



Henrik Rydmark
May 2005
Red Hat

Red Hat Security Activities

- § Enterprise Linux model
 - § backport fixes into existing code tree
 - § eliminate Open Source Software challenges
- § Red Hat Network
 - § efficient deployment of relevant errata
- § Red Hat Security Response Team
 - § single point of contact
 - § coordinate fix testing with partners
 - § track known problems and solutions in public database
- § Security certifications: EAL3+, COE
- § Develop security features
 - § configuration / hardening tools
 - § Exec Shield
 - § SELinux

Enterprise Linux as a Model

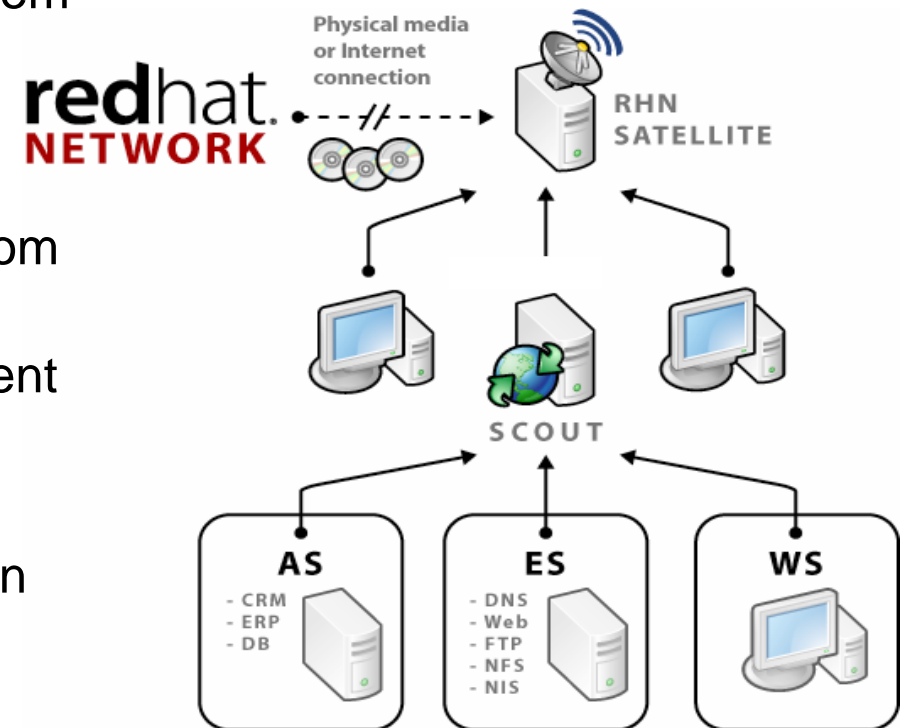
- § Red Hat Enterprise Linux addresses a number of problems:
 - § Architectural stability – makes OSS predictable
 - § Defined platform, certification, compatibility
 - § „Slowing down” the release cycle – only every 12 – 18 months
 - § Long term support and maintenance (7 years) for each version with backports of security fixes
- § Provides a predictable platform for ISVs, OSS projects and the user
- § Resolves the version and certification conflicts
- § Provides long term security

Red Hat Enterprise Linux model
eliminates hindering problem



Red Hat Network

- § Red Hat Network solves the patch problem by providing:
 - § Efficient centralized software deployment based on RPM
 - § Especially: Errata delivery directly from Red Hat
 - § Centralized configuration management
 - § Transparent and credible – clients GPLed
 - § Notification and relevance verification
 - § Makes Open Source deployable – removing the „Guru factor“
 - § Makes actions reproduceable
 - § Hosted and in-house solutions (Satellite) available



With RHN applying fixes is as easy as possible

Existing OS Security Features

- § Security features in RHEL 3 and RHEL 4
 - § Standard Linux features (multi user OS, IPTables, etc.)
 - § Signed Packages (RPM)
 - § POSIX ACLs
 - § Exec-Shield
 - § stack protection for all CPU types (using NX technology if available)
 - § Stack, heap and shared library position randomization
 - § Position Independent Executables (PIE)
 - § ELF Data Hardening
 - § COE and EAL certification

Is this enough?

§ Activities and the traditional security model solves a lot of the problems...

