Comparative study of operating system security using SELinux and Systrace

Jonas Öberg

Kalmar, 2009-06-02
C-nivå, 15hp

Handledare: Martin Blomberg, Högskolan i Kalmar, Institutionen för kommunikation och design
Examinator: Martin Blomberg, Högskolan i Kalmar, Institutionen för kommunikation och design

Institutionen för kommunikation och design
Högskolan i Kalmar

Summary

This thesis makes a comparative study of the security systems Systrace (used primarily with OpenBSD) and SELinux (used exclusively with Linux), trying to answer the question as to which type of security is offered by each respective system, and when each should be used.

Using a qualitative study into the documentation and material available, as well as practical tests, the thesis compares the two security systems based on a representative sample of common system administration tasks. From the analysis of each task, a summary is written for both SELinux and Systrace, which is then used in an analysis based on the grounded theory methodology.

The key difference between SELinux and Systrace seems to be their mode of operation, where SELinux, built around the LSM framework in the Linux kernel, works with type enforcement on files, sockets and other objects, whereas Systrace works on a strict system call basis.

In the thesis, the conclusion is reached that Systrace is useful and desired in many aspects that require absolute security but less flexibility: for instance in embedded devices, firewalls, proxy servers, authentication servers, etc, where users are not an issue, but the key point is to protect and secure specific server programs.

SELinux, on the other hand, can provide the flexibility that is needed in large enterprises and in systems where more than two levels of authentication are needed, for instance in document management systems. The use of SELinux can provide very attractive solutions for ensuring the integrity and security of an enterprise information. The two systems therefor serve two different purposes which sometimes overlap, but in just as many cases provide solutions for entirely different quality priorities.

Sammanfattning

Den här rapporten är en komparativ studie av säkerhetssystemen Systrace (främst använt i OpenBSD) och SELinux (använt endast i Linux). Studien försöker svara på frågan om vilken typ av säkerhet som erbjuds av respektive system, samt när respektive system bäst används.

Genom en kvalitativ metod som drar information från dokumentation och övrigt tillgängligt material, samt praktiska tester, så jämför den här rapporten de två systemen baserat på en uppsättning med vanliga uppgifter vid systemadministration. Från varje uppgift skapas en sammanställning för både SELinux och Systrace, som sedan används i analysen baserat på en grounded theory-metod.

Den huvudsakliga skillnaden mellan SELinux och Systrace verkar vara deras grundläggande funktion, där SELinux som är byggt runt LSM-ramverket i Linux-kerneln arbetar med så kallat Type Enforcement på filer, sockets och andra objekt, medans Systrace arbetar strikt på systemanrop.

I rapporten nås slutsatsen att Systrace är användbart och önskvärt i många aspekter som behöver absolut säkerhet men mindre flexibilitet: till exempel i inbyggda system, brandväggar, proxy-servrar, authentifierings-servrar, med mera, där användare inte är ett problem och huvudproblemet är att skydda och säkra specifika serverprogram.

SELinux, på andra handen, kan ge den flexibilitet som behövs i större organisationer och i system där fler än två nivåer av säkerhet behövs, tlil exempel i dokumenthanteringssystem. Användningen av SELinux kan ge en väldigt attraktiv lösning för att säkerställa integritet och säkerhet i ett större företags information. De två systemen fyller därför två separata behov som ibland överlappar, men som i lika många tillfällen ger lösningar för helt olika problem.

Abstract

This thesis makes a comparative study of the security systems Systrace (used primarily with OpenBSD) and SELinux (used exclusively with Linux), trying to answer the question as to which type of security is offered by each respective system, and when each should be used. The key difference between SELinux and Systrace seems to be their mode of operation, where SELinux, built around the LSM framework in the Linux kernel, works with type enforcement on files, sockets and other objects, whereas Systrace works on a strict system call basis. The two systems are seen to serve two different purposes which sometimes overlap, but in just as many cases provide solutions for entirely different quality priorities.

Key words: OpenBSD, Linux, security, SELinux, Systrace,

Foreword

This thesis was written as part of my continued studies in software engineering and where I with the kind help of the University of Kalmar was able to complete my education by writing about a subject I'm passionate about. The writing of this thesis started with a very practical test where I with the help of AEleen Frisch identified a number of common system administration tasks, and then implemented the various solutions to that task using both Systrace and SELinux.

This was my first attempt at using Grounded Theory as a methodology to analyse the systems and arrive at a concrete and accurate description of each. While it worked in part, I must also concede that a lot of aspects to the systems was not covered in the theory which emerged.

I believe that I've learned a lot about Grounded Theory and qualitative methods in this work, and in future research I will be able to put those theories to better use.

A big thank you to Martin Blomberg from University of Kalmar who has helped me navigate past the bureaucratic obstacles in my path!

Table of Contents

1.Introduction..1

2.Method..2

3.Demarcations..4

4.Background...5

4.1UNIX Authority model...5

4.2Principle of least privilege..6

4.3SELinux implementation...7

4.4Systrace implementation...9

5.Analysis...13

5.1Adding a user..13

5.2Limiting modification of system files.....................................16

5.3Limiting access to user or group files....................................20

5.4Running a service via an account other than root.................21

5.5Limiting/preventing access to a device..................................23

5.6Logging actions that occur (failed and successful)................25

5.7Bind a process to a privileged port..26

6.Results..28

7.Discussion..31

8.Conclusion..34

9.References...35

1. Introduction

This thesis compares the operating systems Linux and OpenBSD from a security point of view, taking the security systems Systrace (used primarily with OpenBSD) and SELinux (used exclusively with Linux) as the starting point to investigate the ways in which these security systems contribute to the security of each operating system.

The two systems are contrasted to each other and compared to understand their differences and relative strengths, as well as the environmental situations related to their use and when and how each system should be used.

The question that this thesis seeks to answer is: what technical security is provided by SELinux and Systrace, and in what situations should they be used?

I will take the operating system to mean the combination of the operating system kernel and all software used in a computer. In many situations, the security of a system is highly dependant upon the individual configuration, and both Linux based systems and OpenBSD are highly flexible in this regard. The capability of a user to send a signal to any process, regardless of the effective owner of the process, is dependent upon the operating system kernel and the authority levels which it implements, whereas the capability of a user to add another user would in many cases be dependent upon user-land tools and effective file permissions in the filesystem.

Unless otherwise stated, the systems discussed in this thesis are default installations of Debian GNU/Linux 5.01 (for Linux-based systems) and OpenBSD 4.4.

2. Method

Using a qualitative study into the documentation and material available, as well as practical tests, this thesis will compare the two security systems for some specific operations which are believed to be a representative sample of common system administration tasks. These tasks are (Frisch 2009):

- Adding a user
- Limiting modification of system files
- Limiting access to user or group files
- Running a service via an account other than root
- Limiting/preventing access to a device
- Logging actions that occur (failed and successful)
- Bind a process to a privileged port

Æleen Frisch, who helped with defining these tasks, have written numerous books on system administration, including Essential system administration, Unix System Administration and Essential Windows NT System Administration.

