Proceedings of the International MultiConference of Engineers and Computer Scientists 2008 Vol I
IMECS 2008, 19-21 March, 2008, Hong Kong

A Comprehensive Analysis of MAC Enhancements for Leveraging Distributed MAC

Shahbaz khan[1], Muhammad Amin[2], Muhammad Nauman[3], Tamleek Ali[4]

***Abstract*—Increased dependability of users, businesses and government on computing systems for research and computations on sensitive data raises serious issues regarding security. Operating systems conventionally provide Discretionary Access Control (DAC, superuser logic) to maintain security. Malicious users and applications are major threats to organizations and DAC seriously fails to address the required level of security. Former research has proven that enforcement of Mandatory Access Control (MAC, security labels/contexts for subjects and objects) restricts the malicious agents to their own domain, thus eliminating risk of damage it can cause to rest of the system and data.**

MAC implementations depend upon access control mechanisms. These mechanisms are added to operating systems as kernel enhancements. There are four prominent kernel enhancements: RSBAC, SELinux, GRSecurity and AppArmor. This paper assesses and reviews the differences in their implementation and enforcement techniques to achieve their similar goals and then compares their effectiveness. We give special attention to the implementation details and network labeling, in order to evaluate and understand this underlying layer of security. Our research objective is to propose a framework for Distributed MAC (DMAC), which depends on a clear perception of the architecture, internals and features of these enhancements.

Keywords: Security, Mandatory Access Control, Operating Systems, Network Security

I. Introduction

Nowadays, it is a common practice to manage and exchange data electronically. Unfortunately, a significant amount of presumably secure IT systems have proved to provide deficient security measures. These systems have been compromized in the past and they still have vulnerabilities, which are a serious threat to the information societies. Efforts made in the past suffer from the flawed assumption that security can adequately be provided in application (user) space without certain security support and features in the operating system [1].

Most operating systems today are based around discretionary access control (DAC) mechanisms. These mechanisms allow the 'owner' of an object to allow access to the object on their discretion. These decisions by individuals may be in contradiction to the organizational policy. Moreover, rights are assigned based on 'owner', 'group' and 'world' previlidges. This course-grained security may allow malicious software access to many critical components of the system once an individual user's critical information has been compromised [1].

Mandatory Access Control (MAC) mechanisms address these deficiencies. MAC enforces a system-wide security, which is a manifestation of the organization's security policy. In simple words MAC can be defined as "the definition of policy logic and the assignment of security attributes or labels to all resources of a system, tightly controlled by a system security administrator [2]." From another perspective we can also say that MAC is the dynamic or runtime secure information flow, although, stable support is currently available on operating system level i.e. considering processes and files.

Currently MAC controls are enforced at a single system i.e. a computer node. There are security mechanisms like Kerberos [3], LDAP [4] and NIS [5], that maintain centralized access control logic for an organization's complete system, but they are limited by the flexibility and granularity of MAC. There is a need for bringing coherence between these two mechanisms. Extending current MAC, without considering such authorization and authentication servers, to enforce security logic on networked resources is also a possibility. Although it would make a neat and consistent development cycle and end result but will result in a great loss in current security infrastructures. This issue is going to be considered in our future work and in this paper our focus is on which kernel enhancement is most suitable with respect to design, implementation, maintenance and future support. The enhancements analysed in this paper are RSBAC [6], SELinux [7], GrSecurity [8] and AppArmor [9]. This evaluation will enable us to select an enhancement, which is appropriate for leveraging DMAC[10]. The overall approach is kept as general as possible but this paper aims to cover all those aspects, which are necessary for our future study. We have left out benchmarks and syntactical analysis of policies because they are not significant for this evaluation.

The rest of the paper is organized as follows: Section II gives a brief concept of access control. It is followed by an overview of Linux Security Module (LSM) framework, which is the standard facilitation mechanism for MAC enhancements in the Linux kernel. In section III, we will give an overview of frameworks and architectures of each kernel enhancements. Section IV will present detailed analysis of their implementations. Section VI sums up the assessments. Future work is given in section VII. Section VIII concludes this study with the possibilities for the next phase of this research.

