RSBAC - a framework for enhanced Linux system security

Marek Jawurek*
RWTH-Aachen

Abstract

Operating systems traditionally bring their own means of protection against any kind of threats. But often the system's security is based on an evolved composition of different projects and tools, historically grown with their operating system, ensuring and controlling system security. Therefore only rarely a coherent security solution is provided by the system. This existing structure of security suffers from different problems and inconsistencies, caused by its evolution, and often lacks universality due to bad or shortsighted design decisions in its history.

The RSBAC framework is focused on the elimination of these drawbacks in Linux kernels and has been designed this way from its start. It introduces a modular framework to allow fine-grained access control enabling system administrators to compose a system using a combination of the preferred security models. This paper presents the RSBAC approach and outlines RSBAC's concept of integration into the Linux kernel. Furthermore, this work presents similar solutions to kernel security and discusses the advantages and disadvantages of RSBAC to those other solutions.

1 Introduction

When Linux was developed in the early 1990s it inherited the standard security mechanisms from Unix. The built-in discretionary access control (DAC) for instance, which does not separate ownership of filesystem objects and the right to set permissions on these objects. Then the Linux capabilities (CAPS) were added. Capabilities that can be taken from or given to processes during execution. Eventually, access control lists (ACL) were introduced into the kernel. These features comprise the security features of Linux. Although these facilities may suffice for the home user, there are businesses and applications which need fine-grained control of system resources. This is where Rule Set Based Access Control (RSBAC) comes into play. The RSBAC kernel-extension was developed in 1997 by Amon Ott as his diploma work and is in stable production since 2000. It introduces a security framework designed to install different security models dynamically into a Linux system in order to control access to system resources.

1.1 This work's focus

Four major different approaches to Linux security exist: SELinux, grsecurity, LIDS and RSBAC. A direct comparison between these would be desirable but is due to the following reasons not feasible:

Classification The different approaches classify themselves differently (kernel patch, security framework, etc.).

Interpretation Each improvement to Linux security has a different understanding of a 'Secure Linux System'.

Concept Each approach can be conceptually different.

Combination Multiple approaches could be combinable in one system.

Because of these four points, and there are probably more, a direct comparison of SELinux, grsecurity, LIDS and RSBAC is impractical. However, the following questions can be deduced from the problems mentioned above:

1. Is RSBAC a good security framework?
2. How 'much' security can one achieve using RSBAC?
3. What is its main focus - does it aim at protection against special attack vectors?
4. Is it easily combinable with other approaches?

This leads to the main goal of this work: This work focuses on the design and implementation of RSBAC and the potential it has because of its superior concept. The advantages/disadvantages the RSBAC approach has over the other existing approaches are discussed where appropriate. Finally, this article examines the RSBAC framework with respect to the questions posed above.

*This paper was written as part of the conference seminar "dependable distributed systems" which was organized by the laboratory of dependable distributed systems at RWTH Aachen University during winter term 2005/2006.

1.2 Structure

Firstly, this work presents the terminology needed for the discussion of the RSBAC framework and other solutions. Secondly it gives a short introduction to the structure and functionality of RSBAC. Thirdly, other security enhancements for Linux are introduced. Then, the RSBAC framework is described in detail. Finally, in the last section, the RSBAC system is assessed using the previously developed questions and a conclusion about its usability is drawn.

1.3 Related Work

Linuxsecurity.com features a short introductory article on RSBAC followed by an interview with Amon Ott ([8]). The table at [13] gives a good overview of RSBAC/grsecurity/SELinux but only lists main attributes. An unpublished paper [16] deals with grsecurity and SELinux and a quite vivid discussion on RSBAC vs grsecurity can be found in the grsecurity Forums at [12].

1.4 Terminology

In order to discuss the benefits of RSBAC to the Linux kernel one must understand the different aspects and terms of security in computer systems.

Subject, Object: A subject in a computer system is either a user or processes acting in the name of the user. The subject's sensitive work usually includes access to objects of the computer system, resources of different kinds: files, pipes, network devices and others.