From the analysis of each task, a summary will be written based on both SELinux and Systrace for that particular task, I will then attempt to examine those summaries using grounded theory as a qualitative method. Grounded theory is a research method which derives the final theory from a system of codes, concepts and categories drawing from the original research material (Glaser and Strauss 1967). As such, it seems aptly suitable for research where we don't have a theory beforehand, but where one emerges from the material in the study.

In practice, the summaries of each task is first coded and organised to form concepts, which are then grouped in categories from which the final theory will emerge, which will answer the original question as to what security is provided by SELinux and Systrace.

Grounded theory has been criticised because of its time consuming coding process, and the precondition that researchers should have no preconceived ideas, which is often difficult (Allan

2003). However, I believe that the conceptual idea of grounded theory, in which the theory can be seen clearly emerging from the original data, and in this case the summaries of the analysis, is useful in this work, and I am making use of that concept in this thesis.

3. Demarcations

The primary access control mechanism in SELinux is type enforcement (McCarty, 2005). While there are other access control mechanisms in SELinux, such as role-based or multi-level security, this thesis will focus on type enforcement as it is the most commonly used in SELinux today.

4. Background

4.1 UNIX Authority model

The authority model of both the Linux kernel and OpenBSD inherits from older UNIX implementations, and the protection of individual files is under the control of the file system which can implement any type of protection that the developers choose. This usually follows a traditional UNIX model where each file or directory is associated with a user and group identification (UID and GID). The access levels of a file is then expressed in terms of a 3-tuple (read, write and execute permission) for the user, the group, and anyone who doesn't fall into either user or group (“other”). A specific user exists normally called root with UID 0 which bypasses all normal security checks (Stuart 2008)

One additional feature of the UNIX security model is worth to mention as well, since it has significant security implications. The so called Set UID (SUID) and Set GID (SGID) bits that can be activated on a file also inherits from older UNIX implementations. Their purpose is to allow a user to run a program, and let that program function as if it was executed with the permission of another user (Stuart 2008). This example demonstrates the use of the SUID bit. The SGID bit functions in the same way, but gives the running program the group permissions and not the user permissions.

> ”An instructor wants the students in a class to have access to a file, but only in a controlled way. One possible example is the grading file. In this example, the students should not be allowed to read the whole file; otherwise they'd be able to see other students' grades. Likewise, they should not be allowed to modify any part of the file. Preventing them from writing the file is easy with the normal permissions, but read access applies to the whole file. To handle this case, the instructor makes the grade file non-readable by the group and world and makes it readable by the owner. Then the instructor provides a program that reads the file and displays only the appropriately selected information. The program is executed by the students and have the SUID bit set. When a student runs the program, it runs as if it were the instructor and has access to the grade file. If the student attempts to access the grade file in any other way, the lack of read permission prevents access.” (Stuart 2008)

If the access program contains faults, this can lead to unauthorised privilege elevation, in which the user who runs the program gains access to the permissions of the owner of the file if the SUID bit is in effect.

This is particularly harmful if the owner is root, but in the example above this can also lead to that the user gains access not only to other students' grades from the students' own course, but also for other courses, exams, and any other information owned by the instructor. Using specific security domains means that the effect is limited, and if the user would gain access to that security domain, the only effect would be the ability to read the grade file which is protected by that security domain, and not other files owned by the instructor.

Another way which can be used to run programs temporarily as another user without using SUID programs is the use of the sudo program. The sudo program provides a flexible means by which the administrators of a system can define which programs can be run by who as whom. More clearly: the sudo program allowed users to run programs as another user, if they are given the authority to do so by the administrators.

The sudo program is often used to allow administrators to run programs as the root user without logging in as root. The sudo program also provides an audit trail in that it records which user ran which program, as which other user. In addition to this, the sudo program allows an administrator to define access to programs in a larger network, so that a user can be allowed permission to run programs as the root user on her own computer, but not on any other computer in the network.

4.2 Principle of least privilege

In this analysis I will investigate the relative strengths and weaknesses of security in Systrace and SELinux by contrasting how each system helps or functions in a set of representative system administration tasks. One of the principles that will be employed here is that of least privilege, which is a commonly used security principle which states that a program or user must be able to access only such resources as is strictly required for their functioning (Saltzer, 1975).

The use of SUID programs is in violation with this principle, since a a SUID root program, executed by a non-root user, would give the program access to all privileges of the root user and not only those that are required for its functioning. Giving a program only the strict privileges that it needs is tricky, since it is almost never known from the start exactly which privileges a program might need during its lifetime (Whittaker, 2003).

Both SELinux and Systrace attempt to follow the principle of least privilege, and Systrace is able to run in a learning mode in which it captures and recognises the (supposedly legitimate) system calls that a program makes, and records them so that it in the future can allow the same system calls, but disallow those which it has not encountered before while in learning mode.

Following the principle of least privileges, programs will often run with no particular privileges, and for system calls or resource access which requires a higher privilege than it has (such as root privileges), the privilege of the program can be elevated. This process is called privilege elevation in Systrace (Palmer, 2004). In SELinux, this process is made possible by allowing a program to transition between security domains.

4.3 SELinux implementation

SELinux is implemented on top of the Linux Security Modules (LSM) framework which provides a common interface for the Linux kernel for a lot of different security systems, not only SELinux (Wright et al, 2002) and as such, SELinux must adhere to a common interface. This is different from several other security systems, for instance Grsecurity which is implemented as replacements for the default security model of the kernel (Fox et al, 2008) instead of add-ons to it.

Another consequence of this is that SELinux is available only as an add-on to kernels supporting the LSM framework, namely the Linux kernel. It would not be possible, or associated with significant work, to port the SELinux module to work with OpenBSD or another operating system kernel.

The LSM framework consists of a number of hooks implemented in the kernel for access to restricted resources. When a program wishes to access a resource, the kernels own security model is first enforced, second, any security modules which have hooked into the LSM framework, such as SELinux, are queried in turn. If the security model of the kernel denies access to a particular resource, then no other security modules are queried and the program is denied access.

Only if both the kernel and all security modules allow the program access to the resource will it be allowed access. An important aspect of using the LSM framework is that the Linux kernels default security model always comes first. If the Linux kernel denies access to a resource, there is no way for SELinux to override this to provide access to the resource.

This is critical when it comes to the principle of least privileges. If a file has permissions which disallow editing of the file by an ordinary user, and we want to enable a specific user to edit that file, then we must first give that user access to write to the file using the normal Linux kernel security model, and can not rely on SELinux to grant this permission.

In this particular case, it might involve giving all users access to write to the file, and then using SELinux to restrict that same access for all but the user who should be able to write to the file (McCarty, 2005). This requires some particular attention and knowledge of SELinux and the way in which it functions to implement correctly, either in this way or through the use of domain transitions.