[1]S. Khan is with Institute of Management Sciences, Peshawar, Pakistan. (e-mail:shazalive@gmail.com phone no. +92 91 0300 5944647)

[2]M. Amin is with Institute of Management Sciences, Peshawar, Pakistan. (e-mail:clickforamin@gmail.com)

[3]M. Nauman is with Institute of Management Sciences, Peshawar, Pakistan. (e-mail:recluze@gmail.com)

[4]T. Ali is with International Islamic University, Islamabad, Pakistan and on leave from Institute of Management Sciences, Peshawar, Pakistan. (e-mail:tamleek@iiu.edu.pk)

II. Related Concepts

A. Reference Monitor

A reference monitor implemented in an operating system. It sets apart discrete resources *objects* into passive entities such as files and active entities such as processes into *subjects*. The *reference validation mechanism* would then validate access between subjects and objects by applying a security policy. Objects can be files, directories, named pipes, symbolic links, devices, interprocess communication data, system control data, users and processes, while subjects are processes. [11]

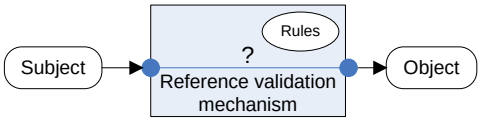

Fig. 1. Reference Monitor Concept

B. Linux Security Module

Earlier projects resorted to system call interposition to control kernel operations, which had serious limitations [12]. Secondly, there was lack for a standard mechanism and enhancements hooked to the kernel in a manner that only suited their own requirements. In addition, creating effective security module was a problematic task because the kernel had no infrastructure to mediate access of the security module to kernel objects.

To facilitate these issues the LSM [13][14] project was developed as a lightweight, general purpose, access control framework for the Linux kernel. It enables many different access control models to be implemented as loadable kernel modules. Figure 2 is a visual representation. The LSM kernel modifies the kernel in five primary ways.

1) It adds opaque security fields as `void*` pointers, which enable security modules to associate security information with kernel objects.
2) It inserts calls to security hook functions at various points in kernel to mediate access to kernel objects.
3) It adds generic security system calls to implement new calls for security aware applications.
4) It provides functions to allow kernel modules to register and unregister themselves as security modules.
5) It moves most of the capability logic into an optional security module.

LSM has an important feature that is module stacking but it pushes most of the work to modules themselves. Our future study may require this feature if we are to use integrity measurements [15] for remote attestation.

LSM kernel control flow ,while assesing access control rights, first checks the DAC decision and if it is positive then LSM hook gives control to hook implementation, which mediates access between kernel data structures and the decision module, else system call is returned an error code.

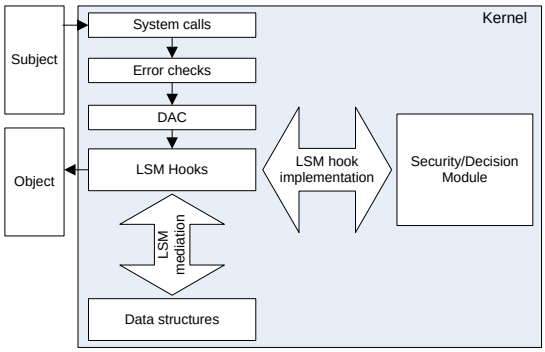

Fig. 2. LSM Framework

III. Overview of Frameworks and Architectures Behind These Enahancements

In this section we give a brief overview of these enhancements. In the next section we will discuss them in details. Table I shows the general comparisons of these enhancements.

A. RSBAC and GFAC

Rule Set Based Access Control [6] is an open source security extension for the Linux type kernels based on the Generalized Framework for Access Control (GFAC) [16].

The GFAC framework targeted the integration of multiple policy components. This was achieved by modularizing the framework into access control enforcement, access control decision and access control information (security attributes) facilities. Access Decision Facility (ADF) implements the MAC security policies and a metapolicy to decide whether process' requests satisfy those policies. Access Enforcement Facility (AEF) uses the ADF decisions to enforce the operations at system call level.

RSBAC [16] is a direct implementation of the GFAC framework. It has been extended to supports more object types, includes generic list management and network access control, contains several additional security models and supports runtime registration of decision modules and system calls for their administration. The components ADF, AEF and ACI are hard-linked into the kernel.

B. SELinux and Flask

SELinux's origins are found in the research of an access control framework based on Distributed Trusted Operating System (DTOS) [17] and Distributed Trusted MACH (DTMACH)[18], two MACH [19] based kernels. The resulting framework was named Flask [20], when ported to the Fluke [18] operating system. The Flask security framework is similar to GFAC where it separates the security policy decision from the enforcement mechanism.

The Security Server (SS), being the decision module, separates the policy logic with well-defined interfaces for obtaining security policy decisions. The object managers are the enforcers for their specific objects (file system, process management, IPC, sockets, e.t.c.). There is an Access Vector Cache (AVC), which is present inside the object managers.

	RSBAC	**SELinux**	**GrSecurity**	**AppArmor**
Origin	GFAC	DTOS, DTMACH, Fluke	Openwall	Immunix SubDomain
Framework	GFAC	FLASK	N/A	CHROOT Concept
Distributions	Hardened Gentoo, Mandriva, T2, Alt Linux	Hardened Gentoo, Fedora, RH Linux, Debian	Hardened Gentoo	Suse
Contributors	Five [6]	Many [7]	Brad Spendler	Suse

TABLE I
GENERAL COMPARISONS

It caches access decisions for subsequent use by the object managers.

