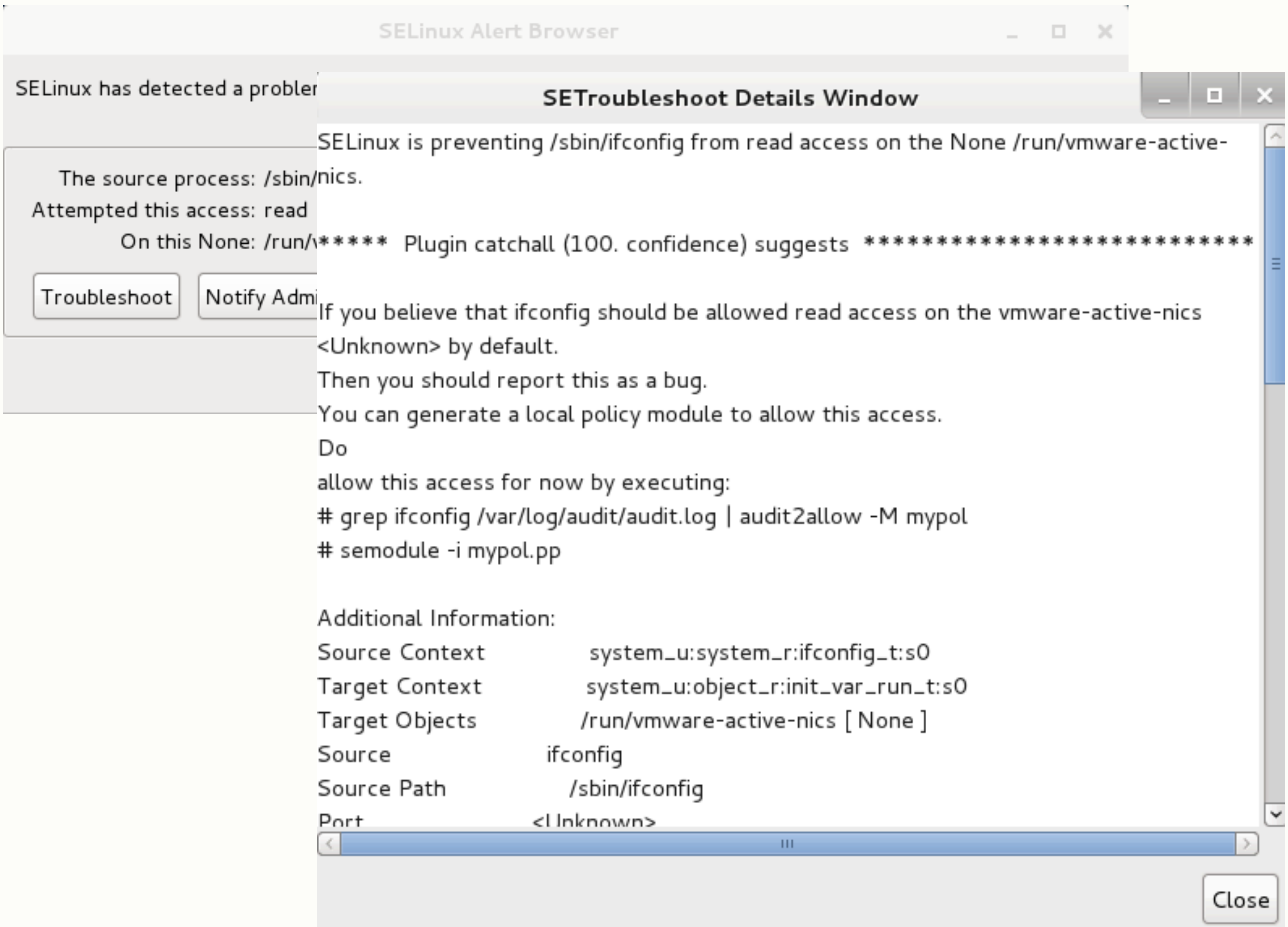
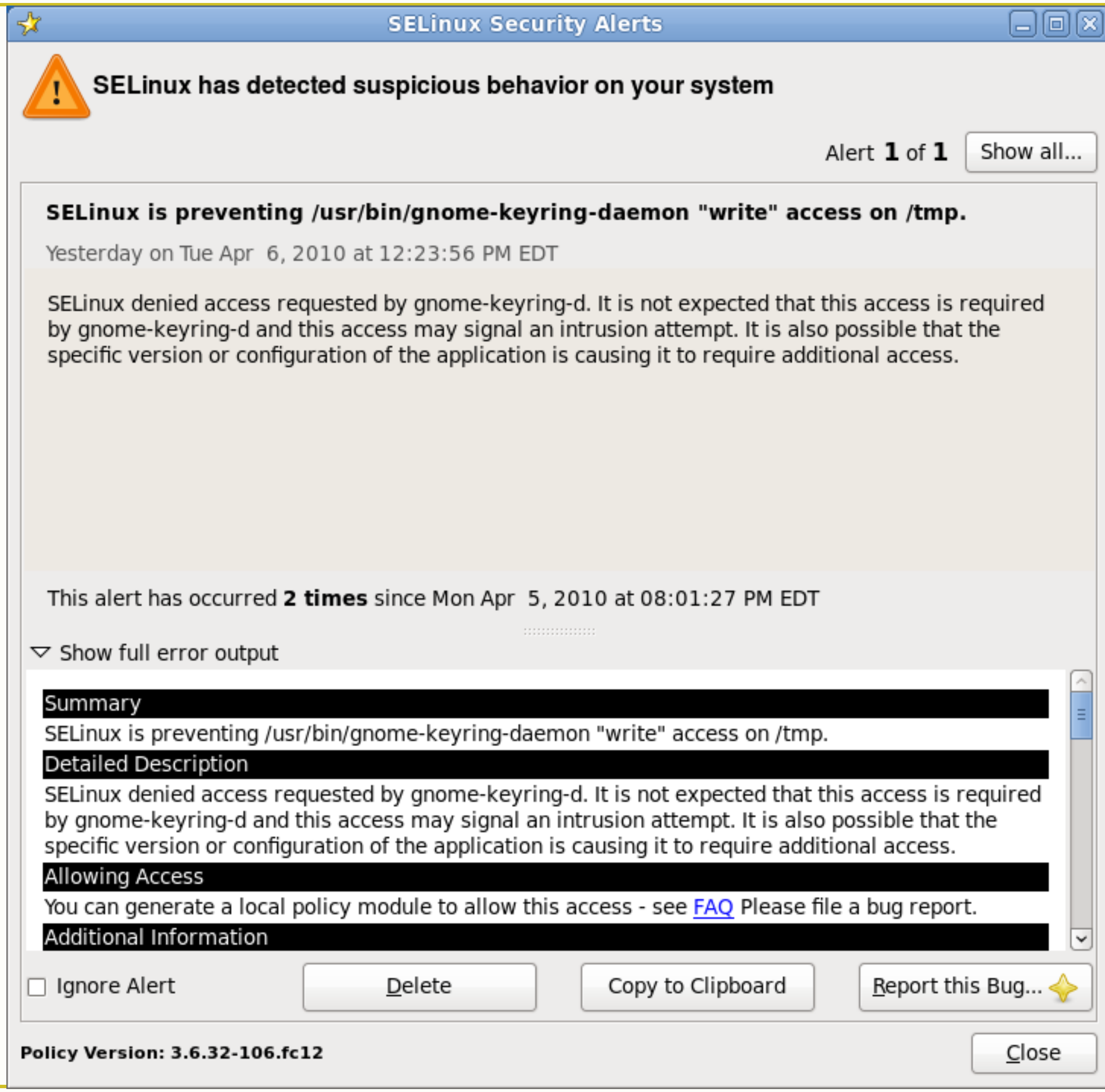
Hands-on SELinux: A Practical Introduction