Security model: The security model is a formal description according to which system subjects (users/processes) are granted/denied access to system objects (files/devices). The security model usually defines rules or criteria in order to map real-world security measures to the computer system.

DAC- discretionary access control: In a DAC system the owner of an object can grant permissions to other users on that object. In contrast to this a MAC (mandatory access control) separates the ownership of an object and the rights to manipulate security related object attributes.

RBAC - Role Based Access Control: This model assigns a role to every user. The role can be a role mapped from real-life to the security model. According to the its role a user may or may not perform operations on targets. RBAC can include hierarchies of roles with inherited permissions.

ACL - Access Control Lists: ACL are extended access attributes for files and directories beyond the owner/group/other separation.

TPE - Trusted Path Execution: TPE restricts execution of programs to paths that are considered 'trusted', i.e. if the parent directory is owned by root and neither group, nor world-writable.

Pseudo-anonymity: The identity of a user is substituted by an alias in order to hide the identity but keep the possibility of logging the actions of this user.

Security policy: A security policy is a set of rules and constraints that control access of system subjects on system objects. The security policy implements and applies the security model to a system.

RSBAC framework: In this work the term 'RSBAC framework' refers mostly to the framework part of the RSBAC patches. The framework itself implements no security policy whereas the modules available for RSBAC offer functionality and do implement a security policy. Every module comes with an own default policy that is created upon first activation of the module and can afterwards be manipulated.

2 Introduction to the RSBAC framework

2.1 Goals of the RSBAC framework

RSBAC's main goal is the implementation of LaPadula's Generalized Framework for Access Control (GFAC) [14] without sacrificing efficiency, stability and flexibility of the Linux kernel. Additionally the following features are built-in:

- activation and deactivation of security modules in runtime.
- filesystem independent storage of security data.
- pseudo-anonymity for enhanced logging without the sacrifice of privacy.

2.2 Structure of the RSBAC framework

The RSBAC framework is an implementation of the GFAC which is divided into three separate facilities, the Access Decision Facility (ADF) composed of the security modules, the Access Enforcement Facility (AEF) and an Access Control Information (ACI) part (figure 1).

Figure 1 illustrates an access request: Every time a subject attempts to access system objects in a security relevant manner the AEF is invoked in the kernel. It then asks the

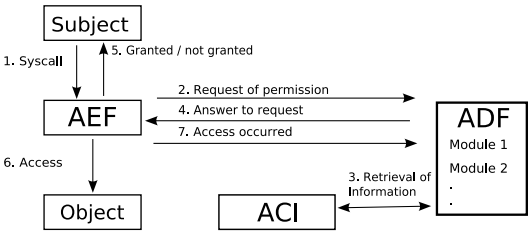

Figure 1. Structure of the RSBAC framework and decision-making process

ADF for clearance. The ADF decides whether access is granted or denied on the basis of the following data:

- The object, subject and the request of the system access in question.
- Security data stored in the system for those entities (ACI).
- The security modules activated at this time.

The answer to this request is returned to and enforced by the AEF. After a successful access the ADF is informed by the AEF in order to eventually update security data.

The separation of ADF and AEF yields several benefits:

- Only the AEF must be implemented into the kernel where security relevant syscalls appear. The ADF is a separate part that communicates with the AEF but does not interfere with the rest of the kernel.
- Only the AEF must be adapted to the machine's architecture. The ADF is a system independent part of the framework.

When an ADF gets a request for system resources, all active security modules' policies are checked for their assessment. Answers of the security models comprise 'granted','not granted', 'don't care' and 'not defined' and are combined to calculate the final decision. Security modules can be switched on/off by the Security Officer (see section 2.3) during runtime.

2.3 Separation of Duties

A feature of the Functional Control Model (LaPadula) , the separation of a system administrator and a security officer, is directly implemented into the RSBAC framework. Initially a general security officer, a user with a special UID, is introduced with the rights to control the framework and its modules. But furthermore, each module can decide on its own and change who the security officer for it is and what his rights, regarding this module, are.