Flask being implemented on Fluke, micro kernel architecture, had to be adjusted with the monolithic design of Linux. Its implementation was a huge patch to the bare kernel, explicitly extending a lot of data structures it used internally to handle the operations concerning files, communication and other I/O. The object managers were ported as additional functionalities implemented inside the kernel subsystems. Security server is also part of the kernel now, which was originally a server in Flask.

After LSM became the standard access control farmework for the Linux kernel, SELinux adapted. Now SS is located as a LSM compliant module [21] and kernel subsystems use the LSM features to communicate with the security server. AVC has also become centralized but an interface is provided in libselinux for user space object managers like the X server. The LSM architecture has made the SELinux design to become a bit scattered due to LSM's coherence with the kernel. Internal components, in addtion to SS, enforcement at kernel subsystems and AVC are network interface table (mapping of network interfaces to security contexts), network notification code (keep libselinux consistent with kernel module), SELinux pseudo file system (SS's API to processes, selinuxfs) and hook functions' implementation. Figure 2 is a visual representation of this enhancement.

User space object managers are suported by *libselinux*. X server and Postgresql are already security enhanced servers. Policy Management Server (PMS), a user space feature lately added, can also be extended to a user space SS, which will reduce burden on kernel SS [22]. Currently PMS only provides policy management in an object oriented manner but it also aims at handling a networked policy.

C. GrSecurity

GrSecurity is a suite of solutions that address different shortcomings of the operating system security. Thus, it can be stated that it does not implement any framework. It is a set of patches knitted in harmony with each other. It was originally a port of the security features of Openwall [23] but with time it has grown.

Philosophy behind GrSeecurity is that currently operating systems avoid, identify and fix software bugs. The history of software development has proven that there are always going to be bugs in the software no matter what security measures are taken. GrSecurity as a solution detects, prevents and contains. Detection of vulnerabilities is available through auditing and logging of attacks, prevention by PaX (address space protection) with a combination of additional techniques. Containment is enabled through access control lists. The ACL feature incorporates MAC into this enhancement and encourages a modular design for future extensions.

GrSecurity [8] being a knit of patches does not have a well defined kernel security module because it is inserted into various parts of the kernel to address the lack of current security mechanisms. Its ACL engine acts as the decision-making facility and stores the ACL policy along with RBAC extensions.

D. AppArmor and SubDomain

AppArmor previously know as SubDomain [24], is yet another implementation, which is not derived from a formal framework. It utilizes the concept of chroot jail with access control functionality enabled through program profiles. It provides a similar containment value to chroot, but without the need to physically move the application and its required resources into a separate container. AppArmor access control concentrates on file system resources, which means process transitions are not consider adequately.

AppArmor has an LSM compliant security module which is similar to SELinux in this regard. It uses LSM to communicate with *securityfs*, where it stores its security attributes and configurations. It uses profiles for the containment.

Seperating decision module from enforcement mechanism enables flexibility in the security by ensuring that the enforcing subsystems always have a consistent view of policy decisions irrespective of how these decisions are made or how they may change over time. This approach can easily allow application transparency and implementation of new policy models. Rsbac and Flask truely follows this approach, while AppArmor and GrSecurity to some extent.

IV. IMPLEMENTATIONAL ANALYSIS

We formulated the following criteria based on the general needs of a MAC security implementation and ISO10181-3. An access decision technique, an access enforcement technique, *security attributes handling* to leverage decision enforcement, and a set of *policy models/modules* for policy granularity and flexibility. The overall security modules have been discussed in the previous section. Figure 4 & 5 are visual representations of the security modules and flow control of RSBAC and SELinux respectively. Table II, a modified version of [25], gives a detailed comparison of these implementations.

