Operating Systems & C

Lecture 9: Monitors

Willard Rafnsson
IT University of Copenhagen

Bad Code

- Adversary may control the code that you run
- Examples
  - Classical: Viruses, Worms, Trojan horses, …
  - Modern: Client-side scripts, Macro-viruses, Email, Ransomware, …
- Easier to update/add software (malware) than ever
- What are the problems with adversary code on your machine?

Bad Code - Example

- You run an adversary-controlled program
  - What can an adversary do?
- Anything you can do
  - Do you have anything you would want to protect?
    - Secret data on your computer
    - Communications you make with your computer
- Well, at least these are only “user” processes
  - They do not *directly* compromise the host
    - Beware “local exploits”

Bad Code - Defenses

- What can you do to avoid executing adversary-controlled code?
- Defenses
  - Only run "approved" code
    - How do you know?
    - Use automated installers or predefined images
      - Let someone else manage it
  - "Sandbox" code you are uncertain of
    - How do you do that?

Good Code

- Fortunately, most code is not adversary controlled
  - I think…
- What is the problem with running code from benign sources?
  - Not really designed to defend itself from a determined, active adversary
- Functions performed by benign code may be exploited – i.e., have vulnerabilities

Vulnerabilities

- A program vulnerability consists of three elements:
  - A flaw
  - Accessible to an adversary
  - Adversary has the capability to exploit the flaw
- Often focus on a subset of these elements
  - But all conditions must be present for a true vulnerability

Good Code – Goes Bad

- Classic flaw: Buffer overflow
- If adversary can access, exploits consist of two steps usually
  - (1) Gain control of execution – IP or stack pointer
  - (2) Choose code for performing exploitation
- Classic attack:
  - (1) Overwrite return address
  - (2) Write code onto stack and execute that

Good Code – Defenses

- Preventing either of these two steps prevents a vulnerability from being exploited
- How to prevent overwriting the return address?
  - ???
- How to prevent code injection onto the stack?
  - ???
- Are we done?
  - End the semester early…

Good Code – Evading Defenses

- Unfortunately, no
- (1) Adversaries gain access to the control flow in multiple ways
  - Function pointers, other variables, heap variables, etc.
  - Or evade defenses – e.g., disclosure attacks
- (2) Adversaries may perform desired operations without injecting code
  - Return-to-libc
  - Return-oriented attacks

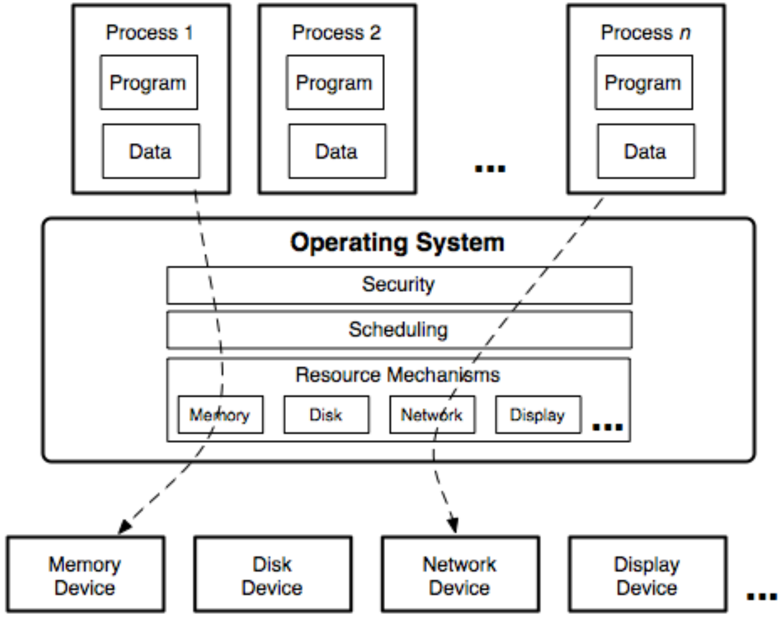
Good Code – Confused Deputy

- And an adversary may accomplish her goals without any memory errors
  - Trick the program into performing the desired, malicious operations
- Example "confused deputy" attacks
  - SQL injection
  - Resource access attacks
  - Bypass attacks
  - Race condition attacks (TOCTTOU)

Result

- Adversaries have a variety of ways to try to get code under their control running on your computer
- Software defenses may not prevent exploitation
  - And still lots of room for improvement
- Malware and intrusion detection is a hard problem
  - How do we know whether code is bad or good?
- Systems security is about blocking damage or limiting damage from adversary-controlled execution
  - Not doing well enough yet

Operating Systems



Control Bad Code

- What mechanism does an OS use to restrict the rights of processes (i.e., running code) from system resources?

WORRYING = WASTE OF TIME. GOOD AND BAD THINGS WILL HAPPEN IN LIFE. YOU JUST HAVE TO KEEP LIVING AND NOT STRESS OVER WHAT YOU CAN'T CONTROL.

KUSHANDWIZDOM

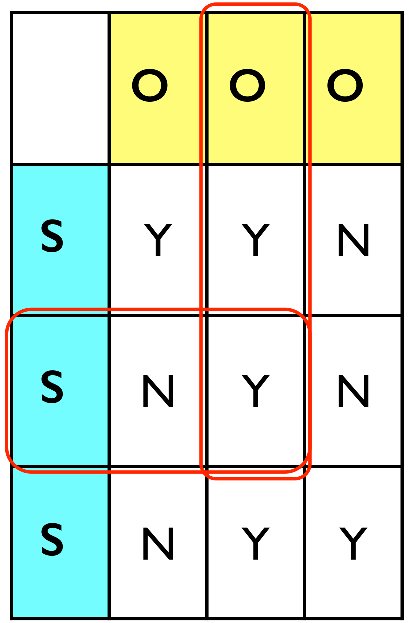
Access Control

- System makes a decision to grant or reject an access request
  - from an already authenticated subject
  - based on what the subject is authorized to access
- Access request
  - Object: System resource
  - Operations: One or more actions to be taken
  - Subject: Process that initiated the request
- Access Control Mechanisms enforce Access Control Policies to make such decisions

Access Matrix

PENNSTATE 1855

- Lampson formalizes the model of access control in his 1970 paper “Protection”
- Called Access Matrix
  - Rows are subjects
  - Columns are objects
  - Authorized operations listed in cells
- To determine if S_i has right to access object O_j, compare the request ops to the appropriate cell