§ But what, if an attack is successful?

§ What about misconfiguration?

§ What about treacherous users?

→ Objective: Reduce the remaining risk by limiting the potential damage caused by the failure of these security mechanisms

A Mandatory Access Control system is needed

Current Linux Security Model

- § Referred to as Discretionary Access Control
 - § Traditional OS/Unix mechanism
- § Programs runs with the permission of the user executing it
 - § File permissions and ACL
 - § setuid/setgid binaries: mount, passwd, ping
 - § chroot – a restricted filesystem namespace
- § Root is all powerful
- § Too coarse grained
 - § root vs non-root gives boolean security model for many cases
- § Users & programs have complete control
- § The level of system security is left to the discretion of the applications running on it.

Virtually any compromise or misconfiguration of the above can easily lead to *total* system compromise

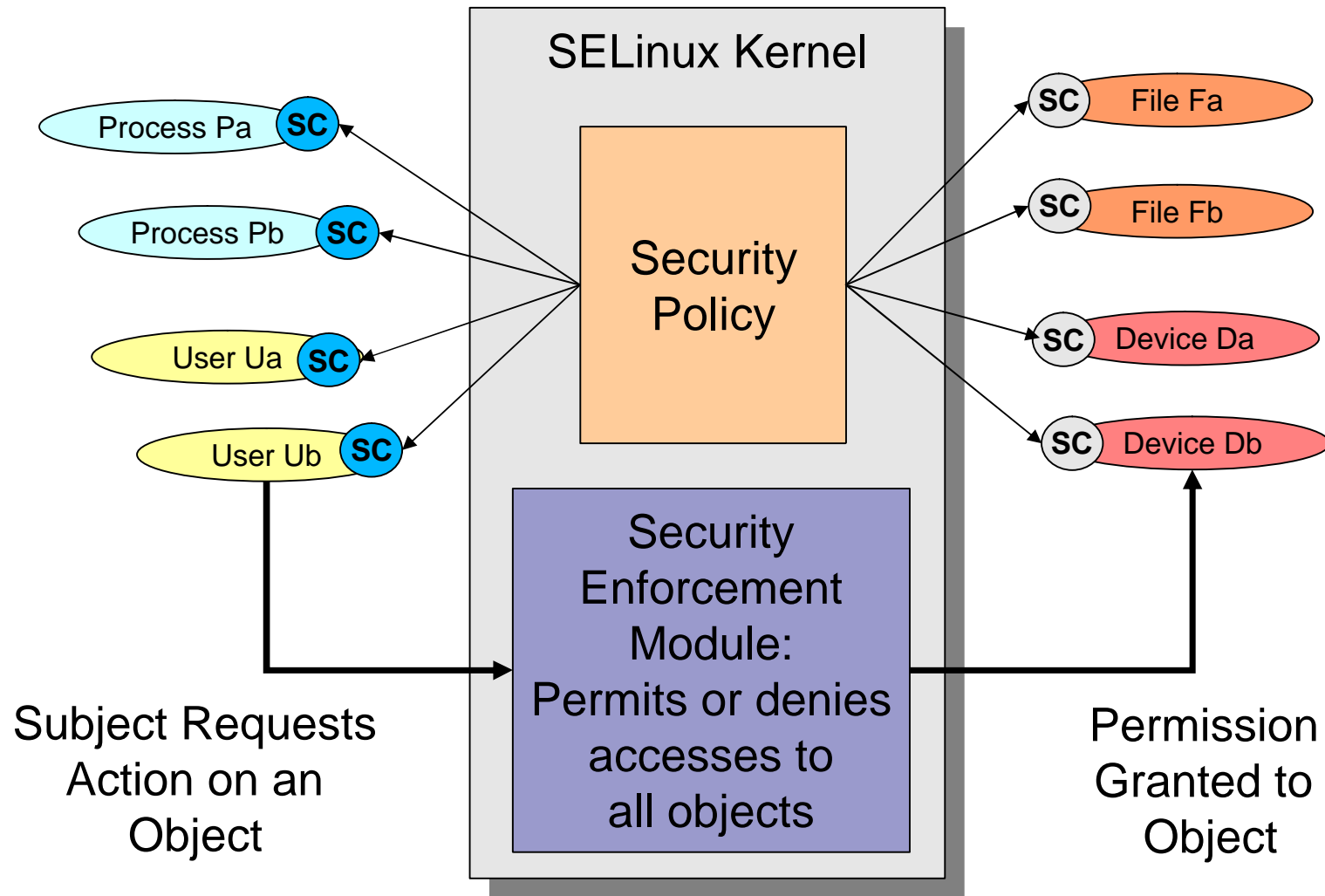
Mandatory Access Control

- § Defined by three major properties:
 - § Centralized defined security policy.
 - § Policy set by administrator and enforced by the system
 - § Policy cannot be overridden either accidentally or deliberately by malicious software and users
 - § Control over all subjects (processes) and objects.
 - § May be applied to object not protected by normal access controls, such as network sockets and processes.
 - § Prevents information leaks or privilege escalation even from accidental misconfiguration
 - § Decisions based on all security-relevant information.
- § Security administrator can ensure two domains never interact, even if they're both compromised
 - § For example, the web server and your raw database
- § Allows protection against untrusted code
- § Follows the principle of least privileges

What is SELinux?

- § A system for Mandatory Access Control (MAC) based on the Linux Security Modules (LSM) framework, integrated in the mainstream 2.6 kernel
- § Building on 10 years of NSA's OS security research
- § Application of NSA's Flask security architecture
 - § Cleanly separates policy from enforcement using well-defined policy interfaces
 - § Allows users to express policies naturally and supports changes
 - § Fine-grained controls over kernel services
 - § Transparent to applications and users
- § SELinux model is independent of normal Linux model, checked after normal Linux checks
- § Policies define what SELinux does
- § Implements MAC via a combination of Type Enforcement and Role-Based Access Control

SELinux – How it works



SELinux Type Persistence

- § Type assignments on regular files stored in xattr
