UNIX/LINUX SECURITY FEATURES

What do we have in this Module?

1. INTRODUCTION

Operating system (OS) is a kernel. Linux operating system is Linux's kernel. Currently the version is 2.6.22.6 [1]. There are hundreds versions of Linux distributions [2] but all still based on the same kernel.
Generally, Linux OS can be represented in the following sphere with three layers: User land, System land and kernel. In the innermost, kernel houses all the operating system resources such as file systems, memory managements, input/output modules and libraries. The outer layer, system land hosts system resources such as Application System Interface (API).

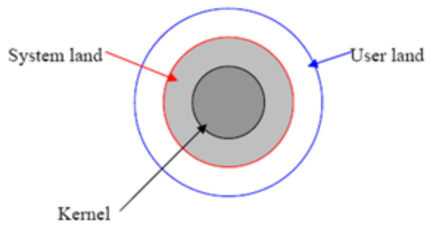

Figure 1. The OS layered sphere representation

The outermost layer is the user land where all the user resources will reside such as application programs. It is where the user accounts have privileges such as running permitted programs and issuing restricted commands however, the Root (Administrator in Windows OS) has privileges in all layers. Keep in mind that for Windows OS there is another higher privileges account than Administrator that is LocalSystem. LocalSystem normally used to run Windows services and can be viewed through the Windows Services snap-in.
Linux is a multi-users and multi-tasking OS. Single Linux OS can provide services for more than one user at any time either locally and/or remotely. Every user has their own profile with custom settings that can be set by the user herself for the permitted settings or enforced by Root from the system side. For every user, there will be multi process running 'concurrently' for him, locally and/or remotely and it is said multi-tasking OS. In another simple word, single user can run many programs at any time. In order to optimize the resources such as memory, in every process there can be many threads and it is said multi threading.
In Linux, systems' processes or services (in Linux term it is a daemon) normally run by Root. Originally, Root can be considered as the king with unlimited privileges that can control the whole OS. However, non-root group's users will have limited privileges. The many problems start when the users' privileges have been escalated to Root. When normal users have controlled or could access the kernel, it is a very bad situation.
To make the thing worse, the memory management (allocation and de-allocation) for the running processes also has many holes.
For example, the area that marked as read could also be executed. The unchecked size memory allocation on stack and heap already generated many exploits and the famous one is buffer overflow [3] problems.
That is why in the previous few decades the OS security enhancements concentrated on the access permissions and memory protections.

2. STANDARD BASIC SECURITY FEATURES

For the basic security features, Linux has password authentication, file system discretionary access control, and security auditing. These three fundamental features are necessary to achieve a security evaluation at the C2 level [4]. Most commercial server-level operating systems, including AIX (IBM), Windows NT, and Solaris, have been certified to this C2 level. By expanding the basic standard security features we have:

1. User and group separation
2. File system security
3. Audit trails
4. PAM authentication

2.1 User and Group Separation

User accounts are used to verify the identity of the person using a computer system. By checking the identity of a user through username and password credentials, the system is able to determine if the user is permitted to log into the system and, if so, which resources the user is allowed to access.
Groups are logical constructs that can be used to group user accounts together for a particular purpose. For example, if a company has a group of system administrators, they can all be placed in a system administrator group with permission to access key resources of the OS. In addition, through group creation and assignment of privileges, access to restricted resources can be controlled for those who need them and denied to others.
The ability for a user to access a machine is determined by whether or not that user's account exists. Access to an application or file is granted based on the permission settings for the file. This helps to ensure the integrity of sensitive information and key resources against accidental or purposeful damage by users.
After a normal user account is created, the user can log into the system and access any applications or files they are permitted to access. Linux determines whether or not a user or group can access these resources based on the permissions assigned to them.
There are three permissions for files, directories, and applications. Table 1 lists the symbols used to indicate each of them. Each of the three permissions is assigned to three defined categories of users. The categories are listed in Table 2.

Table 1. Permission character symbols

Symbol	Description
r	Indicates that a given category of user can **read** a file.
w	Indicates that a given category of user can **write** to a file.
x	Indicates that a given category of user can **execute** the file.
-	A fourth symbol indicates that no access is permitted.

Table 2. Permission categories

Category	Description
Owner	The owner of the file or application.
Group	The group that owns the file or application.
Everyone	All users with access to the system.

