



Towards Intuitive Tools for Managing SELinux: Hiding the Details but Retaining the Power

March 12, 2007

James Athey, Christopher Ashworth, Frank Mayer,
Don Miner

Copyright ©2007 Tresys Technology, LLC. All Rights Reserved.

Other names and brands may be claimed as the property of others. Information regarding third party products is provided solely for educational purposes. Tresys Technology, LLC is not responsible for the performance or support of third party products and does not make any representations or warranties whatsoever regarding quality, reliability, functionality, or compatibility of these devices or products.

Abstract

The details of the SELinux access control mechanisms lead to the perception that SELinux is too complex for non-expert users to manage. We present techniques that bridge the gap between the comprehensive, low-level SELinux access controls and the intuitive, high-level abstractions familiar to system administrators. These techniques shield the user from SELinux implementation details without sacrificing the power and flexibility of the SELinux policy language.

1 Introduction

SELinux provides flexible, comprehensive, and fine-grained mandatory access control using type enforcement. While type enforcement itself is fairly straightforward, the details of its implementation across all Linux resources lead to the perception that SELinux is too complex for non-expert users to manage.

We view the complexity of comprehensive and effective security policies as similar to the complexity of sophisticated software applications. The fundamental fact that software is built upon—and indeed *requires*—detailed, low-level computer instructions does not cause programmers to abandon all but the simplest, least useful applications. The low levels of computational implementation have been abstracted away by higher level programming languages, allowing programmers to manage the complexity of programming systematically and successfully. We believe the same principles apply to SELinux security management. Developing effective management tools leads to questions from the same engineering mold: What techniques allow SELinux “assembly language” (i.e. the policy language) to be hidden from the user? How do we build frameworks to automate tedious tasks? What higher level abstractions give us instruments matched to our goals rather than to the implementation, yet avoid dramatically sacrificing power and flexibility?

This paper presents lessons learned during the implementation of a new SELinux management tool called Tresys Brickwall™ Security Suite. We include techniques for managing SELinux as it exists today and suggestions to improve the manageability of SELinux in the future.

2 Intuitive Resource Management

SELinux provides an extensive list of low-level controls over such resources as processes, files, sockets, nodes, ports, and interfaces [3]. Many of these individual access controls can be meaningfully translated to more abstract, more intuitive, and more useful ideas.

2.1 Targets

The first kind of resource that an administrator must manage is the collection of services and programs running on the system. The administrator often identifies these processes by the name of the product, such as *Apache*, or by the service they provide, such as *Web*

Server. SELinux, however, identifies these processes in terms of the labeled domains that the processes run in (`httpd_t`) and the entrypoints that start those processes (`httpd_exec_t`).

To make these domains manageable and their permissions configurable, Tresys Brickwall groups related domains, entrypoints, and permissions into *targets*. Each target represents one service or application, such as Apache or MySQL. A target may represent one SELinux domain or several related domains that are part of the same package. Each target has a unique name and the capacity to store descriptive notes. The administrator manages the target by assigning specific network resources and files to it.