Due to this separation the root user of an RSBAC-enhanced Linux system can be restricted to limited control of the machine and the data stored within. This lowers the abilities of the root user and therefore can also limit damage in case of intrusion into a process running with root privileges.

2.4 Security module data storage

Security models usually need to store data on subjects/objects or requests. Different approaches are available to store this data exist:

- It could be loaded into the system by the system administrator after the boot process but this leaves the system unprotected up to this moment.
- Storing the data on a separate partition would be possible, but the partition holding the information must be connectable to the partition whose objects are described by the information.
- The data could as well be stored in a special place designated by the filesystem driver in the same partition. This approach, however, would demand changes to the filesystem drivers that are used on the RSBAC system.

In order to circumvent these problems the security modules' data is stored in a special directory 'rsbac.dat' on every mounted filesystem. This way it survives a reboot of the machine and the storage is filesystem independent. Access to these directories is restricted to the kernel and the security modules themselves. However, in case the system is booted with another not RSBAC-secured kernel, these directories are exposed and data stored within is accessible by usual means.

2.5 Available security modules for RSBAC

At the moment there are several security modules available that implement security models: RC (Role Compatibility), ACL, MAC, CAP (Linux capabilities), JAIL (alternative to chroot), RES (Linux resources limitation), FF (File Flags) and PM(Simone Fischer-Hübner's Privacy Model). Additionally there is a PaX support module and a Dazuko File Access Control [4] module that allows plugging-in a compatible virus scanner for on-access virus-scanning.

3 Other security enhancements for the Linux kernel

Since the end of the 90s several other security enhancements have been developed for the Linux kernel. These are the still active projects:

- SELinux developed by the National Security Association (NSA),
- grsecurity
- and LIDS both from the open source community.

The following subsections give a short introduction to the three competitors of RSBAC and to PaX, a security patch aimed at preventing buffer overflows and similar attacks.

3.1 SELinux

SELinux is probably the most famous extension to Linux security due to its origins, the National Security Association. For some its origins are reason enough not to use it. For others this fact supports the idea that it is 'really' secure because it is probably used by the NSA itself. However, question 22 from the SELinux FAQ points out:

> 'Does NSA have plans to use it internally?'
>
> 'For obvious reasons, NSA does not comment on operational uses.'

The first release was made public in Dec 22, 2000 and released under the GPL. Since then it has been updated regularly several times a year. SELinux is an implementation of the FLASK [19] architecture which evolved in cooperation with the University of Utah from older system implementations made by the NSA and the Security Computing Cooperation. SELinux offers the enforcement of Mandatory Access Controls (MAC), Role Based Access Control (RBAC) and Type Enforcement (TE) [15]. Type Enforcement assigns domain attributes to subjects on one side and types to objects on the other side. A lookup mechanism determines whether a subject has the right to conduct its action on a specified object. The Type Enforcement approach separates functional parts of the system that do not need interaction, i.e. even if a process is compromised the attacker will only be able to access objects accessible by this process. SELinux uses the Linux Security Module (LSM) hooks in the kernel to implement its policy. For some time there was uncertainty about a patent on the 'Type Enforcement' technology held by the Security Computing Cooperation but it has expired now.

3.2 grsecurity

grsecurity is a combination of patches, firstly released in 2001 and has been in development since then. The latest version (2.1.6) was released on June 14 2005, for kernels 2.4.31/2.6.11.12. grsecurity's approach is named 'detection, prevention and containment' and aims at the 'avoid, identify, fix' mentality employed by system administrators, according to grsecurity. Fixing software bug by bug is not an attainable solution as one can never say when the software is finally secure. Furthermore it is a 'rat race ' (mentioned in the presentation available on [18]) where programmers have to fix bugs that others have already discovered and, potentially, misused.