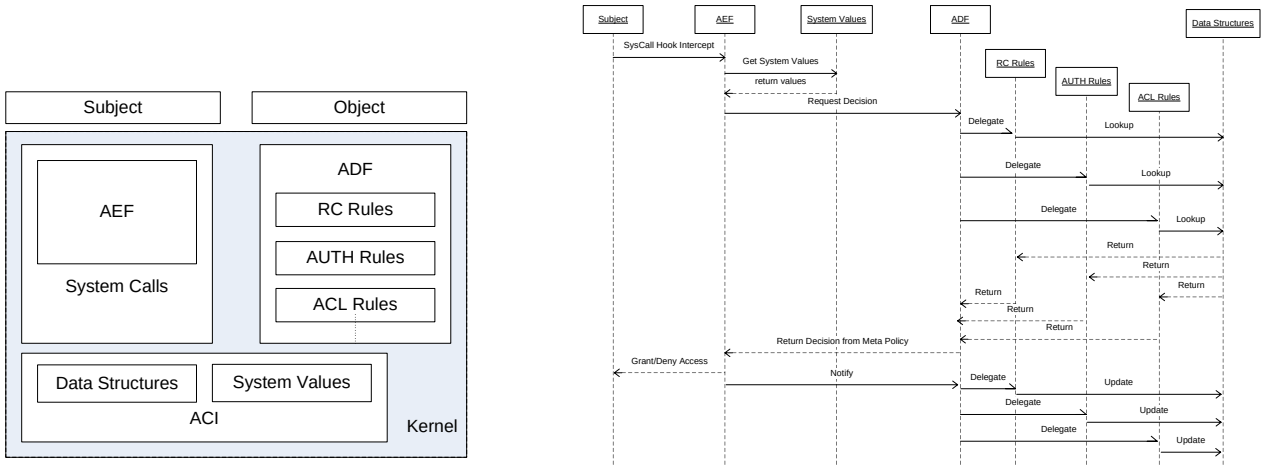

Fig. 3. RSBAC Framework and Control Flow

A. Access Decision Mechanism

RSBAC's decision facility is ADF [16]. With every *security relevant* system call, called by the subject to access an object, the AEF is invoked by intercepting the call. AEF compiles a request and sends it to the ADF for decision using an abstract function with relevant request type. The request has certain parameters; desired type of functionality is known from the request type, identification of the subjects and *possibly* some attributes of the object. These values are extracted by the AEF from the ACI against the subject and object.

Security policies are evaluated with respect to the policy rules for the request type by the ADF. ADF consults ACI for policy rules. When these rules are determined the metapolicy is evaluated to make the decision. Metapolicy uses the set of results from different security policies present to build the final result. Metapolicy is the union of the set of decision modules present in ADF.

ADF can be split into two parts: the main part, which handles the general work and the second part being the installed modules. The general work mainly consists of handling requests from AEF, dispatching it to the modules and performing generic logging. The modules compute the actual decisions against its policies and security attributes of the subject and object.

SS is the decision facility for SELinux. It uses security labels to compute an access decision comparing an object's label with the authorization level of a subject that attempts to access the object. To handle this objective [26] there are two data types to map an object with its access rights. The *security context* is a string representation of the object's rights which is used by the SS to compute its decision against the policy. A *security identifier* (SID), an integer value, is mapped to the security context and is passed to the kernel subsystem for labeling of objects. The kernel subsystems pass SIDs to the SS so that it performs the translation to security context and computes the decision.

The ACL core of GrSecurity is the decision making facility, which gets requests from its enforcement mechanism. These requests contain identity of the requesting subject and the object being accessed. It utilizes hooks similar to what LSM provides, but are much more in number as it uses access control in multiple layers.

AppArmor's decision facility is similar to SELinux as it is also LSM compliant. AppArmor is itself the core module [27], which handles most of the functionality. When a process invokes a system call, LSM hook implementation hooks the information from kernel data structures and provides it to the AppArmor module. Kernel data structures and identity of the subject and object, is checked against the profile of the application and reply is sent back via the hook function's return to the system call.

In all these enhancements, the decision module is abstract to the kernel, which enables extension to policy models. GrSecurity's ACL core being modular, still has interdependance on non-MAC features for enforcement so it can be said that it is not truely modularity.

B. Access Enforcement Mechanism

RSBAC's AEF enforces the decision based on the decision of ADF. Either the system call is permitted to do its function or an error is returned to the process with associated control data if necessary. In case of a positive result, ADF is notified so that ACI can be updated, the ADF tells the decision modules to update their attributes from the data structures, sends an acknowledgement to AEF and then the object is accessed normally.

As AEF is embedded in the kernel subsystems it is the only RSBAC component, which cannot be modularized. It is hooked into several locations of the existing kernel code. Every security relevant system call and pseudo file handling function is extended by two calls to ADF. One is made before the original code, which requests for a decision and one after the original code to send update information to ADF. These two security-handling calls differ by functionality with the change in request type. Thus, it cannot be ported to LSM activated kernels. It would require a lot of change and loss in flexibility to implement new policy models.

The enforcement of policy decisions in SELinux, is accomplished in the kernel subsystems [2][21]. When the SS returns