One of the criticisms of the LSM framework is that this makes it impossible for any LSM security modules such as SELinux to implement proper auditing (Mayer et al, 2008). If the kernel denies access to a resource, then SELinux will never know that access to that particular resource has been denied and can not log this event.

LSM has been criticized heavily by the author of Grsecurity for this problem, and has also been criticized on the basis that LSM might lead to the introduction of new rootkits (Spengler, 2009). LSM is an integral part of Linux 2.6 and many suppliers ship their products with LSM enabled, in order to facilitate the integration and use of SELinux and other security products (Weight el al, 2002). As LSM provides very clear and advanced hooks into the kernel, it is believes that malicious users might use this to their advantage when breaking into a system (Spengler, 2009).

Since SELinux works using the LSM framework, it can also not on its own decide which resources to restrict or audit. For instance, the system call mmap() does not contain any LSM hooks, and it is therefor impossible for SELinux to restrict access to mmap() even if it would want to. The number of resources available to SELinux to potentially restrict is in principle limited to files, pipes and sockets (McCarty 2005 and Smalley et al. 2008).

SELinux stores information about the security domain of a file in the inode of that file. This means that even if a user creates a symbolic link to a file and accesses it through that symbolic link, the security domain would still remain the same. However, if a file is moved and recreated by a program, it is not certain that the security domain would remain the same.

This is unless the application is running under that particular security domain or takes active measures to retain the security domain of the file. Operating systems such as Debian GNU/Linux which has been used in this thesis include wrapper utilities for several common system administration tools, which function exactly as the original tools but include functions to retain the security domain of files that it touches, or put specific security domain labels on files that it creates new.

The policies that control the SELinux environment and the security that it offers depend upon in which method SELinux is made to use. One of the security methods used in SELinux is Type Enforcement (TE), in which the access to a resource is allowed only if the requester type has access to the resource type (McCarty 2005). Other types of security models include Role-Based Access Control (RBAC) in which a requester can also be granted permission to a resource based on the role of the requester, or Multi-Level Security (MLS) in which information follows a strict hierarchical pattern where users of a specific security level can read information of lower or the same security level, but only write information of the same security level, or a higher security level, thus ensuring that information can only flow in one direction through the security layers.

4.4 Systrace implementation

Systrace follows a principle in which is limits a programs access by restricting the system calls which that program makes. Permissions are not enforced specifically for files, but the relevant system calls for opening and writing to a file can be caught by Systrace and then access granted or denied based on those system calls. For writing and reading to files, the Systrace program will keep track of which file is opened so that it can be referred to by name in the policy files.

While Systrace was built originally to work with OpenBSD, it has since been adopted so that it is possible to use in other BSD-style operating systems, and in newer versions it can also be used with the Linux kernel.

The principal method for operating with Systrace is to first create policy profiles for each program that is to be protected by Systrace. When starting the program using the learn mode of Systrace, it will record the system calls made by the program during normal operation.

When the program exits, the administrator can review the created policy file and adapt it to local needs. If the policy file was created by running the ls program on the /etc directory, then the policy file will include a policy which specifically allows the program to read the /etc directory, but disallows reading of any other directory. The administrator can change this to allow the program to read any directory in the filesystem if relevant.

Operations allowed by the policy file of a program must still be allowed by the default access methods. If the policy file allows the ls program to read all directories and files, this will still only be allowed if the user running the ls program have those privileges. In this way, Systrace can be used to limit the privileges of a program, but it can not grant additional privileges where those do not exist beforehand, except for in the case of privilege elevation which will be discussed separately as it refers to Systrace.

When a policy file for a program has been created, the program can be run with Systrace. Any attempts to access resources granted by the policy file will be allowed. Attempts to access resources not granted by the policy file will result in an interactive query to the operator whether to allow that resource access or not. This is commonly used to validate whether a policy file will allow the program access to the resources it needs during normal operation.

Once this validation is done, the program can be started with Systrace in enforced mode, meaning that any attempts to access a resource not allowed by the policy file will be denied.

The Systrace access control will additionally only be enforced on programs started using the Systrace facility. It is possible, and in fact common, for operators to start programs without using the Systrace facility. The use of Systrace in this way do not cover the whole system, but only those parts which are explicitly protected.

If a program protected by Systrace executes another program in turn, that program will also be protected by Systrace. In this way, it's possible to use Systrace in a way that significantly restricts access to a system by ordinary users by starting the users login shell using Systrace. Since all programs started by the user inherits from the login shell, all programs started by that user will fall under Systrace.

However, if doing so, all programs that can be started by a user will also need to have a Systrace policy associated with them, or they will not be covered by the Systrace policies. One way around this is to start Systrace by saying that all child processes should inherit a specific policy.

If done correctly, it could lead to a well functioning protection of the users interactive login environment against the rest of the system, but this is difficult to achieve and requires a deep understanding of the system and the relevant system calls (Palmer, 2004).

One of the uses of Systrace is to remove the need for SUID root programs, but the process of removing this need is not always clear. As has been described before, Systrace can not grant permissions other than those which the user originally has. In order to be able to grant permission to edit the password file of a system using Systrace, such as is done when using the program passwd which is SUID root, the user first needs to have the permission to edit the password file.

Systrace can then be used to limit this permission so that it covers only exactly the password file and not any other file in the system which would otherwise be the case. There are two ways in which privilege elevation can be accomplished in a safe way using Systrace.

The first way is relevant for server programs which interact with the network. Those often require elevated privileges in order to bind to a port under 1024, such as a web server which often needs to bind to port 80 (Peikari 2004). These server programs are often started by the root user. In the case of the Apache httpd server, it includes some features that allow it to drop those root privileges after it has bound to port 80. All other access after this will be done as a normal user.

But using Systrace can further enhance this security. The Systrace policy for the program can include specific instructions saying that the program is allowed to bind to port 80, as if it were root. When starting the program via Systrace, Systrace can also be instructed to start the program as a specific user, typically the www user. This user will have no special privileges in the system and will be used only for running the server program. But since it was launched by the root user, Systrace can elevate the privileges of the running program, even if it's running as the www user, to those of root for specific system calls which require this privilege.

The other way which is relevant for SUID root programs is to move the program to another name and remove the SUID root permissions of it. A Systrace policy is then created which uses privilege elevation in the way described above to give permission to specific systems calls as if the program was running as the root user, which can include opening and reading or writing to specific files. A so called wrapper program is then created, which is given SUID root privileges.

This wrapper program is given the name of the original program. When started, the program will in turn start the original program, but ensure that this is done using Systrace, and with the privileges of the original user. This allows the program to run with the permissions of the original user, but with privilege elevation for the system calls which require this.

It should be noted also that Systrace includes the possibility to log both authorised and unauthorised system calls through the use of the logging functionality in the policy file for an application. In this way, Systrace can be used to audit a program and record when and how specific files were accessed and if access was authorised or not.