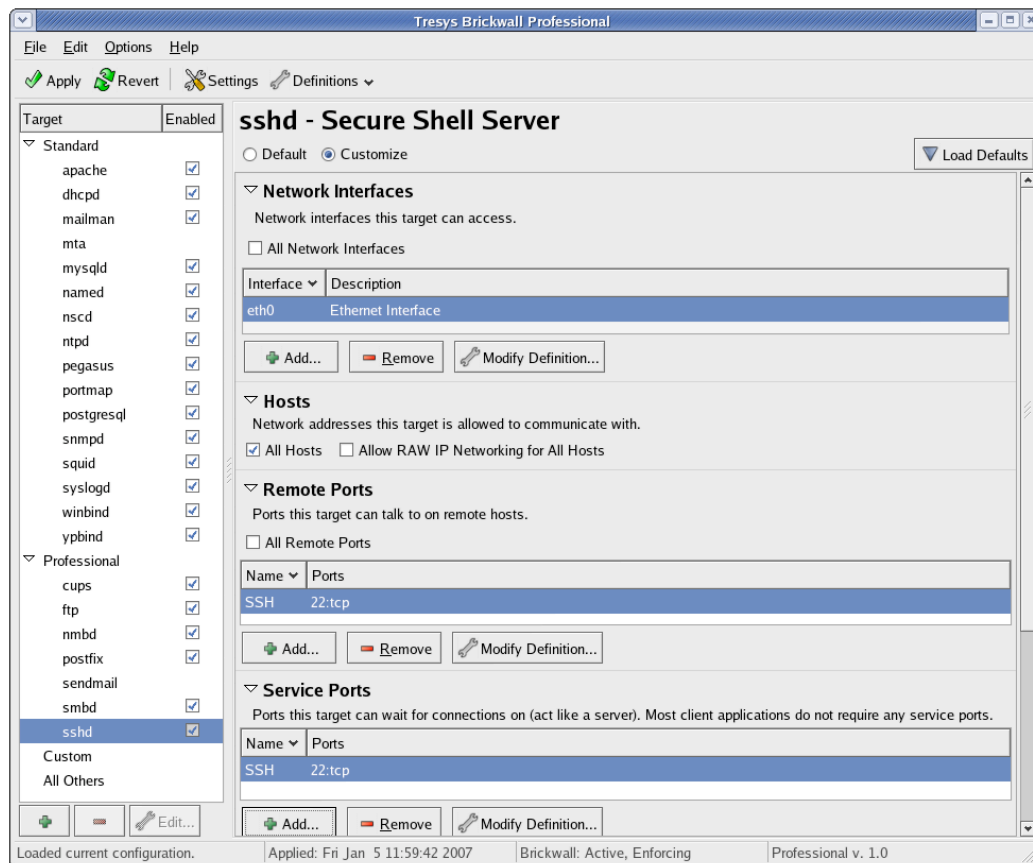


Figure 1: Target overview.

2.1.1 Custom Targets

Tresys Brickwall comes with many built-in targets that represent the domains already defined in the SELinux policy. For proprietary or modified programs unique to the organization, the administrator can also create *custom targets*. The administrator specifies the path to the executable or executables that start the program, and those paths become the entrypoints for the target in the resulting SELinux policy. Custom targets support the same flexible network and file security controls as built-in targets, and additionally offer other security options that Linux administrators are familiar with. These additional

controls include access to specific POSIX capabilities, control over the execution of code in memory (the “execmod”, “execheap”, and “execstack” permissions), and access to operating system features such as the system log.

Custom targets make it possible for a network administrator to create domains for programs with a far more secure result than running the programs in `unconfined_t`, all by using concepts familiar to the administrator and without knowing anything about SELinux. It is true that custom targets cannot be confined as tightly as SELinux policy modules written by hand, such as those used in the built-in targets. Certain SELinux security features are not currently exposed in Tresys Brickwall, such as controls over IPC mechanisms, and thus, out of necessity, custom targets get broad access to these features. Instead, custom targets offer the security customizations that are most important to an administrator concerned with the health of a network and the security of the files on the machine.

As mentioned above, an administrator manages the access each target has to network and file resources by assigning them to the target. Managing these resources effectively requires similarly powerful abstractions using familiar concepts, and Tresys Brickwall accomplishes this by grouping resources into unique *definitions*.

2.2 Network Resources

From a network administrator’s perspective, the network is described in terms of subnets, protocols, services, and named machines. Any tool that seeks to make SELinux network security more manageable should present the network in the same terms that the administrator uses.

By convention, some network resources have a built-in “well known” meaning. For example, port 22 customarily means the port for SSH traffic. Other network resources may only have meaning in a specific context. For example, subnet 255.255.0.0 may map to the more meaningful idea of “our corporate intranet”. The names “SSH port” and “our corporate intranet” capture the security relevance of these network resources much more clearly than “port 22” and “subnet 255.255.0.0”. Naturally, the particular ports or subnets associated with any higher-level abstraction are subject to change in different environments. Even “well known” ports are not guaranteed to mean the same thing in different networks.