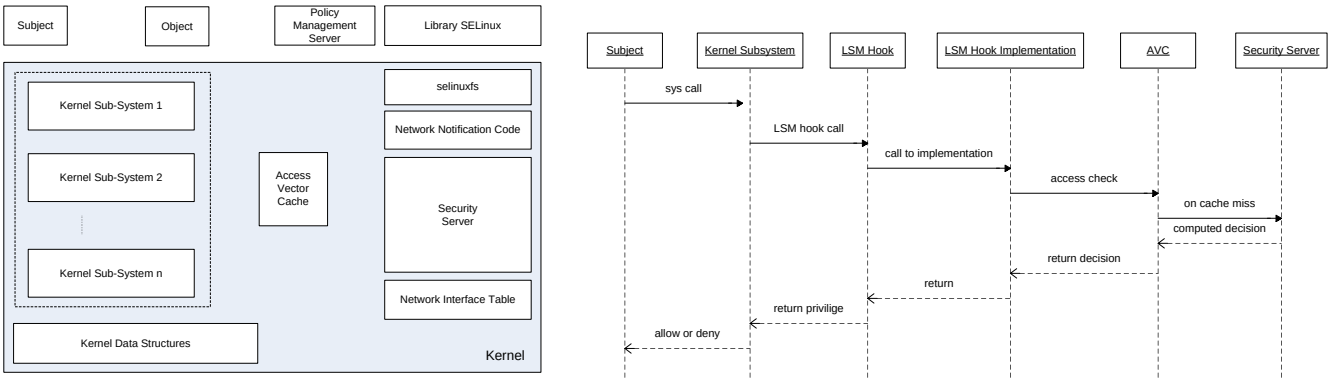

Fig. 4. SELinux and its Control Flow

the decision as a return to the hook function, the enforcement code either allows by letting the system call to continue or denies by returning an error code to it.

The kernel subsystems have the duty to label its objects with the SIDs passed from the security server. In case of subjects, they are non-persistent so these SIDs work fine for their data structures. But file system objects are persistent and labeling them means to release SIDs to user space. SELinux does not allow this. This problem is addressed in the next subsection.

GrSecurity enforcement being similar to SELinux and AppArmor as it has hook functions to mediate the control flow but it also has additional functionality. It has to manage the various security mechanism. These are address space layout randomization, extensive auditing capabilities, netfilter module, stealth netfilter module, enhanced trusted path execution, OpenBSD randomness features at multiple layers, enhanced chroot jail, ptrace and Glibc restriction and various others [8]. These require lots of hooks and other mechanisms. This is the reason why it can't be ported to LSM activated kernels.

AppArmor earlier used system call modifications to check the confinement of processes[24]. Now it uses the LSM module to perform these tasks[27]. LSM is the leveraging medium between AppArmor module, system calls and kernel data structures.

In all these enforcement facilities, we could not find any modularity. The reason was the underlying implementation of a monolithic kernel. They all use some form of abstract functions to escape the problems of system call interpositioning. SELinux and AppArmor implements LSM hooks, while RSBAC and GrSecurity have developed their own abstract functions (hooks). They did this in the interest of greater control, which RSBAC strictly utilized for MAC and GrSecurity to gain access to different layers of the security in the operating system. LSM has been tuned for performance so the compliant enhancements have gained complications to facilitate the required controls.

C. Security Attributes and Policy Storage

RSBAC's ACI [6] is used to store and maintain system values and data structures (for policy modules). System values portion stores and maintains label information for subjects and objects. It implements and store the contexts and data structures that are not handled by the kernel data structures. Persistent attributes are additionally saved in the secondary storage. Dynamic, module specific data, like roles, user groups, access control lists and other relevant information are handled separately in the data structure portion. A file descriptor cache, *fdcache*, is used to reduce the performance hit because of the storage on secondary storage.

SELinux is straight forward in labeling non-persistent objects but allowing the labeling of persistent objects needs a solution. This is enabled by storing the security context in extended file system attributes (EA) [28]. These attributes are special settings stored in the *vnode* of the file, which is more efficient by storing security attributes in an *inode*, for each file (an earlier solution), more localized manner. Although it is a logical solution but it has the limitation of which file systems can be used with SELinux. Currently ext2, ext3, ReiserFS and XFS file systems supports extended attributes.

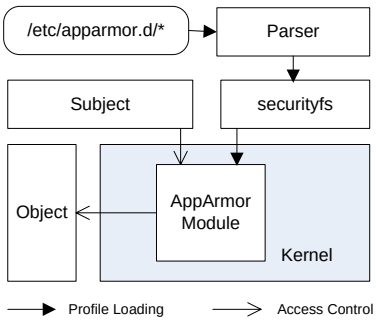

Fig. 5. AppArmor Architecture

The policy configurations are stored in *selinuxfs*, the pseudo file system. This is the binary policy, while the readable version is stored on secondary storage. It can be found in $/etc/selinux/policy.conf$. The generation of this file is a complex and long process. A generous detail can be found in [22].

GrSecurity attribute management for ACL module, is similar to LSM in that it does not centralize them. This was necessary for the various security features it supports. It extends various kernel data structures in order to facilitate all the patches it makes to the kernel.

AppArmor uses *securityfs*, normally mounted on $/sys/kernel/security$ to store its binary profile. The text files that make up the profile are parsed by `apparmor_parse()` called by the user space $/ets/apparmor.d$. The parser loads the binary policy onto securityfs via a `sysctl()` call from where AppArmor module loads it [29]. By convention $/etc/apparmor.d/hello\_world$ will confine hello_world.