5. Analysis

5.1 Adding a user

Under both Linux-based systems and OpenBSD, assuming that the system is using local authentication, adding a user is completely dependent upon the ability of the user to manipulate files in the /etc directory (Peikari 2004). The details differ in so far that with OpenBSD, the information used by the system is kept in a database file generated from the plain-text files, and updating this database requires a temporary file to be created in /etc (Palmer 2004), whereas in Linux-based systems, it is enough to be able to manipulate the /etc/passwd and /etc/shadow files in place (Ward 2004).

When it comes to the question if Systrace or SELinux can contribute to securing this process, we need to consider the case of privilege elevation and least privileges. There is a tool in UNIX called useradd which can be used to add a user to the system. This tool exists under both systems. In addition to this tool, there exists also a adduser program which is an interactive program that can be used instead of useradd and is more “user friendly”.

The program adduser is a Perl script in both systems, and it in turn calls on other programs, such as useradd, to perform the actual addition of the user to the system. We can therefor limit this analysis to the useradd program, which is more specifically tasked with adding a user to the system. When run, the useradd program adds the user to the system by modifying the relevant plain text files and potentially rebuilding the user database.

This tool is traditionally run only by the root user. The question thus becomes; can we, using Systrace and SELinux, enable this program to be run by an ordinary user (a system administrator), using privilege elevation to give the program access to the right files and directories in order to add a user, and at the same time limit the access to the minimum required for this functionality.

One way to do this without using neither Systrace nor SELinux would be to make the program SUID root, but executable only by a specific group. Any user thus in that group would be able to execute the program which would then run with an effective user ID of root. This violates the principle of least privileges, since the program would gain access not only to the files and directories relevant for adding a user, but any file or directory in the system, as well as any other privileges granted to the root user. The same would be true if the program is executed using the sudo command.

5.1.1 Systrace

Using Systrace, it would be tempting to try to create a policy such as:

> native-fswrite: filename eq “/etc/passwd” then permit as root, if group = wheel

Using this policy would enable the program to write to the passwd file in the /etc directory as if it was root, but only if the user running the program is a member of the wheel group. In this way, we could give specific privileges (following the principle of least privileges) to individual users which are part of a pre-defined group, also enabling them to run the program without first becoming root.

There is however a problem with this: as I've explained in “Systrace implementation“ (page 9), Systrace can not elevate privileges beyond what it already has. That means that unless Systrace itself is started with the privileges to write to the /etc/passwd file, then it can not grant such privileges to another program, regardless of the policy in place.

In principle this means that we must start Systrace either as root, or by giving it SUID root privileges (highly insecure). If neither of these are true, then the program can not elevate its privileges. Thus we find ourselves in the situation that Systrace can easily be used to restrict privileges that a program has from being used, but it's more difficult to elevate privileges, even where that would be allowed according to the system policies.

There can be found a way using Systrace to accomplish our task while adhering to the principles of least privileges. We can use the method described above to make the program SUID root, while executable only by members of the wheel group. This would in theory make the program run with unrestricted root privileges, but we can then use Systrace to limit those privileges to exactly those that are needed, namely reading and writing files in /etc, and deny access to any other file in the filesystem.

Another possibility is to write a wrapper program which is SUID root and executable only by members of the wheel group. That wrapper could execute Systrace on the useradd executable, giving appropriate parameters to Systrace to start the program as the calling user. This would allow the program to run as a specific user while still allow some privilege elevation, since Systrace was started as the root user with unlimited privileges.

5.1.2 SELinux

SELinux contains the functionality of privilege elevation through domain transition. Such a transition can occur when a user needs to change her own password, but also when adding a new users, since the files affected are the same. The files in question are in a security domain with the type shadow_t. A user would not have permission to edit these files directly, since no user domain is given access to the shadow_t type.

The program passwd used to change the users own password is in a security domain with the type passwd_exec_t. There are policies in the SELinux reference policy saying that a program in a security domain with the type passwd_exec_t is allowed to enter the security domain with type passwd_t. This is referred to as a domain transition. There are also policies that allow a program running in a security domain with type passwd_t to access files of type shadow_t.

When a user starts the passwd program, the security domain for that program will transition into passwd_t which can then update the password of the user in the relevant files (Red Hat, 2008).

Using SELinux and type enforcement we can in this way place the system administrators in a security domain, let's say sysadm_t and allow the security domain with type sysadm_t to transition into type passwd_t when running the useradd program.

This whole process involves creating the additional type passwd_exec_t, giving passwd_t an entry-point from passwd_exec_t, and creating a policy to automatically transition from sysadm_t to passwd_t, but all of this is already included in the SELinux reference policies, although for a larger enterprise they would need to be audited independently to make sure that they are correct according to the enterprise' security policy.

5.1.3 Summary

Systrace can easily restrict privileges, but it is more difficult to elevate privileges when this is needed. Wrappers need to be used to make use of Systrace for adding users to a system. Using SELinux, privileges can be granted using domain transitions, whereby a program can transition into a security domain which has access to the files that needs updating without needing root privileges. SELinux does not make use of SUID programs.

5.2 Limiting modification of system files

The system files in a UNIX systems are traditionally those in the /etc directory which are used for configuration of the system. However, by system files I will also include all programs that are installed in the system and which are part of the running environment, as well as log files created by those programs. In order to prevent or lessen the effects of an attack on a system, we must ensure that the configuration of a system is intact. If an attacker is able to change the configuration of the system, this would include the ability to change the authentication methods or add or remove users from the system.

Additionally, we must secure the programs that are run in the system. Since many of them are run as the root user with unlimited privileges, an automatic start of a modified program, or an inattentive administrator running a modified program, can have significant security implications.

When an attack is underway, the attacker will usually focus on the system log files in order to hide any trace of their activities. It therefor seems prudent to in the definition of system files include both configuration, program files as well as log files.

It should further be noted that with Systrace, if an attacker is able to change the contents of a program file, and if that program has a Systrace policy, the modified program file would be covered by the Systrace policy as well, which could lessen the effects of the attack. In SELinux, if an attacker is able to replace the contents of a program file, it's possible that this means replacing the entire inode of the program, in which case the SELinux domain type would be lost and the program would run in an unprotected domain.

This can be solved by making the default policy in SELinux be to deny everything. Should a program then lose its original type, it would not have access to any resources at all, not even those it originally had. In contrast, if a program under Systrace loses its Systrace policy, the program would run unprotected and not limited beyond the normal UNIX permissions.

Both Systrace and SELinux contain the necessary functions to limit access to system files. It should be noted that since both Systrace and SELinux build upon the existing security model of the respective kernels, it's always possible to use the standard UNIX security flags on files to prevent unauthorised modification, and indeed this ought to be the first step when limiting access to system files before turning to Systrace and SELinux.

5.2.1 Systrace

With Systrace, it's important to note as I've done before that only programs running specifically under Systrace protection will be covered by the further limits put by Systrace policies. A program which does not have a Systrace policy will revert back to the system defaults, which could mean that if a program in turn starts another program, that program might have access to files which the original program does not, unless the Systrace policy specifies that specific policies should be inherited.