The fundamental goal of a management tool is to translate intentions to implementations. The management tool should therefore handle translating between network resources and their security relevant semantics, such as their names and descriptions. The tool should also support the arbitrary assignment of low-level network resources to semantic groups, to decouple the semantics from particular implementation details.

A stock SELinux system cannot meet these goals, because it has no notion of the local environment. A stock SELinux system does not know what subnets are part of the corporate intranet and which belong to the public Internet. Likewise, there is no way for the original policy authors to predict what non-standard or custom ports may be in use on a particular network. While the *semanage* utility [7] does allow the SELinux-savvy

administrator to label previously unlabeled network resources, it does not let the administrator create new types, or change the meaning of existing types. Moreover, the meaning of raw SELinux types is frequently unclear. The administrator cannot add notes or descriptions to network resources, and resources that are labeled in the source policy cannot be altered.

2.2.1 Mapping Network Resources to Meaning

To meet the goal of describing network resources in specific, real-world terms, Tresys Brickwall provides the abstraction of *resource definitions* (or just *definitions*). Tresys Brickwall currently allows administrators to specify three kinds of network resource definitions: *port definitions*, *host definitions*, and *network interface definitions*.

Resource definitions closely map to an administrator's conceptualization of system resources. Each definition has a unique name and a description, and can contain multiple resources of the same type. For example, the port definition "SNMP" might contain three ports, 161:udp, 162:udp, and 199:tcp, which correspond to the three ports typically used by the Simple Network Management Protocol. Similarly, the host definition "Private" might contain three IP address ranges: 10.0.0.0/255.0.0.0, 172.16.0.0/255.240.0.0, and 192.168.0.0/255.255.0.0.

Definitions can also overlap; the port definition "All TCP Ports" by default contains the range 1-65535:tcp, overlapping all port definitions which contain any TCP ports. The administrator may also create new network definitions which more accurately describe the specific characteristics of the local environment.

2.2.2 Configuring Access to Network Resources

Grouping the network resources into definitions that reflect the real-world environment makes it possible for an administrator to quickly select resources to control. The administrator now needs to define the access to these resources. Network administrators know that a program needs permission to communicate to a given host using a specific port and network interface, but to accomplish this simple task in SELinux requires numerous permissions across a wide range of types and object classes.

In Tresys Brickwall, the administrator assigns the required resource definition to the target, and the software provides the combination of SELinux permissions to match the administrator's intentions. For example, if the administrator assigns the "eth0" Network Interface to the *httpd* target, Apache gets permission to send and receive network traffic on that interface. Other interfaces that have not been assigned to Apache can not be used. The administrator may also choose to assign only the "DNS" port definition to *httpd*'s Remote Ports, and as a result, the only kind of UDP traffic Apache can send is to port 53. Similarly, unless the administrator adds "RAW Networking" Apache's configuration, the target will not be allowed to use protocols other than TCP and UDP, despite having the permission to use eth0.