In order to give system administrators a head start the grsecurity solution involves auditing and logging of the systems processes to detect intrusion (detection).

Prevention is attained by: The PaX [10] mechanism prevents buffer overflows which pose a major security threat. An improved chroot functionality enables better process separation than the original chroot. 'OpenBSD randomness features' introduce randomness into system behavior, i.e. IP IDs or PIDs, to name a few.

grsecurity's containment consists of RBAC/ACL and Trusted Path Execution (TPE) among other means.

3.3 LIDS

The development of the Linux Intrusion Detection System [7] was started in October 1999 by Huagang Xie. Although it is named an intrusion detection system it actually is much more than that. The kernel built-in port-scan-detection facility of LIDS does detect scans and reports them to logs or via mail, but LIDS does also include ACL facilities for files/devices and Linux capabilities. The usual procedure on a LIDS enhanced system is to boot with full capabilities for the root user and processes until the system is 'sealed' by the root user. From then on only priorly defined capabilities remain accessible for the root user and processes. Additionally the LIDS approach allows reactivating capabilities later on. Therefore a secure access to the security system is available via a password secured 'LIDS free session'. This additional level of system administration is comparable to the RSBAC security officer but any user who knows the password can access the 'LIDS free session', thus becoming a 'security officer' in LIDS.

3.4 PaX - Page Execution

Because PaX is integrated in grsecurity and combinable with RSBAC and because it plays a vital role in securing a system in both approaches, it is listed here as well. It provides a distinction between pages of memory that are executable and others that are writable (data pages) in order to fight the problem of buffer overflows. The PaX kernel extension was initially released in October 2000 but its main developer is still unknown. Meanwhile, its development has been ceased and passed over to Brad Spengler of grsecurity. On machines without the NX (no execution) bit for memory

pages PaX ensures that data pages are never executed. Furthermore, pages can only transition from code pages to data pages in order to protect the system from on-the-fly generated code as often introduced by buffer overflows or similar attacks. After a wipe of data pages, they can be used for code again. Additionally PaX employs address space layout randomization (ASLR) making it harder for attackers to use return to libc attacks. But PaX poses problems for programs that create code in runtime, like Java for instance, and prevents normal program behaviour. These unwanted side effects can be circumvented by excepting an executable file from PaX protection.

4 RSBAC in detail

As previously mentioned RSBAC is a framework that allows to implement several different security policies and to activate and deactivate them during runtime. Therefore the success of the framework (w.r.t. acceptance and usage) also depends on the ease of use for security module developers (apart from its maintainability for system administrators). This section deals with the details of which and how subjects, objects and access methods are exported to the security modules, and how security module data is handled and stored.

4.1 Security hooks in the Linux kernel

In order for the framework to allow security policies to be as fine-grained as possibly wanted by a programmer it has to offer hooks in the kernel code in every imaginable place that might be of interest to the module programmer, i.e. that is security relevant. The number of embedded hooks does decrease performance but this is a tradeoff an administrator might be willing to accept. If, in contrast, a certain security policy could not be implemented because the necessary hooks are not in place the administrator would more likely switch the security framework instead of tinkering with the kernel code himself, possibly introducing bugs or, even worse, security issues.

LSM, the Linux Security Modules kernel enhancement, does export hooks for security modules to implement a security solution since kernel 2.4. The development of LSM is a response to a presentation of SELinux at the 2.5 Linux Kernel Summit [1].

LSM offers about 130 hooks to security modules. A security module can register functions with the hooks that are called if program execution triggers a hook. However, LSM does not allow multiple functions to be registered with one hook. If one wants multiple modules in LSM the combination has to be handled by the 'main' module. Brad Spengler of grsecurity and Amon Ott, the developer of RSBAC share a common opinion, they do not use and do not like the LSM hooks. They claim that LSM hooks are not general enough, that they only include access control and no auditing possibilities, lessen default security due to openly exposed hooks and according to Amon Ott the way LSM stores security attributes (additional fields in kernel structs) endangers kernel stability. Both developers express doubts about the motives that led to the introduction of LSM into the kernel. The fact that LSM was introduced as an answer to SELinux which was developed by the NSA, could lead to several interesting conspiracy theories. Further information on their arguments can be found at [5] under 'LSM' and at [11] in the documentation under 'Why RSBAC does not use LSM'.

