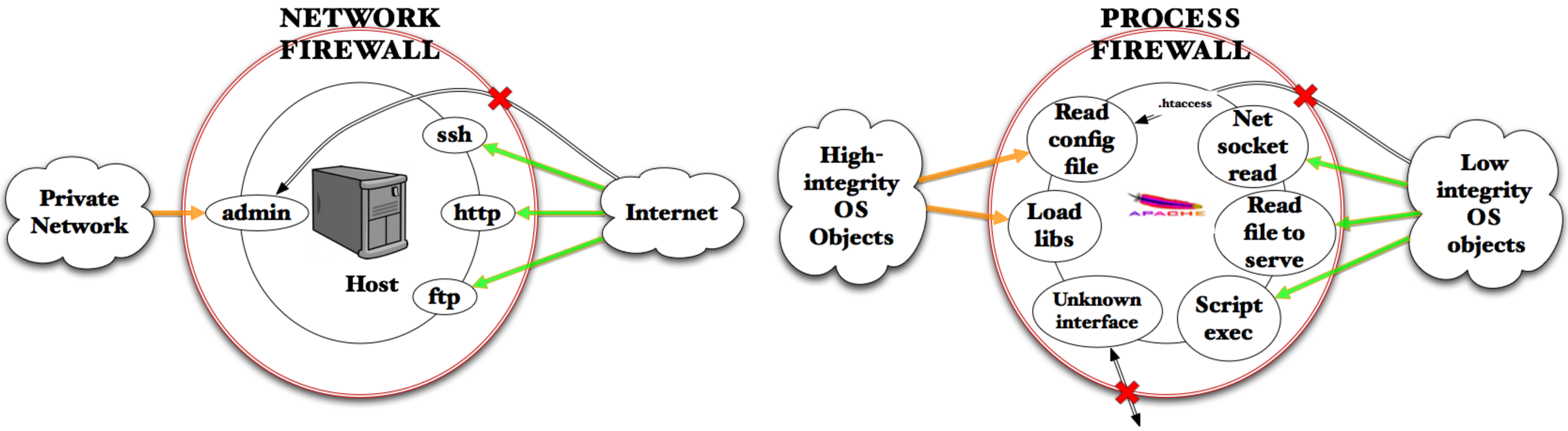
Process Firewalls – Mechanism and Utility

HAYAWARDH VIJAYAKUMAR, SANDRA RUEDA, DIVYA MUTHUKUMARAN, JOSHUA SCHIFFMAN, TRENT JAEGER



Modern operating systems have several protection mechanisms like mandatory access control, and integrity protection mechanisms (e.g., tripwire). However, there is no enforcement of which interfaces in programs access which resources in an OS, resulting in attackers supplying low-integrity data through interfaces do are not meant to receive them. This situation is analogous to what existed before network firewalls were introduced. Hosts were trusted to access the network securely, by opening only certain ports. However, other ports were opened to the outside world – some reasons being misconfiguration (e.g., allowing public access to administrative interfaces) or code injected by attackers after compromising a bug in the host (e.g., bot control through IRC). Network firewalls enforced what the system administrators expected of each network port of the host when accessing the network.

A similar situation exists for processes – misconfigurations may cause low-integrity data to be read through a wrong interface, and code injected into a process may cause it to open new interfaces. We propose a similar mechanism for processes, a *process firewall*, to solve the above problems and in general, to enforce what system administrators expect from each program interface when accessing an operating system object.

Mechanism

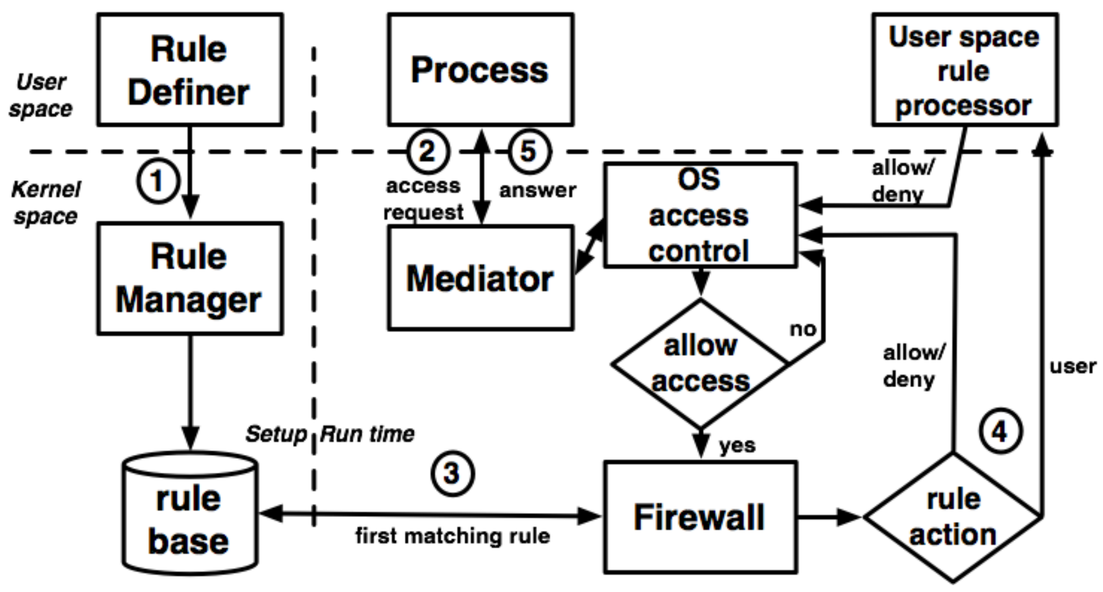
Our process firewall is analogous to iptables.

Called when LSM hooks are invoked (indicating security-sensitive object access).

Interfaces are identified by the address, binary executing at that address, and the context in which the program is executing.

Sample rule.
Allow only shared-library files labeled `lib_t` to be loaded for all applications.

```
pftables --table FILTER                --append INPUT
         --object-label lib_t    --interface 0x15d04
         --vm_area_name /lib/ld.so --binary *
         --process-label *          --jump ACCEPT
```



Challenges

Automation.
- Large amount of fine-grained rules to specify, some sort of automation is required.
- Runtime analysis is performed to identify which interfaces access which objects, and rules are generated automatically for interfaces in applications.
- To reduce the number of rules, we have default rules (e.g., allowing high-integrity objects to be accessed by all interfaces). We have an algorithm to generate high- and low-integrity objects for a particular subject.

Scalability.
- For an Ubuntu-8.04 server system, we identify of 154 interfaces in various trusted programs that need to accept low-integrity input.
- Hashing of rules to speed up rule match.

Other Firewall Uses

Runtime Monitoring.
- Similar to iptables, we have a QUEUE jump target, which allows a user-space rule processor to make a decision when certain objects are accessed.
  This could allow program integrity mechanisms (eg., VM introspection) to be turned on selectively only when a low integrity input is encountered, thus reducing their overhead.