Security Namespace : Making Linux Security Frameworks Available to Containers

Yuqiong Sun, David Safford, Mimi Zohar, Dimitrios Pendarakis,
Zhongshu Gu and Trent Jaeger

IBM Research

PennState

Container vs. Virtual Machines

- Containers are operating system level virtualization environment for running multiple isolated Linux systems on a single Linux control host

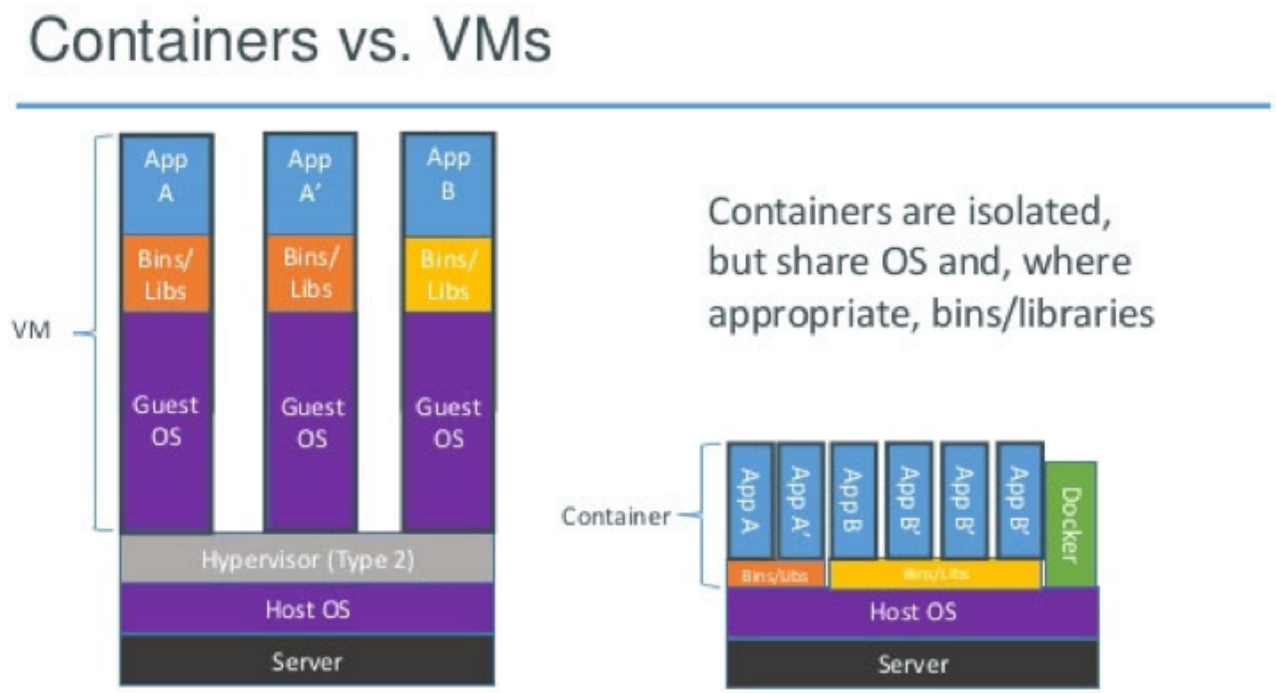


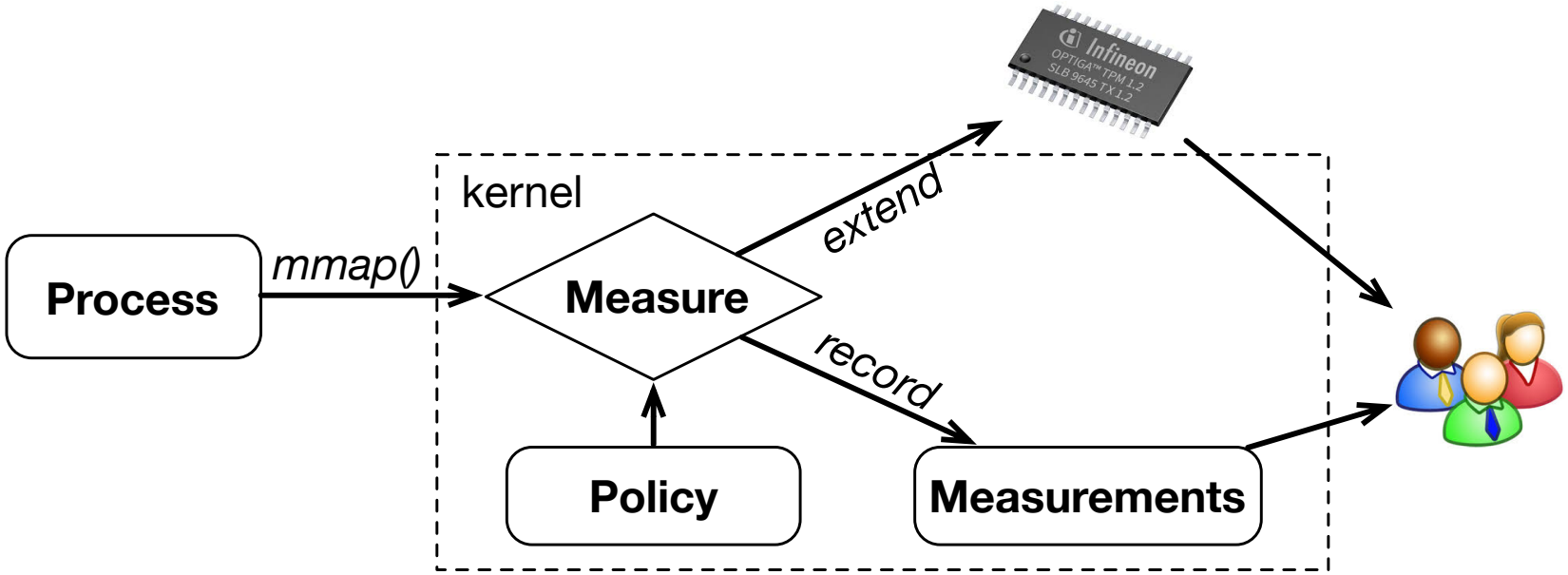
Image credit: Docker Inc. and RightScale Inc.

Is Kernel Sharing All Good?

- Container owners cannot leverage kernel security frameworks to protect their containers
  - I.e., cannot apply local security policies to govern integrity measurement, code execution, mandatory access control, etc.

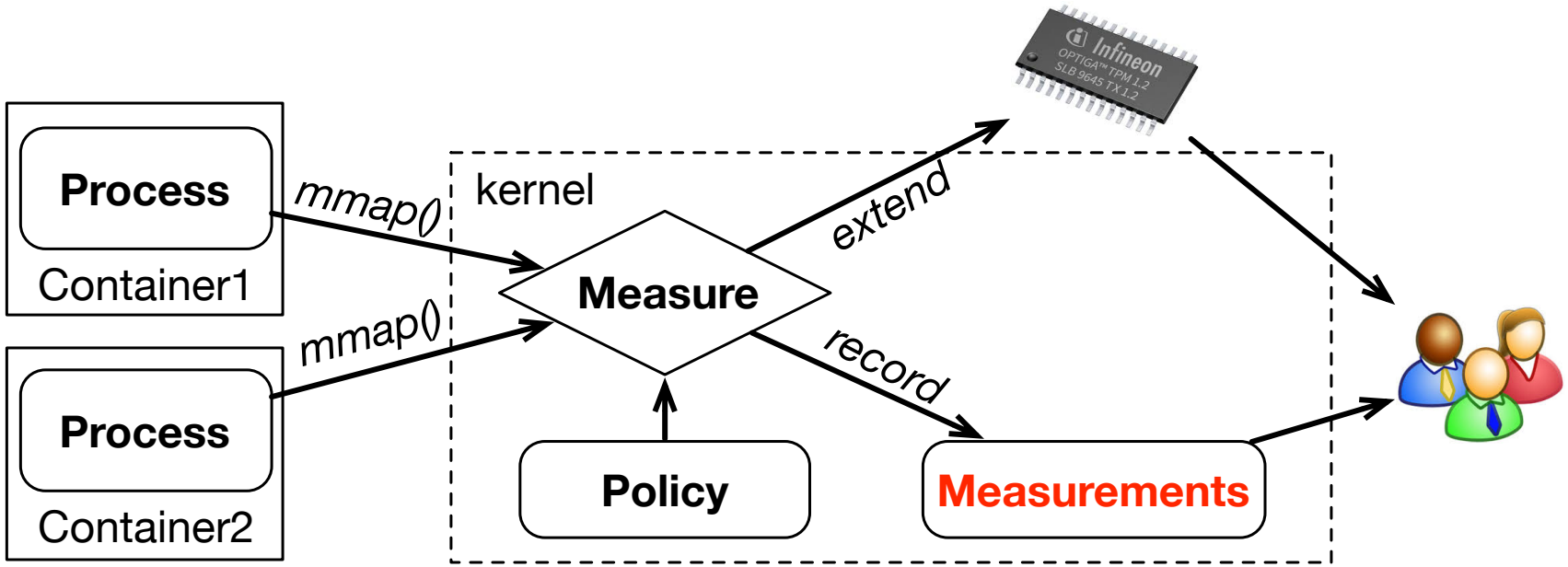
Integrity Attestation for Container

- On a container cloud, can a container owner attest integrity of his/her containers to his/her customers?
  - Exactly what Linux Integrity subsystem (a.k.a. IMA) is designed for