RSBAC follows a different approach. All security relevant syscalls (a listing is available at [6]) are intercepted and the information about the syscall, the request type, is passed along with the subject and the object information to the decision function provided by every activated security module:

Decision function: This is the interface for a security module to implement its policy. Information on the request, the access target, the access subject and attributes associated with the target are passed along to the function. Based on this information and the security policy to be implemented by the module the decision is returned. It is either 'granted','not granted', 'don't care' or 'undefined' (undefined is an exceptional error and is not supposed to happen). A comprehensive list of targets and request types available in RSBAC can be found at [9].

Notification function: The notification function is called by the AEF after a granted request has been completed. Using this, a module can either log the access or update its information. Especially if new objects have been created, i.e. files, information related to these can be manipulated or created accordingly.

Overwrite decision function: Offers the module the possibility to order a secure delete (overwrite the file's contents). It is invoked whenever a file is about to be deleted.

Mount/Unmount functions: Called when a filesystem is mounted or unmounted. Since all security related data is stored in the inaccessible RSBAC directories on a per mounted filesystem basis, a security module might want to update its information accordingly if mount/unmount operations take place.

RSBAC administration functions: These functions are used by the RSBAC daemon to order a security module to write its security data to disk or to issue consistency checks.

In addition to these functions a module can register its own functions as syscalls. This way a security module can be administrated from user space, so that the implemented security policy can be controlled by a security officer.

4.2 Data Storage and Memory Allocation

In order to pursue a sensible security policy, modules have to store information about subjects/objects or policy specific data. To help module developers overcome a steep development curve, the RSBAC framework offers facilities that do the job. Lists, and lists of sublists, are provided by the RSBAC framework for security module data. By using these lists the programmer does not have to care about memory allocation, the methods to manipulate these lists or store them on disk, because all this is managed by the RSBAC framework. Additionally, in contrast to the LSM approach, dangerous pointer handling is concentrated in one point - in the framework itself. As well as data handling, memory management can bind development time that could be better spent on the security policy itself. Thus, the RSBAC framework does also offer a few methods for memory management. More information on these facilities can be found in the documentation at [11].

4.3 Combination of modules

Whenever the AEF asks the ADF for a decision about a request all active modules at that moment are queried. Their answers are then combined with the symmetrical operator defined by figure 2.

operand 1	operand 2	result
granted	granted	granted
granted	don't care	granted
not granted	granted	not granted
not granted	don't care	not granted
undefined	*	undefined

Figure 2. Security module decision function combination

This approach offers much flexibility when combining security modules but does also require a good understanding of the security models implemented by all modules. Models can overlap in their capability of expression or even contradict themselves. One can switch modules off at runtime (as a security officer only) or switch a module to the so-called 'Softmode'. Both features are available if compiled into the kernel. In 'Softmode' the security officer can debug the security models' answers one at a time and eliminate problems, because decisions are made and logged but not enforced.

The next subsections explain the 'Auth' and the 'User-management' module to give the reader an insight into functionality available for the RSBAC framework.

4.4 Auth Module

The Auth module restricts the UID a process may acquire using 'setuid'. A process may be given a list of UIDs it can change to or the general permission to change to any ID. The list of available UIDs is called the 'capability set' (not the same as Linux capabilities) and it is inherited from the executable file of a process.

4.5 User-Management Module

This module replaces the usual way of administrating user data in the system. Instead of the '/etc/shadow' file which holds authentication information the module stores and authenticates users. All programs that rely on PAM authentication can be used with the User-Management module by default. The module is simply installed as a PAM authentication method. If user-management is subject to the system's security policy, i.e. if it is not controlled by other means (Kerberos, LDAP), then this module is a natural consequence of RSBAC usage on a system.

