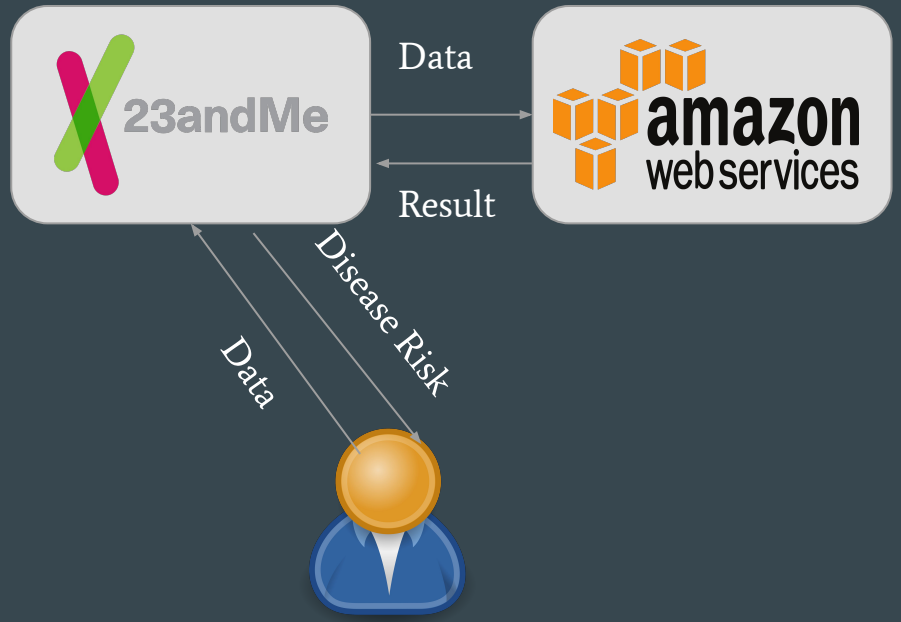
Ryoan: A Distributed Sandbox for Untrusted Computation on Secret Data

CS5352 In Class Presentation
Presented By: Faraz Saeed

Overview

- Data Processing Services
  - Involve Sensitive Data
- Outsource Services → SaaS
- Service Becoming User

- What Ryoan Provides?
  - Sandbox + Trusted Hardware such as SGX

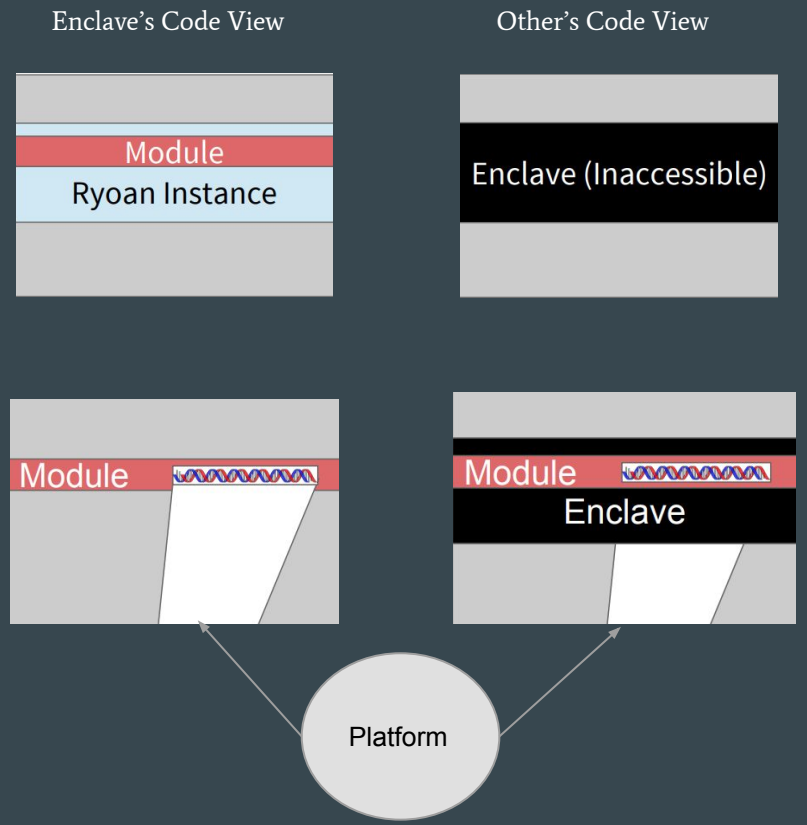


Related Work

- Haven
  - Uses Trusted Program and Library OS
- VC3
  - Uses Trusted MapReduce
- Overshadow / InkTag
  - Uses Trusted Hypervisor
- Difference between these systems and Ryoan
  - Trusted Application in Untrusted Environment
  - Untrusted Code Processing Sensitive Data
- Homomorphic Encryption
  - Limited Application Scenario and Performance Overhead