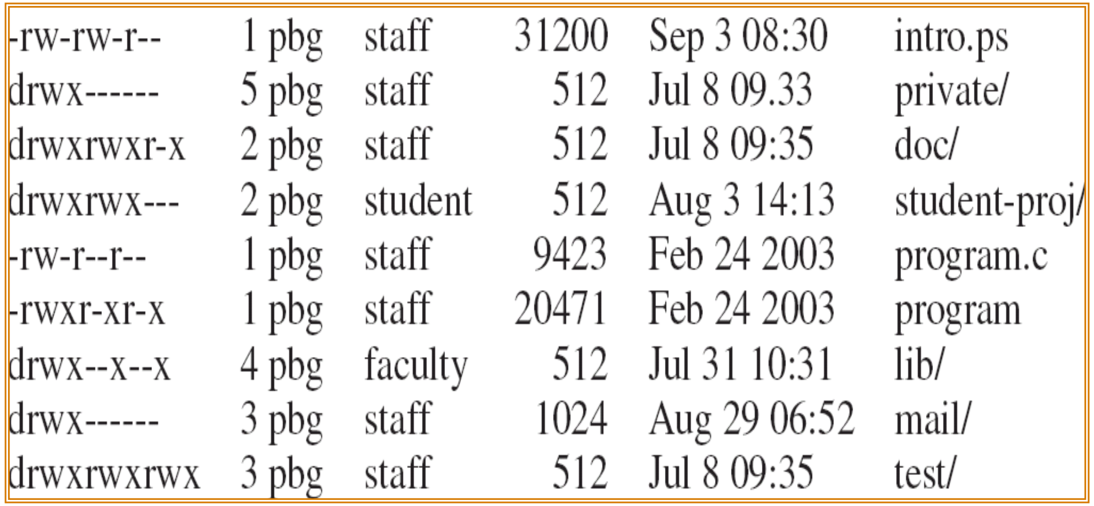


	O	O	O
S	Y	Y	N
S	N	Y	N
S	N	Y	Y

UNIX Access Control

- On Files
  - All objects are files
  - Not exactly true
- Classical Protection System
  - Limited access matrix
  - Discretionary protection state operations
- Practical model for end users
  - Still involves some policy specification

UNIX Mode Bits



```
-rw-rw-r--    1 pbg  staff    31200  Sep 3 08:30    intro.ps
drwx------    5 pbg  staff      512  Jul 8 09.33    private/
drwxrwxr-x    2 pbg  staff      512  Jul 8 09:35    doc/
drwxrwx---    2 pbg  student    512  Aug 3 14:13    student-proj/
-rw-r--r--    1 pbg  staff     9423  Feb 24 2003    program.c
-rwxr-xr-x    1 pbg  staff    20471  Feb 24 2003    program
drwx--x--x    4 pbg  faculty    512  Jul 31 10:31   lib/
drwx------    3 pbg  staff     1024  Aug 29 06:52   mail/
drwxrwxrwx    3 pbg  staff      512  Jul 8 09:35    test/
```

Access Matrix

- Using the Access Matrix
- (1) Suppose J wants to prevent other users' processes from reading/writing her private key (object O_1)
- (2) Suppose J wants to prevent other users' processes from writing her public key (object O_2)
- Design the access matrix
- Are these the rights on your host to your SSH public and private keys?

	O	O	O
J	?	?	?
S	?	?	?
S	?	?	?

Access Matrix

- Using the Access Matrix
- (1) Suppose J wants to protect a private key (object O_1) from being leaked to or modified by others
- (2) Suppose J wants to prevent a public key (object O_2) from being modified by others
- Design the access matrix
- Will this access matrix protect the keys' secrecy and integrity?

	O	O	O
J	?	?	?
S	?	?	?
S	?	?	?

Consider Bad Code Again

- Claim: Any code you run may be able to compromise either of the key files
- For the private key
  - Any process running under your user id can read and leak your private key file
- For the public key
  - Any process running under your user id may modify the public key file
    - Often people make the public key file read-only even to the owner
    - Is that enough?

Consider Bad Code Again

- Claim: Any code you run may be able to compromise either of the key files
- For the private key
  - Any process running under your user id can read and leak your private key file
- For the public key
  - Any process running under your user id may modify the public key file
    - Often people make the public key file read-only even to the owner
    - No. Processes running on behalf of the owner may change perms

Bad Code - Examples

- Suppose you download and run adversary-controlled code (e.g., Trojan horse)
  - It will run with all your permissions
  - Even can modify the permissions of any files you own
- Suppose you run benign code that is compromised by an adversary – becoming bad
  - Is effectively the same as above if adversary can choose code to execute (e.g., return-oriented attack)
  - Adversaries can also trick victims into performing operations on their behalf (e.g., confused deputy attack)

Protection vs. Security

- Protection
  - Secrecy and integrity met under *benign* processes
  - Protects against an error by a non-malicious entity
- Security
  - Security goals met under *potentially malicious* processes
  - Enforces requirements even if adversary is in complete control of the process
- Hence, for J: Non-malicious process shouldn't leak the private key by accident to a specific file owned by others
- A potentially malicious process may contain a Trojan horse that can write the private key to files chosen by adversaries

Fundamentally Flawed

- Conventional operating system mechanisms enforce protection rather than security
  - Protection is fundamentally incapable of defending from an active and determined adversary

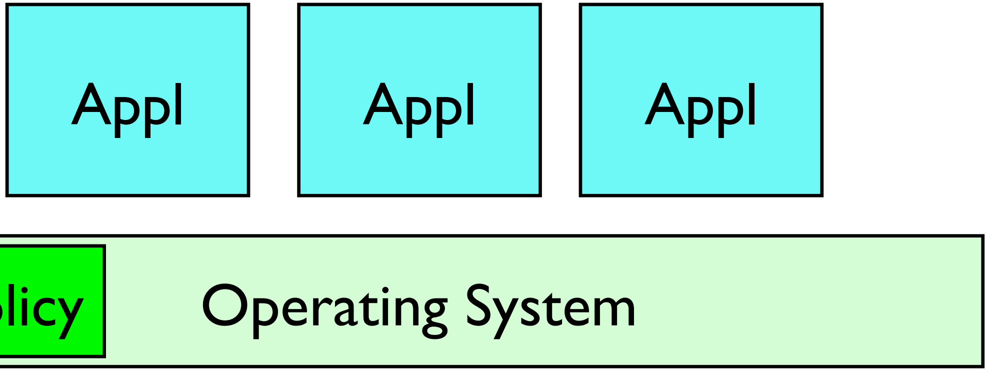
Secrecy

- Process secrecy requires that the process not communicate with unauthorized parties
  - But what about a process that services requests?
  - This is a very difficult requirement to meet
- Suppose a benign process can write to a file controlled by an adversary
- Unless the process is trusted to contain no vulnerabilities then the process could be compromised (is *potentially malicious*)