Security Training Course

Dr. Charles J. Antonelli
The University of Michigan
2013


SELinux Alert Browser
SELinux has detected a probler
The source process: /sbin/
Attempted this access: read
On this None: /run/
Troubleshoot
Notify Admi
SETroubleshoot Details Window
SELinux is preventing /sbin/ifconfig from read access on the None /run/vmware-active-nics.
***** Plugin catchall (100. confidence) suggests ***************************
If you believe that ifconfig should be allowed read access on the vmware-active-nics <Unknown> by default.
Then you should report this as a bug.
You can generate a local policy module to allow this access.
Do
allow this access for now by executing:
grep ifconfig /var/log/audit/audit.log | audit2allow -M mypol
semodule -i mypol.pp
Additional Information:
Source Context system_u:system_r:ifconfig_t:s0
Target Context system_u:object_r:init_var_run_t:s0
Target Objects /run/vmware-active-nics [None]
Source ifconfig
Source Path /sbin/ifconfig
Port <Unknown>
Close


SELinux Security Alerts
SELinux has detected suspicious behavior on your system
Alert 1 of 1
Show all...
SELinux is preventing /usr/bin/gnome-keyring-daemon "write" access on /tmp.
Yesterday on Tue Apr 6, 2010 at 12:23:56 PM EDT
SELinux denied access requested by gnome-keyring-d. It is not expected that this access is required by gnome-keyring-d and this access may signal an intrusion attempt. It is also possible that the specific version or configuration of the application is causing it to require additional access.
This alert has occurred 2 times since Mon Apr 5, 2010 at 08:01:27 PM EDT
Show full error output
Summary
SELinux is preventing /usr/bin/gnome-keyring-daemon "write" access on /tmp.
Detailed Description
SELinux denied access requested by gnome-keyring-d. It is not expected that this access is required by gnome-keyring-d and this access may signal an intrusion attempt. It is also possible that the specific version or configuration of the application is causing it to require additional access.
Allowing Access
You can generate a local policy module to allow this access - see FAQ Please file a bug report.
Additional Information
Ignore Alert
Delete
Copy to Clipboard
Report this Bug...
Policy Version: 3.6.32-106.fc12
Close

Introduction

- Welcome to the course!
- Instructor:
  - Dr. Charles J. Antonelli
    The University of Michigan
    cja@umich.edu, 734 926 8421

Logistics

- Class
  - Wednesdays 6-9 PM (connect from 5:30 on
- Breaks
  - About once an hour (idea: get up, move around)
- Instruction
  - AT&T Connect remote experience
    - Please use the feedback icons
  - Lecture, Demonstration, Experiments
- Lab
  - Linux CentOS 6.3 lab environment via VMware Player
- Listserv
  - selsec2013@umich.edu

Prerequisites

- Nice to have
  - Familiarity with Linux architecture & tools
  - Familiarity with popular Linux applications
  - Working knowledge of network apps
  - Some system administration experience
  - Familiarity with white- and black-hat tools
  - Open source mindset

Take-Aways

- Understand SELinux architecture
- Install and configure SELinux
- Interpret SELinux log records
- Use SELinux permissive domains and Booleans to adjust SELinux policies
- Create and modify SELinux policies for your applications
- A healthy paranoia

Meet the instructor

- High-performance computing, security, and networking
- Systems research & development
  - Large-scale real-time parallel data acquisition & assimilation
  - Be Aware You're Uploading
  - Advanced packet vault
  - SeRIF secure remote invocation framework
- Teaching
  - HPC 101, 201 Basic & Advanced Cluster Computing
  - Linux Platform Security, Hands-on Network Security, Introduction to SELinux
  - ITS 101 Theory and Practice of Campus Computer Security
  - SI 630 Security in the Digital World, SI 572 Database Applications Programming
  - EECS 280 C++ Programming, 482 Operating Systems, 489 Computer Networks; ENGR 101 Programming and Algorithms

Meet the class – Poll

Level of Linux Experience:

1. *Novice*
2. *Experienced*
3. *Expert*

Poll

SELinux status on machines you administer:

1. *Enforcing, and I write my own policies*
2. *Enforcing, and I use permissive domains, Booleans, and/or audit2allow*
3. *Permissive*
4. *Disabled*
5. *Don’t know*
6. *What? You can change that?*

Roadmap

- Day 1:
  - Why SELinux?
  - Overview of SELinux
  - Using SELinux
  - SELinux Permissive Domains
- Day 2:
  - SELinux Booleans
  - SELinux audit2allow
  - SELinux Policy Theory
  - SELinux Policy Praxis

Why SELinux?

Why SELinux?

- Discretionary access control
  - ```
    $ ls -l /etc/passwd /etc/shadow
    -rw-r--r--. 1 root root 2174 2010-05-25 11:19 /etc/passwd
    -rw-r--r--. 1 root root 1459 2010-05-25 11:19 /etc/shadow
    ```
  - ```
    $ ls -la ~/bin
    total 52
    drwxrwxrwx.  2 cja cja  4096 2010-05-18 18:22 .
    drwx--x--x. 39 cja cja  4096 2010-05-25 20:41 ..
    -rwx--x--x.  1 cja cja  7343 2010-05-18 18:22 ccd
    -rwx--x--x.  1 cja cja  7423 2010-05-18 18:22 ctime
    -rwx--x--x.  1 cja cja 11656 2010-05-18 18:22 ctp
    -rwx--x--x.  1 cja cja  7423 2010-05-18 18:22 tbd
    -rwx--x--x.  1 cja cja  7109 2010-05-18 18:22 titleb
    ```

Why SELinux?