Using centralized and generic data structures facilitate abstraction and portability, although it affects performance. RSBAC is LSM independent so it has such facilitation. SELinux and AppArmor being LSM compliant have to use LSM mediation to interact with kernel data structures. GrSecurity implementing hooks to extended kernel data structures has lots of portability issues.

D. Policy Models

RSBAC uses multiple modules and almost all research modules are implemented in it [6]. In this section we describe those that are essential to system security.

The *Authentication* (AUTH) module controls which process can change to which user id (UID). A range or list of UIDs is given for a process, which the process can change to. All other modules depend on this identification scheme.

The *Role Compatibility* (RC) module is a role based model. Every user has a default role, which is inherited by all his processes. This role defines which objects of certain type can be granted accessed or denied.

Linux Capabilities (CAP) defines a group of all root privileges into a POSIX Capabilities list. It enables a set of minimum and maximum capabilities set for both users and processes. It is used to handle root privileges for a user or a process when it needs root privileges to accomplish a task. It restricts privileges to the extent of accomplishing that certain function.

Jail: All Linux kernels provide the chroot() system call to confine a process in a subdirectory. However there are no restrictions on the usual privileges so several ways can be used to achieve root access and break out. RSBAC's Jail module provides a superset of the FreeBSD jail functionality by adding more restrictions. This model is used for service encapsulation and objects still need to be protected with the use of other modules.

SELinux has three mandatory policy models, which work together to formulate a flexible and complete MAC policy. There are two optional models: Multi Level Security (MLS) and Multi Category Security (MCS).

Identity Based Access Control (IBAC) is used to identify the subject, with the right to use some system identity, during login. SELinux identities are orthogonal to Linux UIDs. Whenever a UID of a process is changed, its SELinux identity component will be preserved.

Role Based Access Control (RBAC) [30] restricts the actions of a user according to a set of roles allotted to the user. The transition between roles is controlled by the policy.

Type Enforcement, a simplified version of [31], associates a type with every subject and object i.e. it binds program/code with a type. It is useful in terms of integrity, separation and containment. Nothing can escape this granularity if the policy is properly constructed. This constitutes most of the SELinux policy.

AppArmor does not use any sophisticated policy models but a straightforward profile for the subdomained application. It restricts the resources that a process can access. The profile lists the resources with their respective UNIX permissions and POSIX capability.

GrSecurity uses its ACLs as a policy model to handle MAC. It has included the use of RBAC to assign roles to users. This has added flexibility to it. The ACL system is not orthogonal to standard DAC of Linux as it uses DAC identities with its ACLs. Its maintainer is considering this limitation.

V. Network Controls

RSBAC provides network templates to describe a set of connection end points, which shall be controlled in a mutually configuration [16]. This eases administration as network connections normally have a short life span. Each endpoint inherits the access control settings of the first template that matches it. To set up access control for network connections, to or from a network area, requires definition of a matching template and set the desired attribute values of the network area in the template. Template attribute values are only default values. This enables to configure individual object values.

SELinux provides labeling of network interfaces [32], IP addresses, sockets and ports. These labels can then be used in normal SELinux rules. Additionally, iptables can be used to label packets with *SECMARK* (Red Hat server) extension to Netfilter and then use these labels with SELinux rules. A recent extension to LSM enables labeled network communication via IPSEC [33]. The functionality [34] is leveraged by assigning labels to IPSEC associations for subjects. Work is under progress to include object labels as well.

Following is an example of SECMARK, which labels HTTP packets:

```
[root@some1]\#  iptables -t mangle -p tcp -dport 80
 -t eth0 -s 192.168.0.1/24 -j SECMARK -sectx
 system\_u:object\_r:http\_packet\_t
```

This will label packets on eth0, coming from 192.168.0.1 and on tcp port 80 as http_packet_t. Another rule can be added:

```
[root@some1]\#  iptables -t mangle -m state -status
 RELATED, ESTABLISHED -j CONNSECMARK -restore
```

This will copy the label for related packets so when an http client uses a related port (dynamically assigned) will receive the same label. This takes advantage of netfilter's connection tracking features.

For labeling IPSEC associations we need to add a -ctx statements to the setkey files:

```
[root@some1]\#  cat dev/ipsec/setkey/some1.tst
spdflush;
flush
spdadd 192.168.0.1 192.168.0.2 any
-ctx 1 1 "system\_u:object\_r:default\_t:s0"
-P in ipsec esp/transport//require;
spdadd 192.168.0.2 192.168.0.1 any
-ctx 1 1 "system\_u:object\_r:default\_t:s0"
-P out ipsec esp/transport//require;
```

GrSecurity uses its ACL to control access of subjects on the bases of IP addresses and ports. It has other features inherited