Integrity

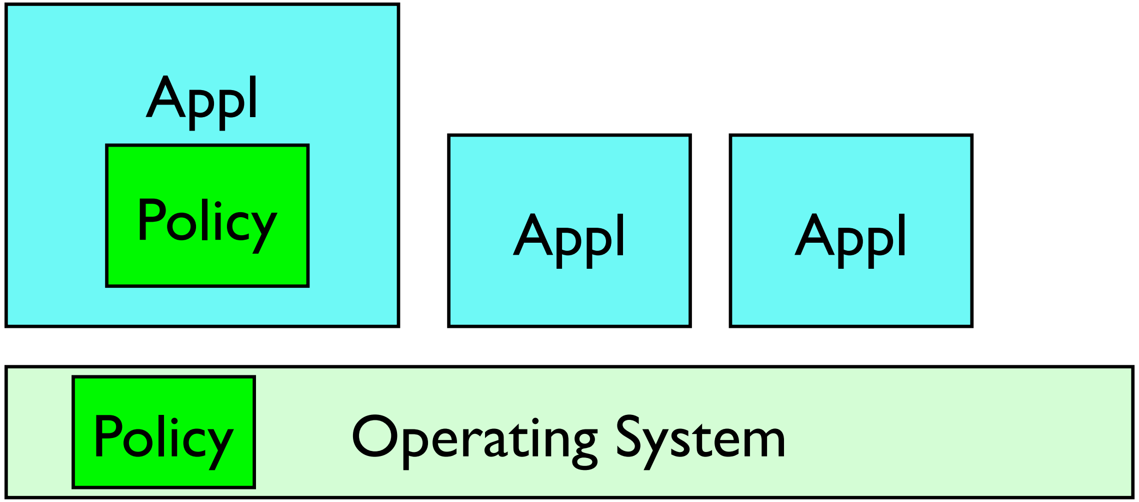
- Process integrity requires that the process not depend on adversary input
  - What does “depend on” mean?
  - This is a very difficult requirement to meet
- Suppose a benign process can read from a file controlled by an adversary
- Unless the process is trusted to contain no vulnerabilities then the process could be compromised (is *potentially malicious*)

Trusted Computing Base



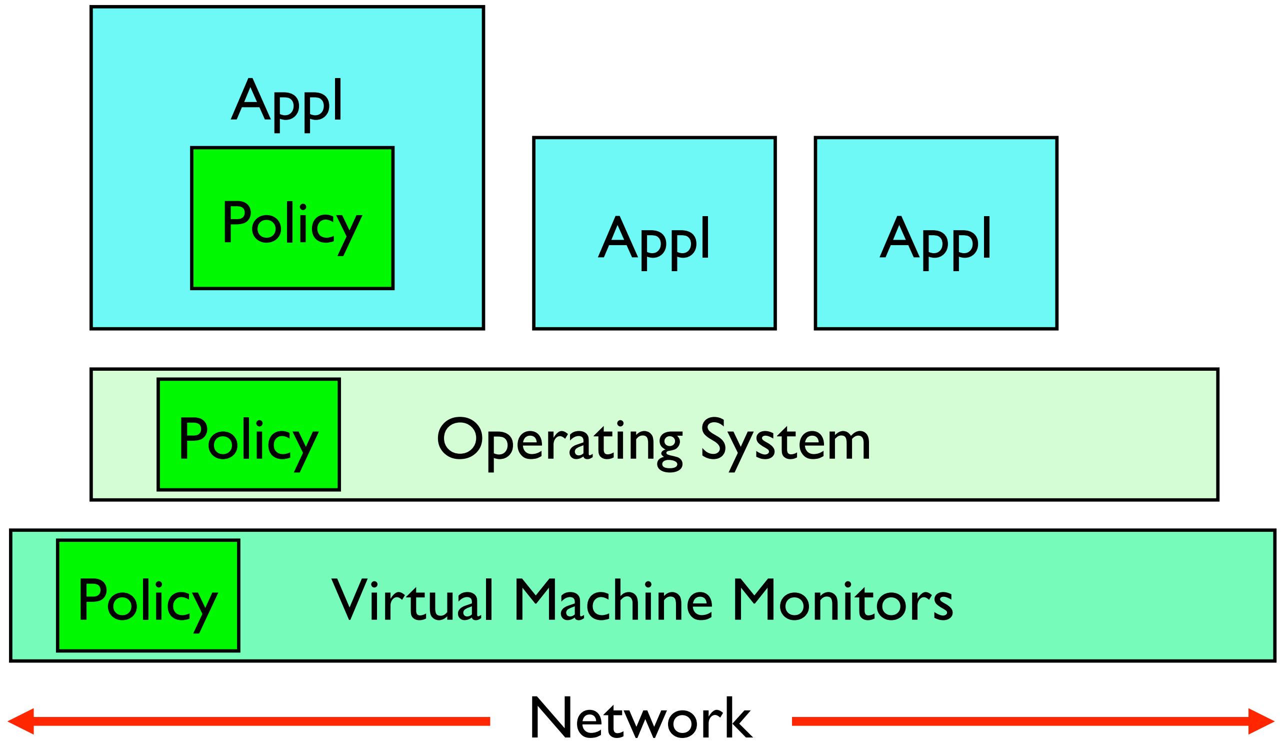
- Historically, OS treats applications as black boxes
  - OS controls flows among applications
  - Security requirements determined by allowed flows

Policy Enforcement in Apps



- Application policy enforcement: databases, JVM, X Windows, daemons, browsers, email clients, servers

Multi-Layered Enforcement



Security Enforcement

- Several applications include access control
  - *Databases, window servers, web servers, browsers, …*
- Some programming systems include access control to system resources
  - *Java, Safe-Tcl, Ruby, Python, Perl – Jif, Flow Caml (information flow);*
- Some systems recognize that programs may contribute to access control
  - *User-level policy server for SELinux*
  - *Information Flow Control*
- *Requirement: Ensure that all layers are using their authority in a manner consistent with system security goals*

Questions for This Class

- How do we keep bad code off our systems?
- How do we keep benign code from becoming bad code?
- How do we prevent benign code from being tricked into being a confused deputy?
- How do we restrict code that may be/go bad from propagating damage?
- How can we leverage the myriad of system defenses to control code efficiently?
- How do we know what we configured works?