One can easily view the permissions for a file by invoking a long format listing using the command ls -l. For instance, if the user kambing creates an executable file named foo, the output of the command ls -l foo would look something like this:

```
-rwxrwxr-x 1 kambing kambing 0 Sep 2 12:25 foo
```

The permissions for this file are listed at the start of the line, starting with set of rwx.

- This first set of symbols defines owner access.
- The next set of rwx symbols define group access,
- The last set of symbols defining access permitted for all other users.

This listing indicates that the file is readable, writable, and executable by the user who owns the file (user kambing) as well as the group owning the file (which is a group named kambing). The file is also world-readable and world-executable, but not world-writable.

2.2 File System Security

A very true statement of a UNIX/Linux system, everything is a file; if something is not a file, it is a process. Most files are just files, called regular files; they contain normal data, for example text files, executable files or programs, input to or output from a program and so on. While it is practically safe to say that everything you encounter on a Linux system is a file, there are some exceptions as listed below:

- Directories: files that are lists of other files.
- Special files: the mechanism used for input and output. Most special files are in /dev for example USB and CD-ROM.
- Links: a system to make a file or directory visible in multiple parts of the system's file tree. It is a shortcut.
- (Domain) sockets: a special file type, similar to TCP/IP sockets, providing inter-process networking protected by the file system's access control.
- Named pipes: act more or less like sockets and form a way for processes to communicate with each other, without using network socket semantics.

The following table gives an overview of the characters determining the file type:

Table 3. File types character symbols

Symbol	Meaning
-	Regular file
d	Directory
l	Link
c	Special file
s	Socket
p	Named pipe
b	Block device

On Linux system, every file is owned by a user and a group user. There is also a third category of users, those that are not the user owner and don't belong to the group owning the file. For each category of users, read, write and execute permissions can be granted or denied.
The long option to list files using the ls -l command, also displays file permissions for these three user categories; they are indicated by the nine characters that follow the first character, which is the file type indicator at the beginning of the file properties line. As seen in the following examples, the first three characters in this series of nine display access rights for

the actual user that owns the file.

```
ls -l Mine
-rw-rw-r-- 1 mike users  5 Jul 15 12:39 Mine
ls -l /bin/ls
-rwxr-xr-x 1 root root 45948 Aug 10 15:01 /bin/ls*
```

The next three are for the group owner of the file, the last three for other users. The permissions are always in the same order: read, write, execute for the user, the group and the others. The first file is a regular file (first dash). Users with user name mike or users belonging to the group users can read and write (change/move/delete) the file, but they can't execute it (second and third dash). All other users are only allowed to read this file, but they can't write or execute it (fourth and fifth dash).
The second example is an executable file, the difference is everybody can run this program, but you need to be root to change it.
For easy use with commands, both access rights or modes and user groups have a code shown in Table 4 and 5.

Table 4. Access mode codes

Code	Meaning
0 or -	The access right that is supposed to be on this place is not granted.
4 or r	read access is granted to the user category defined in this place
2 or w	write permission is granted to the user category defined in this place
1 or x	execute permission is granted to the user category defined in this place

Table 5. User group codes

Code	Meaning
u	user permissions
g	group permissions
o	permissions for others

This straight forward scheme is applied very strictly, which allows a high level of security even without network security. Among other functions, the security scheme takes care of user access to programs; it can serve files on a need-to-know basis or least privilege and protect sensitive data such as home directories and system configuration files. We can use the chmod command to modify the file permission, changing of the access mode of a file. The chmod command can be used with alphanumeric or numeric options, whatever you like best. The following shows the examples.

```
>/hello
bash: ./hello: bad interpreter: Permission denied

>cat hello
#!/bin/bash
echo "Hello, World"

>ls -l hello
-rw-rw-r-- 1 mike  mike 32 Jul 1 16:29 hello

>chmod u+x hello

>./hello
Hello, World

>ls -l hello
-rwxrw-r-- 1 mike mike 32 Jul 1 16:29 hello*
```

The + and - operators are used to grant or deny a given right to a given group. Combinations separated by commas are allowed. The following is another example, which makes the file from the previous example a private file to user mike:

```
>chmod u+rwx,go-rwx hello

>ls -l hello
-rwx------ 1 mike mike 32 Jan 15 16:29 hello*
```