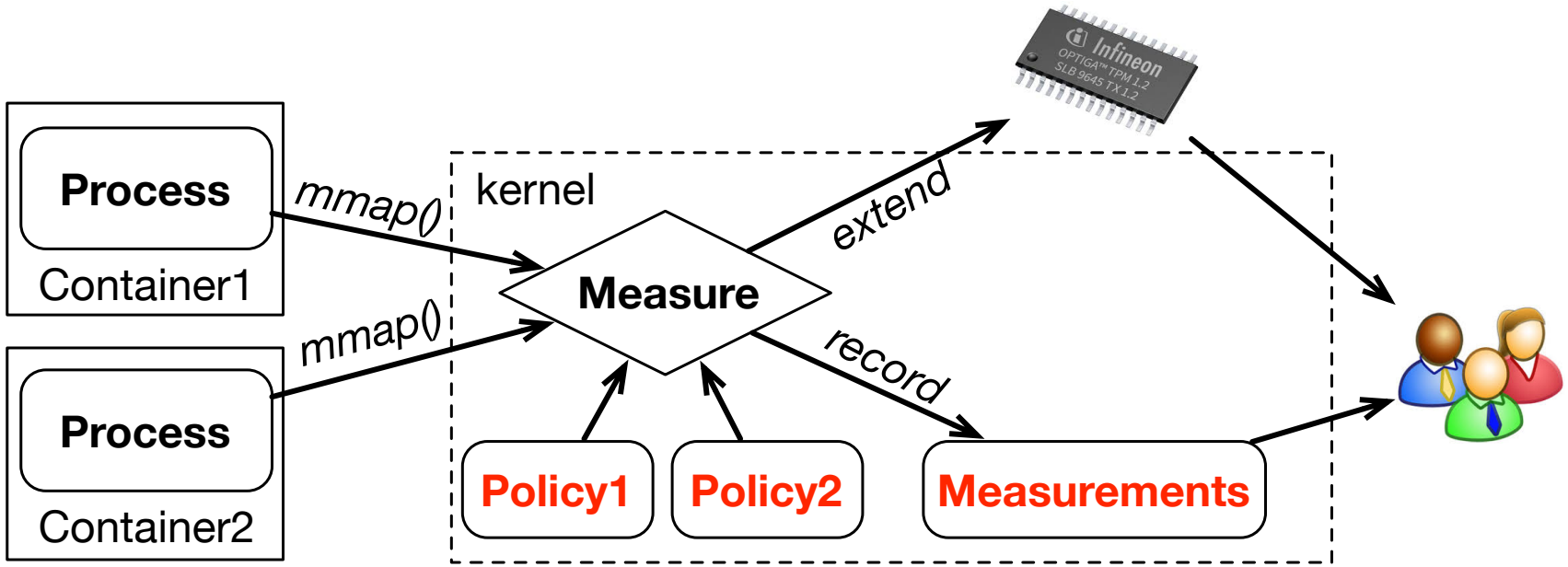
Integrity Attestation for Container (Cont.)

- But…
  - Mixed measurements from different containers and host



Integrity Attestation for Container (Cont.)

- But…
  - Mixed measurements from different containers and host
  - Different versions of policies



Integrity Attestation for Container (Cont.)

- But…
  - Mixed measurements from different containers and host
  - Different versions of policies
    - And policies do not always agree with each other

Mandatory Access Control for Container

- MAC mechanisms can only be used to protect container host, but not container
  - An excerpt from Ubuntu LXC documentation (section AppArmor):

Programs in a container cannot be further confined — for instance, MySQL runs under the container profile (protecting the host) but will not be able to enter the MySQL profile (to protect the container).

Goal: Security Namespace

- Can we virtualize/isolate security frameworks in Linux kernel to make them available to containers?
  - Just like how other kernel resources are virtualized/isolated for containers
- Ideally, we want:
  - Each container can govern the security of its containerized processes
  - Each container can independently define its security policies and access its security states (e.g., audit logs, measurements)
  - **Security Invariant**: a container **cannot** invalidate the security goal of other containers or the container host, as expressed via their respective security policies

Background: Namespaces in Kernel

A namespace wraps a global system resource in an abstraction that makes it appear to the processes within the namespace that they have their own isolated instance of the global resource. Changes to the global resource are visible to other processes that are members of the namespace, but are invisible to other processes. One use of namespaces is to implement containers.

Background: Namespaces in Kernel

A namespace wraps a global system resource in an abstraction that makes it appear to the processes within the namespace that they have their own isolated instance of the global resource. Changes to the global resource are visible to other processes that are members of the namespace, but are invisible to other processes. One use of namespaces is to implement containers.

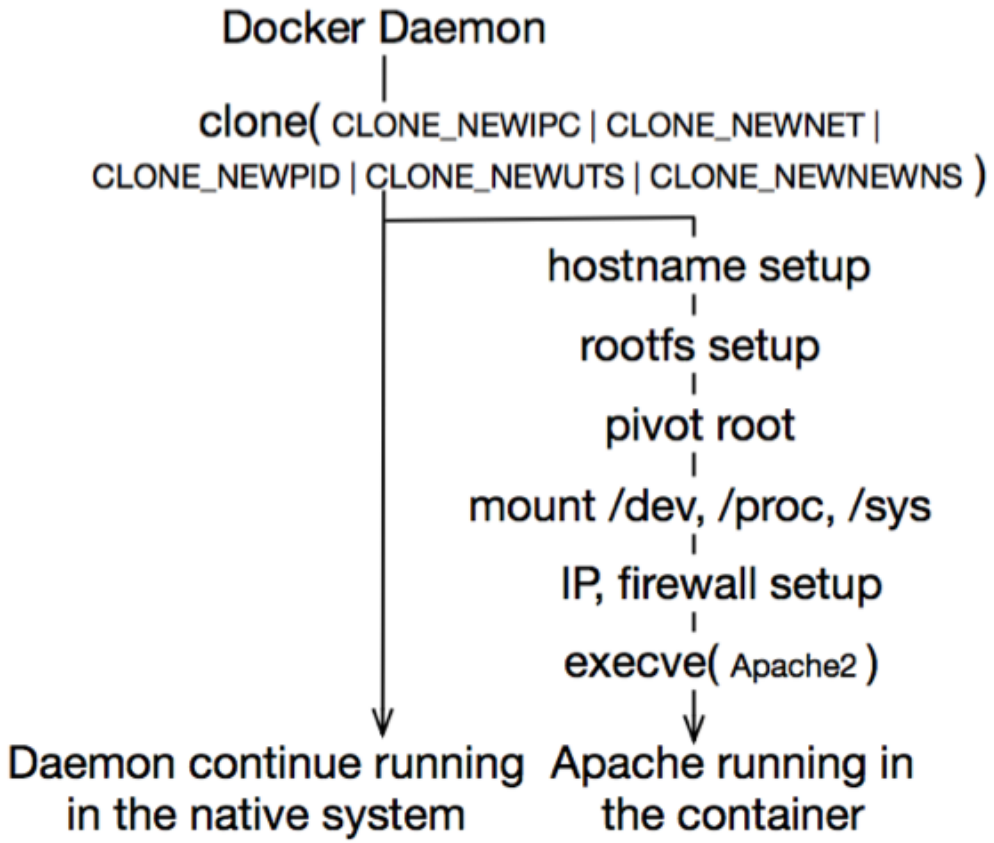
Background: Namespaces in Kernel (Cont.)