Take Away

- Traditional OS access control
  - Is for protection, not security
- So it cannot confine an active adversary
  - Build attacks that work despite access control
  - They can change the access control policies
- Access control is enforced in many places now
  - Can we utilize them comprehensively and efficiently?

Protection System

- Manages the authorization policy for a system
    - It describes what operations each subject (via their processes) can perform on each object
- Consists of
    - **State:** *Protection state*
    - **State Ops:** *Protection state operations*

Access Matrix Protection System

- Protection State
    - Current state of matrix
- Can modify the protection state
    - Via protection state operations
    - E.g., can create objects
    - E.g., owner can add a subject, operation mapping for their objects
- Lampson’s “Protection” paper
    - Can even delegate authority to perform protection state ops

Protection System

- Why is Protection State insufficient to enforce security?
- Goal: a protection state in which we can determine whether an unauthorized operation will ever be allowed (Safety)

Protection System Problems

- Protection system approach is inadequate for security
  - Suppose a process runs bad code
- Processes can change their own permissions
  - Processes may become untrusted, but can modify policy
- Processes, files, etc. are created and modified
  - Cannot predict in advance (safety problem)
- What do we need to achieve necessary controls?

Define and Enforce Goals

- Claim: *If we can define and enforce a security policy that ensures security goals, then we can prevent such attacks*
- How do we know what policy will be enforced?
- How do we know the enforcement mechanism will enforce policy as expected?
  - Look into this today
- How do we know the policy expresses effective goals?
  - Will look into this in depth later

Mandatory Protection System

- Is a *protection system* that can be modified only by *trusted administration* that consists of
  - A *mandatory protection state* where the protection state is defined in terms of an immutable set of *labels* and the *operations that subject labels can perform on object labels*
  - A *labeling state* that assigns system subjects and objects to those labels in the mandatory protection state
  - A *transition state* that determines the legal ways that subjects and objects may be relabeled
- MPS is *immutable* to user-space process

Mandatory Protection State

- Immutable table of
  - Subject labels
  - Object labels
  - Operations authorized for former upon latter
- How can you use an MPS to control use of bad code?
  - E.g., Prevent modification of kernel memory?

Mandatory Protection State

- Immutable table of
  - Subject labels
  - Object labels
  - Operations authorized for former upon latter
- How can you use an MPS to control use of bad code?
  - E.g., Prevent modification of kernel memory?
  - Subject labels for all subjects running "bad code" are not allowed modify kernel memory
    - Or that may run "bad code" (be compromised)
  - How do subjects (processes) get their labels?

Labeling State

- Immutable rules mapping
  - Subjects to labels (in rows)
  - Objects to labels (in columns)
- How can you use labeling state to control bad code?
  - E.g., Prevent modification of kernel memory?
  - Assign all processes that may run bad code …
  - With a label that cannot modify kernel memory
  - What about objects created by these processes?

Protecting Good Code

- How can you use labeling state to prevent good code from going bad?

Protecting Good Code

- How can you use labeling state to prevent good code from going bad?
  - E.g., Prevent dependence on untrusted input?
  - Assign object labels to all objects that may be adversary-controlled
  - Do not grant subject labels that should run good code access to those labels
  - Verify that you are running good code (how?) and assign to one of these protected subject labels
  - What integrity model does this approximate?

Protecting Good Code

- What if good code needs to access some adversary-controlled resources?

Mandatory Protection State

- What if good code needs to access some adversary-controlled resources?
  - (1) if a process reads adversary-controlled object label, remove privileged permissions (e.g., to modify kernel memory)
  - (2) if a process reads adversary-controlled object label, remove permission to write to any object that may be accessed by a subject whose label grants privileged permissions
- How do we achieve this change with the MPS?

Transition State

- Immutable rules mapping
  - Subject labels to conditions that change their subject labels
  - Object labels to conditions that change their object labels
- How can you use labeling state to control bad code?
  - E.g., Achieve (1) and (2)

Transition State

- Immutable rules mapping
  - Subject labels to conditions that change their subject labels
  - Object labels to conditions that change their object labels
- How can you use labeling state to control bad code?
  - E.g., Achieve (1) and (2)
  - Change subject label of subject accessing adversary-controlled resources to remove these permissions
  - What integrity model does this approximate?

Transition State

- Is it possible to launch processes with more permissions than the invoker with MPS?

Managing MPS

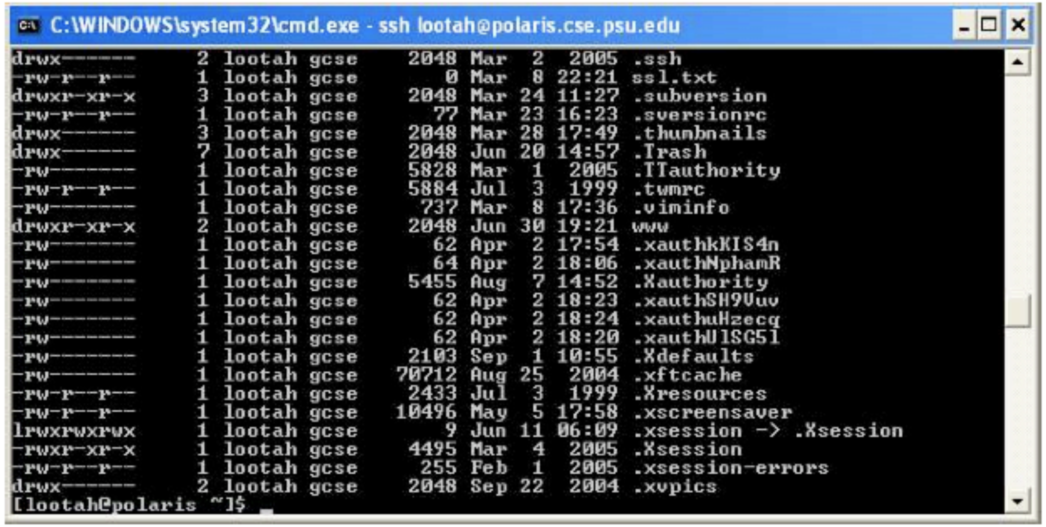
- Challenge
  - Determining how to set and manage an MPS in a complex system involving several parties
- Parties
  - What does programmer know about deploying their program securely?
  - What does an OS distributor know about running a program in the context of their system?
  - What does an administrator know about programs and OS?
  - Users?