If you encounter problems resulting in an error message saying that permission is denied, it is usually a problem with

access rights in most cases.
When using chmod with numeric arguments, the values for each granted access right have to be counted together per group. Thus we get a 3-digit number, which is the symbolic value for the settings chmod has to make. The following table lists the most common combinations:

Table 5. File protection with chmod

Command	**Meaning**
chmod 400 file	To protect a file against accidental overwriting.
chmod 500 directory	To protect you from accidentally removing, renaming or moving files from this directory.
chmod 600 file	A private file only changeable by the user who entered this command.
chmod 644 file	A publicly readable file that can only be changed by the issuing user.
chmod 660 file	Users belonging to your group can change this file; others don't have any access to it at all.
chmod 700 file	Protects a file against any access from other users, while the issuing user still has full access.
chmod 755 directory	For files that should be readable and executable by others, but only changeable by the issuing user.
chmod 775 file	Standard file sharing mode for a group.
chmod 777 file	Everybody can do everything to this file.

If you enter a number with less than three digits as an argument to chmod, omitted characters are replaced with zeros starting from the left. There is actually a fourth digit on Linux systems that precedes the first three and sets special access modes.

2.2.1 The File Mask

When a new file is saved somewhere, it is first subjected to the standard security procedure. Files without permissions don't exist on Linux. The standard file permission is determined by the mask for new file creation. The value of this mask can be displayed using the umask command:

```
>umask
0002
```

Instead of adding the symbolic values to each other, as with chmod, for calculating the permission on a new file they need to be subtracted from the total possible access rights. In the example above, however, we see 4 digits displayed, yet there are only 3 permission categories: user, group and other. The first zero is part of the special file attributes settings. It might just as well be that this first zero is not displayed on your system when entering the umask command and that you only see 3 numbers representing the default file creation mask.
Each UNIX-like system has a system function for creating new files, which is called each time a user uses a program that creates new files, for instance, when downloading a file from the Internet, when saving a new text document. This function creates both new files and new directories. Full read, write and execute permission is granted to everybody when creating a new directory. When creating a new file, this function will grant read and write permissions for everybody, but set execute permissions to none for all user categories. In this case, before the mask is applied, a directory has permissions 777 or rwxrwxrwx, a plain file 666 or rw-rw-rw-.
The umask value is subtracted from these default permissions after the function has created the new file or directory. Thus, a directory will have permissions of 775 by default, a file 664, if the mask value is (0)002. This is demonstrated in the following examples:

```
>mkdir newdir

>ls -ld newdir
drwxrwxr-x 2 mike mike 2096 Jul 28 13:45 newdir/

>touch newfile

>ls -l newfile
-rw-rw-r-- 1 mike mike 0 Jul 28 13:52 newfile
```

A directory gets more permission by default, it always has the execute permission. If it wouldn't have that, it would not be accessible.
If you log in to another group using the newgrp command, the mask remains unchanged. Thus, if it is set to 002, files and directories that you create while being in the new group will also be accessible to the other members of that group; you don't have to use chmod. The root user usually has stricter default file creation permissions as shown below:

```
[root@tenouk root]# umask
022
```

These defaults are set system-wide in the shell resource configuration files, for instance /etc/bashrc or /etc/profile. You can change them in your own shell configuration file.

2.3 Audit Trails

Linux kernel 2.6 comes with auditd daemon. It's responsible for writing audit records to the disk. During startup, the rules in /etc/audit.rules are read by this daemon. You can open /etc/audit.rules file and make changes such as setup audit file log location and other option. The default file is good enough to get started with auditd. In order to use audit facility you need to use following utilities:

Table 6. Audit utility

Utility	Description
auditctl	A command to assist controlling the kernel's audit system. You can get status, and add or delete rules into kernel audit system
ausearch	A command that can query the audit daemon logs based for events based on different search criteria.
aureport	A tool that produces summary reports of the audit system logs.