- There are ~~6~~ 7 namespaces isolating different types of kernel resources

Namespace	Constant	Isolates
Cgroup	**CLONE_NEWCGROUP**	Cgroup root directory
IPC	**CLONE_NEWIPC**	System V IPC, POSIX message queues
Network	**CLONE_NEWNET**	Network devices, stacks, ports, etc.
Mount	**CLONE_NEWNS**	Mount points
PID	**CLONE_NEWPID**	Process IDs
User	**CLONE_NEWUSER**	User and group IDs
UTS	**CLONE_NEWUTS**	Hostname and NIS domain name

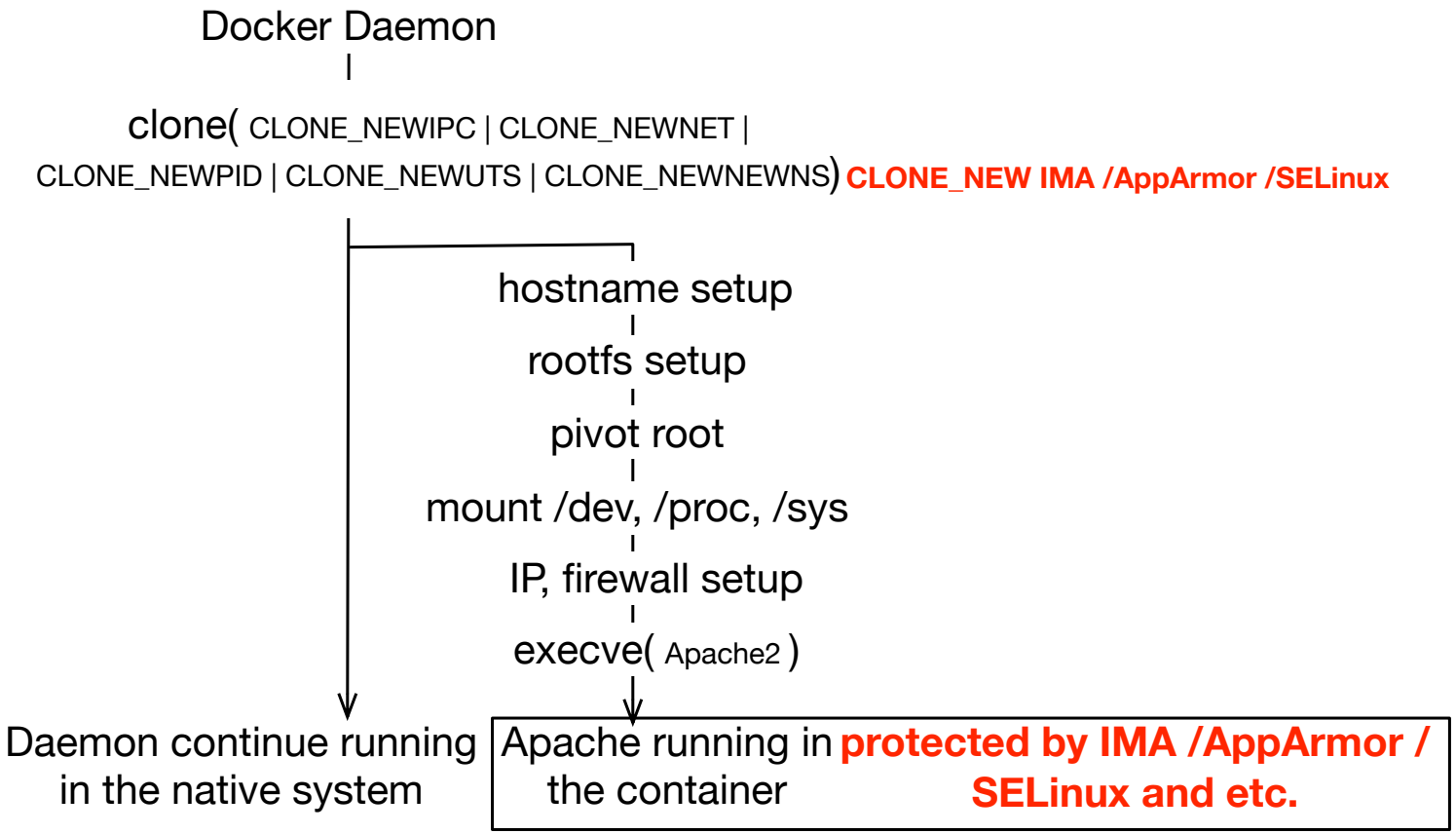
Background: Namespaces and Container

- There are ~~6~~ 7 namespaces isolating different types of kernel resources