There are essentially three system calls relevant for Systrace when limiting access to system files: stat, fsread and fswrite, referring to the ability to get meta-data information about files in the filesystem, read files, and write files respectively.

For all three system calls it's possible to specify more granularly which files should be affected. The default policy of Systrace for a program is to not allow any of the system calls to any files, so one must therefor specifically grant permission to the files and directories which the program using the policy should be able to access.

This would allow a program to access and write only those files which are in the users home directory, as well as read files in the /tmp directory.

```
native-fsread: filename inpath "$HOME" then permit
native-fsread: filename eq “/tmp/*” then permit
native-fswrite: filename inpath "$HOME" then permit
native-stat: filename inpath "$HOME" then permit
```

5.2.2 SELinux

For SELinux, limiting access to a file means putting that file in a type to which the user has no access, and ensuring that the user is not able to transition to a security domain of that type. Even if the user by the standard UNIX permissions would have permission to read the file, SELinux will deny access if the file is not in the users domain. Creating a specific security type involves creating a policy module that will plug into the policies of SELinux and should contain information about the type itself, the transitions allowed (if any) as well as the access permissions if relevant.

This is an example of the creation of a policy module called mymod which creates the security type mymod_t and the getattr permission to files of that type to the sysadm_t type:

```
policy_module(mymod,1.0)
require {
        type sysadm_t;
}

type mymod_t;

allow sysadm_t mymod_t:file {getattr};
```

If compiled and inserted into the SELinux policy with the semodule command, this will result in the creation of the mymod_t type, which if applied to a file will grant access to that file only to the sysadm_t security domain, but only permissions enough to view permissions, size, owners, and other meta-data attributes of the file (getattr). No user (not even the root user) would be allowed to view any other information about the file, nor read its contents.

In the test system, I created the above policy and applied the type mymod_t to the file /home/jonas/testfile. The user jonas was placed in the staff_r role (and staff_t type), which can transition into the sysadm_r, but this does not happen automatically. The user must transition manually to sysadm_r through the use of the newrole command, and is then allowed to view the attributes of the testfile. However, reading it is still not allowed:

```
jonas@debian:~$ id -Z
staff_u:staff_r:staff_t:s0
jonas@debian:~$ ls -l
[   71.633808] type=1400 audit(1242672620.183:8): avc:
denied { getattr } for  pid=2202 comm="ls"
path="/home/jonas/testfile" dev=dm-1 ino=146316
```

scontext=sta ff_u:staff_r:staff_t:s0
tcontext=system_u:object_r:mymod_t:s0 tclass=file
ls: cannot access testfile: Permission denied
jonas@debian:~$ **newrole -r sysadm_r**
Password:
jonas@debian:~$ **id -Z**
staff_u:sysadm_r:sysadm_t:s0
jonas@debian:~$ **ls -l**
total 4
-rw-r—r-- 1 root root 22 May 18 20:25 testfile
jonas@debian:~$ **more testfile**
[81.867994] type=1400 audit(1242672630.415:9): avc: denied { read } for pi d=2220 comm="more" name="testfile" dev=dm-1 ino=146316
scontext=staff_u:sysadm_r :sysadm_t:s0
tcontext=system_u:object_r:mymod_t:s0 tclass=file
testfile: Permission denied

This shows that protection of system files in SELinux is feasible, although slightly impractical by the need to create specific policy modules. The default policy for SELinux, called the reference policy, does contain several types that are applied by default to several system files, such as etc_t for files in the /etc directory. Such default policies could be modified to allow only the system operators access to etc_t type files, although doing so might break other parts of the installation which relies on users ability to read etc_t type files.

However, it should also be noted that the default policy for SELinux is to deny all access which is not explicitly granted. This is different from Systrace where the access depends on the application wanting access to a specific resource, and not the user requesting it.

5.2.3 Summary

Systrace does not require changing of the object that one wants to limit. If a Systrace policy is lost, the program will be granted unrestricted access to the system (other than that given by UNIX permissions). The default permission of a Systrace policy is to deny all access, and permission can then be granted to specific files or directories. With SELinux, files that are to be protected needs to be put in a different security domain. Reference policy contains reasonable defaults that can be built upon, but this requires deeper knowledge of SELinux and the running system.

5.3 Limiting access to user or group files

The user with effective user id of 0 (root) can read and write any file on both OpenBSD and Linux-based systems (Stevens 1992), unless Systrace or SELinux denies it. Limiting access to user or group files are in essence not different from limiting access to system files. However, there's a clear difference here which should be noted since it is significant to both systems.

Assuming a situation with two distinct users, where one wants to limit their ability to see each others files, allowing them only access to their own files. This is a situation perhaps solved best by removing the access permissions from world and group in the UNIX permissions for the files and directories in question, thus relying on the normal UNIX permissions to deny access.

As both SELinux and Systrace can be used to restrict access to files which a user would normally be able to see, but can not, as explained earlier, grant permissions where a user would not normally be able to access a file (Red Hat 2008, Palmer 2004), this makes using UNIX permissions to deny access a viable option.

5.3.1 Systrace

Doing the same with Systrace, assuming a situation where the policy is placed on the users login shell, and policies are then inherited from it, one could use the environment variable $HOME in the Systrace policy, as shown before:

```
native-fsread: filename inpath "$HOME" then permit
```

This would allow each user read access to the files which are in her home directory, but not to any other files.

If looking at limitations for groups, there are no clear ways for how to do this in Systrace. One possibility would be to place each groups file in separate directories, such as /export/group1, /export/group2 and so on. Then for each user, a separate Systrace policy would have to be created, granting read and write access to files in the group directories that the user should have access to, which will increase in complexity with the amount of groups in the system.

5.3.2 SELinux

If wanting to do this with SELinux, one would in practice have to create different types for the two users, for instance user1_t and user2_t, allowing each user access to her own type, but not the other.

In SELinux, one could accomplish the group division by creating a type for each group, and giving users the ability to transition into a security domain which has access to the group files they need, however doing so means that the user needs to transition into a new security domain each time a new group file should be worked with although this process can also be automated so that the security domain transitions automatically if allowed to do so when needed.

5.3.3 Summary

SELinux requires the creation of new types for protecting files, but automatic domain transitions can be used to make it easier to transition to the correct security domains when needed. Systrace allows using $HOME and other environment variables in policies. But using Systrace can also make the security system very complex if having many groups.

5.4 Running a service via an account other than root

Running a service via an account other than root is a technique used as default by the Apache web server, the BIND name server, and several other server programs as well as server programs made to run via another account by the administrators of the system.

This would be a typical usage of the sudo program which has been explained earlier, and which gives the possibility for an administrator to grant access to run a service via an account other than root to any user on the system. SELinux doesn't contain any tools or features which would make this process of running a service via an account other than root directly neither more secure or easier.