- Buffer overflows

Jan 02 16:19:45 host.example.com rpc.statd[351]: gethostbyname error for ^X÷ÿ¿^X÷ÿ¿^Y÷ÿ¿^Y÷ÿ¿^Z÷ÿ¿^Z÷ÿ¿^[÷ÿ¿^[÷ÿ¿bffff750 8 0 4 9 7 1 0 9 0 9 0 9 0 9 0 6 8 7 4 6 5 6 7 6 2 7 4 7 3 6 f 6 d 6 1 6 e 7 9 7 2 6 5 2 0 6 5 2 0 7 2 6 f 7 2 2 0 7 2 6 f 6 6 b f f f f 7 1 8 bffff719 bffff71a b f f f f 7 1 b _
_ _
_ _
_ _
_ _
_ ! _ _ ! _
_ _
_ _
_ _ _ _ _ _ _

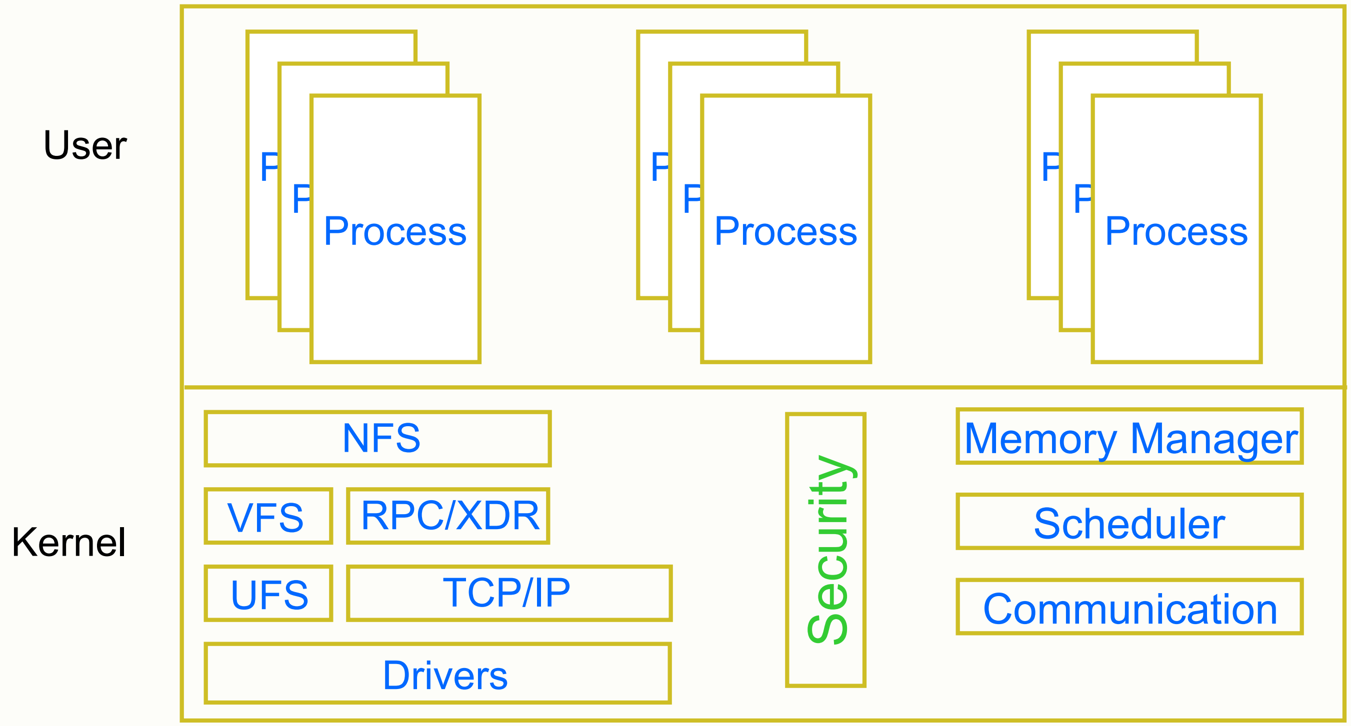
Why SELinux?

Propagation Mechanisms

Rank	Propagation Mechanisms	2011	Change	2010
1	**Executable file sharing** The malicious code creates copies of itself or infects executable files. The files are distributed to other users, often by copying them to removable drives such as USB thumb drives and setting up an autorun routine.	76%	+2% ↑	74%
2	**File transfer, CIFS** CIFS is a file sharing protocol that allows files and other resources on a computer to be shared with other computers across the Internet. One or more directories on a computer can be shared to allow other computers to access the files within. Malicious code creates copies of itself on shared directories to affect other users who have access to the share.	43%	-4% ↓	47%
3	**Remotely exploitable vulnerability** The malicious code exploits a vulnerability that allows it to copy itself to or infect another computer, such as a Web-based attack or a drive-by download.	28%	+4% ↑	24%
4	**File transfer, email attachment**. The malicious code sends spam email that contains a copy of the malicious code. Should a recipient of the spam open the attachment, the malicious code will run and the recipient's computer may be compromised.	14%	-4% ↓	18%
5	**File sharing, P2P.** The malicious code copies itself to folders on an infected computer that are associated with P2P file-sharing applications. When the application runs, the malicious file will be shared with other users on the same P2P network.	7%	-1% ↓	8%

Figure B11: Propagation Mechanisms
Source: Symantec Internet Security Threat Report, Vol. 17, April 2012

Linux Architecture



Linux Architecture

- Creating a process
  - Two intertwined system calls
  - A *parent* process calls *fork()*
    - Creates a *child* process
      - » An exact copy of the parent
      - » Including uid, open files, devices, network connections
  - The child process calls *exec(executable)*
    - Overlays itself with the named executable
      - » Retains uid, open files, devices, network connections