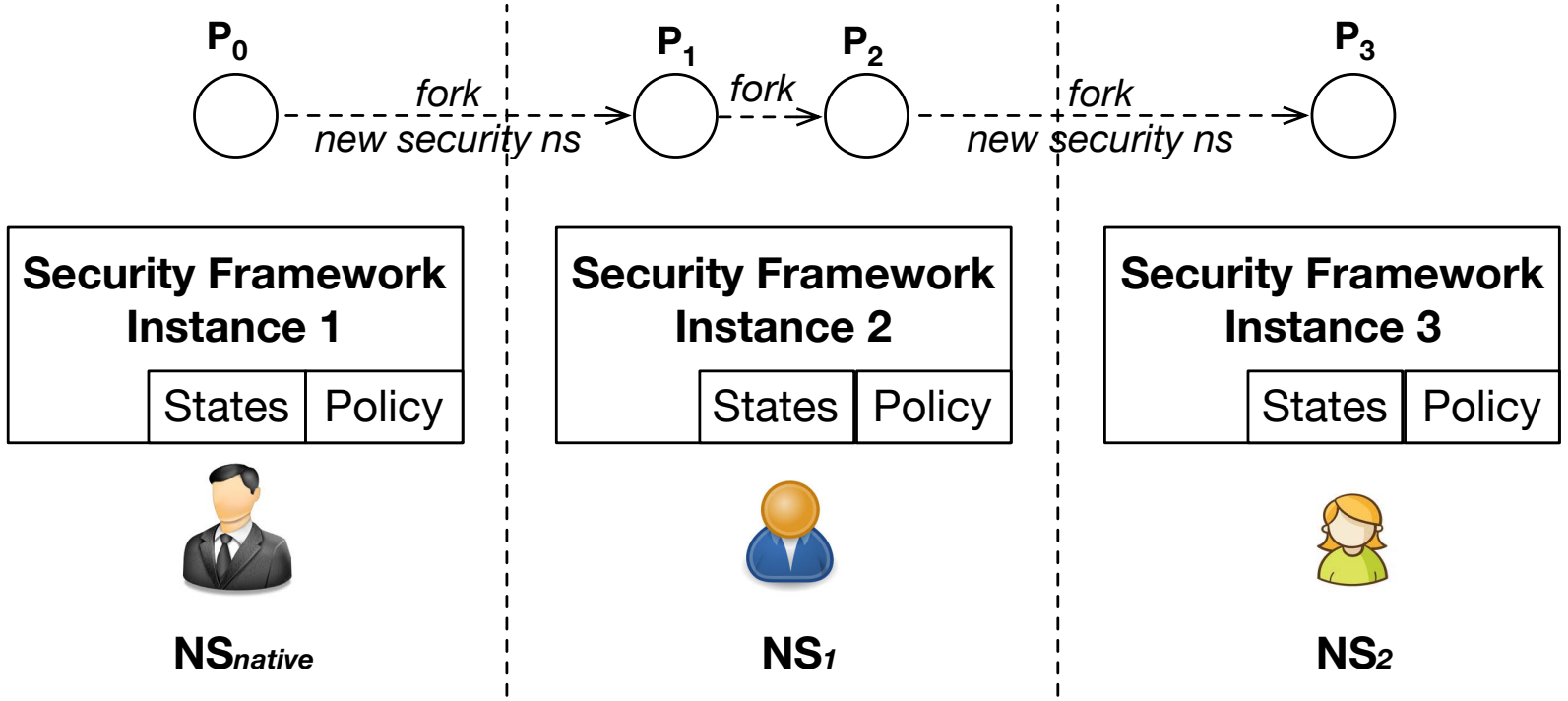
Background: Namespaces and Container

- There are ~~6~~ 7 namespaces isolating different types of kernel resources



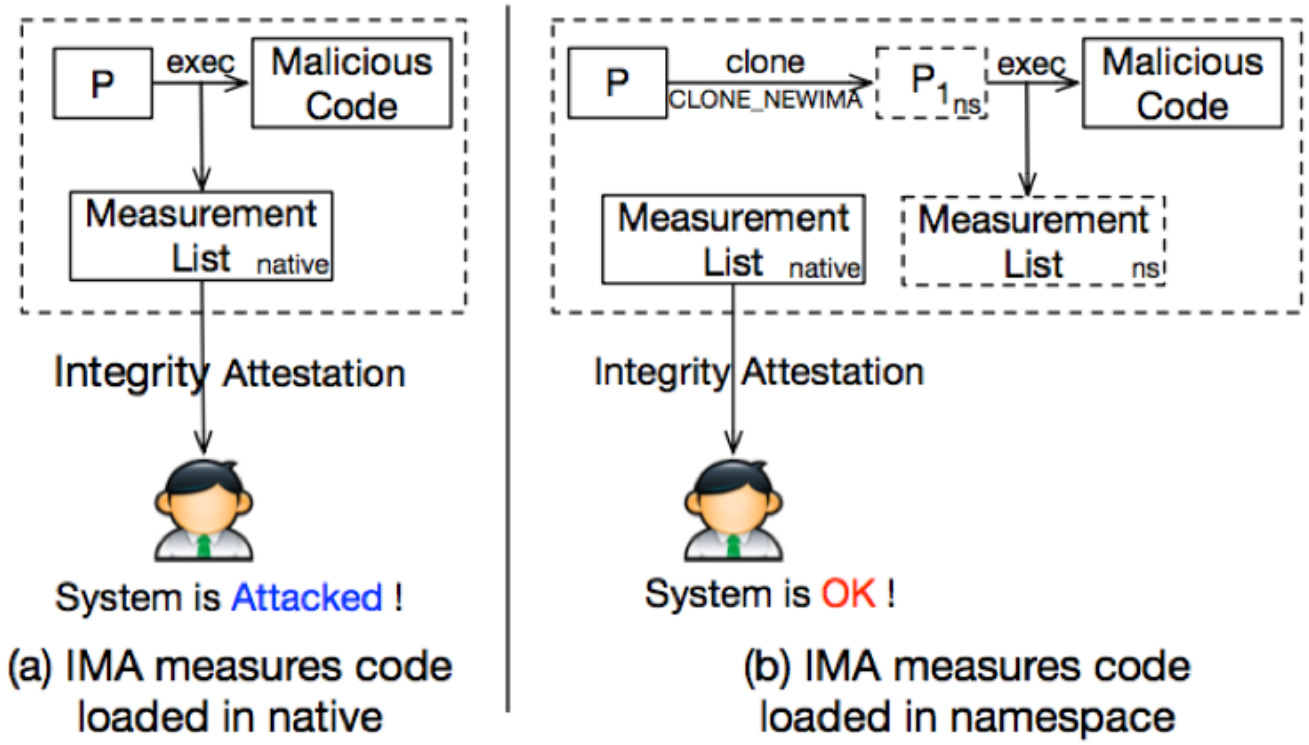
A Strawman Design

- Virtualize security framework into instances and divide control
  - NS_{native} controls P_0, NS_1 controls P_1 and P_2, *and* NS_2 controls P_3
  - NS_{native}, NS_1 and NS_2 independently applies security policies



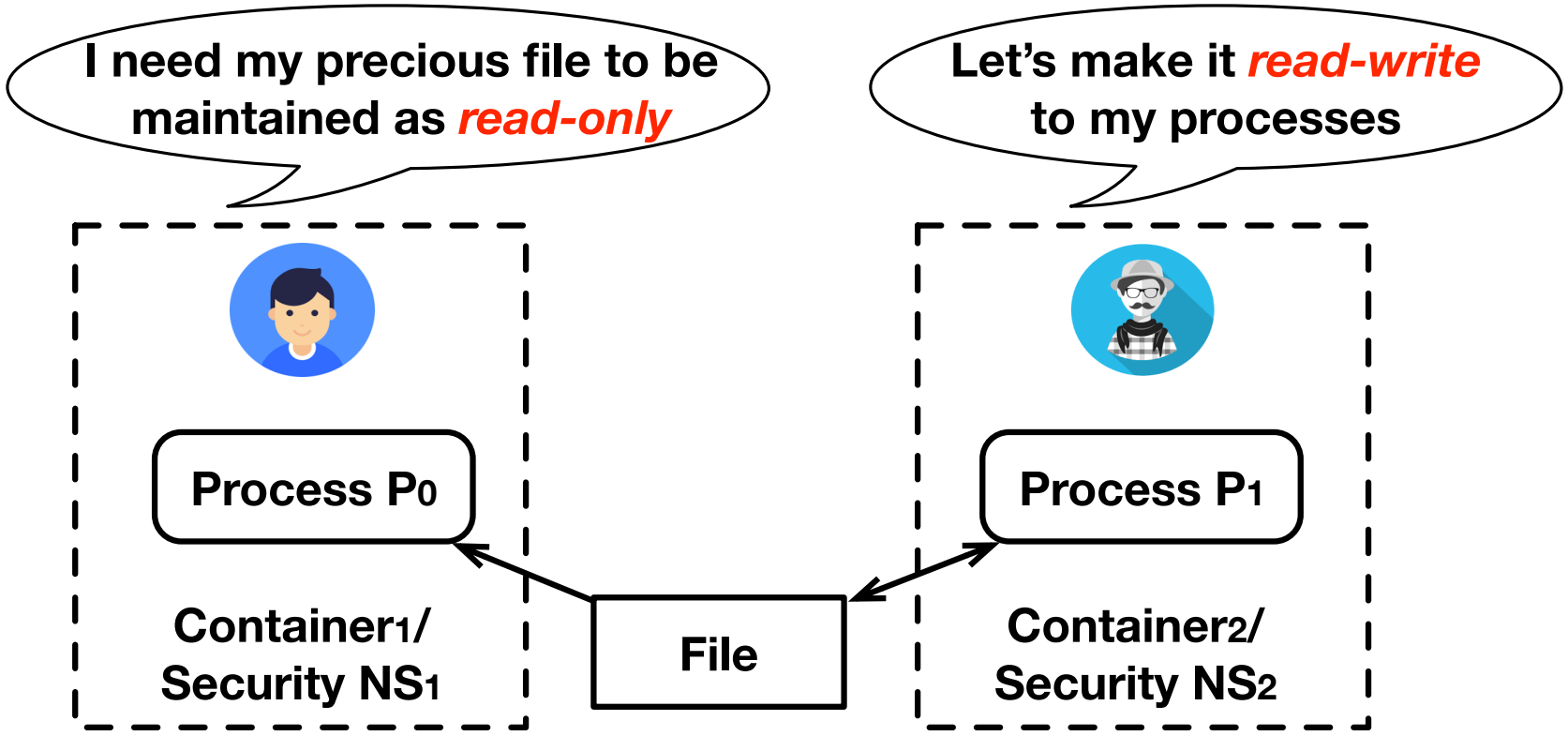
Attack in Strawman Design

- Kernel security frameworks are no longer **global**



Attack in Strawman Design (Cont.)

