the same execution state after TCB (Trusted Computing Base) interrupts
- New TCB modules in SGX context:
  - Enclave memory (local/global variables)
  - Non-volatile memory (monotonic counters)
  - Persistent storage (sealed data)
- New threat model in SGX context:
  - Controls the privileged code (OS and application code)
  - Arbitrary thread and process instantiation
  - Permute, reorder enclave calls
  - Access to ecall or ocall arguments and returns
  - Replay, modify of data in untrusted code
State Continuity for Enclave Programs

- Enclave program states always executes on the expected TCB state under the SGX threat model and TCB interrupts.
Example SGX Application -- Sawtooth

- Permissioned Blockchain Framework
- Consensus algorithm: Proof-of-Elapsed-Time (PoET)
- Leverages Intel SGX for fair node participation
- Each node workflow
  - Signup and register into the blockchain network
  - Participate in the block leader election
## Sawtooth Block Leader Election

**Ecall E1**

<table>
<thead>
<tr>
<th>Election 1</th>
<th>Create Reference Objects</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Generate random wait duration</td>
</tr>
<tr>
<td></td>
<td>Create reference monotonic counter ( \text{MC}_{\text{ref}} )</td>
</tr>
<tr>
<td></td>
<td>Seal the duration and ( \text{MC}_{\text{ref}} )</td>
</tr>
</tbody>
</table>

- Wait random duration

**Ecall E2**

<table>
<thead>
<tr>
<th>Election 2</th>
<th>Verify Proof of Elapsed Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Unseal and verify the sealed object</td>
</tr>
<tr>
<td></td>
<td>Verify elapsed time</td>
</tr>
<tr>
<td></td>
<td>Compare ( \text{MC}_{\text{ref}} )</td>
</tr>
<tr>
<td></td>
<td>PoETCertificate</td>
</tr>
</tbody>
</table>

- Monotonic Counter ++
Sawtooth Expected TCB States

1. Monotonic Counter Value < MC_Ref

   PoETCertificate

2. Monotonic Counter Value = MC_Ref

   PoETCertificate

3. Monotonic Counter Value > MC_Ref

   Abort
What Could Go Wrong?

- Ecall E1
  - Seal
    - Unseal
      - Ecall E1

- Ecall E1
  - Ecall E1
  - Ecall E2

- Ecall E1
  - Ecall E1
  - Ecall E2

- Ecall E1
  - Ecall E1

- Ecall E1
  - Ecall E2

- Ecall E2
Outline

- Background
- **Motivation**
- Our approach
- Case study - Sawtooth
- Conclusion
Maintaining State Continuity is Important

State continuity TCB modules are prevalent in many open SGX applications.

196 open source SGX applications

- 59 -- Sealing
- 15 -- Monotonic Counters
- 05 -- Global variables.
The Research Problem

State continuity properties are difficult to verify in the SGX environment. Why?

Manual efforts is tedious and error prone

1. Clearly understand trusted & untrusted boundary
2. Correct coordination of heterogeneous TCB modules
3. Carefully apply thread synchronization and locks

Is there a systematic approach to verify state continuity?
Outline

- Background
- Motivation
- **Our approach**
- Case study - Sawtooth
- Conclusion
Our Approach

- Use Symbolic Verification Tool -- Tamarin, to verify state continuity property
Key Observation

Cryptographic Protocols and SGX Environment share common features

Key Exchange Protocols

Tamarin MSR and query language

SGX Environment

State Cont. Properties

SGX Thread Model
Our Approach

Our Contribution

- SGX primitives
- Model templates for 3 open source SGX apps

Transform icon cite: https://icon-library.com/icon/transformation-icon-3.html
Model Primitives used in our work

<table>
<thead>
<tr>
<th>SGX primitives</th>
<th>Programming primitives</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Enclave threads</td>
<td>1. Locks</td>
</tr>
<tr>
<td>2. Association network of SGX entities</td>
<td>2. Loops</td>
</tr>
<tr>
<td>4. Local/Global variables</td>
<td>4. Database (Read only)</td>
</tr>
<tr>
<td>5. SGX threat model</td>
<td></td>
</tr>
<tr>
<td>6. Key derivation</td>
<td></td>
</tr>
<tr>
<td>7. Sealing</td>
<td></td>
</tr>
</tbody>
</table>
## SGX Threat Model

<table>
<thead>
<tr>
<th>SGX Threat Model Construction</th>
<th>Realized by</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread and process instantiation</td>
<td>Using the thread policy based on the ecall facts $F_{ecall}$ in the first enclave thread rule and binding ecall sequences of rules using thread facts $F_{thread}$</td>
</tr>
<tr>
<td>Permute or reorder ecalls</td>
<td>Modeling the first enclave thread rule open to executability without order dependencies of timepoints and facts</td>
</tr>
<tr>
<td>Pause enclave execution at instruction level</td>
<td>Modeling instructions in individual rules and utilizing atomic rule executability</td>
</tr>
<tr>
<td>Read access to ecall returns; Read/Modify access to ecall or ocall arguments and returns</td>
<td>Arguments and returns pass through public channel</td>
</tr>
<tr>
<td>Replay, modify of sealing, ecall or arguments and returns</td>
<td>Public channel use in combination Tamarin's inbuilt Dolev Yao adversary capabilities</td>
</tr>
</tbody>
</table>
Outline

- Background
- Motivation
- Our approach
- **Sawtooth -- Tamarin Model**
- Conclusion
Recall -- Sawtooth

Each node workflow

1. Signup and register into the blockchain network
2. Election Ecall 1
3. Election Ecall 2
# Recall -- Sawtooth Block Leader Election