2.4 Pluggable Authentication Modules authentication (PAM)

PAM [5] was invented by SUN Microsystems. Linux-PAM provides a flexible mechanism for authenticating users. It consists of a set of libraries that handle the authentication tasks of applications on the system. The library provides a stable general interface to which privilege-granting programs (such as login) defer to perform standard authentication tasks. Historically, authentication of Linux users relied on the input of a password which was checked with the one stored in /etc/passwd. At each improvement (e.g. /etc/shadow, one-time passwords) each program (e.g. login, ftp) had to be rewritten. PAM is a more flexible user authentication mechanism. Programs supporting PAM must dynamically link themselves to the modules in charge of authentication. The administrator is in charge of the configuration and the attachment order of modules. All applications using PAM must have a configuration file in /etc/pam.d. Each file is composed of four columns:

Table 7. PAM's pam.d content

Column	Description
Module type	■ auth: user authentication ■ account: user restriction (e.g.: hour restriction) ■ session: tasks to perform at login and logout e.g.: mounting directories ■ password: update of the user authentication token
success control	■ required: a least one of the required modules ■ requisite: all the requisite modules ■ sufficient: only one sufficient module ■ optional: a least one of the required modules is necessary if no other has succeeded
path to the module	Usually /lib/security.
optional arguments	-

Other PAM functionalities are listed in the following Table.

Table 8. Other PAM functionality

Functionality	Description
/etc/pam.d/other file	Provides default configuration for all modules not specified in the configuration file of the application.
pam_cracklib	Uses the cracklib library to check the "strength" of a password and to check it was not built based on the old one.
pam_limits	This module can restrict, depending on the user and/or group, the number of simultaneous processes, CPU time, the number of files simultaneously opened, their size, and the maximum number of simultaneous connections. The configuration file is: /etc/security/limits.conf
pam_rootok	Enables root to access a service without using his password. To be used with chfn or chsh and not with login.
pam_time	Control the access time. The configuration file is: /etc/security/time.conf.
pam_wheel	Allow access to root only to users of the wheel group. For use with su.
pam_cap	This module can force all privileges to a user.

Keep in mind that PAM however does not itself have an authenticated access to the kernel.

3. LINUX SECURITY EXTENSIONS

The Linux family of products has provided a highly secure environment since its original delivery in early 2002. The features discussed in the following sections have been added to the Linux OS. For example, the Red Hat Enterprise Linux Update 3, shipped in September 2004 contains:

1. ExecShield [6] – With the **N**o e**X**ecute (NX) [7], [8], or e**X**ecute **D**isable (XD) and Segmentation features.
2. Position Independent Executables (PIE) [9]

Then, in Red Hat Enterprise Linux v.4, shipped in February 2005 contains the following security features:

1. SecurityEnhanced Linux (SELinux) [10]
2. Compiler and library enhancements [11]
3. Advanced glibc memory corruption checker [11]
4. Secure version of the printf and other string manipulation functions. [11]
5. gcc buffer bound checking [11]

In term of the Linux OS security breaches, most of the problems originated from the buffer overflow issue. The buffer overflow exploits unprotected and or unchecked fixed sized buffers, overwriting the area beyond it. The overwritten area may be filled with the malicious codes, containing code that pointing to the customized return address. There are many buffer locations in the memory area. It is used to temporarily store data.

3.1 ExecShield

The ExecShield supports two technologies that protect application from being compromised by most of the buffer exploit types. The goal of these features is to prevent code that is maliciously written in the data areas of an application from being executed. These NX/XD and Segmentation features use different techniques but to achieve the similar result. PAX [12], [13], [14] is similar, earlier technology that will not be discussed here.

3.1.1 The NX/XD

The NX term is used by AMD for its Opteron/Athlon64 processors, while the XD is used by Intel for its Itanium2 and the x86/EM64T processors. These capabilities provides a new memory management feature that that allows individual pages of an application's memory to be marked as non executable. The problem is, previously the only level of control over memory pages was read and write. However, a page that was enabled for read could also be executed.
This meant that data areas such as the stack, heap and I/O buffers, which are typically only used for read/write could also be used to execute codes. It is a common form of exploit that involves writing code in a stack buffer and then executing it. So the ability to disable execution enhances the application and system security. The NX/XD support is available for most new processors including the recent model Intel x86 CPUs.

3.1.2 Application Segmentation

Application Segmentation provides a capability that is similar to NX/XD but it is designed for the mast installed base of x86 systems that do not have the NX/XD hardware feature. As with NX/XD, the goal is to prevent execution of code from stack or data buffer areas. Segmentation uses the little known x86 processor memory management segmentation feature to split an application into two segments that is executable and non executable as shown in Figure 2. As with NX/XD, Segmentation makes application exploits much more difficult to construct. It is enabled/disabled in the same manner as described for NX/XD earlier. Segmentation provides a less granular approach to preventing execution of data as code at the segment level as opposed to NX/XD, which operates at the per page level, but it is equally effective.