	RSBAC	**SELinux**	**GrSecurity**	**AppArmor**
Kernel support	Patch	Mainline kernel	Patch	Patch
Hook type	Abstract function + request type	LSM	GrSecurity	LSM
Label storage	rsbac.dat, file system independant	EA/metadata (file system dependant)	Extending data structures	internal-only (file system independant)
Policy Learning	Built-in for non-critical modules	audit2allow	Extensive, built-in	Initial policy generation and incremental updates
Optimization	Hashed list lookups 0(1), ordered generic lists, attributes inheritance	AVC	Hashed lookups 0(1)	Access granted once and rechecked only on change in rights
Buffer overflow protection	PaX	Exec-Shield	PaX	N/A
Policy Models	AUTH, RC, CAP, JAIL,ACL,FF, UM, PM, DAZ, MLS	IBAC, TE, RBAC, MLS, MCS	ACL, RBAC	Profiles
Additional features	Secure delete, process hiding, file system hiding, symlink redirection, in-kernel usermanagement, on-access virus scan	N/A	Random IPID, process hiding, modified chroot, TPE, symlink restrictions	N/A
Portability	Yes	Yes	N/A	N/A

TABLE II
DETAILED COMPARISON

from OpenBSD and association with Netfilter but they are mechanisms to defend against common exploits. [8]

AppArmor [9]access controls are limited to confining network agents by inspecting open ports using netstat `-inet -n -p`. It is on its way to integrate SECMARK functionality in its profiles.

There are user space mechanisms to handle labels for remote resources as well. These labels can be passed in headers of RPC [35] in RPC based servers like NFS and messages in message passing clusters and grids. These labels need to be interpreted meaningfully, which is not possible without keeping policy copies on all systems. Label understanding remotely and policy distribution are being researched [36] extensively. We are not satisfied with current efforts and our future work will exhaustively consider these issues.

VI. RELATIVE STRENTHS AND WEAKNESSES

RSBAC has the best modular design, which has been taken care of during implementation. It provides fine grained and flexible MAC. It's a perfect toolkit for implementing new security models and it has adequate network controls, which are also extendable.

SELinux is also suitable for implementing new policy models. Being part of the mainline kernel gives it an edge for mass testing, development and acceptance. A lot of work has been done to implement usable network controls. SELinux enforcement support has also been extended to application space for complex servers, while others lag behind in this aspect.

AppArmor lacks true MAC implementation due to its application-oriented approach but it attracts administrators due to easy management of policy. It lacks granularity and flexibility. Network controls are adequate and it is following SELinux for more usable controls. Yast [29], the profile builder and manager, is a GUI based application that eases management of AppArmor.

GrSecurity has a moderate design but addresses vast threat models. It's ACL and RBAC provides a flexible MAC but cannot achieve the granularity of RC and TE policy models. It is administrator friendly because of its user space tools that can automate policy development by its learning mode. Network controls are adequate and extendable.

VII. FUTURE DIRECTIONS

It is necessary to determine all the possible ways in which network labels can be distributed and all needs of network security are to be considered. Current policies are not appropriate to handle the complexity of networked resources so a remote resource aware policy model or a translation/equivalence mechanism needs to be devised. We have already identified *some* solutions and work is in progress.

Individual nodes might need to establish trust before releasing their information based upon the access control policies of other nodes. This trust can be concretely established using remote attestation techniques. We will be considering the integeration of Integrity Measurement Architecture [37] or an equivalent approach for remote attestaion and use TCG's TPM [38] as a root of trust in the attestation process.

VIII. CONLUSION

We have learned from this study that RSBAC is the tool, which is most suitable for the development of new policy models for security. On the other hand, SELinux is the enhancement, which is more practical in terms of its integration in mainline kernel. SELinux also provides the set of latest network controls. These can also be implemented in other enhancements with some effort.

It would be better to prototype the design on RSBAC and then port a working model to SELinux. The learning curve for this approach would be very steep because its not easy adjusting to such complex enhancements. Thus it will be sensible to use SELinux as a security enhanced kernel for the design and development of DMAC framework.

REFERENCES

[1] P. Loscocco, S. Smalley, P. Muckelbauer, R. Taylor, S. Turner, and J. Farrell, "The Inevitability of Failure: The Flawed Assumption of Security in Modern Computing Environments," *Proceedings of the 21st National Information Systems Security Conference*, vol. 10, pp. 303–314, 1998.

[2] P. Loscocco and S. Smalley, "Integrating Flexible Support for Security Policies into the Linux Operating System," *Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference table of contents*, pp. 29–42, 2001.

[3] "MIT Kerberos," *Available at: web.mit.edu/Kerberos/*, 2000.

[4] "LDAP," *Available at: www.openldap.org/*, 2000.

[5] "NIS," *Available at: www.linux-nis.org/nis/*, 2000.

[6] "Rule Set Based Access Control," *Available at: www.rsbac.org*, 2000.

[7] "NSA SELinux website," *Available at: http://www.nsa.gov/selinux*, 1999.