- Kernel security frameworks are are no longer **mandatory**



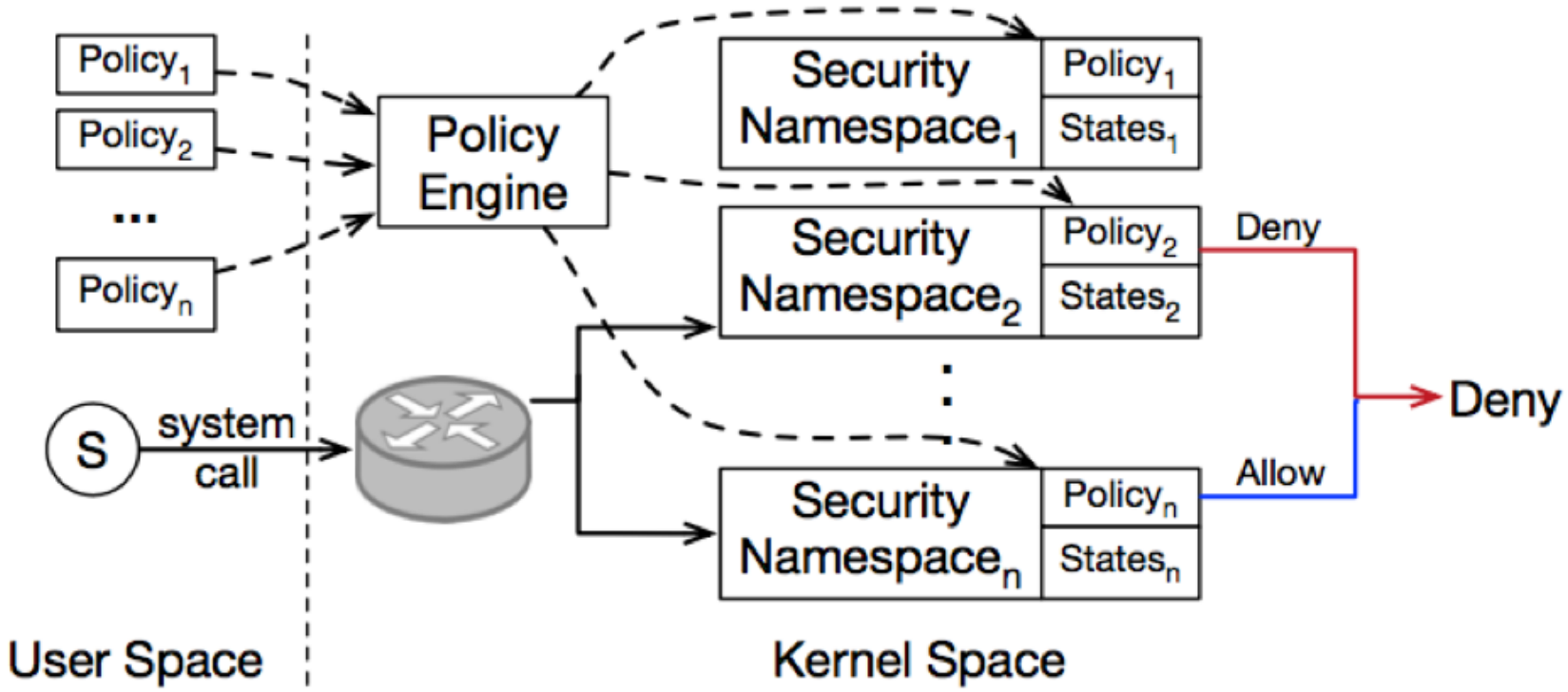
Challenges

- Kernel security frameworks are designed to be **global**
  - They control ALL processes running on a system (completeness for reference monitor)
  - But we should allow container owners to exercise control over limited set of processes (i.e., his/her own containers)
- Kernel security frameworks are designed to be **mandatory**
  - Only system admin may define security policies
  - But we should allow container owners to make his/her security decisions independently
- Relaxing the two requirements in a naïve way may compromise security offered by security frameworks

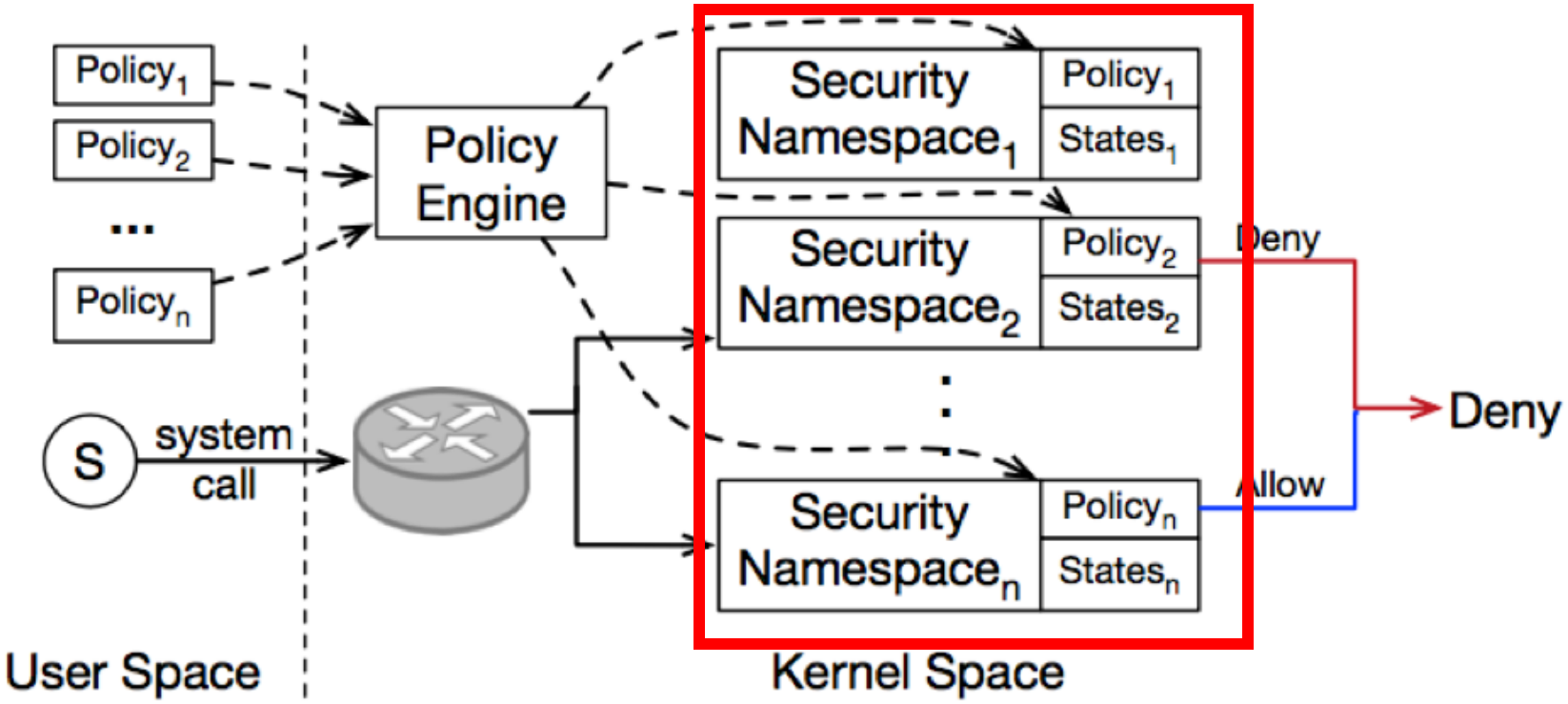
Insights

- Insight 1 (to relax global requirement)
  - Route an operation (i.e., system call) to ALL security namespaces whose security might be affected by the operation
- Insight 2 (to relax mandatory requirement)
  - Only allow an operation if all security namespaces affected by the operation allow the operation