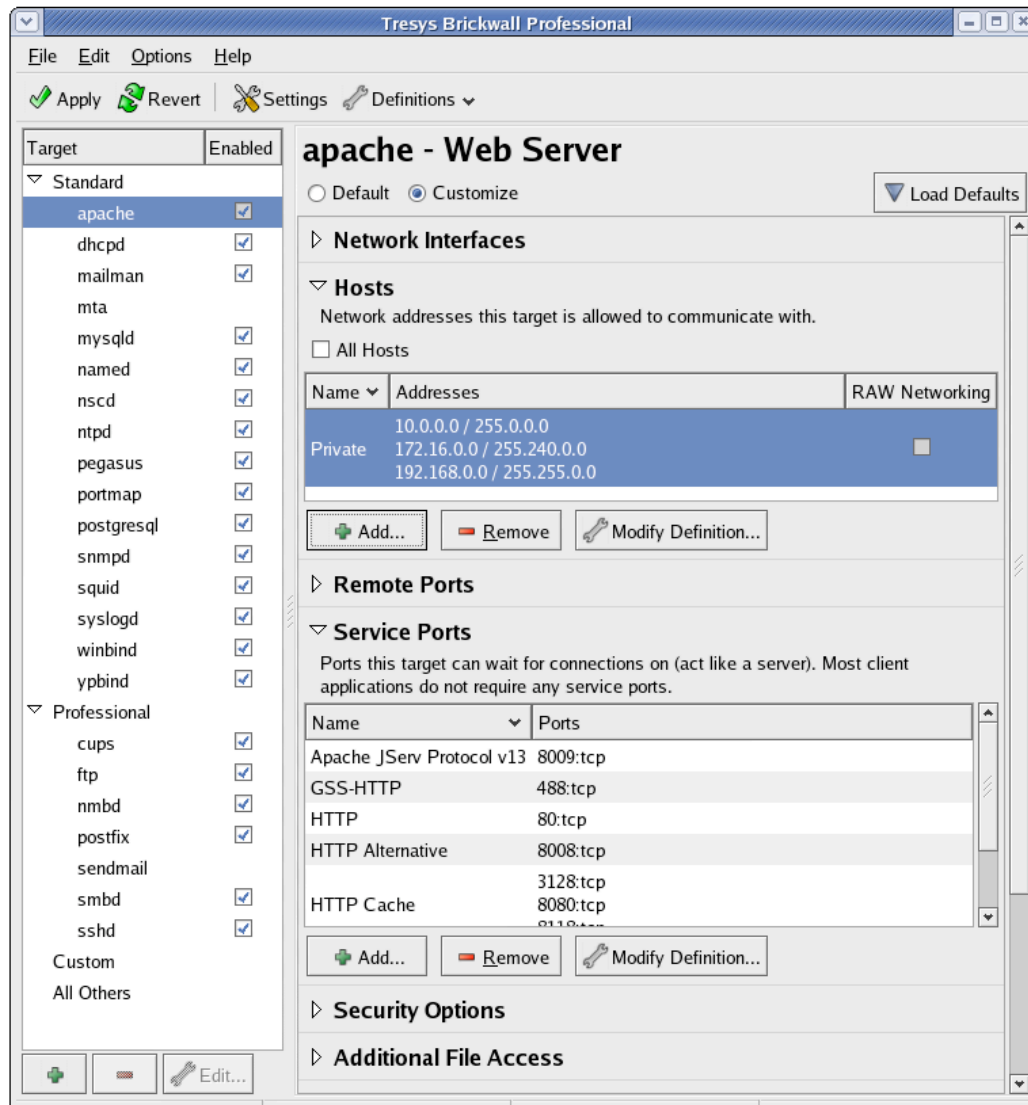


Figure 2: Network controls.

2.3 File Resources

Just as a natural way to describe a network is to use real-world ideas such as subnets and named servers, a natural way to identify a file is through its path. When describing more than one file, such as all temp files in `/tmp`, the most common way to describe the group is through the use of wildcards, which in this case would be `/tmp/*`.

Although paths seem fixed, each file in a Linux system is actually represented by an inode data structure, and uniquely identified by an inode number [6]. The label on the file that SELinux uses to make access decisions is stored in the inode, because the inode is the only unambiguous reference to the file. A path, on the other hand, may point to entirely different inodes at different times, due to the use of chroot jails or different

mount points. Similarly, more than one path can represent the same inode if more than one hard link points to it.[1]

Nevertheless, the labeling rules, called file contexts, in the SELinux policy use paths to identify files, because there is no other consistent, viable way to identify them. This system works, despite the potential for ambiguity in file paths, because it makes a reasonable assumption: when the system is labeling inodes, the filesystem is mounted in the same way it will be used in practice. Tresys Brickwall makes the same assumption, and this enables an administrator to manage permissions on labeled files.

2.3.1 Mapping File Paths to Meaning

File definitions are conceptually similar to network resource definitions in Tresys Brickwall. Each definition has a unique name, a description, and a set of resources associated with it. In the case of file definitions, each definition corresponds to one file type from `file_contexts`, and the resources associated with that definition are all of the contexts which use that type as the label.

