Ryoan: A Distributed Sandbox for Untrusted Computation on Secret Data

PRESENTED BY:
NICHOLAS BURTON

WRITTEN BY:
TYLER HUNT
ZHITING ZHU
YUANZHONG XU
SIMON PETER
EMMETT WITCHEL
Big Data, Your Information, and Security
Related Work

- Haven
- MiniBox
- OverShadow and others
Issues
Ryoan
Threat Model

**In Scope**
- User is distrustful of the service
- Service can outsource work, becoming a user
- User does not trust software at any privilege level
- Covert software channel attacks

**Out of Scope**
- DoS attacks
- Hardware limitation based side-channel attacks
- Execution time based side-channel attacks
Hardware Limitations

- SGX page faults
- Cache Timing
- Address Bus Monitoring
- Processor Monitoring
Main Components of Ryoan
Intel SGX, and how Ryoan uses it

- Available on all recent Intel processors
- Provides a Hardware Isolated Execution Environment via Enclaves
- SGX provides enclave identities (hash of enclave’s initial state)
- Assume initial state of an enclave cannot be impersonated
- Ryoan checks enclave’s identity to ensure it is of Ryoan origin
Google Native Client, and how Ryoan uses it

- Sandbox for running x86/86-64 code using fault isolation
- Two parts: verifier, service runtime
- Verifier guarantees a module cannot break out of NaCl.
- Intercepts System Calls and passes them to Ryoan instead of the OS
How is it put together?
A Single Ryoan Instance

Figure 1: A single instance of Ryoan’s distributed sandbox. The privileged software includes an operating system and an optional hypervisor.
A Single Ryoan Instance

Figure 2: The Ryoan chain of trust. SGX hardware attests that a valid instance of Ryoan is executing (Hash) with an intended SGX configuration (Meta). Ryoan ensures that the expected binary is loaded with a signed hash from the software provider (grey).
Directed Acyclic Graph

- A finite graph, with no loops
- Improved security
Use of labels to mark secret data, and which previous enclaves have seen it
Each module can remove its own label
Labels together are a tag
Labels and DAG

- Non-confining and Confining Labels
- Audit Trail
- Data Oblivious Communication

Figure 3: Ryoan’s distributed sandbox. Ryoan instances manage labels on data and modules. The user’s tag is propagated to all modules, making them confined after receiving input; For example, 23andMe’s tag is kept when it outsources to Amazon Machine Learning to prevent leaking secrets from 23andMe.
Individual Module Security
Module Security

- Verifier ensures code adheres to Ryoan format
- Memory access is constrained to the provided memory for the module
- NaCl guarantees still stand
- Ryoan disallows SGX instructions in modules
- Single processing opportunity per unit of work/input
- After processing and sending output, Ryoan instance is deleted or reset
Module Security

- When a Ryoan instance is confined it disallows communication with the OS
- Ryoan provides a system API
- Virtual File System
- mmap calls handled by Ryoan API to satisfy memory allocation
- Fixed and Quantized processing times
- System calls to OS intercepted by Ryoan-libc
- Checkpoint Module Resetting
Checkpoint Module Resetting

Figure 4: Instance life cycle: unoptimized vs checkpoint-based.
Implementation
Implementation

- Ryoan-libc
- Module Address Space
- I/O control
- Key Establishment Between enclaves
- Checkpoint Code
Examples and Use cases
Overhead
Overhead

<table>
<thead>
<tr>
<th></th>
<th>Load Size</th>
<th>Init Size</th>
<th>Init Time</th>
<th>CPU Time</th>
<th>CPR Size</th>
<th>Sys. Calls</th>
<th>PF Intrp Size</th>
<th>Intrap Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>Email Distribute</td>
<td>18.8 MB</td>
<td>18.1 MB</td>
<td>0.59 sec</td>
<td>1.32 sec</td>
<td>11.6 MB</td>
<td>47k</td>
<td>60k</td>
<td>473</td>
</tr>
<tr>
<td>DSPAM</td>
<td>19.6 MB</td>
<td>273.5 MB</td>
<td>11.15 sec</td>
<td>22.10 sec</td>
<td>45.3 MB</td>
<td>1.29 MB</td>
<td>1.81 MB</td>
<td>6k</td>
</tr>
<tr>
<td>ClamAV</td>
<td>21.1 MB</td>
<td>403.9 MB</td>
<td>24.96 sec</td>
<td>29.17 sec</td>
<td>83.3 MB</td>
<td>247k</td>
<td>423k</td>
<td>7k</td>
</tr>
<tr>
<td>Combine</td>
<td>18.8 MB</td>
<td>18.1 MB</td>
<td>0.59 sec</td>
<td>0.11 sec</td>
<td>16 KB</td>
<td>12k</td>
<td>2k</td>
<td>77</td>
</tr>
<tr>
<td>LoadModel</td>
<td>19.3 MB</td>
<td>19.4 MB</td>
<td>0.58 sec</td>
<td>12.52 sec</td>
<td>28 KB</td>
<td>82k</td>
<td>280k</td>
<td>56k</td>
</tr>
<tr>
<td>Classifier</td>
<td>19.3 MB</td>
<td>19.4 MB</td>
<td>0.58 sec</td>
<td>18.23 sec</td>
<td>36 KB</td>
<td>1.84 MB</td>
<td>389k</td>
<td>151k</td>
</tr>
<tr>
<td>Return</td>
<td>18.8 MB</td>
<td>18.1 MB</td>
<td>0.59 sec</td>
<td>6.77 sec</td>
<td>16 KB</td>
<td>668 K</td>
<td>162k</td>
<td>3k</td>
</tr>
<tr>
<td>Images Distribute</td>
<td>18.0 MB</td>
<td>18.1 MB</td>
<td>0.59 sec</td>
<td>0.42 sec</td>
<td>632 KB</td>
<td>2k</td>
<td>2k</td>
<td>36</td>
</tr>
<tr>
<td>Recognize</td>
<td>26.6 MB</td>
<td>27.1 MB</td>
<td>0.63 sec</td>
<td>24.79 sec</td>
<td>83.2 MB</td>
<td>88k</td>
<td>174k</td>
<td>6k</td>
</tr>
<tr>
<td>Combine</td>
<td>18.0 MB</td>
<td>18.1 MB</td>
<td>0.59 sec</td>
<td>0.36 sec</td>
<td>2.5 MB</td>
<td>14k</td>
<td>3k</td>
<td>129</td>
</tr>
<tr>
<td>Translation</td>
<td>25.3 MB</td>
<td>386.9 MB</td>
<td>2.34 sec</td>
<td>26.65 sec</td>
<td>29.1 MB</td>
<td>303k</td>
<td>248k</td>
<td>8k</td>
</tr>
</tbody>
</table>

- Email Input: 250 emails, 30% with 103KB-12MB attachment
- Health Input: 20,000 1.4KB Boolean vectors from different users
- Images Input: 12 images, sizes 17KB-613KB
- Trans. Input: 30 short paragraphs, sizes 25-300B, 4.1KB total
Questions?