Example: Application-based RM

A security mechanism embedded within applications

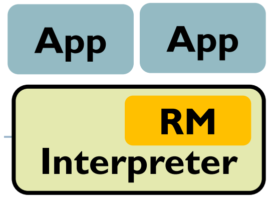
- Enforce access control policies, provide fine-grained control over application behaviors, and prevent unauthorized actions.

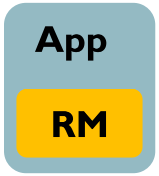
Integrating RM with interpreter

- Every operation will be checked against security policies before execution
- Example: JavaScript engine enforces sandboxing by restricting access to certain APIs or resources during script execution.

Inline RM

- Inserting RM directly into the application's code. This could be achieved with source code instrumentation, or binary rewriting.
- Example: StackGuard





Example: Hardware-based RM

Responsible for monitoring and regulating all the software activities, including OS kernel.

- Any operation violating the security policy will throw a hardware exception

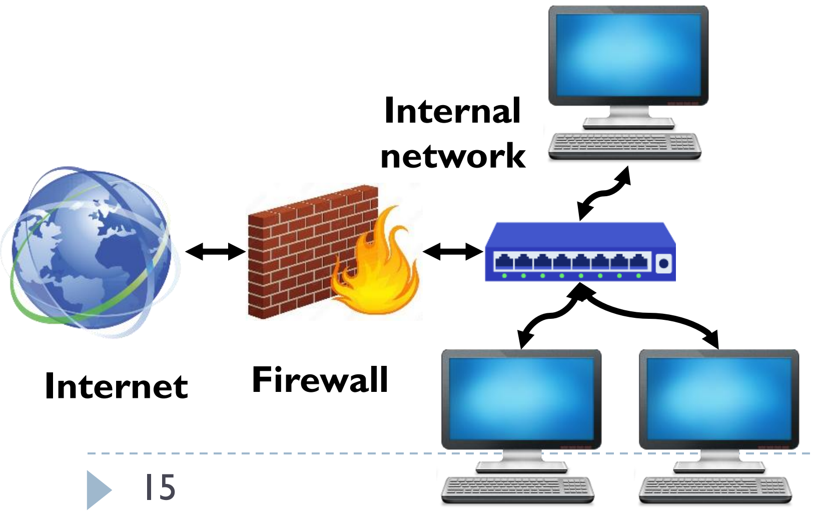
Hardware-based RMs conduct various checking

- Memory access management.
  - If each memory access is within the process' memory range.
  - If each access follows the allowed permission (read, write, executable, set in the Page Table Entry). Recall the Non-executable Memory mechanism.
- Privilege mode management.
  - At any time, CPU can be in one mode, either user or kernel.
  - Privileged instructions can only be issued in kernel mode.
  - Context switch is required for user mode to call privileged functions.

Example: Network-based RM

Firewall

- Monitor and regulate the network traffics based on the security policy.
  - Outbound policy: define what traffic is allowed to exit the network
  - Inbound policy: define what traffic is allowed to enter the network
- Possible actions:
  - Allow: permitted through the firewall.
  - Deny: not allowed through the firewall.
  - Alert: send alert to the administrator.



Protocol	Source Addr	Source Port	Dest. Addr	Dest. Port	Action
TCP	Any	Any	192.168.42.0/24	>1023	Allow
TCP	192.168.42.1	Any	Any	Any	Deny
TCP	Any	Any	192.168.42.1	Any	Deny
TCP	Any	Any	192.168.42.55	25	Allow
TCP	**Any**	**Any**	**Any**	**Any**	**Deny**

Outline

- **Protection Strategies**
  - Confinement
  - Reference Monitor

- **Hardware-assisted Protection**
  - Basic Functionalities
  - Trusted Platform Module
  - Trusted Execution Environment

Using Hardware to Protect Software

Software is not always trusted

- Privileged software (OS, hypervisor) usually has very large code base, which inevitably contains lots of vulnerabilities.
- Once it is compromised, the attacker can do anything to any apps running on it.

SW	Line of codes
Linux Kernel 5.12	28.8M
Windows 10	50M
VMWare	6M
Xen	0.9M

Commercial software typically has 20 to 30 bugs for every 1k lines of code

Hardware is more reliable

- After the chip is fabricated, it is hard for the attacker to modify it. The integrity of hardware is guaranteed.
- It is also very hard for the attacker to peek into the chip and steal the secret (e.g., encryption key). The confidentiality of hardware is guaranteed.
- It is more reliable to introduce security-aware hardware to protect the operating system and applications