Ryoan: A Distributed Sandbox for Untrusted Computation on Secret Data
“Ryoan: A Distributed Sandbox for Untrusted Computation on Secret Data”

Tyler Hunt, Zhiting Zhu, Yuazhong Xu, Simon Peter, Emmett Witchel 2016

• Tyler Hunt (UT Austin PhD student)
• Zhiting Zhu (UT Austin PhD student)
• Yuazhong Xu (UT Austin PhD, Google)
• Simon Peter (UT Austin Asst. Prof) – Sloan Fellow
• Emmett Witchel (UT Austin Prof) – ASPLOS’19 PC Chair

OSDI’16
Best Paper
Ryoan: A Distributed Sandbox for Untrusted Computation on Secret Data

Tyler Hunt, Zhiting Zhu, Yuanzhong Xu, Simon Peter, Emmett Witchel
Disease risk assessment: Trust issues?
Disease risk assessment: Trust issues!
Ryoan’s goals

◎ Provide user data secrecy
  ○ Without trusting the application
  ○ Without trusting the platform (OS, Hypervisor)
◎ Support cooperation between service providers
Ryoan’s goals

- Provide user data secrecy
  - Without trusting the application
  - Without trusting the platform (OS, Hypervisor)
- Support cooperation between service providers
Ryōan-ji
Threat model

Users
◎ Don’t trust service providers for secrecy
◎ Don’t trust platforms for secrecy

Service Providers
◎ Control platforms
◎ Don’t trust other service providers for secrecy

Everyone
◎ Trusts Ryoan
◎ Trusts Intel SGX

- User
- User Data
- Untrusted Code
- Untrusted Platform
- Ryoan
- SGX
Ryoan’s world

Modules
◎ NaCl x86 binaries from service providers
◎ Application logic

Platforms
◎ More service providers’ code
◎ Host computation

Sandboxes
◎ Trusted code
◎ Confine modules
◎ Based on Google’s Native Client (NaCl)
Ryoan applications

**Modules**, limited to:
- Request oriented
- Well defined unit of work
  - One request → one result
  - e.g., 1 email, 1 photo

**Composable**
- Modules can be connected to build services
Intel SGX in 2 minutes (or less)

◎ Provides Enclaves
  ○ Regions of a process's virtual address space

◎ Enclaves
  ○ Can only be accessed by enclave code
  ○ Still have access to the rest of memory

◎ Attestations
  ○ Hardware signed hashes of initial code and data
Chain of trust

- SGX provides unforgeable attestation of the sandbox

- Statements Ryoan makes about the module can now be trusted
Ryoan’s view of SGX

SGX gives you:
- *Trusted* computation on secret data

Ryoan uses SGX to give you:
- *Guarantees on Untrusted* computation
Confining untrusted code

Problem:
◎ Platform can read secrets out of memory

Solution?
◎ Execute module inside of an enclave
Confining untrusted code

Problem:
- Module can copy secrets to non-enclave memory

Solution?
- Restrict accessible memory with a sandbox
  - Property of NaCl
Confining untrusted code

Problem:
◎ Modules can use system calls to write out user data

Solution?
◎ NaCl modules call sandbox to access system calls
◎ Enforce encryption
Confining untrusted code

Problem:
Modules can collude with users to steal data

Solution?
Confining untrusted code

Problem:
◎ Modules can collude with users to steal data

Solution?
◎ Don’t let modules keep state between requests

How?

Later
Modules cannot keep state

- Module life cycle imposed by Ryoan
  - Read, process, write, destroy

- Sandbox enforces one request per module execution
  - Represent a complete unit of work
  - Only contain content from one user
Discussion: Summary Question #1

- **State the 3 most important things the paper says.** These could be some combination of their motivations, observations, interesting parts of the design, or clever parts of their implementation.
Covert and side channels

- Output, via some externally visible property of execution
- **Ryoan**: Software covert channels
  - System calls
  - Execution time
- **Hardware covert channels**:  
  - Hardware vendor’s responsibility
System call covert channel

0101110

write(8bytes); write(16bytes);
write(8bytes); write(16bytes);
write(16bytes); write(16bytes);
write(8bytes);

0101110

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>8bytes</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>16bytes</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Eliminating system call channel

- Remove modules ability to make system calls. How?
- Ryoan performs all data input and output independent of the content
Initialization is expensive

ClamAV (virus scanner):
24 seconds to initialize
0.12 seconds to process a request

Checkpoint

Confined; Module cannot make system calls.