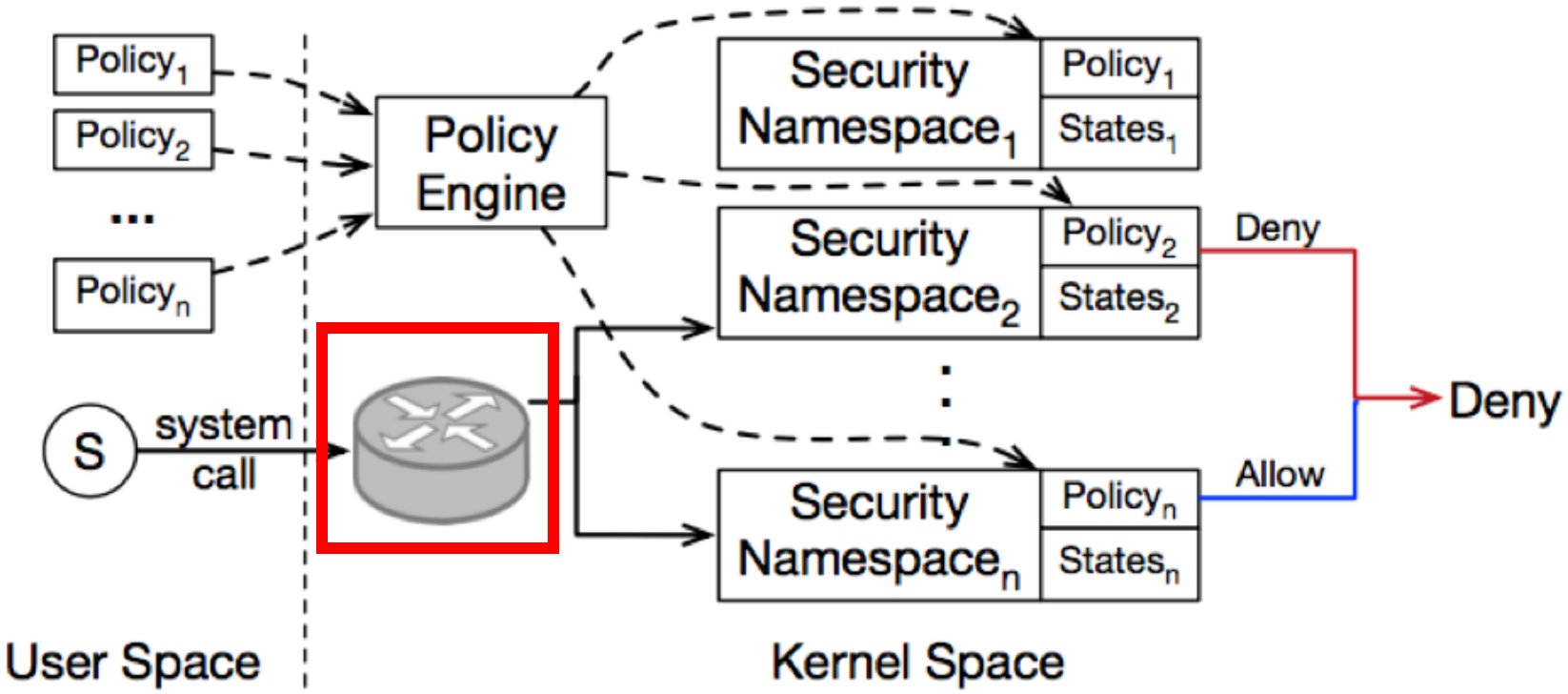
Solution Overview



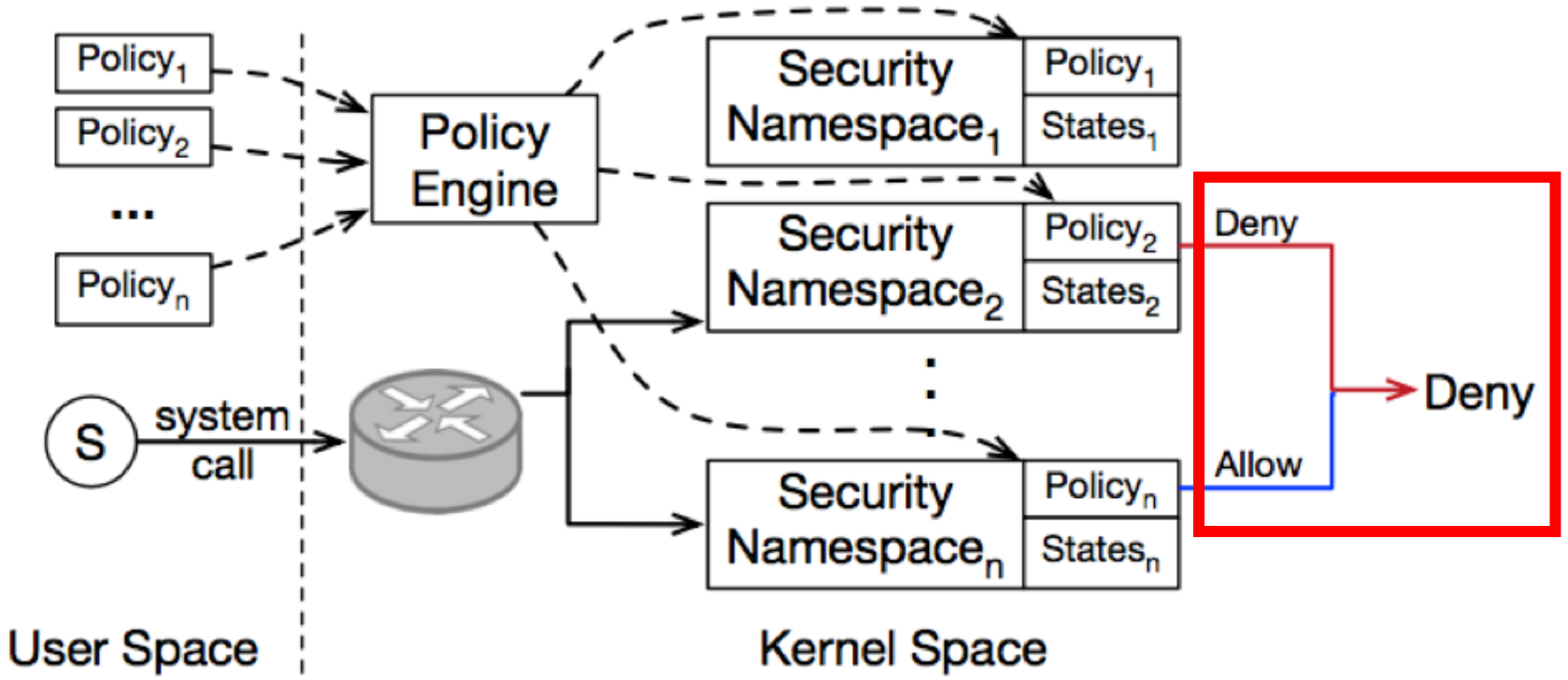
Solution Overview



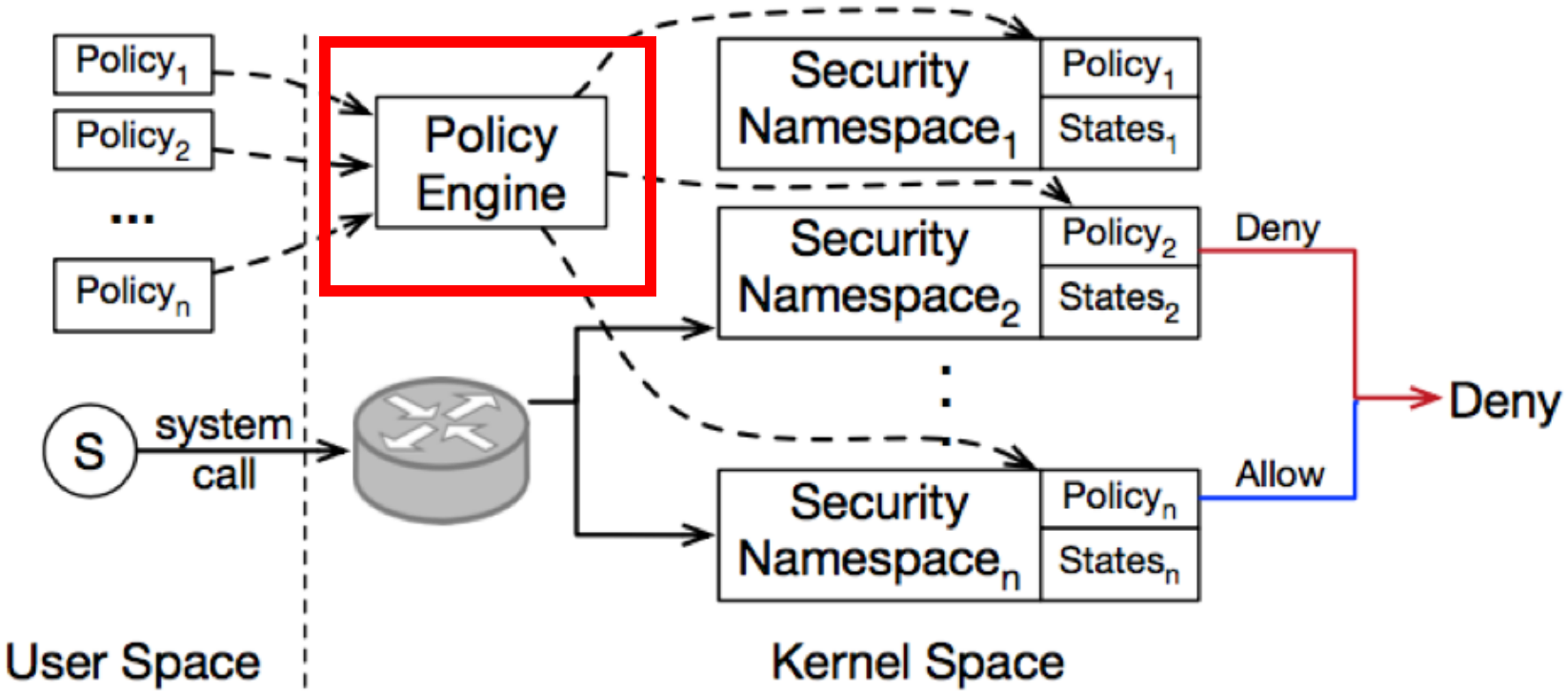
Solution Overview



Solution Overview



Solution Overview



Operation Router

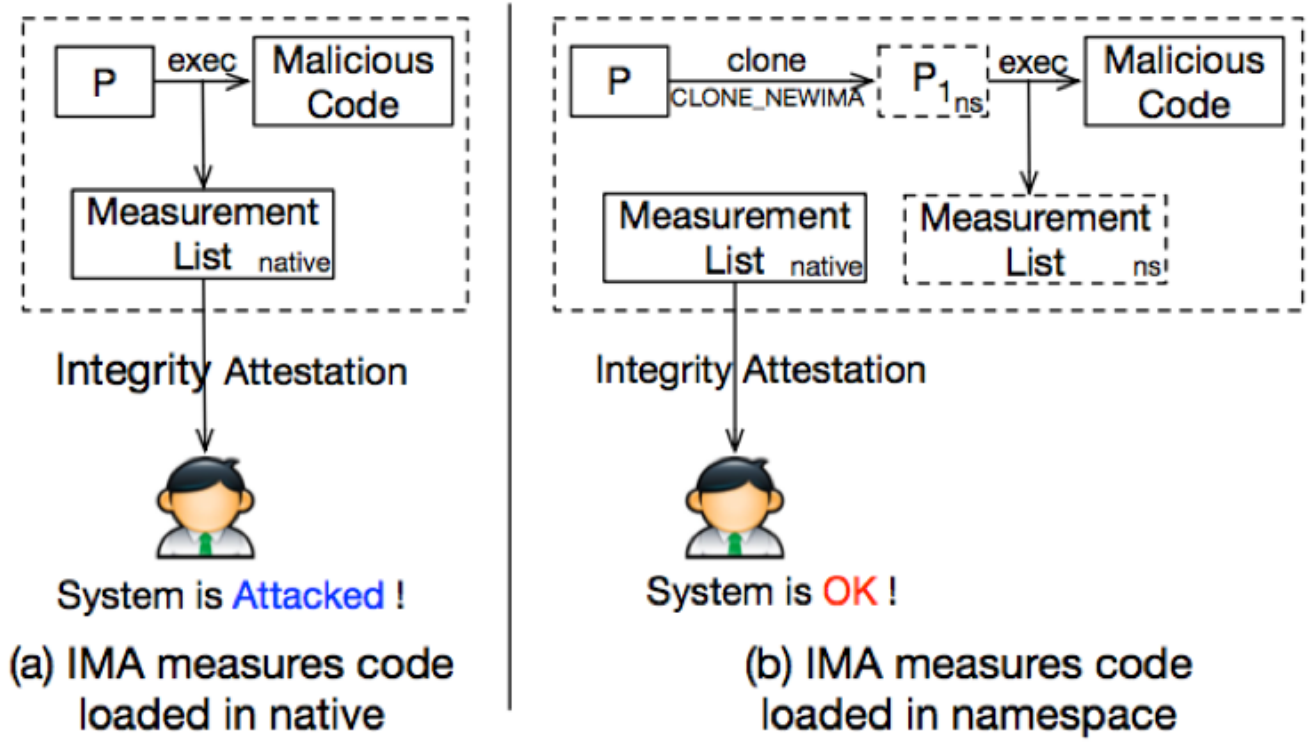
- Key Task:
  - Route an operation to all security namespaces whose security might be affected by the operation
- How:
  - Operation can be written as 3-tuple <*subject*, *object*, *operation*>
  - Security namespace has *implicit assumptions* over subject and object

A Subject's Perspective

- Implicit assumption of global
  - A security framework controls ***all*** subjects stemming from the first subject that it sees