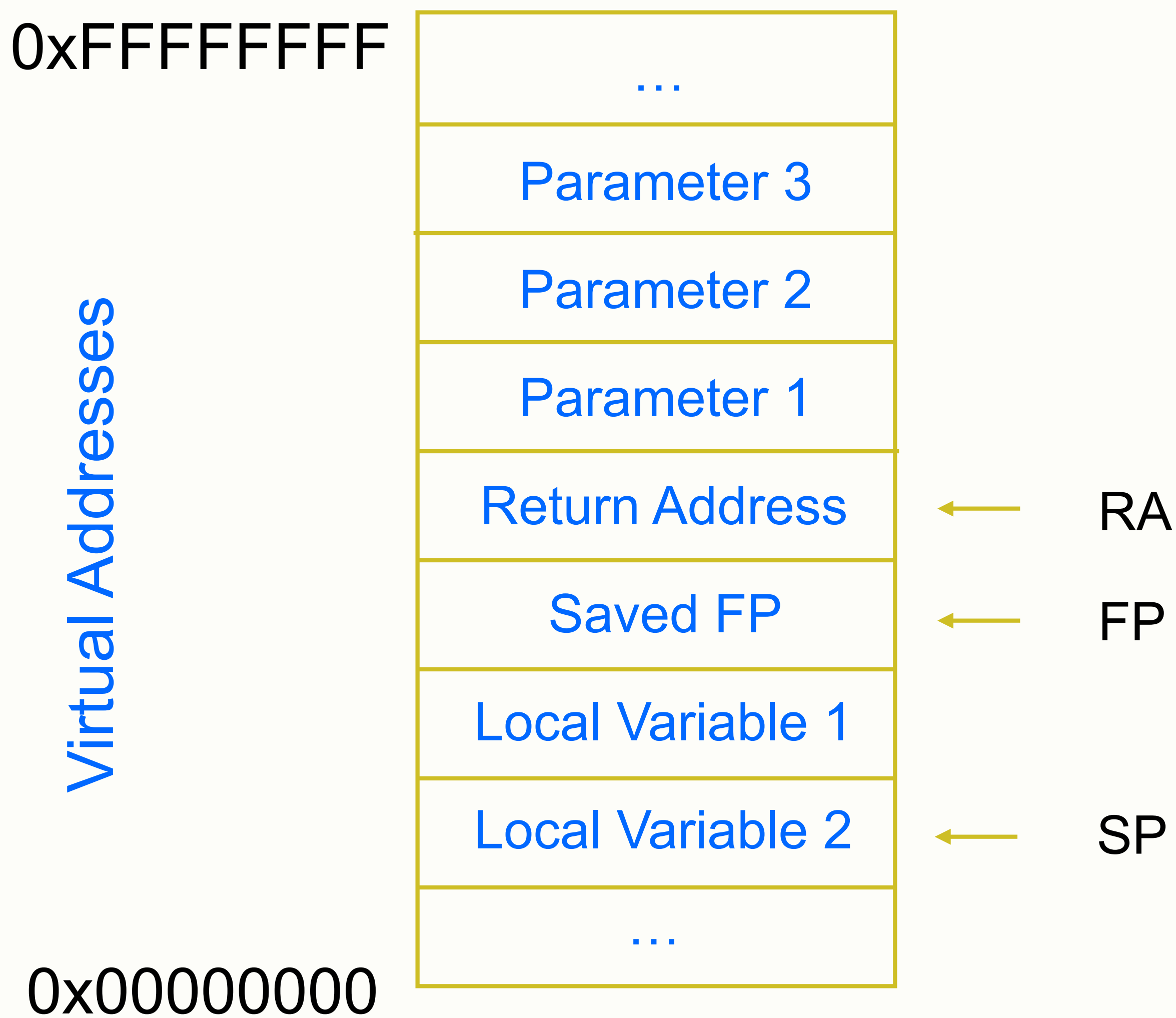
Linux Architecture

- Creating trouble
  - exec() may be called without fork()
  - Useful paradigm
    - `tcpd` execs the wrapped application after validation
  - So what happens if a process calls exec("/bin/sh") ?
    - Process becomes a command shell
    - Running with the overlaid process's credentials
      - If the process was running as root, so is the shell
    - Connected the same network connections
      - If the process was connected to your keyboard, so is the shell
      - If the process was connected to a client, so is the shell

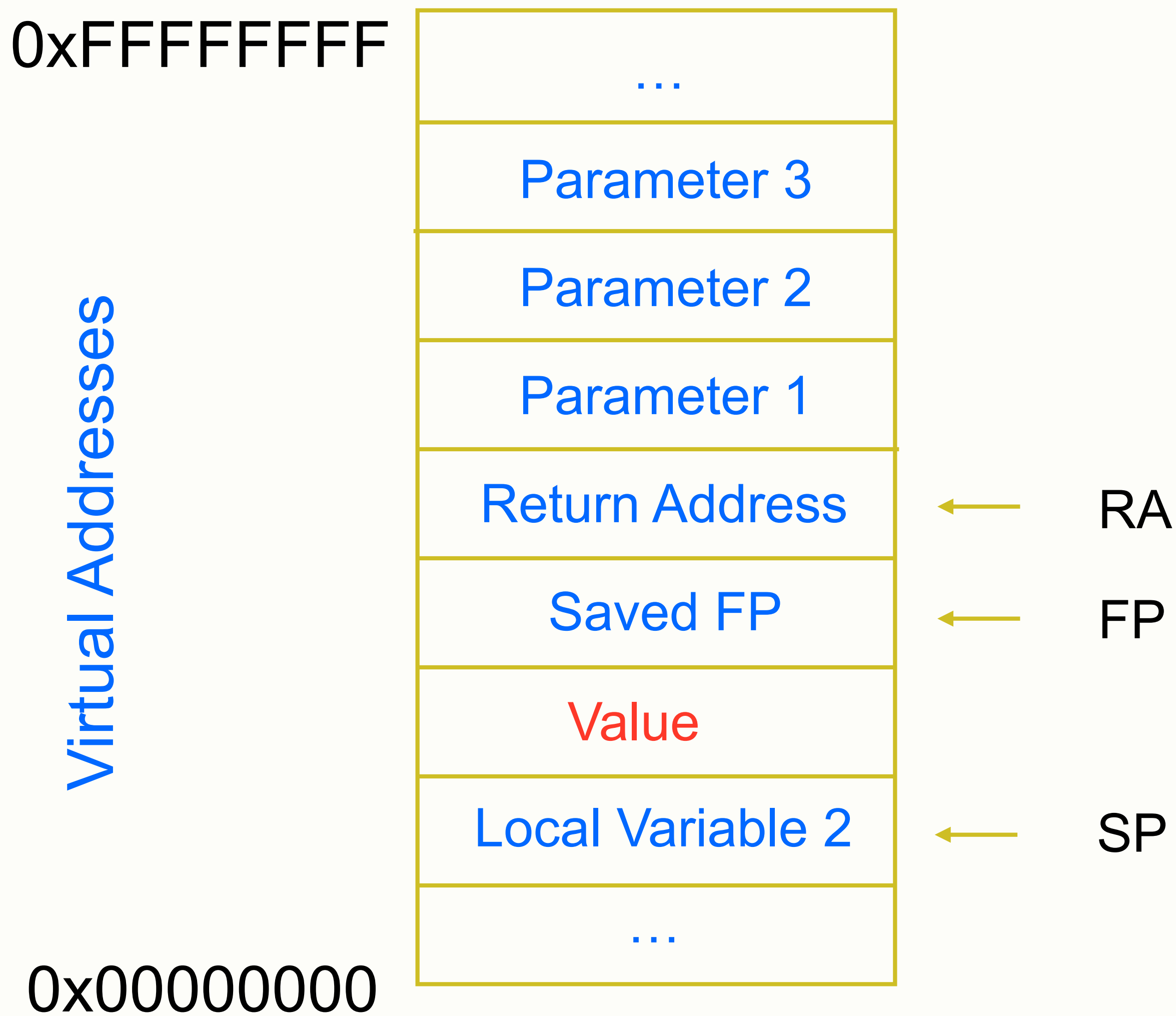
Smashing the stack Part I

- A calling function will write its return address into a memory data structure called the *stack*
- When the called function is finished, the processor will jump to whatever address is stored in the stack
- Suppose "Local Variable 1" is an array of integers of some fixed size
- Suppose our called function doesn't check boundary conditions properly and writes values past the end of the array
  - The first value beyond the end of the array overwrites the stack
  - The second value overwrites the return address on the stack
- When the called function returns, the processor jumps to the overwritten address