 - § check with ls -Z
 - § set with chcon
 - § apply policy assignments with restorecon, setfiles or fixfiles
 - § after policy change automatic relabeling initiated
 - § \$MNTPOINT/.autorelabel restores policy assignments on every boot
- § Filesystems without xattr support
 - § vfat, nfs, smbfs, tmpfs etc.
 - § set context for all files with mount option -o context=....
 - § set context for filesystem types in genfs_contexts
 - § set context transition in policy of your program
- § non-file resources
 - § network ports, fs mount, kernel module load etc.
 - § type assignment defined in policy

SELinux Policy

- § Defines whole security model
 - § Type declarations of system resources
 - § Class declarations with their operations
 - § Role declarations
 - § Domain transitions
 - § Access vectors: operations on types as granted operations on classes
- § Written at a high level with M4 macros
- § Compiled into a binary form that is understood by the kernel
- § Loaded by /sbin/init at the start of the boot process before any other programs are executed
- § A modified policy can be loaded at any time by the administrator

Everything not explicitly allowed is denied

Type Enforcement (TE)

- § Seven types of declarations; attribute, type, typealias, boolean, transition, access vector and conditional
- § Associates each program with a domain and each nonprocess with a type.

```
type httpd_config_t, file_type, sysadmfile;
```

- § Permissions are encoded as access vectors, which specify the operations that a domain is authorized to perform on objects of a giving type, such as files.

```
allow passwd_t shadow_t:file { read write };
```

Result	Access permitted	Result logged
No matching policy rule	No	No
allow	Yes	No, unless auditallow
auditallow	No, unless allow	Yes
dontaudit	No	No

Role-Based Access Control (RBAC)

- § Defining rules for every user to use every program to access every object would result in an enormous set of rules.
- § Need for a second security model. Working alongside Type Enforcement
- § Under RBAC, administrators define roles and allow certain users access to those roles.
 - § Specifies roles that are authorized for each user
`user root roles { user_r sysadm_r system_r };`
 - § Defines allowed transition between roles
`allow system_r sysadm_r;`
 - § Specifies domains that can be entered by each role
`role system_r types httpd_t;`

Example – snort.fc

```
# SNORT
/usr/(s)?bin/snort --      system_u:object_r:snort_exec_t
/etc/snort(/.*)?         system_u:object_r:snort_etc_t
/var/log/snort(/.*)?     system_u:object_r:snort_log_t
```