5.4.1 Systrace

Systrace includes the ability to run a service via an account other than root per default, and it's a common way of using Systrace to start a program as another user while still allowing some limited form of privilege elevation for specific system calls. Systrace can also be used generally to, as the root user, start programs as another user. However, Systrace can not be used by a non-root user to start programs as another user.

However, both Systrace and SELinux provides the ability to grant specific privileges to non-root users, which could be used to facilitate this process. Assuming the situation where one wants to allow certain users the ability to run the passwd program to set the password of another user, one could give those users a specific Systrace policy, which would include:

> native-execve: filename eq "/bin/passwd" then permit as root

This would only work in so far that Systrace can be used for privilege elevation (such as starting Systrace as root, but telling Systrace to start the individual program as the user herself, or using wrappers), but the effect would be that the user is able to run the /bin/passwd program as if she was the root user on the system, and Systrace would elevate the privileges to root when running that command. This can also be used not only to elevate to root but to allow system calls to run with the privileges of any user or group on the system.

5.4.2 SELinux

With SELinux, it would be possible to grant the security domain of the system administrators the access to read and write the passwd files and other relevant files in the system. Doing so would allow them to change the password of another user, either manually or through some program, without actually needing root permissions, or the ability to run a program as root.

This is associated with a particular danger though. As SELinux can not grant more permissions than the default UNIX permissions allow, then write permission to the password files must be granted to the world, or at the very least to the group which includes all administrators. Once this is done, SELinux can be used to restrict that write access only to the administrators.

But should the system for any reason, by malfunction or administrative error, disable SELinux or its security mechanisms, the write permissions on the password file would be available to the world and would be unrestricted.

5.4.3 Summary

Systrace can use privilege elevation to perform a system call as another user. SELinux can not be used to run a program as another user, but can be used to allow programs to be run, giving them permissions to the resources needed without needing to elevate the privileges.

5.5 Limiting/preventing access to a device

In UNIX, there are few differences between a file and a device, and devices are accessed through special files in the regular filesystem. As such, they can be protected by SELinux and Systrace exactly as any other file, and this specific task therefor shares much in common with the task of limiting access to system files.

5.5.1 Systrace

Using Systrace, exactly the same method can be employed to limit access to system files as it can to limit access to devices. There are no particular hooks or features available in Systrace to make this easier or more complicated.

Network devices represent a class of devices which both Systrace and SELinux has specific support for, as it's a commonly occurring requirement to be able to limit access to the network for programs or users. The following statement would allow a program running under Systrace to connect to any other system on the Internet, but only on port number 53:

```
native-connect: sockaddr match "inet-*:53" then permit
```

This is typically used to allow programs access to DNS servers, but deny any other access which is not also explicitly allowed. In this way, Systrace doesn't directly control access to the device, but it controls access to the device indirectly by limiting a programs ability to make use of it to send network traffic. Network devices which are not connected or in use will not be affected by this, nor would it allow the program to change any details of the network connection.

5.5.2 SELinux

In SELinux, the reference policy includes a number of types specifically for devices. This includes the sound_device_t for access to any sound device (including audio input, output and mixers) and usb_device_t for controlling access to USB devices). There is no support built in the reference policy to limit access only to some USB devices, and access to USB devices presents a particular problem here.

USB devices are enumerated in the filesystem in accordance to the port to which they are connected, and sometimes even the order in which they are connected. The file name can thus be very different from one time to another, and in order to control access to specific USB devices, such as allowing USB web cameras to be connected but not USB memory sticks, require significant efforts which are outside of the scope of SELinux, although SELinux can after that be used to place the USB web camera devices in a specific security domain to which the user has access.

Using SELinux, network access policies can be created which are flexible. This example is from the policy of the BIND name server, which runs in the security domain with type named_t. The policy allows the type named_t access to an udp socket of type port_t, but only when connecting to other hosts on the port name_bind, which has been defined elsewhere as port 53.

```
allow named_t port_t:udp_socket name_connect;
```

These policies are based on the premise that the network device it setup and working. It does not provide any access to unused devices, not does it provide any access to the device directly or to the ability to configure the device, but only limits the programs interaction with the network.

5.5.3 Summary

Systrace can limit access to devices just as to files and directories. Network devices can be limited. SELinux reference policies include types for some devices, and can otherwise limit access to devices if they are devices in the filesystem. Network devices can be limited.

5.6 Logging actions that occur (failed and successful)

5.6.1 SELinux

In Linux-based systems, the normal UNIX permissions take precedence over the LSM modules, including SELinux. As such, if access to a file, device or socket is denied, SELinux is effectively excluded from any further processing and is not able to log any information about that access request. Only if the UNIX permissions allow access will SELinux be able to log if SELinux itself denied or allowed access to a device.

5.6.2 Systrace

Systrace is significantly different in this sense and allows the possibility to log access requests, regardless of if they are permitted or denied. However, it's important to note that Systrace logs the access according to its own policy, not that of the underlying system. Systrace is intercepting system calls before they are actually made, which means that it can log all system calls, as well as deny or allow access before the UNIX permissions come into play. But even if the Systrace policy allows access, and logs an action as being permitted, this does not mean that the system call was ultimately successful since subsequent processing by the kernel can deny access to the resource.

This example demonstrates the logging of Systrace. In the example, I'm using a Systrace policy which has been told specifically to permit, but log, access to the /home/jonas/d file. I first run the program cat on the file, when the file is readable. I then remove the read permissions, and attempt the same command again. This time, the system denies access to the file, but the Systrace audit still logs the call as successful:

$ **systrace -a cat d**
This is a test file.
May 19 09:50:38 bsd systrace: permit user: jonas, prog: /bin/cat, pid: 24083(0)[0], policy: /bin/cat, filters: 14, syscall: native- open(5), filename: /home/jona s/d, oflags: ro
$ **chmod -r d**
$ **systrace -a cat d**
cat: d: Permission denied
May 19 09:50:45 bsd systrace: permit user: jonas, prog: /bin/cat, pid: 8309(0)[0], policy:

/bin/cat, filters: 14, syscall: native-open(5), filename: /home/jonas /d, oflags: ro

While Systrace does provide ways by which system calls can be logged, it can only provide an audit trail of its own access policies, and not for the system generally.

5.6.3 Summary

SELinux can not log actions which are denied by the kernel. Systrace can log actions, but can not provide information about if the kernel then ultimately denied or allowed access beyond the Systrace policy.

5.7 Bind a process to a privileged port

Normal user-land binaries without any particular capabilities will usually bind to ports above 1024. Binding a port below 1024, such as port 80 commonly used for webservers (Peikari 2004), usually require specific capabilities. In Linux-based systems, as well as in OpenBSD, any process with the effective user id of 0 (root) can bind to any privileged port. In order to allow for a more fine-grained control of the authority levels, Linux kernel implements a capability structure which has its basis in POSIX draft 1003.1e, which allows the root user to assign the capability NET_BIND_SERVICE to a program, which would enable that program to bind to a privileged port (Böhm 2006). However, this would also allow the process to bind to port 81, or any other privileged port. Both SELinux and Systrace therefor implement a more comprehensive system for capabilities.