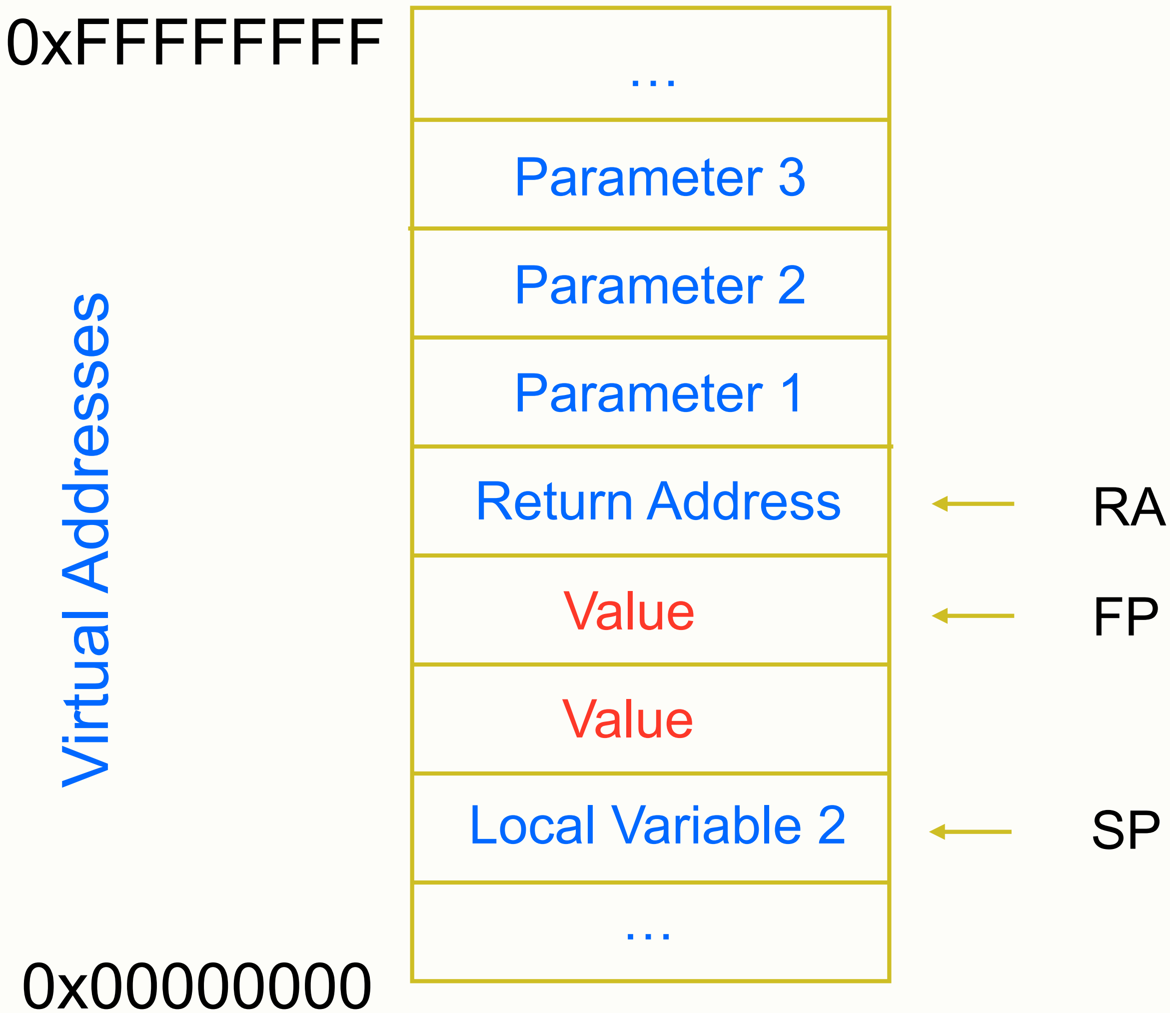
Smashing the stack



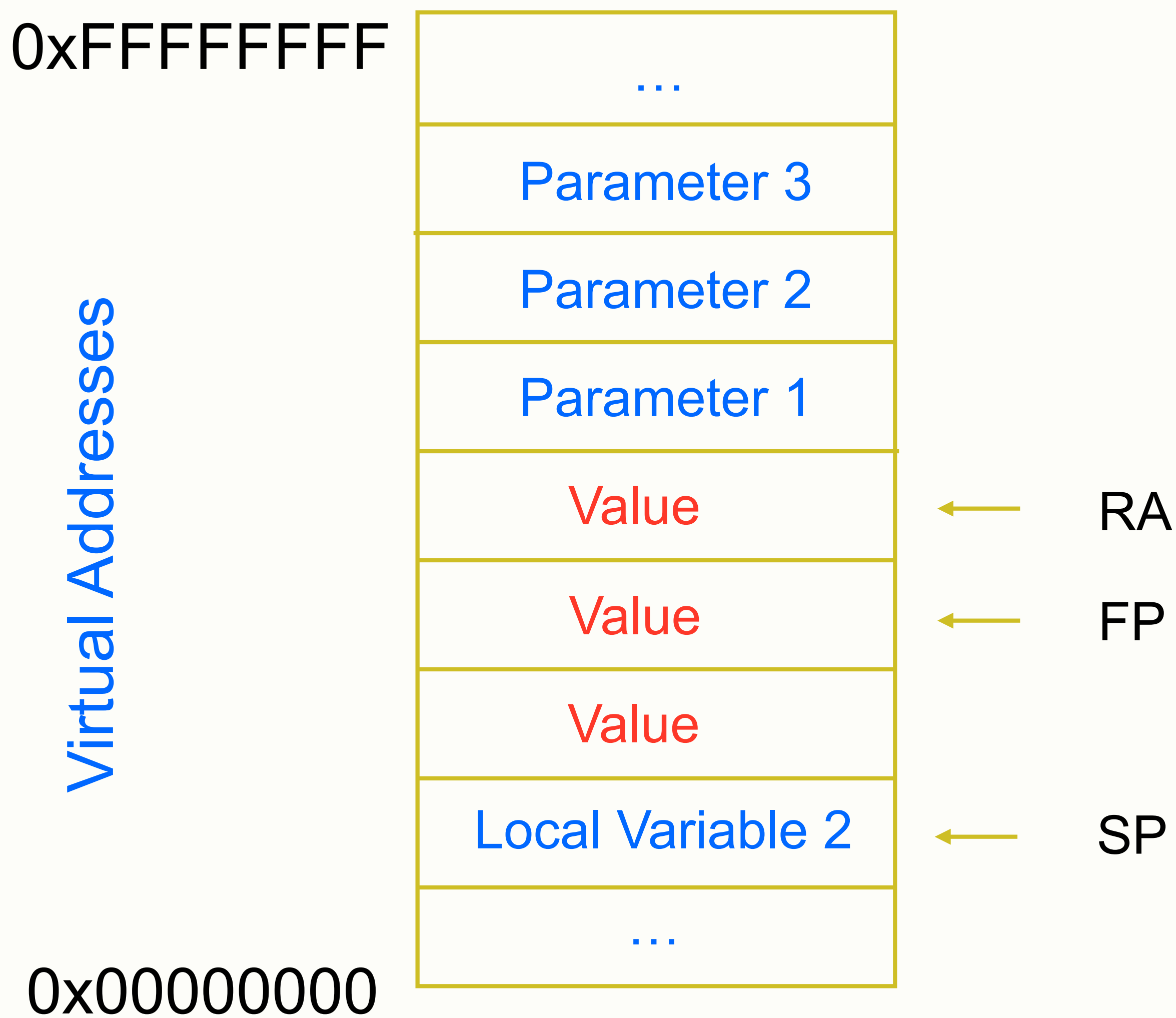
Smashing the stack



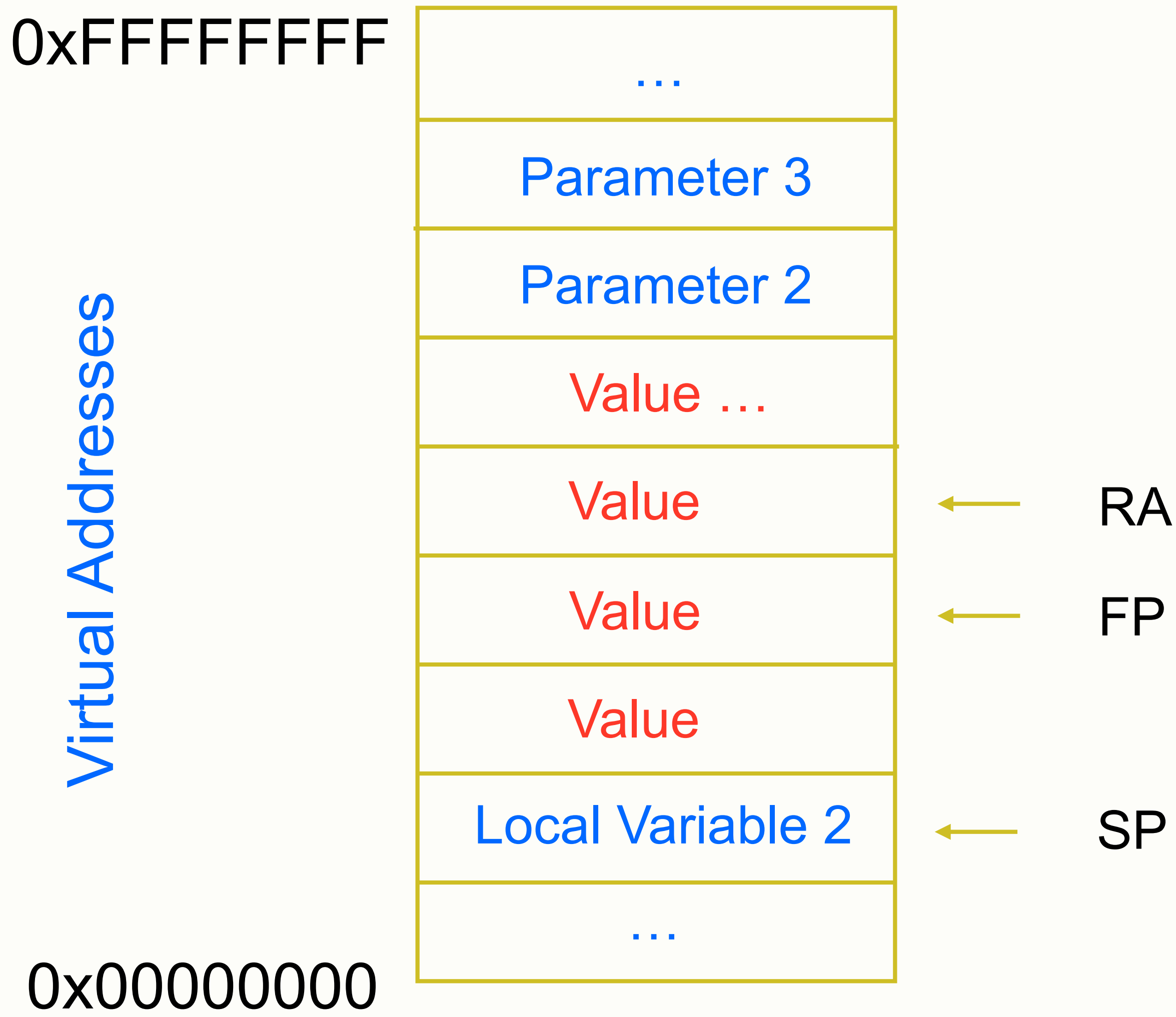
Smashing the stack



Smashing the stack



Smashing the stack



Smashing the stack Part II

- Suppose the attacker has placed malicious code somewhere in memory and overwrites *that* address on the stack
  - Now the attacker has forced your process to execute her code
- Where to place the code?
  - Simplest to put it in the buffer that is being overflowed
- How to get the code into the buffer?
  - Examine the source code
    - Look for copy functions that don't check bounds
      - gets, strcpy, strcat, sprintf, …
    - Look for arguments to those functions that are under the attacker's control and not validated by the victim code
      - Environment variables, format strings, URLs, …

Lab – stopping buffer overflows

1. Copy selsmash.tgz from Supplemental Information on course web page
   - `wget http://www-personal.umich.edu/~cja/SEL13/supp/selsmash.tgz`
   - `tar zxf selsmash.tgz`
   - `cd ~/selsmash`
   - `make`
     - `… enter your password when prompted`
2. Run the executable
   - What happened?
   - Examine the SELinux audit
3. Change SELinux to permissive mode
   - System | Administration | SELinux management
     - … enter root password when prompted
     - … may take a while to come up
   - Set current enforcing mode to permissive
4. Rerun the executable
   - What happened this time?

Lab – supplemental

- We'll be using gdb