Initialize

Read Input

Process

Done

Restore Checkpoint
Confined compatibility API

Dynamic Memory
- Modules can call mmap for “new” memory
- Return memory from a pre-allocated pool

In-memory file API
- File system operations in memory
- Examples:
  - Temp files
  - Preexisting files

Replaced system calls:
- mmap

Replaced system calls:
- open, close, read, write, stat, lseek, unlink, mkdir, rmdir, getdents
• **Describe the paper's single most glaring deficiency.** Every paper has some fault. Perhaps an experiment was poorly designed or the main idea had a narrow scope or applicability.
Health

In: Genome/health data  
Out: Disease risk

Translation

In: French text  
Out: English text

Images

In: Pictures  
Out: Array of objects

Email

In: Emails  
Out: Spam & virus status
Evaluation

- Implementation requires SGX v2 instructions (spec: Fall 2014, coming soon)
  - Dynamic memory allocation/protection

- SGX performance model
  - Measured SGX v1 latencies on our hardware
  - Estimated SGX v2 latencies (sensitivity study in paper)
  - Flush TLB on all system calls, page faults, interrupts
<table>
<thead>
<tr>
<th>Workload</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Email</td>
<td>20,000 1.4KB Boolean vectors from different users</td>
</tr>
<tr>
<td>Health</td>
<td>30 short paragraphs, sizes 25-300B, 4.1KB total</td>
</tr>
<tr>
<td>Images</td>
<td>12 images, sizes 17KB-613KB</td>
</tr>
<tr>
<td>Translation</td>
<td>250 emails, 30% with 103KB-12MB attachment</td>
</tr>
</tbody>
</table>

**Cost of Confinement**

- **Email**: 48%
- **Health**: 419%
- **Images**: 27%
- **Translation**: 91%
Ryoan summary

- Allows untrusted code to operate on secret data on untrusted platforms

- Sandbox with SGX
  - Eliminates explicit channels

- Module can’t call platform
  - Eliminates covert channels

- Mostly backwards compatible
  - Sandbox code implements system calls
Limitations

• Threat model doesn’t handle:
  – Bugs in code that leak its own secrets
  – Denial of service
  – HW security limitations, e.g., page faults, performance counters, address bus monitoring, HW timing attacks

• Unable to protect applications that include:
  – Storage or too much memory
  – Network metadata
  – Repeated Computations on the Same/Similar Input Data
  – Multi-user computation
  – Dynamic code generation, JIT compiler, OS, hypervisor
## Summary of Properties

<table>
<thead>
<tr>
<th>Module property</th>
<th>Enforce</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS cannot access module memory (§2.2).</td>
<td>SGX</td>
<td>Security</td>
</tr>
<tr>
<td>Initial module code and data verified (§2.2).</td>
<td>SGX</td>
<td>Security</td>
</tr>
<tr>
<td>Can only address module memory (§2.4).</td>
<td>NaCl</td>
<td>Security</td>
</tr>
<tr>
<td>Ryoan intercepts syscalls (§2.4,§4.3).</td>
<td>NaCl</td>
<td>Security</td>
</tr>
<tr>
<td>Cannot modify SGX state (§5).</td>
<td>NaCl</td>
<td>Security</td>
</tr>
<tr>
<td>User defines topology (§4.1).</td>
<td>Ryoan</td>
<td>Security</td>
</tr>
<tr>
<td>Data flow tracked by labels (§4.2).</td>
<td>Ryoan</td>
<td>Security</td>
</tr>
<tr>
<td>Memory cleaned between requests (§5).</td>
<td>Ryoan</td>
<td>Security</td>
</tr>
<tr>
<td>Module defines initialized state (§5.4).</td>
<td>Ryoan</td>
<td>Perf.</td>
</tr>
<tr>
<td>Unconfined initialization (§5).</td>
<td>Ryoan</td>
<td>Compat.</td>
</tr>
<tr>
<td>In-memory POSIX API (§5.1)</td>
<td>Ryoan</td>
<td>Compat.</td>
</tr>
</tbody>
</table>
Wednesday

Midterm #2