Example – snort.te

```
#DESC Snort - Network sniffer
#
# Author: Shaun Savage <savages@pcez.com>
# Modified by Russell Coker <russell@coker.com.au>
# X-Debian-Packages: snort-common
#
```

```
daemon_domain(snort)
```

```
logdir_domain(snort)
allow snort_t snort_log_t:dir create;
can_network_server(snort_t)
type snort_etc_t, file_type, sysadmfile;
```

```
# Create temporary files.
tmp_domain(snort)
```

```
# use iptable netlink
allow snort_t self:netlink_route_socket { bind create
  getattr read write };
allow snort_t self:packet_socket create_socket_perms;
allow snort_t self:capability { setgid setuid net_admin
  net_raw dac_override };
```

```
r_dir_file(snort_t, snort_etc_t)
allow snort_t etc_t:file { getattr read };
allow snort_t etc_t:lnk_file read;
```

```
allow snort_t self:unix_dgram_socket create_socket_perms;
allow snort_t self:unix_stream_socket
  create_socket_perms;
```

```
# for start script
allow initrc_t snort_etc_t:file read;
```

```
dontaudit snort_t { etc_runtime_t proc_t }:file read;
```

```
allow snort_t snort_log_t:dir create;
```

```
can_network_server(snort_t)
```

```
type snort_etc_t, file_type, sysadmfile;
```

```
allow snort_t etc_t:file { getattr read };
```

Permissive versus Enforcing

- § SELinux can run in 3 modes:
- § Permissive mode
 - § Policy loaded
 - § Auditing enabled
 - § No denial, only logging
 - § Useful for policy writing and troubleshooting
- § Enforcing mode
 - § Policy loaded
 - § Auditing enabled
 - § Action denial or grant depending on policy
- § Disabled
 - § Policy not loaded
 - § No auditing

Only enforcing mode provides more security

SELinux Implementation in RHEL

- § Integrated into standard RHEL versions – full ISV support
- § Not only the code but actual policies:
 - § Targeted policy in core product secures key network-facing services with minimal system impact
 - § Support for strict and custom policies through Red Hat Global Professional Services
- § OS integration
 - § Integrated into system boot
 - § Policy information gathering, analyse, compiling tools
 - § Changed programs for role support: su, login, pam etc.
 - § Changed programs for type & domain support: ls, find, id (-Z)
 - § Simplified policy management
 - § system-config-securitylevel
 - § set booleans instead of policy writing
- § Documentation & Support