5.7.1 Systrace

Both Systrace and SELinux allow a fine-grained control whereby a program can be restricted to opening just a specific privileged port (Red Hat 2008, Palmer 2004). The process of doing so is similar to that used for restricting access to the network and involves in Systrace creating a specific policy which permits the program access to the bind system call, optionally limited to binding to specific ports or internet addresses, or allowing it to bind as if it was root.

```
native-bind: sockaddr match "inet-*:80" then permit as root
```

5.7.2 SELinux

The policies in SELinux for binding to a port are similar to those used for restricting outgoing network traffic.

```
allow http_t http_port_t:udp_socket name_bind;
```

With SELinux, it's also possible to allow a process to bind not only to a port but specifically to a designated IP number. Doing this can be used for instance for virtual hosting, whereby several server programs can be run on the same port but with different IP numbers without conflicting. SELinux can be used to enforce a separation between the server programs such that they are prevented to binding to ports on an IP number other than the allowed one.

Doing this involves creating a security domain for the IP number that the program should be allowed to bind to:

```
nodecon 130.241.150.2 255.255.255.255
system_u:object_r:node_gu_t
```

When combined with the name_bind option, SELinux will check whether a socket can be bound to a specific type of node, in this case enforcing that the IP number 130.241.150.2 requires a security domain system_u:object_r:node_gu_t in order to allow the program to bind to it.

5.7.3 Summary

SELinux can limit and grant privileges to bind to specific ports. Systrace can limit binding to specific ports, or use privilege elevation to allow binding.

6. Results

The summaries in the preceding chapter are coded according to SELinux or Systrace and assigned a unique identifier.

SELinux	Id
Privileges can be granted using domain transitions	101
SELinux does not make use of SUID programs	102
Files that are to be protected needs to be put in a different security domain	103
Reference policy contain reasonable defaults that can be built upon	104
This requires deeper knowledge of SELinux and the running system	105
Requires the creation of new types for protecting files	106
Automatic domain transition can be used to make it easier to transition to the correct security domain when needed	107
Can not be used to run a program as another user	108
Can be used to allow programs to be run with resources without needing to elevate privileges	109
Reference policy includes types for some devices	110

Systrace	Id
Systrace can easily restrict privileges	201
It is more difficult to elevate privileges when this is needed	202
Wrappers need to be used	203
Systrace does not require changing of objects	204
If a Systrace policy is lost, the program will be granted unrestricted access to the system	205
Default permission is to deny all access	206
Permissions can be granted to specific files or directories	207
Allows using $HOME and other environment variables in policies	208
Systrace can make the security system very complex if having many groups	209
Can use privilege elevation to perform a system call as another user.	210

Can limit access to devices if they are devices in the filesystem	111
Network devices can be limited	112
Can not log actions which are denied by the kernel	113
Can limit and grand privileges to bind to specific ports	114

Can limit access to devices just as to files and directories.	211
Network devices can be limited.	212
Can log actions	213
Can not provide information about if the kernel then ultimately denied or allowed access	214
Can limit binding to specific ports	215
Use privilege elevation to allow binding	216

From these codes, some concepts can be clearly identified which are then grouped in categories.

SELinux	
Security is comprehensible and does not make use of SUD programs	
Privilege escalation can be used	101, 107
Does not make use of SUID programs	102, 109
Can not run software as another user	108
Can limit access to devices	111
Network devices can be limited	112, 114
Reference policy is a good start, but knowledge is required to adapt	
Reference policy can be expanded upon	104, 106, 110

Systrace	
System access can be restricted or granted	
Privileges can be restricted	201, 206
Permissions can be arbitrarily given	207, 208
Can limit access to devices	211
Can limit access to network	212, 215
Privilege elevation can be used, but can be difficult to get right	
Can use privilege elevation	210, 216
Can be difficult to elevate privileges	202, 203

Files needs changing to reflect security changes	103
Requires deeper knowledge	105
Limited means of traceability exist	
Can not log all actions	113

Configuration can be easy, but dangerous or complex for larger systems	
Does not require changing of objects when security changes	204
Dangerous to lose a policy file	205
Can become complex configuration	209
Traceability can be achieved for Systrace itself	
Some logging can be done	213, 214

This provides a good basis for arriving to a final theory which captures the fundamental details of each security system. Drawing from the categories and concepts from above, the following descriptions of SELinux and Systrace can be seen to capture the various concepts that appeared during the evaluation of the systems, based ultimately in the common system administration tasks which the evaluation was done based on.

> SELinux provides a comprehensible security infrastructure that can both restrict and grant access to files, sockets and devices without making use of SUID programs. The reference policy can provide a good start for new installations, but changing the security policies requires a deeper knowledge of SELinux and the system itself. Some limited means of traceability exist, but SELinux is unable to log all actions.

> Systrace provides a security system where access can be granted or restricted to files, sockets and devices. While privilege elevation can be used to grant access to some resources at a higher privilege level, such functionality can be difficult to implement. Configuration is easy, but can be dangerous and complex for larger installations. Some logging can be done to improve traceability in the system.

7. Discussion

The key difference between SELinux and Systrace seems to be their mode of operation, where SELinux, built around the LSM framework in the Linux kernel, works with type enforcement on files, sockets and other objects, whereas Systrace works on a strict system call basis. Both are built on top of the existing security models of the Linux and OpenBSD kernels, and as such neither is able to grant any permissions on top of what the user already has, but both can be used to limit overly broad permissions (such as SUID root programs) to the least privileges needed by a program.

Systrace does not contain any sense of different levels of authorisation for the users, and indeed seems built with the premise that an application should have the same permissions regardless of who is running it. In this way, Systrace is well places to protect server programs, but is more ill placed when it comes to protecting information in an enterprise, since two different users can not normally be granted different access permissions to files, unless special care is taken in the setup.

SELinux, through its use of type enforcement, but additionally through the possibility of using MLS, seems constructed from the onset to protect information in an enterprise setting, and the ability to affect also server programs or other programs running on a host machine is a side effect of this. This is further seen by the ability in SELinux to setup policies which allow users of one level to read files with information from levels below, and write information only at the same level or levels above.

Additionally, the level of protection that Systrace and SELinux offers to the whole system is different. SELinux is built on the principle that the entire system should be protected through the use of SELinux from the moment the system is started. Systrace does not contain any general protection for the entire system, but rather it is up to the administrator to setup and specifically activate the server programs or other programs that should be protected through the use of Systrace. Securing the whole system with Systrace involves a multi-step process which is both time consuming and prone to errors in the policy generation (Palmer, 2004).

Another key difference between the two security systems is that the policies in SELinux are comprehensive and enforced across the entire system, thus leaving the decision on access control up to the administrators, Systrace offers this capability to some extent but also makes it possible for a user herself to start a program specifically under Systrace protection to limit the authority of that program to be less than that of the running user (Palmer, 2004). This can be used to run user programs which interact with the network, and where one wishes to protect against malicious attacks caused by the malfunctioning of the program itself.