  - For native → all subjects forked from init (PID 1)
  - For a container → all subjects forked from the first container process
- A subject may affect a security namespace
  - If the subject is associated with or a descendant of that security namespace

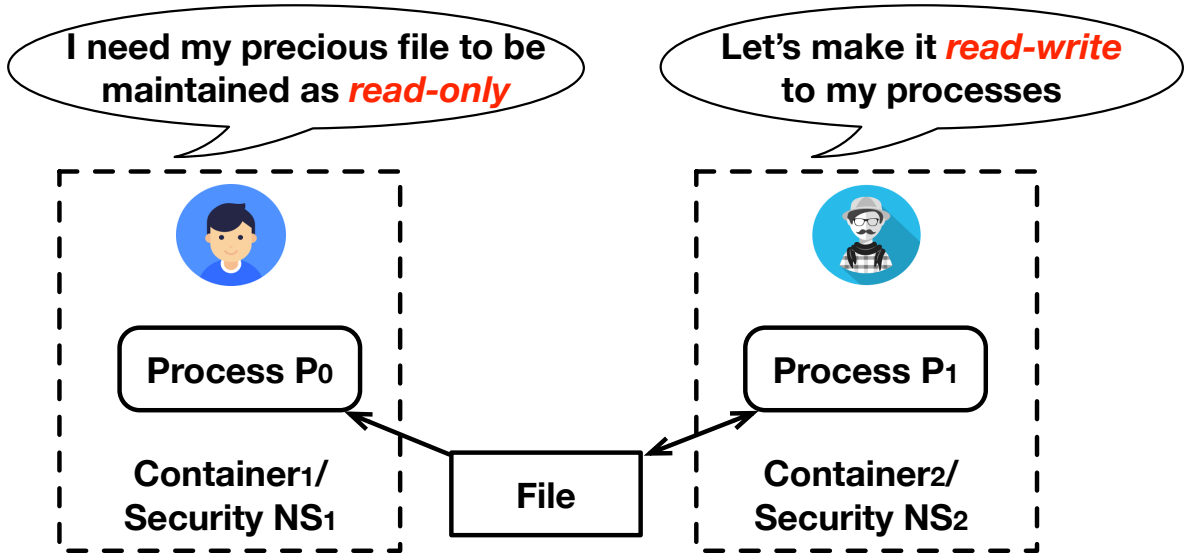
A Subject's Perspective (Cont.)

- A subject may affect a security namespace
  - If the subject is associated with or a descendant of that security namespace

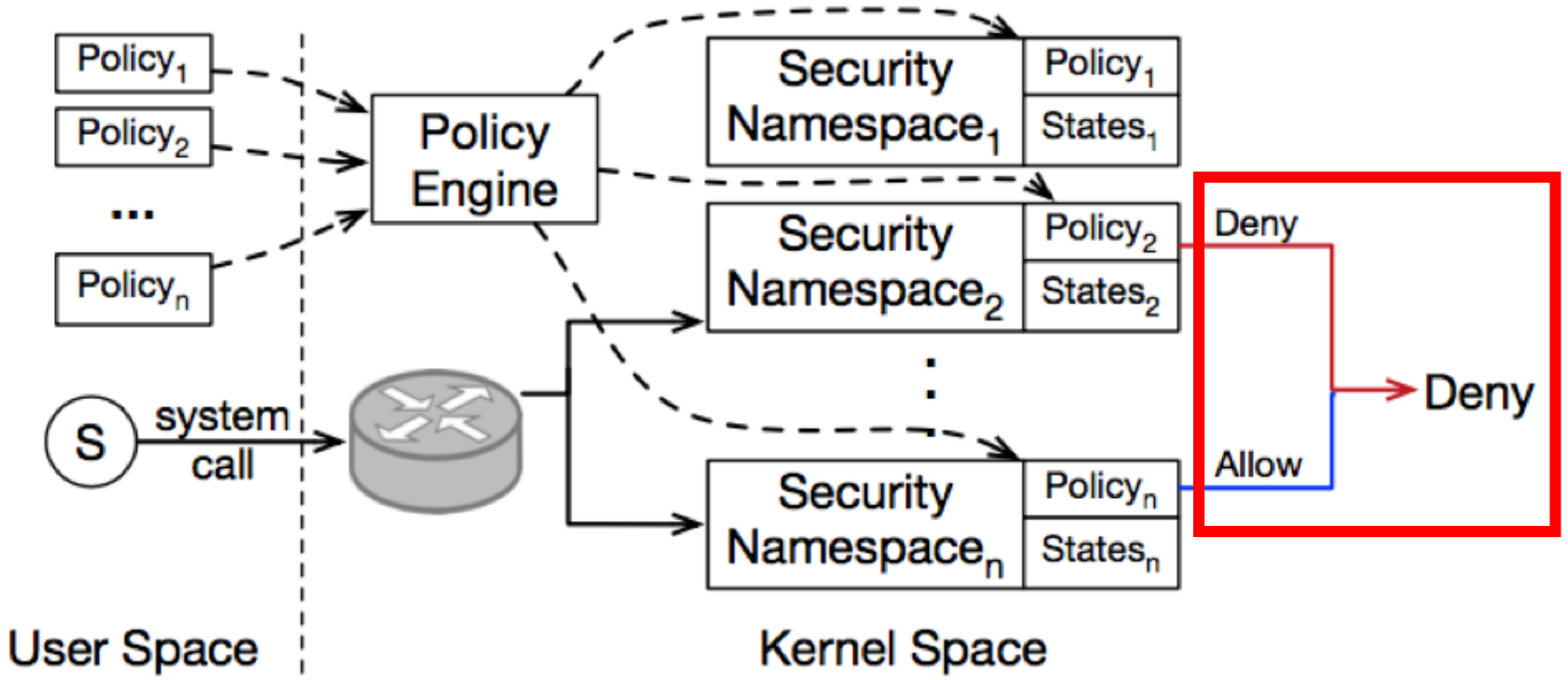


An Object's Perspective

- Implicit assumption of mandatory
  - Only operations explicitly allowed by a policy can be performed
- An object may affect a security namespace
  - If it is visible to the security namespace