For example, the definition “httpd Logs” corresponds to the file type `httpd_log_t`, and contains the specifications `/var/log/httpd(/.*)?`, `/var/log/apache(2)?(/.*)?`, and so on. Associating these specifications with a real-world name and description provides administrators with a more familiar abstraction than just the file type. However, the result may still confuse some administrators, because they may be unfamiliar with the regular expressions that are used to actually specify the files.

2.3.2 The Problem of Regular Expressions

Regular expressions provide a powerful mechanism to express multiple file paths in a compact form. When the specifications are sorted, it is easy for the system to determine if a particular regular expression best matches a given file path. Regular expressions have serious drawbacks, however, for purposes of management, automation, and policy authoring.

With regard to management, regular expressions can be extremely dense and confusing, on top of being unfamiliar to many administrators. Administrators are more accustomed to using shell-style “globbing” to specify a group of files, and the syntax used in globbing conflicts with the syntax used in regular expressions. Tresys Brickwall provides a search feature that bridges the familiar globbing syntax to the actual regular expressions used in the policy.

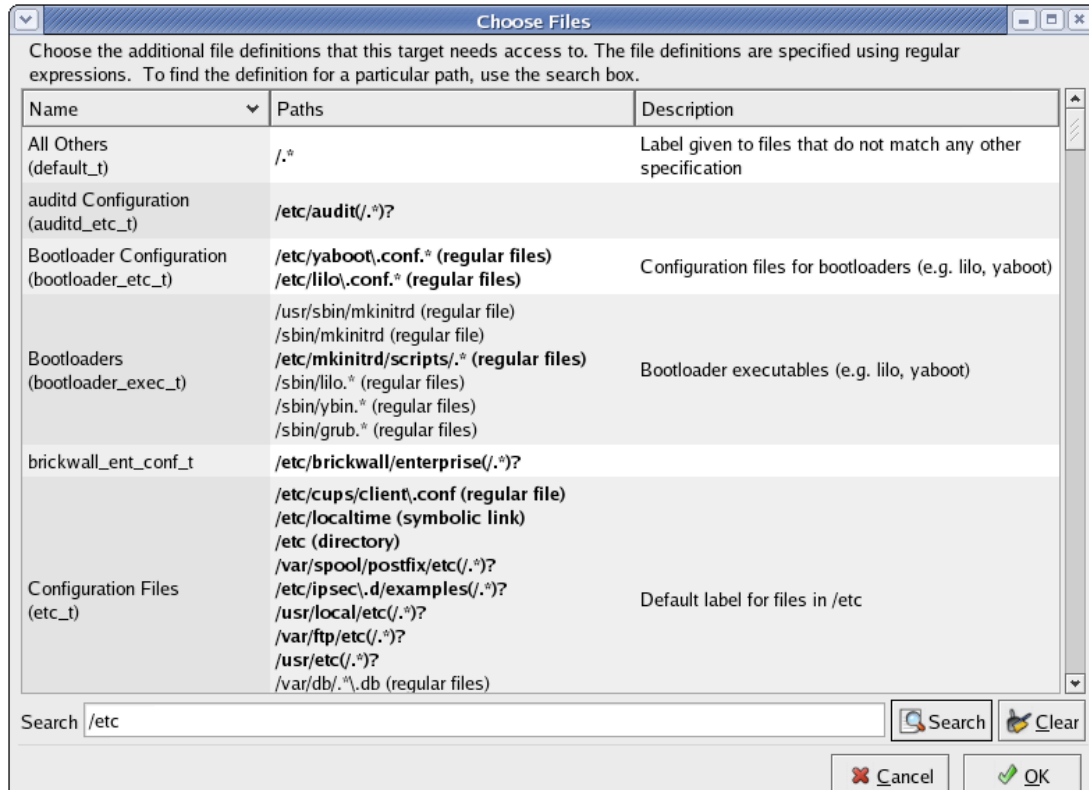


Figure 3: File definition search interface.