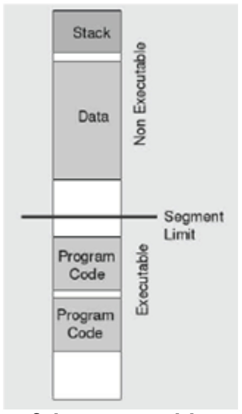

Figure 2. Segmentation of the executable and non-executable area

3.1.3 Position Independent Executable (PIE)

The NX/XD and ExecShield prevent data in buffer or on the stack from being executed as code, but they do not prevent exploits that, for example, change the return address of a function on the stack so that program control is transferred to an incorrect part of the application or pointing to other part of executable code portion such as C libraries or the executable sections itself. This type of exploit might allow functions such as security checks to be bypassed. Without PIE, any given application is typically loaded into the same memory addresses each time it runs. Based on the exploits experiences, security exploits that manage to trigger incorrect sections or functions of an application to run are only effective if they know where the sections are located in memory. With PIE enabled, different sections of an application are randomly loaded into different memory locations each time the application runs. This makes it slightly difficult to guess the memory address and harder for these exploits to succeed.
As an example, in Fedora Core, the memory addresses randomization happens once every two weeks with a daily incremental run in between, making all system ‘look’ different even though running the same applications on the same machine.

4. COMPILER AND LIBRARY ENHANCEMENTS

Another feature, during late 2004, Red Hat developed a new group of features that improve buffer management and security for inclusion in Red Hat Enterprise Linux v.4. After that, these features adopted by other Linux distribution that compiled using new GLIBC libraries.

5. THE GLIB MEMORY CORRUPTION CHECKS

The GLIBC memory allocator functions now perform a set of internal sanity check to detect double freeing of memory and heap buffer overflows. With these checks, regular application bugs and security exploit attempts that use these techniques are detected and the program will be instantly aborted to avoid the possibility of the exploit succeeding. With these checks, double free exploits become entirely impossible and all standard, generic heap type overflow techniques are blocked. These detection functions execute prior to the ExecShield features mentioned earlier. In many cases exploit attempts of this type would ultimately be blocked by ExecShield, but these memory corruption checks provide an extra level of security because the earlier an exploit attempt is detected and aborted the better.

6. THE printf() FORMAT STRING EXPLOIT PREVENTION

The printf() format string exploits were popular around several years ago when the technique was first exposed. Compared to other C functions, the printf() function is a variadic type function that can accept variable number of parameter. The printf() format string exploits abuse a bug in programs that have a faulty call to the standard printf() function, caused by a formatting parameter. When applications are compiled with the "-D_FORTIFY_SOURCE=2" compiler option, the printf() function will check that this rare formatting comes from guaranteed trusted sources and will abort the program if that is not the case, thus preventing printf() format exploits entirely. Other secure version of the string and character manipulation, standard C and C++ functions also were introduced and implemented in newer version of compilers.

7. GCC BUFFER BOUND CHECKING [15], [16]

This feature detects many potential buffer overflow conditions and is the most important of the buffer management improvements. An enhancement has been added to the GCC compiler so that if the size of the destination buffer can be detected at compile time, function such as strcpy(), memcpy() and printf() will use a checking variant of these functions detects if the buffer will overflow. If that happens the program is aborted immediately. While GCC cannot always detect the size of the destination buffer for example, it is not possible for dynamically allocated buffers. Buffer allocation errors usually occur with the types of buffer that can be detected by GCC. The result is that a large percentage of buffer overflow errors are prevented immediately.

In the meantime the secure version of the C functions [17] that involve the use of the buffer, have been added to the standard C library. The secure version can be recognized from the _s suffix for example, printf_s() and strcpy_s(). When using these secure versions, programmer need to include additional parameter for the buffer size. However, the implementation still depends on the programmers whether to use the secure version or not. You will find warning that stated a non-secure version will be marked as deprecated when using those functions for newer compilers.