Specific for Systrace is that it can not reliably be used to deny permissions which a user normally has. For instance, if a Systrace policy is inserted called bin_ls to restrict the viewing of the /etc directory to members of the wheel group, a user could easily upload their own ls executable, /home/user/ls, whose Systrace policy home_user_ls would not match bin_ls and thus be allowed to execute without any limitation (Palmer 2004).

This can however be avoided, since the policies can be configured to be inherited from the parent process, meaning that the system administrator can restrict the user to a specific shell without possibility to change it, and put global policies in the policy for that shell. Care must be taken however to ensure that the user can not execute any commands without the use of that particular shell, such as via cron or other means.

Systrace and SELinux can be more beneficial when granting specific privileges to individual binaries, thereby restricting the use of suid and sgid bits on executables. The use of specific capabilities in this way is particularly interesting since it allows the system administrator to in more detail assign capabilities to specific software, such as granting the permission to bind to port 80 for a webserver, without at the same time giving it access to read any file in the filesystem which would previously have been the case using normal UNIX authentication. Systrace and SELinux can also be used to various degrees for privilege escalation. For SELinux specifically, privileges can be granted to a specific security domain (Red Hat 2008), thus getting rid of the requirement for SUID programs, whereas Systrace limits this to giving privileges to specific executables.

In the practical evaluation of the two systems, it emerged that Systrace was much better at restricting access than granting access, and while privilege elevation did exist, it was very difficult to implement correctly and required a lot of care to be taken, especially in order to make the system continue to be safe across upgrades. In the whole, this can probably be traced back to a difference in ideology behind Systrace and SELinux. Where SELinux was constructed in order to fulfill a specific security classification of a system, with a lot of effort put into information passing between different layers of the system, Systrace was created to be able to restrict privileges of a specific program, often limited to one or a handful of binaries.

8. Conclusion

Overall, it would seem that Systrace is useful and desired in many aspects that require absolute security but less flexibility: for instance in embedded devices, firewalls, proxy servers, authentication servers, etc, where users are not an issue, but the key point is to protect and secure specific server programs. The resulting description of Systrace seems correct and accurately reflect the experience during the evaluation of the systems.

SELinux, on the other hand, can provide the flexibility that is needed in large enterprises and in systems where more than two levels of authentication are needed, for instance in document management systems. The use of SELinux can provide very attractive solutions for ensuring the integrity and security of an enterprise information. The two systems therefor serve two different purposes which sometimes overlap, but in just as many cases provide solutions for entirely different quality priorities.

However, it should be noted that the resulting descriptions of both SELinux and Systrace does not take this into consideration. This is likely due to the fact that SELinux and Systrace was evaluated as if they fulfilled the same functionality. But as has been explained in the discussion, the two provides slightly different angles on security and have different aims as to what they should secure.

It is therefor not as easy as saying that SELinux should be used instead of Systrace or the other way around, but it depends entirely on the context, not only on the where, but also based on what should be protected. More research is needed here to come up with a better metrics for when SELinux and Systrace should be used and where they can provide the most useful enhancements to a running system. Such research needs to focus on one or the other though, and not try to contrast both to each other.

9. References

Allan, G. 2003. 'A critique of using grounded theory as a research method'. *Electronic Journal of Business Research Methods*, 2(1) 1-10.

Andreasson, O. 2005. Linux Packet Filtering and iptables, viewed 5 April 2009, <http://www.linuxtopia.org/Linux_Firewall_iptables/index.html>

Apache, 2009, Apache HTTP Server Version 2.0, viewed 2 April 2009, <http://httpd.apache.org/docs/2.0/>

Böhm, B. 2006. Linux Vserver Project, viewed 4 April 2009, <http://linux-vserver.org/Capabilities_and_Flags>

Fox, M., Giordano, J., Stotler, L., Thomas, A. 2008. 'SELinux and grsecurity: A Case Study Comparing Linux Security Kernel Enhancements'.

Frisch, A. 2009. [Personal communication] 15 April 2009.

Glaser, B. and Straus, A. 1967. *The Discovery of Grounded Theory*. Aldine, New York

Hansteen, P. 2007. *The Book of PF - A No-Nonsense Guide to the OpenBSD Firewall.* No Starch Press, San Francisco

Harris, G. 2008. Executable space protection, viewed 5 April 2009, <http://en.wikipedia.org/wiki/Executable_space_protection>

Herrb, M. 2007. 'Integrating X.org in OpenBSD'. *Proceedings of FOSDEM 2007.*

Love, P., Merilno J., Zimmerman, C., Reed, J., Weinstein, P. 2005. *Beginning Unix.* Wiley Publishing: New Jersey

Mayer, F., MacMillan, K., and Caplan, D. 2006. *SELinux by Example: Using Security Enhanced Linux.* Prentice Hall: Upper Saddle River

McCarty, B. 2005. *SELinux: NSA's Open Source Security Enhanced Linux*. O'Reilly Media: Sebastopol

OpenBSD. 2009. OpenBSD 4.4 Installation Guide, viewed 3 April 2009, <http://www.openbsd.org/faq/faq4.html>

Palmer, B. and Nazario, J. 2004, *Secure Architectures with OpenBSD,* Addison-Wesley Professional, New Jersey

Peikari, C. and Chuvakin, A. 2004. *Security Warrior*.

Red Hat. 2008. Red Hat Enterprise Linux Deployment Guide, viewed 2 April 2009, <http://www.redhat.com/docs/manuals/enterprise/RHEL-5-manual/>

Saltzer, J and Schroeder, M. 1975. 'The protection of information in computer systems'. *Proceedings of the IEEE* 63 (9): 1278–1308.

Smalley, S., Fraser, T., and Vance, C. 2008. Linux Security Modules: General Security Hooks for Linux, viewed 15 May 2009, Linux Kernel 2.6.26 Source Code.

Spengler, B. 2009. Lightbulbs are going off, viewed 15 May 2009, <http://www.grsecurity.net/lsm.php>

Stevens, R. 1992. *Advanced Programming in the UNIX Environment.* Addison-Wesley: Reading

Stuart, B. 2008. *Principles of operating systems: Design and Applications*. Cengage Learning, South Melbourne

Vaughn, G. 2002 *Maximum Security: A Hacker's Guide to Protecting Your Computer Systems and Networks.* SAMS Publishing, Ontario

Ward, B. 2004. *How Linux Works*. No Stach Press: San Francisco

Whittaker, J. 2003. 'Why secure applications are difficult to write'. *IEEE Security and Privacy* 1 (2): 81-83.

Wright, C., Cowan, C., Morris, J., Smalley, S. and Kroah-Hartman, G. 2002. 'Linux Security Modules: General Security Support for the Linux Kernel'. *Proceedings of the 11*th *USENIX Security Symposium.*