5 Assessment of the RSBAC framework

Up to now, the reader gained an insight into the RSBAC framework and its potential. So in this section I try to answer the questions posed in section 1.1 and make an assessment of the RSBAC framework.

5.1 Is RSBAC a good framework for Linux security?

Generally speaking one could say a framework is an entity embedding something else. A framework does not provide any functionality by itself, instead it serves as a surrounding entity providing support to its content. In the context of a computer system the framework would be some software embedding other software and providing support in terms of interfaces. Assuming this interpretation is valid one can assess RSBAC:

Firstly, although the RSBAC developers provide some security modules with the framework they are not an essential part of the framework. One could discard all predefined modules and implement ones own modules. Therefore, strictly speaking, RSBAC does not have to provide functionality by itself. Secondly, RSBAC intercepts all security relevant syscalls and provides an interface that security modules can use and which is well documented. This

empowers security module developers to implement a security model quickly without having to mess with the kernel itself. Thirdly, RSBAC provides certain functions which ease the burden for the developer implementing his model. Memory allocation and data storage in memory and replication to disk can be handled by the framework as well.

Summing it up, RSBAC is a good framework.

5.2 How secure can an RSBAC-secured system be?

As in 'Behind every successful man is a strong woman' one could say 'Behind every successful security solution there is a good security expert'. The framework itself does not provide any security to a system at all. The power of RSBAC reveals itself through the potential it has in the hands of an expert. Because of the sheer complexity of RSBAC one can create any security model imaginable with relatively low effort. Multiple security modules can be combined delivering a combined security policy to the system. However, in the hands of a novice this complexity can be difficult to understand.

The security of an RSBAC secured system could also be limited by the framework itself. If the framework poses security risks itself it would render the security modules useless. Although, RSBAC has been rated stable by its developers in the year 2000 this does not allow any deductions on the quality of its implementation. However, there exists a product offered by Cyberguard [3] that officially uses RSBAC and received a EAL4 certification (common criteria [2]). RSBAC itself has not been verified yet.

So in spite of its enormous potential and its commercial use RSBAC's approach to security is situated quite deeply inside of the system. The idea is to limit the access of processes and users to the system in order to prevent greater damage in the event of intrusion and takeover of a process context. One should evaluate the additional usage of security mechanisms that protect the system more proactively (Firewalls,Intrusion-Detection-Systems). LIDS for example, integrates such mechanisms, it can detect port scans.

But there is a difference between being a complete security solution and a security framework, and while the first can be constricting, the latter can be too loose, there will always be a tradeoff.

5.3 What does RSBAC offer against known attack vectors?

RSBAC can only protect confidentiality and integrity of a system. Availability is not ensured per se, but can be a side effect of the separation of services on the system. As mentioned in the previous subsection, RSBAC security is positioned relatively deep inside the system. The advantage of this approach:

- RSBAC is equally capable of defending the system from external (network) attacks as from internal (trusted users) attacks. If the security policy restricts capabilities of local users and the usable system services the likelihood of misuse is minimized.
- The motive behind an attack on a computer system is either to gain access to the system or to stop its work. RSBAC can not protect a system from denial-of-service attacks or similar. But it can ensure that once inside the system the attacker has only little to take or to damage. If the security policy is tight enough, so that processes have only access to data that they need (e.g. FTP to user directories, Web-server to server pages, DNS to domain information). One can rule out that a compromise of this service will endanger other data as well. And as all attack vectors aim at using a processes context (social engineering as well as emails with virus attachments or buffer overflows in network services) one can, using RSBAC, divide the system into small compartments in order to confine the potential damage of all attack vectors.