8. DISCRETIONARY ACCESS CONTROL AND MANDATORY ACCESS CONTROLS

As discussed previously standard Linux file permissions use the Discretionary Access Control (DAC) model. Under DAC, files are owned by a user and that user has full control over them, including the ability to grant access permissions to other users. The root account has full control over every file on the entire system. An attacker who penetrates an account can do anything with the files owned by that user. For example, an attacker who compromises a web server has full control over all files owned by the web server account. If an application runs under the context of the root user, an attacker penetrating it now has full control over the entire system.

SELinux (discussed later) supplements Discretionary Access Control with Mandatory Access Control (MAC). Under MAC, the Administrator writes a security policy that defines access rights for all users and applications. MAC in effect provides each application with a virtual sandbox that only allows the application to perform the tasks it is designed for and explicitly allowed in the security policy to perform. For example, the web server process may only be able to read web published files and serve them on a specified network port. An attacker penetrating it will not be able to perform any activities not expressly permitted to the process by the security policy, even if the process is running as the root user. Files are assigned a security context that determines what specific processes can do with them, and the allowable actions are much more finely defined than the standard Linux read/write/execute controls. For example, a web served file would have a context allowing the apache process to read it but not execute or make changes to it, while the log files would be appendable but not readable or otherwise changeable by apache.

Network ports are also assigned a context, which can prevent penetrated applications from using ports not permitted to them by security policy. Standard Linux permissions are still present on the system, and will be consulted before the SELinux policy when access attempts are made. If the standard permissions would deny access, access is simply denied and SELinux is not consulted at all. If the standard file permissions would allow access, the SELinux policy is consulted and access is either allowed or denied based on the security contexts of the source process and the targeted object.

8.1 Security Enhanced Linux (SELinux)

Researchers in the Information Assurance Research Group of the National Security Agency (NSA) worked with Secure Computing Corporation (SCC) to develop a strong, flexible mandatory access control architecture based on Type Enforcement, a mechanism first developed for the LOCK system [18]. The NSA and SCC developed two Mach-based prototypes of the architecture: DTMach [19] and DTOS [20], [21]. The NSA and SCC then worked with the University of Utah's Flux [22] research group to transfer the architecture to the Fluke [23] research operating system. During this transfer, the architecture was enhanced to provide better support for dynamic security policies. This enhanced architecture was named Flask [24]. The NSA has now integrated the Flask architecture into the Linux operating system to transfer the technology to a larger developer and user community.

SE Linux is based on the Flask Architecture in which the security policy is separated from the enforcement logic. The advantage of this approach is that a flexible MAC security model and usually a sort of Role Based Access Control (RBAC) [25] is also integrated, as per the security need of the organization, can be implemented. An additional benefit of this approach is that enforcement of security policies can be transparent to the applications since it’s possible to define the default security behavior. This architecture ensures the data integrity and the trustworthiness or simply put it provides access controls.

SELinux has strong Mandatory Access Control built into the kernel which the process and objects such as files are classified based on the confidentiality and integrity requirement; hence the affect of a security break is reduced to minimum. SE Linux was designed rather to use MAC in contrast to DAC used by traditional Linux systems to make a system which will lessen the affects of security policy breaks to a minimum, by the help of policies which specify the security requirements of a system.

9. OTHER LINUX SECURITY EXTENSIONS

Many Linux distributions have been hardened by the security extensions. The main modifications to these systems were the addition of MAC model as discussed previously. It is called a multi-level security (MLS) [26] model, and auditing capabilities. A MLS model is designed to prevent the leakage of data to unauthorized subjects, but does not address the integrity of the system. That is, such systems prevent the leakage of data, but do not prevent the exploitation of bugs by user on data from untrusted sources that may compromise the entire system.
Firstly, the TCPA consortium [27], [28] aims to provide hardware support for reliable system integrity verification, and tamperproof platforms, such as IBM's 4758 [29], enable the installation of security services in hostile environments.
A large number of advances have been made for Linux as well. The Bastille Linux [30] and Immunix [31] provide hardening tools for preventing bugs from taking over system services that run as root. For example, Immunix is a family of tools designed to cause system services to fail safely when one of a variety of common vulnerability types such as buffer overflow attack happens.
Trustix [32], OpenWall [33] and HP Secure OS Software for Linux [34] provide a variety of tools to improve Linux security such as TripWire [35], [36] and encrypted file systems.
Many of these systems provide MAC policy models for files, but only Argus PitBull [37] provides a model that enables control of network objects as well. The MAC policy models used in these systems are either significantly limited or complex and lack supporting management tools.
Trustix does provide support for assisting system administrators in installing a system with minimal services and preventing accidental initiation of new services.
A fundamental problem with all of the approaches above is that they require kernel modifications to provide the desired authorization flexibility and performance. This is because the actual objects and operations performed are determined deep inside the kernel, so the kernel must be changed to ensure that the intended policy is enforced. Because each system uses different, adhoc kernel modifications none will be accepted into the base kernel.
In kernel 2.6, the Linux Security Modules (LSM) [38], [39] framework adds authorization hooks into the base Linux kernel that intends to cover every controlled operation in Linux kernel. The hooks are independent of the authorization policy, so a variety of MAC policies can be supported. Fundamentally, LSM is only an authorization framework, so many other features are necessary to build a secure Linux system. In this case, we need to identify the tasks that can be performed to install an initial Trusted Computing Base (TCB) for a secure Linux system. Then, we point out where the Linux extension systems provide a similar functionality.
The LSM community is led by Wirex with DARPA sponsorship. NAI Labs with NSA sponsorship has been another major contributor, but many other companies such as IBM and individuals from around the world have also provided input to the LSM framework.