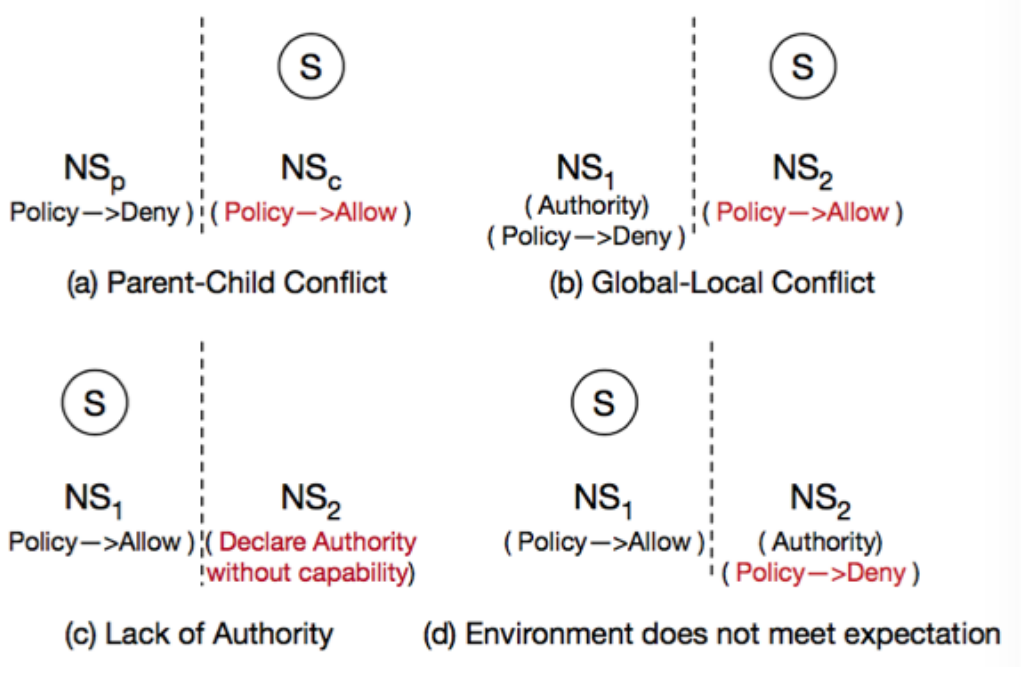


Solution Overview



Policy Engine

- Runtime conflicts affect usability
  - An operation might be eventually denied even if a container allowed it
  - Cannot debug since security namespaces are isolated
  - So we detect and inform potential conflicts ***at policy load time***



Implementation

- Namespace for two kernel security frameworks
  - IMA (Integrity Measurement Architecture) for integrity attestation
    - ~1.1K LOC in C
    - https://git.kernel.org/pub/scm/linux/kernel/git/zohar/linux-integrity.git/log/?h=next-namespacing-experimental
  - AppArmor for mandatory access control
    - ~1.5K LOC in C
- Less than 20 LOC extension to Docker libcontainer
  - Essentially use CLONE_NEWIMA & CLONE_NEWAPPARMOR flags in clone system call

Evaluation: AppArmor Namespace

Programs in a container cannot be further confined — for instance, MySQL runs under the container profile (protecting the host) but will not be able to enter the MySQL profile (to protect the container).

Evaluation: AppArmor Namespace (Cont.)

- Enforce both host profile (Docker profile) and application (in container) profile at the same time
  - Both profiles are shipped as default in Ubuntu

Application Profile	Conflicting Rules
Apache2	/proc/[pid]/attr/current *rw*
NTP	/dev/pps[0-9]* *rw*
firefox	/proc/ *r*
chrome	/proc/ *r*
MySQL, Perl, PHP5 OpenSSL, Samba, Ruby, Python Subversion, BitTorrent, Bash dhclient, dnsmasq, Squid OpenLDAP(slapd), nmbd, Tor	None

Evaluation: AppArmor Namespace (Cont.)

- Enforce both host profile (Docker profile) and application (in container) profile at the same time
  - Both profiles are shipped as default in Ubuntu

Container wants to allow something that container host denies

Application Profile	Conflicting Rules
Apache2	/proc/[pid]/attr/current *rw*
NTP	/dev/pps[0-9]* *rw*
firefox	/proc/ *r*
chrome	/proc/ *r*
MySQL, Perl, PHP5 OpenSSL, Samba, Ruby, Python Subversion, BitTorrent, Bash dhclient, dnsmasq, Squid OpenLDAP(slapd), nmbd, Tor	None

Sharing, Sharing, Conflicts, Conflicts

- /proc, /sys and /dev has been historically used as interfaces between kernel and userspace
  - Many objects under them (e.g., /proc/uptime) may break container isolation
  - Many applications need to access objects under them
- Solution
  - Fine tune both host and application profiles (e.g., do firefox need “/proc/ r”)
  - Avoid sharing
    - Device namespaces to isolate /dev? (e.g., NTP → “/dev/pps[0-9]* rw”)
    - Multi-layered filesystems to conceal /proc?

Evaluation: Performance

- System call latency added due to namespace
  - Common system calls hooked by LSM (e.g., read, write, mmap, execve)

mmap(μs)	IMA (stdev)	AppArmor (stdev)	slowdown
No security	1.08 (0.01)	1.08 (0.01)	
Native	1.26 (0.01)	1.38 (0.01)	
Native + 1NS	1.26 (0.01)	1.39 (0.02)	0.7%
Native + 2 NS	1.27 (0.01)	1.39 (0.02)	0.8%
Native + 5 NS	1.27 (0.01)	1.41 (0.02)	2.2%
Native + 10 NS	1.28 (0.01)	1.43 (0.02)	3.5%

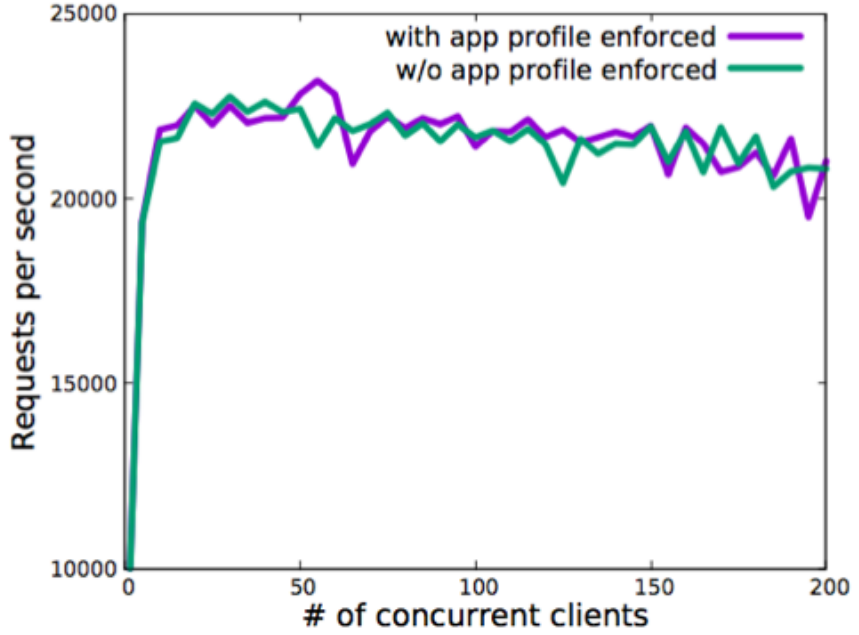
Evaluation: Performance

- System call latency added due to namespace
  - Common system calls hooked by LSM (e.g., read, write, mmap, execve)

mmap(μs)	IMA (stdev)	AppArmor (stdev)	slowdown
No security	1.08 (0.01)	1.08 (0.01)	
Native	1.26 (0.01)	1.38 (0.01)	
Native + 1NS	1.26 (0.01)	1.39 (0.02)	0.7%
Native + 2 NS	1.27 (0.01)	1.39 (0.02)	0.8%
Native + 5 NS	1.27 (0.01)	1.41 (0.02)	2.2%
Native + 10 NS	1.28 (0.01)	1.43 (0.02)	3.5%

Evaluation: Performance

- System call latency added due to namespace
  - Common system calls hooked by LSM (e.g., read, write, mmap, execve)
- Application performance
  - Containerized Apache throughput (w and w/o application profile)



Summary

- Existing containers cannot leverage kernel security frameworks to apply local security control
  - A naïve virtualization may break security offered by security frameworks
- The routing based security namespace design preserves security while making kernel security frameworks available to containers

Backup Slides

SELinux Namespace

- Routing based design largely apply, but…
- Challenge 1: filesystem labeling
  - Subjects and objects will have multiple labels?
    - Multiple security attributes
    - Runtime manipulation of security attributes without reboot
- Challenge 2: policy conflict detection
  - Statically decide potential labels for subjects?
    - Hard to predict due to the complexity of transition rules