5.4 Is RSBAC combinable with other security enhancements?

Generally speaking : 'Of course !', but one has to differentiate the means and the security policy implemented *by* the means. Because RSBAC is such a general approach one could implement the security policy of SELinux, in this respect SELinux and RSBAC are combinable. Even more interestingly, an SELinux module can be found on the todo-list of the RSBAC website, with a helper script to load SELinux configurations into this SELinux RSBAC module. This would enable SELinux users to migrate easily to RSBAC if wanted or needed. Additionally the PaX patches can be used together with the RSBAC framework, there even is a module that allows to store PaX relevant information using RSBAC's data storage facilities so that executables do not have to be marked for PaX. Binaries can be updated without losing the flags and security checkers do not produce false alarms when a flag on a binary is changed.

6 Conclusion

RSBAC is one of the oldest but probably the most farsighted approach to enhanced security in a Linux system. It does not limit the user to a predefined security policy but instead provides a good basis to start from and thus delivers high flexibility in implementing a custom, very fine-grained security policy. The downside of this flexibility is,

of course, the need for expertise and experience from the system administrator. The RSBAC framework does not provide a default policy like grsecurity and SELinux. However the modules shipped with the framework do have default security policies that do provide some security upfront. Since usually the framework will not be used on its own, without the available modules, one could say RSBAC provides a default policy. Nevertheless usage of RSBAC can only be recommended to the experienced system administrator who needs the flexibility and is willing to invest the time needed to master RSBAC and the functionality provided by its modules.

The fact that RSBAC is more universal and consistent than other approaches does not necessarily mean it is better than others, but it has the potential to be better. In the end, it is the scenario, the available time frame, the administrator's personal opinion and the security policy chosen by the administrator that predetermine the choice of an approach.

References

[1] Presentation of lsm at linux kernel summit, 2001. http://lsm.immunix.org/docs/overview/linuxsecuritymodule.html.

[2] Common criteria website, 2005. http://www.commoncriteriaportal.org.

[3] Cyberguard website, 2005. http://www.cyberguard.com.

[4] Dazuko file access control interface, 2005. http://www.dazuko.org/.

[5] Grsecurity website, 2005. http://www.grsecurity.net.

[6] Intercepted syscalls in rsbac, 2005. http://svn.rsbac.mprivacy-update.de/viewsvn.php?project=rsbac1&path=/linux-kernel/2.4/branches/linux-rsbac-1.2/Documentation/rsbac/Interceptions-2.4.

[7] Lids project, 2005. http://www.lids.org.

[8] Linuxsecurity.com, 2005. http://www.linuxsecurity.com/content/view/117460/49/.

[9] List of rsbac targets and request types, 2005. http://rsbac.org/documentation/targets_and_requests.

[10] Pax website, 2005. http://pax.grsecurity.net.

[11] Rsbac website, 2005. http://www.rsbac.org.

[12] Rsbac/grsecurity discussion in grsecurity forum, 2005. http://forums.grsecurity.net/viewtopic.php?p=3076&sid=62bcd6cc0e7e18c6c2d42b6d742f2eb4.

[13] Rsbac/grsecurity/selinux comparison table, 2005. http://gentoo-wiki.com/Access_Control_Comparison_Table.

[14] M. Abrams, K. Eggers, L. L. Padula, and I. Olson. A generalized framework for access control: An informal description. October 1990.

[15] L. Badger, D. Sterne, D. Sherman, K. Walker, and S. Haighighat. Practical domain and type enforcement for unix. pages 66–77, May 1995.

[16] M. Fox, J. Giordano, L. Stotler, and A. Thomas. Selinux and grsecurity: A case study comparing linux security kernel enhancements. http://www.cs.virginia.edu/ jcg8f/GrsecuritySELinuxCaseStudy.pdf.

[17] A. Ott. Regelsatz-basierte zugriffskontrolle nach dem 'generalized framework for access control'- ansatz am beispiel linux. Master's thesis, November 1997.

[18] B. Spengler. Detection, prevention, and containment: A study of grsecurity. Libres Software Meeting, 2002.

[19] R. Sripada and T. F. Keefe. Version management in the star mls database system. August 1998.