Managing MPS

- Current methods use dynamic analysis to setup MAC policies – run the program and collect the permissions used
  - Really a functional policy

Linux Authorization circa 2000

- Linux implements discretionary access control



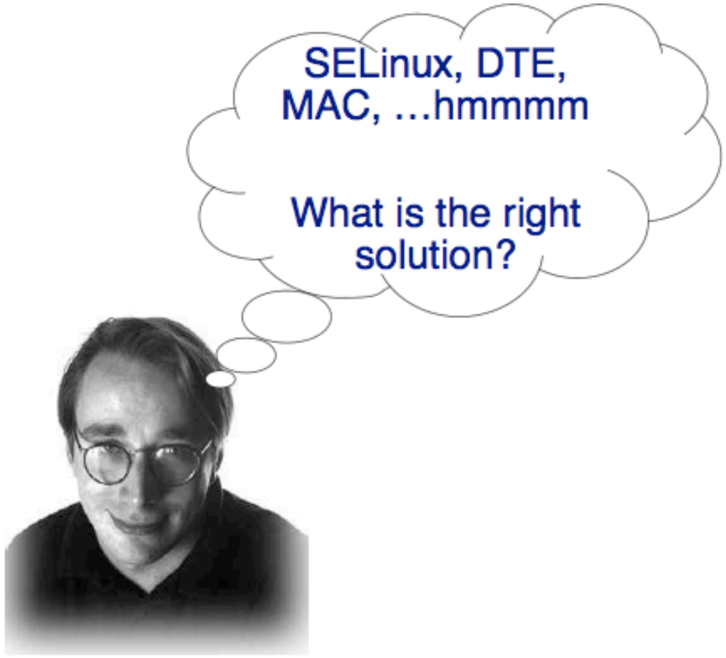
C:\WINDOWS\system32\cmd.exe - ssh lootah@polaris.cse.psu.edu

```
drwx------    2 lootah gcse     2048 Mar  2  2005 .ssh
-rw-r--r--    1 lootah gcse        0 Mar  8 22:21 ssl.txt
drwxr-xr-x    3 lootah gcse     2048 Mar 24 11:27 .subversion
-rw-r--r--    1 lootah gcse       77 Mar 23 16:23 .sversionrc
drwx------    3 lootah gcse     2048 Mar 28 17:49 .thumbnails
drwx------    7 lootah gcse     2048 Jun 20 14:57 .Trash
-rw-------    1 lootah gcse     5828 Mar  1  2005 .TTauthority
-rw-r--r--    1 lootah gcse     5884 Jul  3  1999 .twmrc
-rw-------    1 lootah gcse      737 Mar  8 17:36 .viminfo
drwxr-xr-x    2 lootah gcse     2048 Jun 30 19:21 www
-rw-------    1 lootah gcse       62 Apr  2 17:54 .xauthkKIS4n
-rw-------    1 lootah gcse       64 Apr  2 18:06 .xauthNphamR
-rw-------    1 lootah gcse     5455 Aug  7 14:52 .Xauthority
-rw-------    1 lootah gcse       62 Apr  2 18:23 .xauthSH9Vuv
-rw-------    1 lootah gcse       62 Apr  2 18:24 .xauthuHzecq
-rw-------    1 lootah gcse       62 Apr  2 18:20 .xauthWlSG5l
-rw-------    1 lootah gcse     2103 Sep  1 10:55 .Xdefaults
-rw-------    1 lootah gcse    70712 Aug 25  2004 .xftcache
-rw-r--r--    1 lootah gcse     2433 Jul  3  1999 .Xresources
-rw-r--r--    1 lootah gcse    10496 May  5 17:58 .xscreensaver
lrwxrwxrwx    1 lootah gcse        9 Jun 11 06:09 .xsession -> .Xsession
-rwxr-xr-x    1 lootah gcse     4495 Mar  4  2005 .Xsession
-rw-r--r--    1 lootah gcse      255 Feb  1  2005 .xsession-errors
drwx------    2 lootah gcse     2048 Sep 22  2004 .xvpics
[lootah@polaris ~]$ _
```

Linux Security circa 2000

- Patches to the Linux kernel
  - Enforce different access control policy
    - Restrict root processes
  - Some hardening
- Argus PitBull
  - Limited permissions for root services
- RSBAC
  - MAC enforcement and virus scanning
- grsecurity
  - RBAC MAC system
  - Auditing, buffer overflow prevention, /tmp race protection, etc
- LIDS
  - MAC system for root confinement

Linus' Dilemna



The Answer

- The solution to all computer science problems

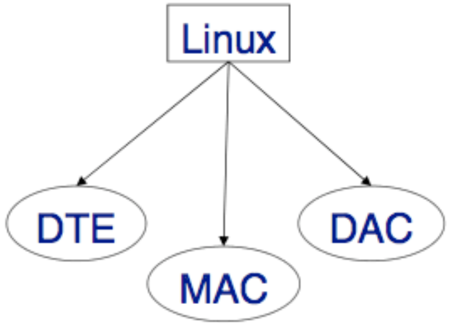
- Add another layer of indirection

Linux Security Modules Was Born

- “to allow Linux to support a variety of security models, so that security developers don't have to have the ‘my dog's bigger than your dog’ argument, and users can choose the security model that suits their needs.”, Crispin Cowan
  - http://mail.wirex.com/pipermail/linux-security-module/2001-April:/0005.html