10. CONCLUSION

The fundamental Linux securities not change so much however there are many Linux security extensions enhancements. These extensions seem overlapped in many aspects. There should be an independent body that coordinates Linux security framework or tools development and adoption.
We can appreciate that although without starting from scratch in designing new secure kernel, the approaches to provide a secure OS start from designing compiler and using new safer C/C++ libraries.
This paper does not discuss other tools that can be used for Linux security implementations such as Linux Intrusion Detection (LIDS) [40] and Linux firewall [41], [42], [43]. There are other similar tools for the items discussed in the Linux security extension sections not included in this paper. For example, StackGuard [44] and Trusted Platform Module (TPM) [45]. Other good and free Linux security related security software include Snort, ClamAV, OpenSSH, OpenSSL, IPSec, AIDE, nmap, GnuPG, Encrypted File System (EFS) and many more.
In dealing with the current vulnerabilities we need to face many new challenges from time to time such as the rootkits [46] and the progressive web technologies development have introduced more complex exploits.

11. REFERENCES

[1] The Linux Kernel Archives site, "The primary site for the Linux kernel source", http://kernel.org/
[2] The Linux Distributions information site, http://distrowatch.com/
[3] Buffer overflows tutorial, http://www.tenouk.com/Bufferoverflowc/Bufferoverflow1.html
[4] USDA's C2 LEVEL OF TRUST information, http://www.ocio.usda.gov/directives/doc/DM3535-001.htm