  - "gdb file" to debug; "info gdb" for manual:
    - type cursor motion keys to move cursor
    - type page motion keys or "f" to page forward or "b" to page back
    - type "p" to return to previous page
    - position cursor on topic (line with ::) and type enter to move to new topic
    - type "u" to return to previous topic
    - type "/", *string*, and return to search for *string* in current topic
    - type "q" to quit
- We'll examine buffer overflows in detail
  - Follow along with instructor
- Code taken from Shellcoder's Handbook
  - Actually, Aleph One's 1996 "Smashing the Stack for Fun and Profit" paper

Lab – supplemental

gdb exec	start gdb on executable exec
gdb exec core	start gdb on executable exec with core file core
l [m,n]	list source
disas	disassemble function enclosing current instruction
disas func	disassemble function func
b func	set breakpoint at entry to func
b line#	set breakpoint at source line#
b *0xaddr	set breakpoint at address addr
i b	show breakpoints
d bp#	delete beakpoint bp#
r	run program
bt	show stack backtrace
c	continue execution from breakpoint
step	single-step one source line
next	single-step, don't step into function
stepi	single-step one instruction
p var	display contents of variable var
p *var	display value pointed to by var
p &var	display address of var
p arr[idx]	display element idx of array arr
x 0xaddr	display hex word at addr
x *0xaddr	display hex word pointed to by addr
x/20x 0xaddr	display 20 words in hex starting at addr
i r	display registers
i r ebp	display register ebp
q	quit gdb