Strict Policy vs Targeted Policy

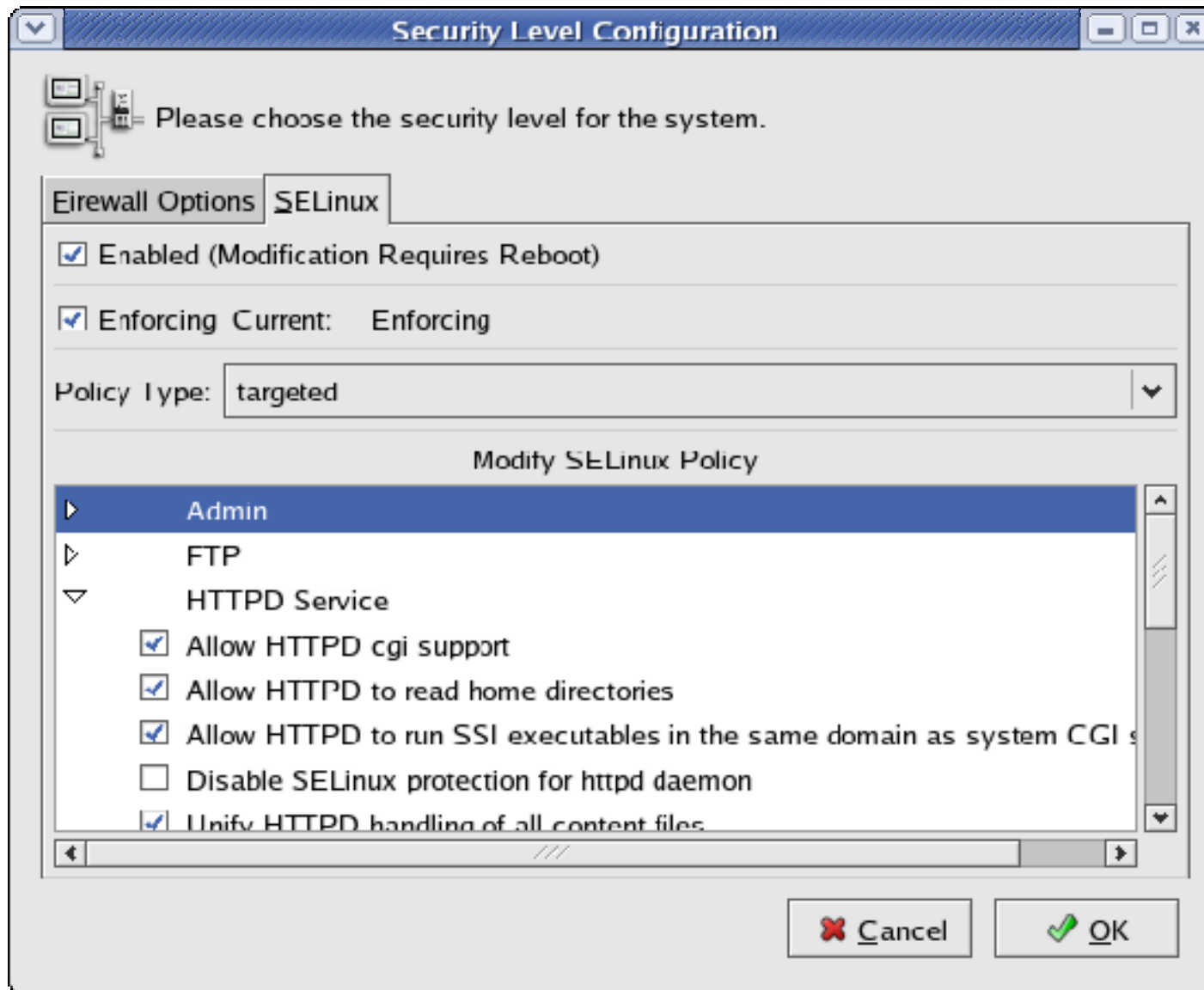
Strict policy

- § All processes, including user logins, run in restricted domain; again, any interaction not explicitly permitted is denied
- § Restricting system heavily - not shipped with RHEL
- § Supported through Red Hat GPS
- § Source +200k lines. Binary ~8M

Targeted policy

- § Only certain key system daemons confined by SELinux denial rules; dhcpd, httpd, named, nscd, ntpd, portmap, snmpd, squid, syslogd, mtas, mailman, mysql, postgresql, winbind, ypbind
- § Everything else runs as unconfined_t, a domain which is granted essentially every SELinux permission
- § Goal is to restrict above daemons, not protect them against the rest of the system
- § Use of policy booleans allows dynamic reconfiguration
 - § Disable protection for squid, disable httpd's ability to read user home directories
- § No extensive use of roles
- § Source +20K lines. Binary ~300K

System-config-securitylevel



Modified Linux commands

```
root@uljana:/etc/selinux/targeted
File Edit View Terminal Tabs Help
[root@uljana targeted]#
[root@uljana targeted]#
[root@uljana targeted]# id -Z
root:system_r:unconfined_t
[root@uljana targeted]#
[root@uljana targeted]#
[root@uljana targeted]#
[root@uljana targeted]# ps -Z
LABEL                                PID TTY          TIME CMD
user_u:system_r:unconfined_t         3758 pts/1        00:00:00 su
root:system_r:unconfined_t           3761 pts/1        00:00:00 bash
root:system_r:unconfined_t           3841 pts/1        00:00:00 newrole
root:system_r:unconfined_t           3842 pts/1        00:00:00 bash
root:system_r:unconfined_t           4009 pts/1        00:00:00 ps
[root@uljana targeted]#
[root@uljana targeted]#
[root@uljana targeted]#
[root@uljana targeted]# ls -Z
-rwx----- root      root      system_u:object_r:selinux_config_t booleans
drwxr-xr-x  root      root      system_u:object_r:default_context_t contexts
drwxr-xr-x  root      root      system_u:object_r:policy_config_t  policy
drwx----- root      root      system_u:object_r:policy_src_t     src
[root@uljana targeted]#
```