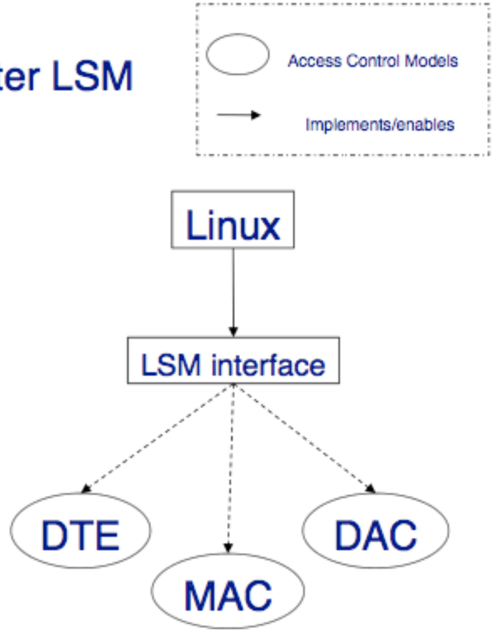
Linux Before and After

Before LSM



Access control models implemented as Kernel patches

After LSM



Access control models implemented as Loadable Kernel Modules

LSM Requirements

- LSM needs to reach a balance between kernel developer and security developers requirements. LSM needs to unify the functional needs of as many security projects as possible, while minimizing the impact on the Linux kernel.
  - Truly generic
  - conceptually simple
  - minimally invasive
  - Efficient
  - Support for POSIX capabilities
  - Support the implementation of as many access control models as Loadable Kernel Modules

LSM – A Reference Monitor

- To enforce mandatory access control
  - We need to develop an authorization mechanism that satisfies the reference monitor concept
- How do we do that?
  - And satisfy all the other goals?

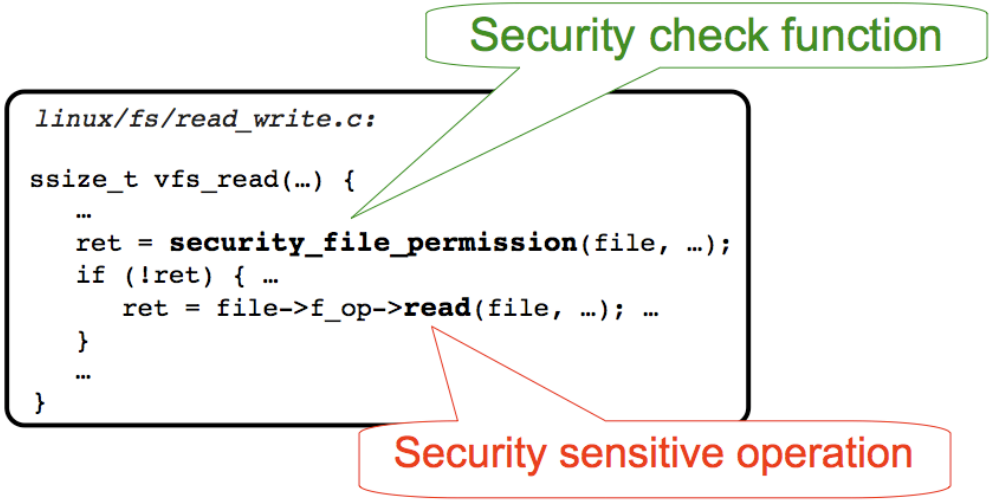
LSM – Complete Mediation

- First requirement is complete mediation
- Add security hooks to mediate various operations in the kernel
    - These hooks invoke functions defined by the chosen module
- These hooks construct “authorization queries” that are passed to the module
    - Subject, Object, Operations

LSM Hooks

- Function calls that can be overridden by security modules to manage security fields and mediate access to Kernel objects.
- Hooks called via function pointers stored in `security->ops` table
- Hooks are primarily “restrictive”

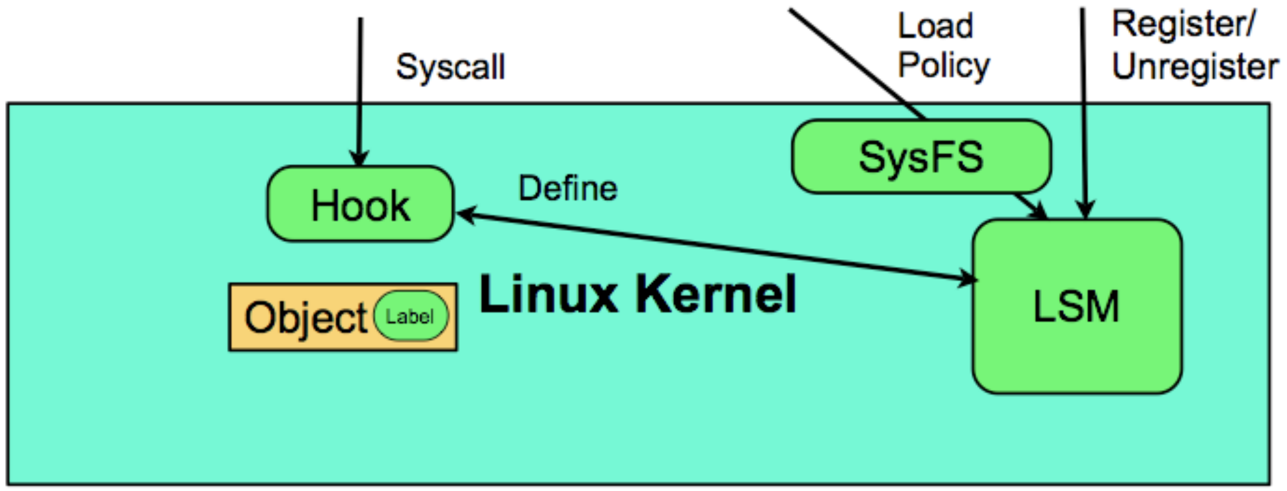
LSM Hooks



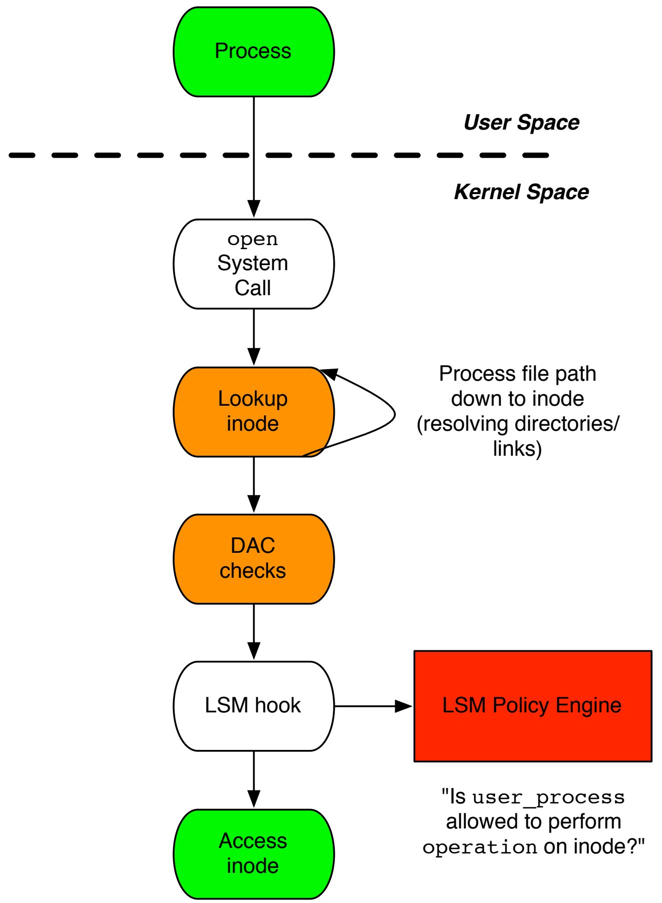
```
linux/fs/read_write.c:

ssize_t vfs_read(…) {
   …
   ret = security_file_permission(file, …);
   if (!ret) { …
      ret = file->f_op->read(file, …); …
   }
   …
}
```