Ryoan Controlling Untrusted Modules

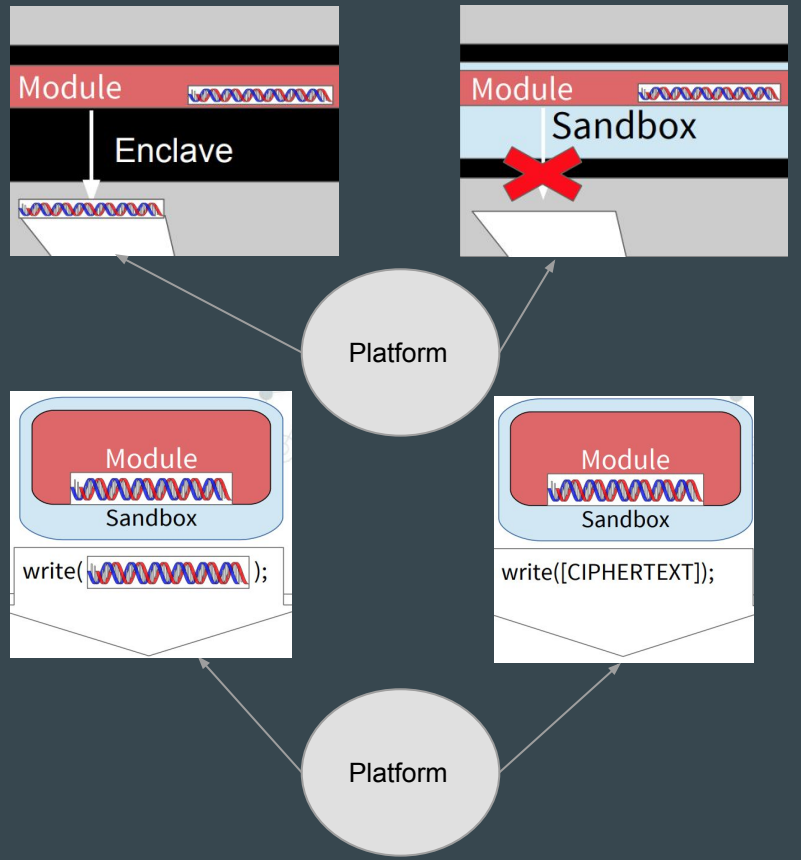
- Intel SGX OverView
- SGX → Ryoan → Module
- Confining Untrusted Code
  - Secrets Out of Memory
  - Module In Enclave



Ryoan Controlling Untrusted Modules

- Confining Untrusted Code
  - Secrets to Non-Enclave Memory
  - Restrict Accessible Memory with Sandbox

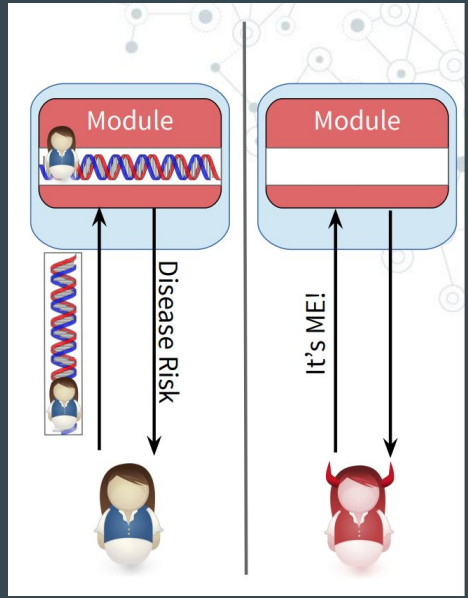
- Confining Untrusted Code
  - Use System Calls to Write Data
  - Nacl Enforce Encryption



Ryoan Controlling Untrusted Modules

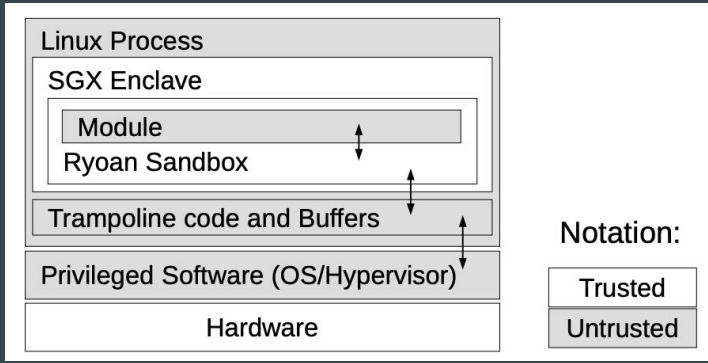
- Confining Untrusted Code
  - Collude with Users
  - Don’t Preserve State Between Requests