[8] "GrSecurity," *Available at: www.GrSecurity.net*, 2000.

[9] "AppArmor," *Available at: http://www.novell.com/linux/security/apparmor/*, 2000.

[10] S. Khan, "The DMAC Infrastructure Project," *Available at: https://sourceforge.net/projects/dmac*, 2007.

[11] J. P. Anderson, "Reference Monitor," *Computer Security Technology Planning Study*, vol. 2, 1972.

[12] M. Bishop and M. Digler, "Checking for Race Conditions in File Accesses," *Computing Systems*, pp. 131–152, 1996.

[13] C. Wright, C. Cowan, J. Morris, I. Ltd, S. Smalley, G. Kroah-Hartman, and I. Center, "Linux Security Module Framework," *Ottawa Linux Symposium*, 2002.

[14] C. Wright, C. Cowan, J. Morris, S. Smalley, and G. Kroah-Hartman, "Linux security modules: general security support for the linux kernel," *Foundations of Intrusion Tolerant Systems, 2003 [Organically Assured and Survivable Information Systems]*, pp. 213–226, 2003.

[15] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn, "Design and Implementation of a TCG-based Integrity Measurement Architecture," *Proceedings of the 13th USENIX Security Symposium*, pp. 223–238, 2004.

[16] A. Ott and S. Fischer-Hübner, *The Rule Set Based Access Control (RSBAC) framework for Linux*. Division for Information Technology, Department of Computer Science, Univ.

[17] "DTOS," *Available at: http://www.cs.utah.edu/flux/dtos/*, 1998.

[18] "Flask," *Available at: http://www.cs.utah.edu/flux/flask/*, 1999.

[19] "MACH," *Available at: http://www.gnu.org/software/hurd/gnumach.html*, 2000.

[20] R. Spencer, S. Smalley, P. Loscocco, M. Hibler, D. Andersen, and J. Lepreau, "The flask security architecture: system support for diverse security policies," *Proceedings of the 8th conference on USENIX Security Symposium-Volume 8 table of contents*, pp. 11–11, 1999.

[21] S. Smalley, C. Vance, and W. Salamon, "Implementing SELinux as a Linux Security Module," *NAI Labs Report 01*, vol. 43, 2001.

[22] F. Mayer, K. MacMillan, and D. Caplan, *SELinux by Example: Using Security Enhanced Linux*. Prentice Hall, 2006.

[23] "Openwall," *Available at: www.openwall.com*, 2000.

[24] C. Cowan, S. Beattie, G. Kroah-Hartman, C. Pu, P. Wagle, and V. Gligor, "SubDomain: Parsimonious Server Security."

[25] "MACcomparison," *Available at: http://gentoo-wiki.com/Access_Control_Comparison_Table*, 2007.

[26] "SELinux Technical Documents," *Available at: www.nsa.gov/selinux/docs.cfm*, 2004.

[27] "AppArmor Technical Documentation," *Available at: http://www.ussg.iu.edu/hypermail/linux/kernel/0706.1/0805/techdoc.pdf*, 2002.

[28] "Extended Attributes," *Available at: http://www.linuxjournal.com/node/7426*, 2004.

[29] "AppArmor Admin Documentation," *Available at: http://www.novell.com/documentation/suse101/pdfdoc/apparmor-admin-guide/apparmor-admin-guide.pdf*, 2002.

[30] D. Ferraiolo, R. Sandhu, S. Gavrila, D. Kuhn, and R. Chandramouli, "Proposed NIST standard for role-based access control," *ACM Transactions on Information and System Security (TISSEC)*, vol. 4, no. 3, pp. 224–274, 2001.

[31] S. Hallyn and P. Kearns, "Domain and Type Enforcement in Linux," Ph.D. dissertation, 2000.

[32] J. Morris, "Networking in NSA security-enhanced Linux," *Linux Journal*, vol. 2005, no. 129, 2005.

[33] S. Kent and R. Atkinson, "RFC 2401,," *Security Architecture for the Internet Protocol", November*, 1998.

[34] T. Jaeger, D. King, K. Butler, S. Hallyn, J. Latten, and X. Zhang, "Leveraging IPsec for Mandatory Per-Packet Access Control," *Securecomm and Workshops, 2006*, pp. 1–9, 2006.

[35] J. Carter, "Implementing SELinux support for NFS," *Available at: http://www.nsa.gov/selinux/papers/nfsv3.pdf*, 2000.

[36] C. S. B. Joshua, K. Vance, "Enforcing Security Enhanced Linux Policies in a Networked Policy Domain," *SELinux Symposium*, 2007.

[37] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn, "Design and Implementation of a TCG-based Integrity Measurement Architecture," *Proceedings of the 13th USENIX Security Symposium*, pp. 223–238, 2004.

[38] T. Part, "Design Principles Specification Version 1.2," *TCG Specification*, vol. 5, 2004.