LSM Hook Architecture



LSM Hook Architecture


Process
User Space
Kernel Space
open System Call
Lookup inode
Process file path down to inode (resolving directories/ links)
DAC checks
LSM hook
LSM Policy Engine
"Is user_process allowed to perform operation on inode?"
Access inode

LSM – Complete Mediation

- First requirement is complete mediation
- Enables authorization by module
- Linux extends “sensitive data types” with opaque security fields
  - Modules manage these fields – e.g., store security labels
- Which Linux data types are sensitive?

LSM Security Fields

- Enable security modules to associate security information to Kernel objects
- Implemented as void* pointers
- Completely managed by security modules
- What to do about object created before the security module is loaded?

LSM – Complete Mediation

- First requirement is complete mediation
- How do we know LSM implements complete mediation?
- Asked one of the lead developers (Cowan)
  - His reply?

LSM – Complete Mediation

- First requirement is complete mediation
- How do we know LSM implements complete mediation?
- Asked one of the lead developers (Cowan)
  - His reply?
- "We don't"

LSM – Tamperproof

- Second requirement is tamperproof
- Prevent adversaries from modifying the reference monitor code or data
- How is LSM code protected?
- How is LSM data protected?

LSM – Tamperproof

- Second requirement is tamperproof
- Add functions to register and unregister Linux Security Modules
  - Implemented as a set of function pointers defined at registration time
- LSM module defines code
- LSM function pointers define targets of hooks
  - These are data – modifiable
- Implications?

LSM – Tamperproof

- Second requirement is tamperproof
- Add functions to register and unregister Linux Security Modules
    - Implemented as a set of function pointers defined at registration time
- Adversaries could modify the code executed by Linux by modifying these function pointer data values
    - Some people opposed this idea and refused to participate
    - Eventually changed to require compiled-in LSM modules

LSM Tasks

- Linux Kernel modified in 5 ways:
  - Opaque security fields added to certain kernel data structures
  - Security hook function calls inserted at various points with the kernel code
  - A generic security system call added
  - Function to allow modules to register and unregistered as security modules
  - Move capabilities logic into an optional security module

Hook Details

- Difference from discretionary controls
  - More object types
    - 29 different object types
    - Per packet, superblock, shared memory, processes
    - Different types of files
  - Finer-grained operations
    - File: ioctl, create, getattr, setattr, lock, append, unlink,
  - System labeling
    - Not dependent on user
  - Authorization and policy defined by module
    - Not defined by the kernel

LSM Performance

- Microbenchmark: LMBench
  - Compare standard Linux Kernel 2.5.15 with Linux Kernel with LSM patch and a default capabilities module
  - Worst case overhead is 5.1%
- Macrobenchmark: Kernel Compilation
  - Worst case 0.3%
- Macrobenchmark: Webstone
  - With Netfilter hooks 5-7%
  - Uni-Processor 16%
  - SMP 21% overhead

LSM Use

- Available in Linux 2.6
  - Packet-level controls upstreamed in 2.6.16
- Modules
  - POSIX Capabilities module
  - SELinux module
  - Domain and Type Enforcement
  - Openwall, includes grsecurity function
  - LIDS
  - AppArmor
- Not everyone is in favor
  - How does LSM impact system hardening?

Take Away

- Aiming for mandatory controls in Linux
  - But everyone had their own approach
- Linux Security Modules is a general interface for any* authorization module
  - Much finer controls – interface is union of what everyone can do
- What does this effort say about
  - Achieving complete mediation?
  - Whether complete mediation should be policy-dependent?

Reference Monitor for Linux

- LSM provides a reference monitor interface for Linux
  - Complete Mediation
- You need a module and infrastructure to achieve the other two goals
  - Tamperproofing
  - Verifiability
- *SELinux is a comprehensive reference validation mechanism aiming at reference monitor guarantees*

SELinux History

- Origins go back to the Mach microkernel retrofitting projects of the 1980s
    - DTMach (1992)
    - DTOS (USENIX Security 1995)
    - Flask (USENIX Security 1999)
    - SELinux (2000-…)
- Motivated by the security kernel design philosophy
    - But, practical considerations were made

The Rest of the SELinux Story

- Tamperproof
  - Protect the kernel
  - Protect the trusted computing base
  - *Use MPS to provide tamperproofing of TCB?*
- Verifiability
  - Code correctness
  - Policy satisfy a security goal
  - *Use MPS to express secrecy and integrity requirements*

Design MPS

- Do not believe that classical integrity is achievable in practice
  - Too many exceptions
  - Commercial systems will not accept constraints of classical integrity
- Instead, focus on providing comprehensive control of access aiming for
  - Confining root processes (tamperproof)
  - Least privilege in general (verifiability)
- *How does 'least privilege' affect security?*

SELinux Policy Rules

- SELinux Rules express an MPS
  - *Protection state – ALLOW subject-label object-label ops*
  - *Labeling state – TYPE_TRANSITION subject-label object-label new-label (at create – objects)*
    - Default is to label to same state as creator
  - *Transition state – TYPE_TRANSITION subject-label object-label new-label (at exec – processes)*
- Tens of thousands of rules are necessary for a standard Linux distribution
  - Protect system processes from user processes
  - User data can be protected by MLS