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



Design Overview

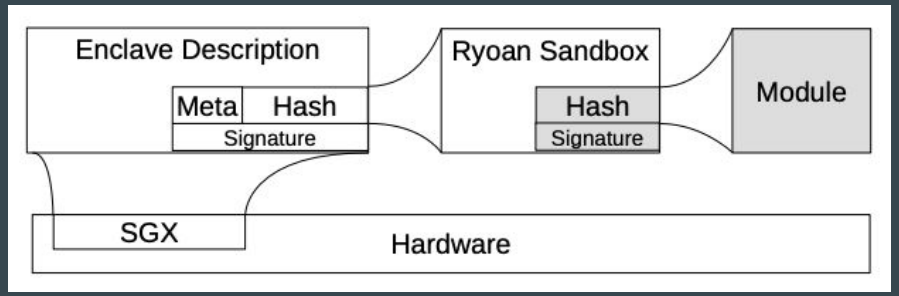
- Ryoan's Primary Job
  - Prevent Modules from Communicating Outside
- SGX limits Externally Visible Behaviour
  - Unprotected Memory
  - Systems Calls
- Systems Calls from NaCl
  - Remove modules ability to make system calls
- Restricted IO
  - Ryoan performs all data input and output independent of the content

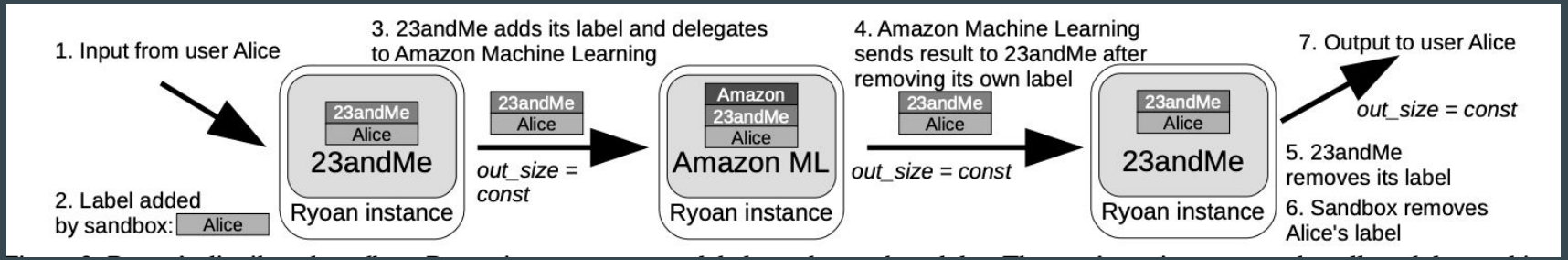


Module property	Enforce	Reason
OS cannot access module memory (§2.2).	SGX	Security
Initial module code and data verified (§2.2).	SGX	Security
Can only address module memory (§2.4).	NaCl	Security
Ryoan intercepts syscalls (§2.4,§4.3).	NaCl	Security
Cannot modify SGX state (§5).	NaCl	Security
User defines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

Design Overview

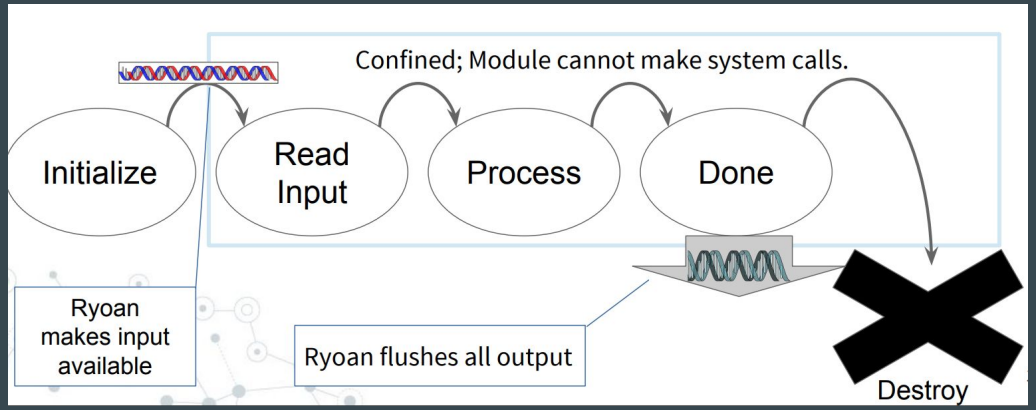
- Ryoan Does Not Trust Others
  - Including OS and Hypervisor
- Master Enclave
  - Creates all Ryoan Instances
  - Establish cryptographically protected Communication
- Label Based Model For Communication





Design Overview

- Module Confinement
  - Enforcement of Lifecycle
- NaCl code Validator
- In-memory virtual file system
  - Pre-loaded files in Memory
- Nmap calls for dynamic memory
- One shot at Input Data



Future Work

- Applying Ryoan to other domains
  - cloud computing or financial services
- Increasing the number of supported system calls
  - Limited number of supported system calls right now
- Expanding the types of untrusted code that can be executed
  - Confined to x86

Conclusion

Allows untrusted code to operate on secret data on untrusted platforms

References

https://www.usenix.org/sites/default/files/conference/protected-files/osdi16_slides_hunt.pdf

https://www.usenix.org/system/files/conference/osdi16/osdi16-hunt.pdf