<table>
<thead>
<tr>
<th>Ecall E1</th>
<th>Ecall E2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Election 1</strong></td>
<td><strong>Election 2</strong></td>
</tr>
<tr>
<td>Create Reference Objects</td>
<td>Verify Proof of Elapsed Time</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>• Generate random wait duration</td>
<td>• Unseal and verify the sealed object</td>
</tr>
<tr>
<td>• Create reference monotonic counter <em>(MC_ref)</em></td>
<td>• Verify elapsed time</td>
</tr>
<tr>
<td>• Seal the duration and MC_ref</td>
<td>• Compare MC_ref</td>
</tr>
<tr>
<td></td>
<td>• PoETCertificate</td>
</tr>
</tbody>
</table>

---

**Wait random duration**

**Monotonic Counter ++**
Tamarin Model for Sawtooth

● What components of the workflow do we need?
  ○ SGX entities -- ISV, User, Nodes, Processes
  ○ Entity association network
  ○ Enclave threads
  ○ Sealed sign-up information
  ○ Monotonic Counter
Tamarin Model for Sawtooth

- What components of the workflow do we need?
  - SGX entities -- ISV, User, Nodes, Processes
  - Entity association network
  - Enclave threads
  - Sealed sign-up information
  - Monotonic Counter
SGX Entities

Blockchain Network deployed by a developer (ISV)

1. Signup
2. Ecall E1
3. Ecall E2

A user at a node

Blockchain Interface

1. Signup
2. Ecall E1
3. Ecall E2

Blockchain Interface

A user at a node
SGX Entities

ISV 1

Blockchain Network deployed by a developer (ISV)

A user at a node

1. Signup
2. Ecall E1
3. Ecall E2

Blockchain Interface

1. Signup
2. Ecall E1
3. Ecall E2

A user at a node

ISV 2

Blockchain Network deployed by a developer (ISV)

A user at a node

1. Signup
2. Ecall E1
3. Ecall E2

Blockchain Interface

1. Signup
2. Ecall E1
3. Ecall E2

A user at a node

P1

1. Signup
2. Ecall E1
3. Ecall E2

P2

1. Signup
2. Ecall E1
3. Ecall E2

1. Signup
2. Ecall E1
3. Ecall E2

P2
Tamarin Model for Sawtooth

- What components of the workflow do we need?
  - SGX entities -- ISV, User, Nodes, Processes
  - **Entity association network**
  - Enclave threads
  - Sealed sign-up information
  - Monotonic Counter
Entity Association Network

- Tamarin Fr(*) Fact produces unique variables
- Tamarin Rules can be instantiated unbounded times
- Variables can be passed on through Rules using Tamarin Facts

\[
\begin{align*}
\text{Fr(ISV)} & \quad \rightarrow \\
\forall \text{ ISV}_1 & \quad \rightarrow \\
\text{Fr(User)} \ \text{Fr(Node)} & \\
\forall \text{ ISV}_1 \ \text{User}_k \ \text{Node}_k & \quad \rightarrow \\
\text{Fr(Process)} & \\
\end{align*}
\]
Tamarin Model for Sawtooth

- What components of the workflow do we need?
  - SGX entities -- ISV, User, Nodes, Processes
  - Entity association network
  - Enclave threads
  - Sealed sign-up information
  - Monotonic Counter
Enclave Thread Construction

- **Linear Fact** \((F)\) can be consumed only once.
- **Persistent Fact** \((!F)\) can be consumed unbounded times.
- Linear and persistent **Fact dependencies** allows configuration of single and multiple thread

![Association Network Diagrams](image-url)
Tamarin Model for Sawtooth

- What components of the workflow do we need?
  - SGX entities -- ISV, User, Nodes, Processes
  - Entity association network
  - Enclave threads
  - Sealed sign-up information
  - Monotonic Counter
State Continuity Property

Fair election participation of each node in the blockchain requires that a node must not generate two certificates with same MC_ref

First Order Logic Query

```plaintext
All
PoETCertificate ( node , MC_ref ) @t1 &
PoETCertificate ( node , MC_ref ) @t2
== > # t1 =# t2
```
Verification

Sawtooth Model
Multiset Rewriting Rule

State Cont. Property
First Order Logic

Apply patch in the model

Tamarin Prover

Property does not hold
⇒ Attack

Property Holds
Verified
Sawtooth Attack Trace
## Summary of Case Studies

<table>
<thead>
<tr>
<th>App</th>
<th>Attack Discovery Time</th>
<th>Verification Time</th>
<th># Rules</th>
<th>Model LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sawtooth</td>
<td>1m 18s</td>
<td>25s</td>
<td>11</td>
<td>300</td>
</tr>
<tr>
<td>Heartbeat</td>
<td>7s</td>
<td>2h 4m 7s</td>
<td>11</td>
<td>250</td>
</tr>
<tr>
<td>BI-SGX</td>
<td>36s</td>
<td>37s</td>
<td>18</td>
<td>450</td>
</tr>
</tbody>
</table>
Conclusion

- First attempt towards using symbolic verification tools to verify the state continuity for SGX enclave programs.

- We demonstrate our approach using three open-source SGX applications, resulting into reusable SGX primitives and model templates.

- Tamarin Prover can effectively model SGX-specific semantics and operations; and state continuity properties.

- Our Tamarin code is released at Github: https://github.com/OSUSecLab/SGX-Enclave-Formal-Verification.
Thank you

Mohit Kumar Jangid
The Ohio State University

Guoxing Chen
Shanghai Jiao Tong University

Yinqian Zhang
Southern University of Science and Technology

Zhiqiang Lin
The Ohio State University

Follow up questions at jangid.6@osu.edu.