The administrator can specify a full or partial path, with an optional trailing wildcard, and the tool displays only those file groups that match the given path. The list is sorted with most specific matches at the top.¹

The main automation challenge that regular expressions present are a consequence of the `dir:search` SELinux permission. For a domain to perform any operation on a path, it must be able to “search” the path’s parent directory type, that directory’s parent’s type, and so on all the way to the root of the filesystem. Because file contexts using regular expressions can contain one or more wildcards in the middle, determining what parent directories belong to a context is impossible.

¹One of the most common computing operations—sorting—is difficult to perform on regular expressions. Consider, for example, the task of sorting default file labels in order of specificity. This is important because during a default labeling operation a generic label must be given only as a last resort, in favor of more specific labels. When SELinux policy was written in a monolithic fashion this ordering could be performed by hand. When modular policy was introduced, however, the policy toolchain required an automatic way to sort regular expressions from least specific to most specific. A theoretically complete algorithm for sorting file context rules was eschewed for a simpler heuristics-based approach. Although this method has been effective, it requires policy authors to be aware of the heuristics to avoid file labeling errors.

```
/usr/doc(/.*)? /lib(/.*)? system_u:object_r:usr_t
```

Figure 4: An impenetrable file context

Therefore, to provide the necessary search permissions for each file type, the parent types are provided instead of generated. To generate the parent types ahead of time, regular expressions are expanded where possible, and when certain wildcards remain, the specification is truncated to at least produce the parent directories before the wildcard. Any missing parent types are then added manually as they are discovered.

The final problem regular expressions as file contexts present is in authoring new contexts. First, because the syntax is complicated, authors frequently make mistakes that either fail to label the correct files or label too many. (One common mistake is to forget that the period is the wildcard metacharacter in regular expressions, and if an actual period is needed, it must be escaped with a backslash.) Second, because writing a new context necessarily takes files away from another file type, it is paramount that authors are careful not to break other application policies. However, if the new context needs to redefine the permissions on a subset of files in an existing file type, it is impossible to do so without also changing the policy of every application that uses that existing type.

As a result, it is impractical to ask administrators to alter labeling policy as part of managing the security of the system. Making the labeling policy static allows Tresys Brickwall to provide *Additional* File Access to targets without breaking other targets and still leverage the huge development and testing effort that goes into making the original SELinux policy work on a variety of systems.

Ultimately, providing better management of file contexts will require a new system for specifying them, one in which the syntax is familiar and where the specifications cannot conflict with each other, removing the need to sort the contexts altogether. One implementation of such a replacement has been proposed. [4]

2.3.3 Configuring Access to Files

Once a file definition has been assigned to a target, the administrator now needs to choose the access on that definition to grant to the target. Many administrators are familiar with the permissions in standard Unix discretionary access, namely read, write, and execute. These three permissions allow more operations than their name implies, however - for example, the write permission gives authorized users the ability to delete files as well as create new ones. On the other hand, SELinux provides dozens of file-related permissions that must be combined to provide any useful sort of access or allow real-world operations.

Tresys Brickwall provides a middle ground between the broad, vague controls in DAC and the precise, rigorous controls in SELinux. There are five kinds of permission - Read, Write, eXecute, Create, and Delete. They group together related access where it makes

sense from an administrator's perspective; for example, the Write permission includes setting attributes, renaming, and appending, in addition to the obvious writing permission. In short, these permissions reflect concepts that administrators are familiar with. Tresys Brickwall translates each of these abstract permissions into the necessary SELinux object class / permission combinations to grant the intended access.

3 Policy Generation

Policy generation is accomplished through a translation layer called the *generator*. The generator takes a security configuration file, which contains Tresys Brickwall definitions and targets but no SELinux details, determines the relationships between the definitions and SELinux labels, and produces an SELinux policy module to be added to the binary policy.