SELinux Transition State

- For user to run passwd program
  - Only passwd should have permission to modify */etc/shadow*
- Need permission to execute the passwd program
  - *allow user_t passwd_exec_t:file execute (user can exec /usr/bin/passwd)*
  - *allow user_t passwd_t:process transition (user gets passwd perms)*
- Must transition to passwd_t from user_t
  - *allow passwd_t passwd_exec_t:file entrypoint (run w/ passwd perms)*
  - *type_transition user_t passwd_exec_t:process passwd_t*
- Passwd can the perform the operation
  - *allow passwd_t shadow_t:file {read write}* (can edit passwd file)

SELinux Deployment

- You've configured your SELinux policy
  - *Now what is left?*
- Surprisingly, a lot
  - Many services must be aware of SELinux
  - Got to get the policy installed in the kernel
  - Got to manage all this policy
- And then there is the question of getting the policy to do what you want

User-space Services

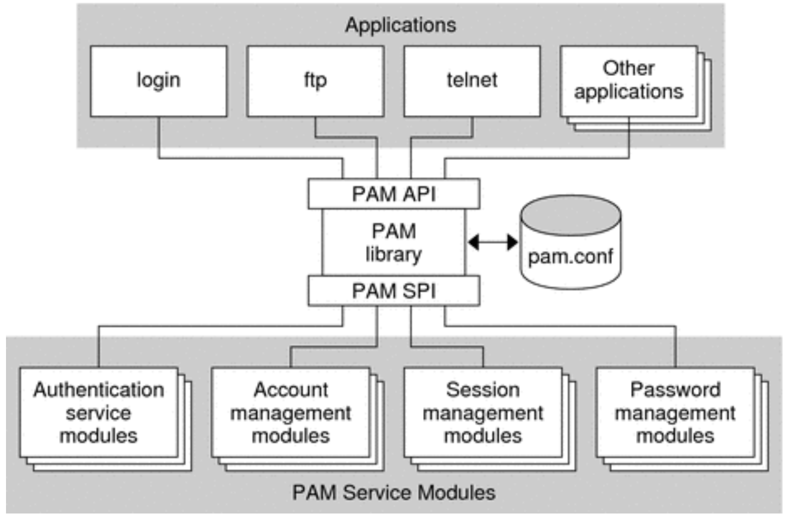
- What kind of security decisions are made by user-space services?
  - Authentication (e.g., sshd)
  - Access control (e.g., X windows, DBs (servers), browsers (middleware), etc.)
  - Configuration (e.g., policy build and installation)
- Also, many services need to be aware of SELinux to enable usability
  - E.g., Listing files/processes with SELinux contexts (ls/ps)

User-space Services

- Authentication
  - Various authentication services need to create a "SELinux subject context" on a user login
  - Like login in general, except we set an SELinux context and a UID for the generated shell
- How do you get all these ad hoc authentication services to interact with SELinux?

Authentication for SELinux

- Pluggable Authentication Modules
  - There is a module for SELinux that various authentication services use to create a subject context


Applications
login
ftp
telnet
Other applications
PAM API
PAM library
pam.conf
PAM SPI
Authentication service modules
Account management modules
Session management modules
Password management modules
PAM Service Modules

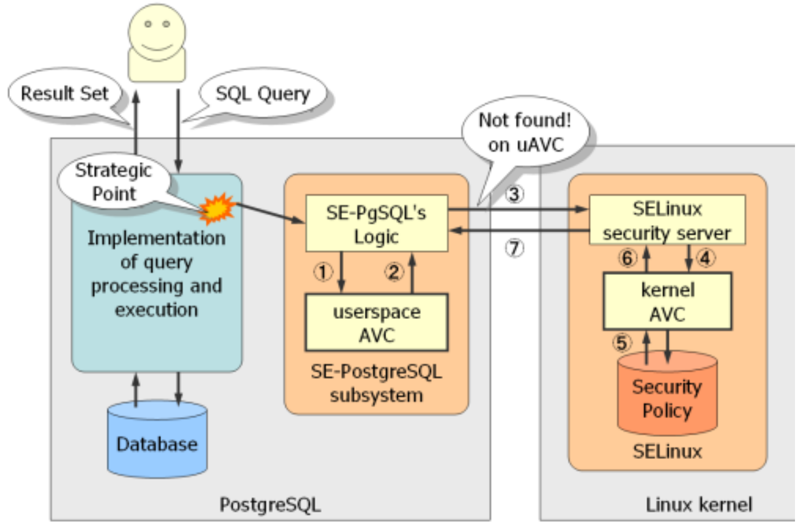
User-space Services

PENNSTATE 1855

- Access Control
  - Many user-space services are shared among mutually untrusting clients
    - Problem: service may leak one client's secret to another
- If your SELinux policy allows multiple, mutually untrusting clients to talk to the same service, what can SELinux do to prevent exploits?

User-space Services

- Add SELinux support to the service
  - X Windows, postgres, dbus, gconf, telephony server
- E.g., Postgres with the SELinux user-space library

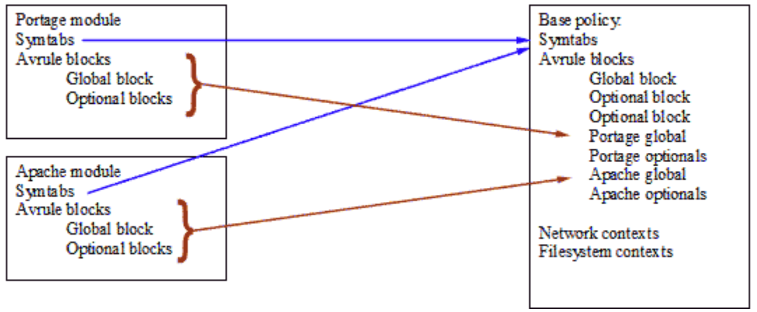


User-space Services

- Configuration
  - You need to get the SELinux policy constructed and loaded into the kernel
    - Without allowing attacker to control the system policy
    - And policy can change dynamically
- How to compose policies?
- How to install policies?

Compose Policies

- The SELinux policy is modular
  - Although not in a pure, object-oriented sense
    - Too much had been done
- Policy management system composes the policy from modules, linking a module to previous definitions and loads them



Take Away

- Problem: Turn the SELinux policy into a working, usable reference monitor
  - Work with user-space services
  - Design the policy that you want
- There are many requirements for user-space services to provide authentication, access control, and policy configuration itself
  - PAM, Policy Mgmt, User-space access, Network support
- Design of MPS can only be semi-automated
  - Prevent network threats and design for app integrity