SELinux Related Packages

- § targeted policy
 - § selinux-policy-targeted contains binary policy
 - § selinux-policy-targeted-source contains sources
- § checkpolicy
 - § required tools to load & enable policy
- § libselinux
 - § basic tools: getsebool, setsebool, getenforce, setenforce, avcstat
- § policycoreutils
 - § basic tools: newrole, fixfiles, restorecon, sestatus
- § setools
 - § advanced tools for selinux status and policy: seinfo, sesearch, seuseradd
- § setools-gui
 - § advanced analyse tools: apol, seaudit, seaudit-report
- § selinux-doc: NSA documentation

SELinux in the filesystem

- § main configuration: /etc/selinux/config
 - § enabled | disabled (kernel param selinux=0|1 overrides)
 - § permissive | enforcing mode (kernel param enforcing overrides)
 - § policy selection – default is targeted
- § binary policy
 - § /etc/selinux/<NAME>/policy/policy.<VERSION> (current version is 18)
- § policy sources
 - § POLICY_SRC=/etc/selinux/<NAME>/src/policy/
 - § Makefile = used for policy (re)compilation
 - § file_contexts = type assignment rules
 - § domains = program policy types & access vectors
 - § users = user & role declarations
 - § flask/security_classes and flask/access_vectors contain valid classes & ops
 - § types contains general type & attribute declarations and rules
 - § macros contains m4 macros used to simplify policy writing

AVC Messages

§ When an access is not permitted by SE Linux an audit message is logged, here is a sample:

```
audit(1089889979.989:0): avc denied { write } for
pid=10317 exe=/usr/bin/vim name=etc dev=hda1
ino=162881 scontext=root:user_r:user_t
tcontext=system_u:object_r:etc_t tclass=dir
```

§ This tells us that at time 1089889979.989 seconds since 1970-01-01 the process vim which had pid 10317 tried to write to an object named etc (which had the Inode number 162881) of class dir on the file system hda1. vim had the context root:user_r:user_t which means that someone logged in as root with an unprivileged user role. The context of the directory was system_u:object_r:etc_t.

§ The program audit2allow can be used to convert AVC messages to allow rules, the above message would be converted to:

```
allow user_t etc_t:dir write;
```

§ But don't do this. Most times the output of audit2allow is not suitable for adding to your policy, it's just an indication of what's being attempted.

Why SELinux?

- § Traditional Linux system security depends on:
 - § Correctness of the kernel
 - § All the privileged applications
 - § Each privileged applications configurations
 - § Problems in any of these areas may cause the entire system to be compromised

- § SELinux system security depends on:
 - § Correctness of the kernel
 - § Security policy configuration.
 - § Problems in any application could allow limited compromise of individual user programs/system daemons, but not a threat to the security of other programs/system daemons or to the security of entire system.

Key Points

- § SELinux provides a secondary, independent layer of mandatory security
- § Choice of targeted versus strict policy, as well as policy booleans allows great flexibility
- § Integrated and on by default in Fedora Core 3 and RHEL 4
- § Extensive documentation and active mailing lists
- § SELinux provides – in difference to other commercial approaches – a highly flexible security framework
- § In general applications work WITHOUT modifications
- § Red Hat is the first to provide an enterprise level supported SELinux solution.
- § Security in RHEL becomes mainstream

Learning More and Getting Involved

- § SELinux Guide for RHEL4: <http://www.redhat.com/docs>
- § FAQ: <http://www.nsa.gov/selinux/FAQ>
- § Tutorial: <http://www.coker.com.au/>
- § Fedora: <http://fedora.redhat.com>
- § Fedora and selinux: fedora-selinux-list@redhat.com
- § FAQ: <http://fedora.redhat.com/docs/selinux-faq-fc3test2/>
- § Upstream SELinux: <http://www.nsa.gov/selinux/>
- § SELinux mailing list: selinux@tycho.nsa.gov
- § M4 tutorial: <http://www.cs.utah.edu/dept/old/texinfo/m4/m4.html>

Questions?