3.1 Harnessing Modular Policy

The SELinux policy ultimately produced by the Tresys Brickwall generation process is a managed, modular policy, even though RHEL 4's default targeted policy is unmanaged and monolithic. Tresys Brickwall uses local copies of recent versions of the SELinux toolchain and support libraries, thereby enabling the use of recent policy features. The resulting binary is still a version 18 policy supported by RHEL 4, and thus the kernel and the system's versions of libselinux et. al., do not need updating. This approach limits Tresys Brickwall's impact on the operating system to the SELinux policy alone. The reason to go to this effort is that using a managed, modular policy confers several benefits.

The current thrust of SELinux tool and policy development is focused on the managed, modular model [2]. Both Fedora Core and Red Hat Enterprise Linux now use managed, modular policies derived from the Reference Policy [5], and these two distributions represent a large percentage of the SELinux user community. As a result, managed, modular policy is now the more widely tested, documented, and understood type of policy, and any project which uses the same model benefits from its widespread use.

Using a modular policy also reduces the time spent generating and compiling the binary policy. Most of the rules in the policy are not affected by the management facilities provided by the Tresys Brickwall editor. It would be wasteful to process the sources that make up the unchanged part of the policy every time the security configuration changes. Instead, the Tresys Brickwall generator produces a single SELinux policy module that implements all of the settings specified in the security configuration, and this module is linked using semodule with the other, non-Brickwall policy modules to produce the Binary Policy.

Rebuilding only this tightly-focused module presents a new challenge: only the base module can contain network resource labeling policy, but the base module is large and, aside from network labeling, would not change. However, Tresys Brickwall frequently needs to alter how the network resources are labeled when the security configuration changes. To avoid recompiling the base module, Tresys Brickwall keeps no network

resource labeling policy there. Instead, Tresys Brickwall leverages `libsemanage` to insert network resource labeling rules into the policy store, and `libsemanage` applies these rules to the binary policy during the link stage. Targets in the Tresys Brickwall policy have hard-coded networking permissions on specific attributes, and Tresys Brickwall assigns types determined at generation time to these attributes to reflect the security configuration.

4 Network Management

Another abstraction currently available to network administrators in many contexts is centralized management of network resources. Firewalls routinely provide a mechanism for centrally maintaining and monitoring the firewall policy over multiple machines. Employing a modular policy scheme lends itself to a centralized mechanism for policy definition.

In many enterprise environments, multiple machines will be configured to perform similar or identical tasks. While tools may exist to control the applications on these machines remotely, no tools are currently available specifically intended to remotely configure the security of the underlying operating system.

Tresys Brickwall includes a centralized network management tool that provides the ability to group machines into functional areas, then define, apply, and monitor a security configuration for each group. Distributing the configuration to the group requires that the Brickwall Daemon exist on each machine under management. Using a modular policy and local generation allows the tool to distribute only the Tresys Brickwall configuration file. This significantly reduces the network traffic required for updates and allows the tool to scale well to large environments. Each machine in the group generates its SELinux policy locally, a process which takes only a few seconds.