[5] The Linux-PAM Guides, http://www.kernel.org/pub/linux/libs/pam/Linux-PAM-html/
[6] The first patch was released by Ingo Molnar of Red Hat and first released in May 2003, ExecShield information, http://people.redhat.com/mingo/exec-shield/
[7] NX/XD bit information at Wikipedia, http://en.wikipedia.org/wiki/NX_bit
[8] Geek.com, “Desktop NX/XD-enabled Intel processors already available”, http://www.geek.com/desktop-nxxd-enabled-intel-processors-already-available/?rfp=dta
[9] linuxfromscratch.org, Position Independent Executables (PIE) information, http://www.linuxfromscratch.org/hlfs/view/unstable/glibc-2.6/chapter02/pie.html
[10] Security-Enhanced Linux homepage at National Security Agency (NSA)/Central Security Service (CSS), http://www.nsa.gov/selinux/
[11] Proceedings of the GCC Developers Summit, Ottawa, Ontario Canada, May 25–27, 2003, gccsummit-2003-proceedings.pdf
[12] Homepage of The PaX Team, http://pax.grsecurity.net/
[13] kerneltrap.org, "Linux: PaX vs. ExecShield, An ExecShield Perspective", January 20, 2005 - 6:40pm, by Jeremy, http://kerneltrap.org/node/4590
[14] kerneltrap.org, "Pax vs. ExecShield: Blowing away the smoke", July 9, 2005 - 5:59am, by bluefoxicy on July 9, 2005 - 5:59am, http://kerneltrap.org/node/5396
[15] Rationale for TR 24731 Extensions to the C Library Part I: Bounds-checking interfaces, www.open-std.org/JTC1/SC22/WG14/www/docs/TR24731-Rationale.pdf
[16] ISO/IEC WDTR 24731-2, Specification for Safer C Library Functions — Part II: Dynamic Allocation Functions, www.open-std.org/jtc1/sc22/wg14/www/docs/n1193.pdf
[17] Specification for Safer, More Secure C Library Functions, ISO/IEC draft Technical Report, www.open-std.org/jtc1/sc22/wg14/www/docs/n1135.pdf
[18] The LOCK project, O. S. Saydjari, J. M. Beckman, and J. R. Leaman. LOCK Trek: Navigating Uncharted Space. In *Proceedings of the 1989 IEEE Symposium on Security and Privacy*, pages 167-175, 1989.
[19] Distributed Trusted Mach (DTMach), T. Fine and S. E. Minear. Assuring Distributed Trusted Mach. In *Proceedings IEEE Computer Society Symposium on Research in Security and Privacy*, pages 206-218, May 1993.
[20] The Distributed Trusted Operating System (DTOS) project, S. E. Minear. Providing Policy Control Over Object Operations in a Mach Based System. In *Proceedings of the Fifth USENIX UNIX Security Symposium*, pages 141-156, June 1995.
[21] The Distributed Trusted Operating System (DTOS) Home Page http://www.cs.utah.edu/flux/fluke/html/dtos/HTML/dtos.html
[22] University of Utah, The Flux Research Group, http://www.cs.utah.edu/flux/
[23] University of Utah, Fluke: Flux µ-kernel Environment, http://www.cs.utah.edu/flux/fluke/html/index.html
[24] Flask: Flux Advanced Security Kernel, http://www.cs.utah.edu/flux/fluke/html/flask.html
[25] Role Set Based Access Control, RSBAC, MAC kernel security enhancement project for Linux. http://www.rsbac.org/why
[26] Multi Level security (MLS), “SELinux and MLS: Putting the Pieces Together”, by Chad Hanson, Trusted Computer Solutions, Inc.
[27] Trusted Computing Platform Alliance (TCPA), an initiative led by Intel, http://www.trustedpc.org/
[28] Trusted Computing FAQ, “TC / TCG / LaGrande / NGSCB / Longhorn / Palladium / TCPA”, Version 1.1, August 2003, by Ross Anderson, http://www.cl.cam.ac.uk/~rja14/tcpa-faq.html
[29] IBM PCI Cryptographic Coprocessor, http://www-03.ibm.com/security/cryptocards/pcicc/overview.shtml
[30] The Bastille Linux homepage, http://www.bastille-linux.org/
[31] Immunix homepage, http://www.immunix.org/
[32] Trustix Secure Linux homepage, http://www.trustix.org/
[33] OpenWall, Linux kernel patch from the Openwall Project, A security-enhanced GNU/Linux-based server platform, http://www.openwall.com/
[34] HP Secure OS Software for Linux, http://h20331.www2.hp.com/enterprise/cache/4231-0-0-0-121.html
[35] Commercial version of TripWire, http://www.tripwire.com/products/index.cfm
[36] Open Source Tripwire® software. The project is based on code originally contributed by Tripwire, Inc. in 2000."http://sourceforge.net/projects/tripwire/
[37] Argus PitBull homepage, http://www.argus-systems.com/
[38] LSM project homepage, http://lsm.immunix.org/
[39] Real-time LSM project page, http://sourceforge.net/projects/realtime-lsm/
[40] Linux Intrusion Detection System, LIDS, http://www.lids.org/
[41] Firestarter, http://www.fs-security.com/
[42] Shoreline Firewall, http://www.shorewall.net/
[43] PCX Firewall, A Toolkit of perl libraries that allow you to define firewall rules for the Linux netfilter/iptables subsystem. http://pcxfirewall.sourceforge.net/
[44] StackGuard, Stack-smashing protection, http://immunix.org/stackguard.html
[45] Trusted Computing (TPM driver in 2.6 kernel, TSS & TPM-tools open sourced), Trusted Platform Module (TPM) Specifications, https://www.trustedcomputinggroup.org/specs/TPM/
[46] Sans.org, Linux RootKits For Beginners - From Prevention to Removal, http://www.sans.org/reading_room/whitepapers

/linux/901.php

| Winsock |< Other TCP/IP Security Related Info |Internet Protocol version 6 (ipV6) >| Main |
Site Index | Download |