Lab – stopping buffer overflows

5. Change SELinux back to enforcing mode
   - System | Administration | SELinux management
     - … enter root password when prompted
     - … may take a while to come up
   - Set current enforcing mode to enforcing

Overview of SELinux

Mandatory Access Control (MAC)

- System-enforced access control
  - But Unix and Linux systems provide only Discretionary Access Control (DAC)
    - Users determine access control settings of their objects
    - Improper access control settings expose data
    - Superusers can access everything
      - » Access checks disabled

Compartmentalization

- "Need to know"
  - But Linux processes have coarse-grained access to system objects
    - e.g. /tmp, /proc, ps
    - A subverted process, say via buffer overflow, can access too much system state

SELinux

- Flask Architecture
  - NSA & SCC 1999
  - A Type-Enforcement model
  - Flexible MAC
- Linux Implementation
  - Loscocco & Smalley 2001
  - Type Enforcement (TE)
  - Role-Based Access Control (RBAC)
  - Multi-Level Access Control (MLS)
    - Bell – La Padula

SELinux

- Red Hat Implementation
  - *Targeted* Policy
  - *Confined* system services
  - Unconfined users

SELinux

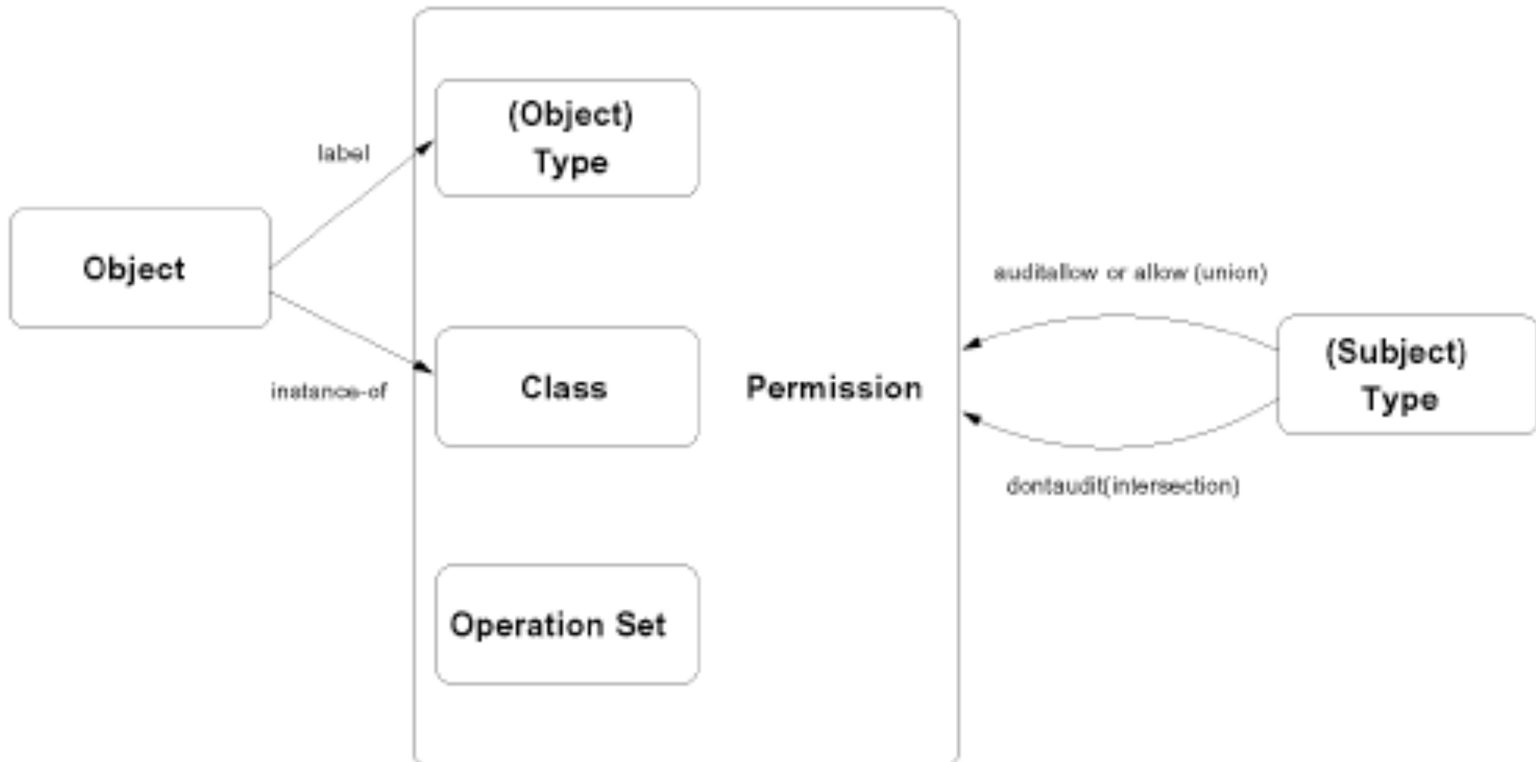


Figure 1: SELinux extended Type Enforcement (TE) policy model basics.