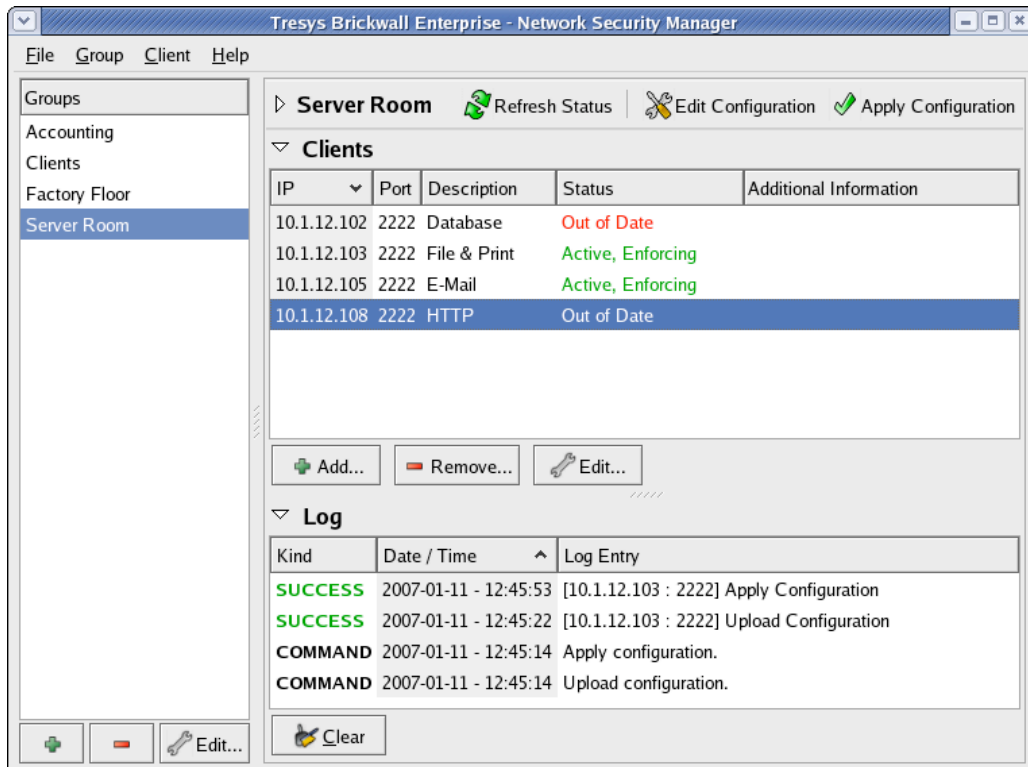


Figure 5: Network manager interface.

5 Summary

The concept of abstraction is well understood in computer science, but its application to SELinux policy is relatively new. Various research tools have been developed to provide a usable interface for policy definition and analysis, but all have required significant knowledge of SELinux concepts. Providing an interface that abstracts virtually all SELinux concepts required a number of tradeoffs.

First, the various SELinux capabilities had to be examined to determine which were the most powerful and likely to be widely applicable. The tool focused on configuring access to network and filesystem resources. This simplifies the management interface at the expense of restricting the ability to configure other SELinux capabilities.

One significant capability of SELinux that is not currently supported is the ability to re-label files. This results in potentially overly permissive file access. This is not a concern for the built-in targets provided with the tool, since the policy already accounts for file locations. However, for custom targets the administrator needs to be aware that granting access to specific files to an application may result in the exposure to unintended files, or exposure of the application's files to other applications.

The Reference Policy is widely used and examined by the SELinux community. Using this modular policy reduced the footprint of changes required by the configuration tool

and helped facilitate the centralized management of groups of machines. However, this required significant effort to back-port the policy to RHEL4.

We believe these tradeoffs are reasonable for most environments. By providing a configuration tool that requires no knowledge of the complexities of SELinux, we hope to broaden the appeal of this powerful security mechanism.

References

- [1] Brindle, Joshua. "Security Anti-Pattern: Path Based Access Control." Brindle on Security. 19 Apr. 2006. <<http://securityblog.org/brindle/2006/04/19/security-anti-pattern-path-based-access-control>>.
- [2] MacMillan, Karl, Joshua Brindle, Frank Mayer, Dave Caplan, and Jason Tang. Design and Implementation of the SELinux Policy Management Server. In *Proceedings of the Second Annual Security Enhanced Linux Symposium*, (2006), pp. 1-6.
- [3] Mayer, Frank, Karl Macmillan, and David Caplan. SELinux by Example. Prentice Hall, 2006.
- [4] Miner, Don and James Athey. FCGlob: A New SELinux File Context Syntax. In *Proceedings of the Third Annual Security Enhanced Linux Symposium*, (2007).
- [5] PeBenito, Christopher, Frank Mayer, and Karl MacMillan. Reference Policy for Security Enhanced Linux. In *Proceedings of the Second Annual Security Enhanced Linux Symposium*, (2006), pp. 25-29.
- [6] Smalley, Stephen, Chris Vance, and Wayne Salamon. Implementing SELinux as a Linux Security Module. The National Security Agency and NAI Labs, 2006. <<http://www.nsa.gov/seLinux/papers/module-abs.cfmi>>.
- [7] <<http://www.die.net/doc/linux/man/man8/semanage.8.html>>