SELinux Reference Policy

TRESYS TECHNOLOGY

Motivations

- Policies are too difficult to develop / maintain
  - no coherent structure
  - application policies are closely coupled
    - no mechanism for creating 3rd party policies
- Strict policy is not strict enough
  - does not meet needs of high security systems
  - situation is only getting better slowly
- Targeted policy is not simple enough
  - creating new targets is very difficult
- Policy needs to support binary modules

Issues with Current Policies

- Lacks clear and consistent security goals
  - some goals implied, but not rigorously applied
  - focused on functionality and preserving legacy behavior
  - macros are ambiguous and inconsistent
    - often over-privilege for convenience
- Roles and administration poorly addressed
  - adding or changing roles is very difficult
- Tight coupling among policy modules
  - type and attribute declaration / use are undisciplined
- Result is difficult to write and understand
  - both for developers and analysts

Reference Policy Goals

- Security goals
  - self-protection
  - assurance
  - improved role separation
- Functional goals
  - enable application policy development
    - including separately distributed applications
  - support system evaluation and accreditation
  - reduce the policy development complexity
    - enable users to make security relevant decisions
  - improve readability and comprehensibility
    - capture lessons learned
  - support source and binary modules

Approach

- Multi-step, incremental improvement of strict policy
  - strict policy contains important information
    - result of years of testing and evolution
  - preserve most existing conventions (e.g., naming)
    - ease conversion and adoption
- Add rigor by restructuring the strict policy
  - layering, modularity, abstraction and encapsulation
  - required to support later steps
- Iteratively tighten policy
  - systematic application of security goals
  - create multiple variations of the same application policy
    - each with different security / functionality tradeoffs
- Add targeted functionality
  - strict and targeted policy from single source
- Use on real systems immediately

Layering

- Not strict computer science layering
- Group similar modules
  - according function in the system
  - ordered from lowest to highest abstraction
- Lower layer modules included most system policies
  - protect kernel resources
  - startup / shutdown the system
- Higher layer modules are included as needed, e.g.,
  - mail daemon included on systems that send mail
  - X windows included desktops

Current Layers

- Kernel layer
  - protects the basic system and resources
    - including kernel image, devices, bootloader
  - does not include init
- System layer
  - lowest level user-space systems and resource
  - included on almost all system
  - covers init to multi-user login
- Key services layer
  - higher level system services
  - included based on system function
- Application layer
  - all other modules

Modularity Definition

- Smallest component in reference policy
- Groups related application policies
  - based primarily on similarity of functionality
  - includes policy rules and labeling
- Basis for encapsulation and abstraction
- Rough correspondence with RPMs
  - practical decision to ease system configuration
  - sometimes requires compromises
- Modules can be dependent on other modules
  - e.g., locallogin depends on authlogin
  - cannot be dependent on modules in higher layers

Module Examples

- Kernel layer
  - kernel
  - devices
- System layer
  - init
  - logging
- Key services layer
  - cron
  - backup
- Application layer
  - apache
  - postfix

Encapsulation and Abstraction

- Encapsulation hides module implementation details
  - allows module changes without effecting dependents
  - reduces close-coupling of policies
- Abstraction creates higher-level concepts
  - allows writers to make security relevant decisions
  - three types of macros
    - type transformation: e.g., make a type a domain or file
    - access: e.g., read all log files
    - template: common policy pattern
- Enforced by conventions
  - macros are used to create module ‘interfaces’
  - set of clearly defined policy writing rules

Policy Conventions

- Each module has three components
  - private policy: declarations and rules private to a module
  - interfaces: macros defining abstract access to module resources
  - labeling: file contexts
- Most important policy writing rules
  - types and attributes are private to a module
    - never referenced outside of the module
  - macros never declare types
  - available access defined by the module that owns the type
    - encoded in interface
  - interfaces following clear naming conventions
    - module name is prefixed
    - consistent verbs describing access: e.g., read, write, modify

Strict Policy Example

```
type initrc_t, domain, …;

allow initrc_t domain:process signal_perms;
allow initrc_t random_device_t:chr_file rw_file_perms;
can_setenforce(initrc_t)
allow initrc_t lockfile:dir rw_dir_perms;
allow initrc_t lockfile:file { getattr unlink };
allow initrc_t var_log_t:dir rw_dir_perms;
allow initrc_t logfile:file { read append };
```

Example Module Policy

```
type initrc_t;
domain_make_domain(initrc_t)

kernel_set_selinux_enforcement_mode(initrc_t)
domain_kill_all_domains(initrc_t)
devices_get_random_data(initrc_t)
devices_add_entropy(initrc_t)
files_remove_all_lock_files(initrc_t)
logging_read_all_logs(initrc_t)
logging_append_all_logs(initrc_t)
```

Example Module Interface

```
define(`devices_get_random_data',`
        requires_block_template(devices_get_random_data_depend)
        allow $1 device_t:dir { getattr search read };
        allow $1 random_device_t:chr_file { getattr read ioctl };
')

define(`devices_get_random_data_depend',`
        type device_t, random_device_t;
        class dir { getattr search read };
        class chr_file { getattr read ioctl };
')
```

Policy File and Directory Structure

- Preserve top level conventions (e.g., make targets)
- Three files per module
  - modulename.te
    - declarations
    - private policy
  - modulename.if
    - interfaces
  - modulename.fc
    - file contexts
- Separate directory for each layer
  - all three module files in the layer directory

Questions?

TRESYS TECHNOLOGY