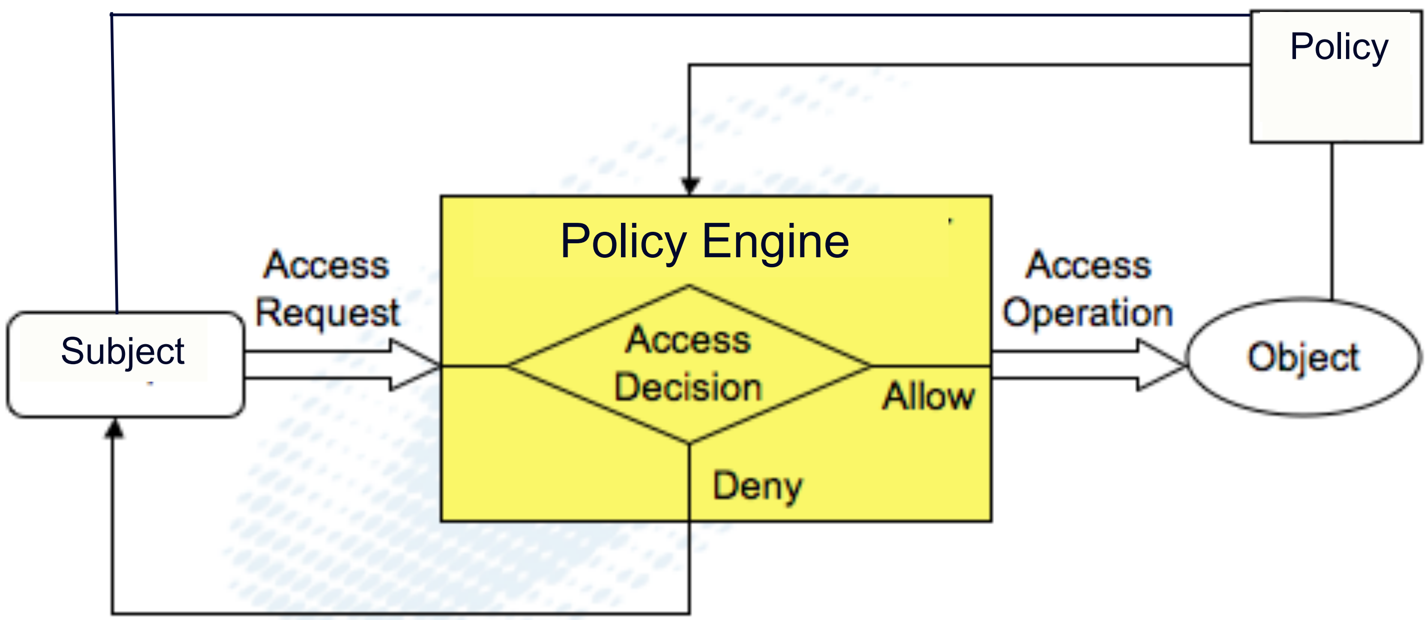
SELinux

- MAC applied after DAC succeeds
  - If DAC fails, access is denied
  - Access granted only if both succeed
- SELinux is thus a security layer
  - Not antivirus software
  - Not a replacement for firewalls, passwords, encryption, …
  - Not a complete security solution

SELinux Components

- Subject
  - AKA SELinux User
    - SELinux User ≠ Linux User
  - Users are *unconfined*
  - System services are *confined*
- Object
  - Have security *label* attached
    - AKA *context*

SELinux Type Enforcement



SELinux Components

- Security context
  - <identity, role, type | domain, securitylevel>
    - *username*_t
    - *role*_r (object_r for files)
    - *type*_t
    - s0 (used only for MLS)
  - Coin of the realm
    - Everything in SELinux has a context

SELinux Components

- An **Identity** identifies the user
- A **Role** determines in which domains a process runs
- A **Type** is assigned to an object and determines access to the object
- A **Domain** is assigned to a subject and determines what that subject may do
  - A domain is a capability
  - “Domain” and “Type” are synonymous

SELinux Components

```
$ ls -ldZ .
drwx------  cja cja system_u:object_r:user_home_dir_t:s0 .
$ ls -lZ .bashrc
-rw-r--r--  cja cja system_u:object_r:user_home_t:s0 .bashrc
$ ps -Z
LABEL                                  PID TTY          TIME CMD
unconfined_u:unconfined_r:unconfined_t:s0 3581 pts/0 00:00:00 bash
unconfined_u:unconfined_r:unconfined_t:s0 3732 pts/0 00:00:00 ps
$ ps axZ | grep sendmail:\ accepting
system_u:system_r:sendmail_t:s0  2756 ?        Ss     0:00 sendmail:
   accepting connections
$ ps axZ|wc -l
203
$ ps axZ|grep unconfined|wc -l
55
$ ps axZ|grep -v unconfined|wc -l
149
```

SELinux Security Implications

- `su` changes UID but not identity
  - Root processes cannot access everything!
  - User identity determines what roles and domains can be used
- File access controlled by context, not UID
  - Compromised processes cannot rummage around in the file system!

SELinux Security Implications

- Every object has a security context
  - For files, contexts are called labels
  - Labels are stored in extended attributes
  - Relabeling a file system takes about as long as a full filesystem check (`fsck`)
    - You don't want to do this!
  - Relabeling a small set of files is okay
    - Often the best way to restore operation

SELinux Modes

- Three (global) *modes*:
  - Enforcing – creates labels, checks and logs, and enforces access decisions
  - Non-enforcing – creates labels, checks and logs, but does not enforce access decisions
  - Disabled – doesn't do anything
    - Including writing labels on new files
    - Which means you will have to relabel the file systems later

SELinux Logging

- /var/log/audit/audit.log   if `auditd` is running
- /var/log/messages   otherwise
- Failure audits include
  - Failing operation (read, etc.)
  - Process ID of executable
  - Name of executable
  - Mount point and path to object accessed
  - Linux inode of object accessed

SELinux Tools

- GUI
  - Configure SELinux
    sudo /usr/bin/system-config-selinux
    System | Administration | SELinux Management
  - Interpret SELinux log errors
    /usr/bin/sealert
    Applications | System Tools | SELinux Troubleshooter

- Command line
  - semanage, setsebool, setenforce, getenforce, audit2allow, …
  - As always, **`man`** is your friend

Using SELinux

Status quo

- SELinux running in enforcing mode
- Users are unconfined
- Services are confined
- Policies defined for all distributed services
  - Fairly well-tuned by now
    - Policy errors less frequent
- Less fun installing new applications

SELinux status

1. Look at your SELinux status
   `sestatus`

SELinux users vs. services

1. Look at your security context
   `id`
2. Examine a service's security context
   `ps axZ | grep rsyslogd`

SELinux Permissive Domains

Permissive domains

- The enforcing mode switch is very coarse
  - Everything is permissive, or nothing is
- SELinux allows you to set a *single domain* to be permissive
  - Investigate a problem with a single process
  - Define policies for new applications
  - Keeps rest of system protected
  - Greatly reduces need for permissive mode

Permissive domains

- Command-line tool: semanage
- man semanage

- Example

```
sudo semanage permissive -l | less
sudo semanage permissive -a httpd_t
sudo semodule -l | grep permissive
sudo semanage permissive -l | less
sudo semanage permissive -d httpd_t
```

End Day 1

References

- P. A. Loscocco, S. D. Smalley, P. A. Muckelbauer, R. C. Taylor, S. J. Turner, and J. F. Farrell, "The inevitability of failure: the flawed assumption of security in modern computing environments," *Proceedings of the 21st National Information Systems Security Conference, pp 303–314, Oct. 1998.* *http://csrc.nist.gov/nissc/1998/proceedings/paperF1.pdf*
- Ray Spencer, Stephen Smalley, Peter Loscocco, Mike Hibler, Dave Andersen, and Jay Lepreau, "The Flask Security Architecture: System Support for Diverse Security Policies," Proceedings of the 8th USENIX Security Symposium, Washington D.C., August 1999.
- Loscocco, P. and S. Smalley, "Integrating Flexible Support for Security Policies into the Linux Operating System," Proceedings of the FREENIX Track, Usenix Technical Conference, June 2001.
- Trent Jaeger, Reiner Sailer, and Xiaolan Zhang, "Analyzing Integrity Protection in the SELinux Example Policy," Proc. 12th Usenix Security Symposium, Washington DC, August 2003.
- Fedora Project Documentation Team, "Fedora 11 Security-Enhanced Linux User Guide," Linux Documentation Library, http://www.linbrary.com/.
- D. E. Bell and L. J. La Padula, "Secure computer systems: Mathematical foundations and model," Technical Report M74-244, MITRE Corporation, Bedford, MA, May 1973.
- http://wiki.centos.org/HowTos/SELinux
- http://fedoraproject.org/wiki/SELinux