Ryoan: A Distributed Sandbox for Untrusted Computation on Secret Data

Tyler Hunt, Zhiting Zhu, Yuanzhong Xu, Simon Peter, and Emmett Witchel
University of Texas at Austin
OSDI 2016

Presented by John Alsop
A **Distributed** Sandbox for **Untrusted** Computation on **Secret** Data

• **Why distributed?**
  • Large computation, can’t run on single core
  • Outsource different compute jobs to specialized services

• **Why untrusted computation on secret data?**
  • Outsourced computation: no control over software
  • Ensuring software will not leak data is hard
  • Software stack is huge, complex: can’t verify everything
Motivating Example

- **Compute Requirements**
  - *Alice* outsources DNA processing to *23andMe*
  - *23andMe* outsources ML classification to *Amazon*

- **Security Requirements**
  - *Alice* must keep her DNA secret from *23andMe* and *Amazon*
  - *23andMe* must keep its DNA processing tech secret from *Amazon*
Background: SGX

- Intel SGX hardware attests for and confines software module
- Enclave memory protected, encrypted off-chip
- Reduces attack surface

Img src: Hamza Omar et al, Intel’s SGX In-depth Architecture, ISCA’15 tutorial
Background: Google Native Client (NaCl)

- Sandbox tool with 2 components: **verifier** and **runtime**

- **Verifier** prevents external control flow, mem access

- **Runtime** intercepts syscalls

- **Sandboxed module cannot communicate with external SW**

Source: http://zqsmm.qiniucdn.com/data/20120628145922/index.html
Background: Distributed Information Control Flow

• Mutually distrustful data sources
  • Secrecy categories specify permissions
  • Privilege hierarchy specifies who is trusted with what

• Each piece of data has attached label
  • Label is a set of secrecy categories
  • Actors can add or remove categories from label

Img source: Zeldovich et al, Securing distributed systems with information flow control, NSDI'08
Ryoan Threat Model

- Performing distributed secure computation
  - Need to communicate secret information
  - Some communicating nodes don’t trust each other

- Only SGX hardware and Ryoan software are universally trusted:
  - Platform OS/hypervisor untrusted
  - Confined software module untrusted
  - SGX attests to Ryoan module and config
  - Ryoan attests to untrusted SW module

- Goal: prevent untrusted software from leaking secret data to external attacker
Ryoan Overview

1. User specifies a DAG of computation
2. Master Ryoan enclave sets up DAG
   - SGX+NaCl+Ryoan on each node
   - Encrypted channel between neighbors
3. User inputs secret data
4. Life cycle at each node:
   - Initialize state
   - Wait for input
   - Process input
   - Send output
   - Reset/destroy

\[\text{= Trusted}\]
### Ways to Leak Secret Data

<table>
<thead>
<tr>
<th>Ways to Leak Secret Data</th>
<th>Ryoan Protections</th>
</tr>
</thead>
<tbody>
<tr>
<td>Machine loads corrupted version of Ryoan</td>
<td></td>
</tr>
<tr>
<td>OS reads Ryoan data</td>
<td></td>
</tr>
<tr>
<td>Attacker probes off-chip DRAM contents/data bus</td>
<td></td>
</tr>
<tr>
<td>Module writes data outside encrypted zone</td>
<td></td>
</tr>
<tr>
<td>Module passes info through system calls</td>
<td></td>
</tr>
<tr>
<td>Module weakens SGX config</td>
<td></td>
</tr>
<tr>
<td>Module uses secret data unconfined</td>
<td></td>
</tr>
<tr>
<td>Module accumulates data from past inputs</td>
<td></td>
</tr>
<tr>
<td>Info leaked through message contents</td>
<td></td>
</tr>
<tr>
<td>Info leaked through message size</td>
<td></td>
</tr>
</tbody>
</table>

### Ryoan Protections

- **SGX**: Verifies Ryoan binary and config.
- **NaCl**: Prevents dynamic code generation.
- **Ryoan**: Prevents accesses outside enclave and intercepts system calls.

---

**Diagram:**

1. Input from user Alice
2. Label added by sandbox: Alice
3. 23andMe adds its label and delegates to Amazon Machine Learning
4. Amazon Machine Learning sends result to 23andMe after removing its own label
5. 23andMe removes its label
6. Sandbox removes Alice's label
7. Output to user Alice

---

**Ryoan**

- **DAG**: Specifies fixed message size.
- **DIFC**: Labels determine whether confinement is necessary.
- **Clears state after outputting result**.
- **One shot at input data**.
Ryoan Performance, Compatibility Measures

• Ryoan must handle system calls for confined modules
  • Preload any files that may be accessed
  • Maintain, manage enclave memory pool (4GB)
  • Marshal other syscalls (e.g. pagefaults)

• Confinement only necessary when handling secret data
  • Module unconfined if input not external secret (defined by DIFC label)
  • Module initialization is unconfined

• Checkpoint-based life cycle saves init time
Ryoan Limitations

• Application Requirements
  • Representable as static DAG
  • Output messages are padded to worst case (or truncated if longer)
  • Each module’s computation must fit in enclave memory
  • All potentially accessed files must fit in enclave memory

• Unprotected Attack Vectors
  • Execution time information channel (suggestion: fixed execution time)
  • No protection against denial of service attacks
  • SGX page faults
  • Cache timing
  • Address bus monitoring
  • Processor monitoring
Evaluation: Use Cases

• Translation

• Healthcare

• Email processing

• Image processing
Evaluation: Confinement Cost

• Report incremental cost for
  • NaCl sandboxing
  • Message encryption
  • Syscall marshalling
  • Checkpoint restore/zero
  • SGX

• Main sources of overhead are sandbox, CPR, SGX

• Health sees biggest overhead due to small work:data ratio
Sensitivity to SGX instruction latency

- SGX v2 instruction delay estimated based on measurement, emulation
- Sensitivity study shows that increasing this delay doesn’t affect performance significantly
Discussion
Evaluation: Detailed Results

- Init time can be huge
- Example kernels are small
Outline

• Motivation/Intro
  • Example: 23andMe + AWS
  • Why distributed
  • Things to protect against

• Background
  • SGX
  • NaCl
  • DIFC

• Ryoan design
  • Threat model:
    • Figure 1, Figure 2: only Ryoan, SGX trusted, not module or external OS/hypervisor
    • Principals, acts-for relation. Only need confinement if handle other principals' data
    • does not protect against DoS, cache timing, SGX page faults, address bus monitoring, PMUs, execution time
  • Intra-module attack planes: information leak through man-in-middle, output size, processing time, system calls, residual state
  • Confinement: At setup, master creates DAG and establishes secure point-to-point connections, NaCl restricts control flow, Ryoan provides confined OS functionality, life cycle reset and one-shot at input data, SGX protects enclave memory, DIFC labels ensure confinement when operating on some other principal's secrets
  • Everything must fit into enclave memory (4GB), files must be pre-loaded, no dynamic code generation, fixed DAG topology

• Results
  • Use cases: healthcare, email, image processing, translation
  • Fig 6: incremental overheads
Ryoan: Overview

• blah