Global Information Assurance Certification Paper

Building Blocks for LINUX-PAM Authentication

GIAC Security Audit Essentials (GSAE)
Assignment 1 version 1.1 Option 1

Lila Zurzolo

February 2003

Table of Contents

1.0 Abstract:

Many Linux distributions use Linux-PAM (Pluggable Authentication Modules for Linux) as the mechanism for authentication. In today's world of cyber space wars launching attacks against the vulnerabilities in confidentiality, availability and integrity of a system we need to understand how our system's authenticate and who is using what. Linux uses PAM to do just that. PAM can be used to tighten security on a system to the point no one can get in, including root, or loosen security so that anyone can get in with no password. Because most Linux distributions now use PAM, certainly the more popular ones, it behooves us to understand the basics of how PAM works and how to use it.

PAM has the potential to be a real security vulnerability or asset. I'm afraid that most system administrators don't understand PAM, how to set it up, or even that it is the authentication mechanism on their systems. Using the defaults, as is usually being done, could lead to a vulnerability exploit as soon as it is noticed that most systems are set up that way. The one saving grace of the default configuration is that PAM tends to error on the paranoid side. One might wonder why there is such a thing as PAM when we have SSH, Kerberos, secure ID cards, and several other secure ways of authentication to a system. What can PAM do to lessen the risks and threats against a host? Because PAM is so widely distributed with Linux and other UNIX systems it is important to understand the nature of the beast so that this handy tool doesn't lead to more vulnerabilities on a system. This paper will cover a brief history of PAM, what it is, how it is used, and an example of configuration files. It is my goal to show the building blocks of what PAM is to get a greater understanding of PAM and therefor be able to use it to a fuller extent.

2.0 History:

"PAM was defined and developed in 1995 by Vipin Samar and Charlie Lai of Sun Microsystems, and has not changed much since. In 1997, the Open Group published the X/Open Single Sign-on (XSSO) preliminary specification, which standardized the PAM API and added extensions for single (or rather integrated) sign-on. At the time of this writing, this specification has not yet been adopted as a standard." [2]http://www.freebsd.org/doc/en_US.ISO8859-1/articles/pam/index.html#PAM-INTRO. It was first released with Solaris 2.3 as a private internal interface. [1]OAF-RFC 86.0 p. 16. Linux-PAM came into existence with Linux in the RedHat 4.0 release in 1996 and has been an integral part of most Linux distribution ever sense. [3]http://lwn.net/2001/0927/history.php3. An incomplete list on Linux-PAM users are: Caldera, Debian 2.2 (alpha, arm, i386, ppc and sparc, sparc64), FreeBSD 3.2, Red Hat Linux 4.0, (all platforms that they support), SuSE Linux 6.2 (all platforms that they support), Apple OS-X ,and MSC.Linux. [4]http://www.kernel.org/pub/linux/libs/pam/whereislinuxpam.html.

The reason for the development of PAM was to be able to separate out the security aspect of a service from the application that way the applications wouldn't have to be recompiled every time a new authentication method or security policy needed to be added or changed. This gives greater control to the systems administrators over their system's authentication methods and ease of changing methods without having to reinstall or recompile services when these changes are implemented. For example you have the application "login" and can decide to authenticated it through /etc/passwd & shadow files, password database, kererbos server, secure ID card, one time password, bio-scan, or any/all of the proceeding without recompiling the login application for the specific means of authentication.

3.0 What is PAM:

Pluggable Authentication Modules is a method for assigning how applications authenticate a user. It is not a security program like ssh but a mechanism of tying programs into the system and making the decisions on which type of authentication to use, what resources to assign, and also to log the event. As the name indicates it uses modules that can be plugged into the mechanism as they become available or need updating. PAM can handle the determination of authentication mode, logging of access, management of account, control user sessions, and manage password updates, without having to recompile the applications. This leads to being able to "allow a system administrator to add new authentication methods simply by installing new PAM modules, and to modify authentication policies by editing configuration files". [2]http://www.freebsd.org/doc/en_US.ISO8859-1/articles/pam/. But PAM is quite complex and not easy to understand. Because of its complication, problems do arise when changing the configuration files, like completely locking out anyone from the system. As is stated in The Linux-PAM System Administrators' Guide section 5.1 [5]http://www.kernel.org/pub/linux/libs/pam/Linux-PAM-html/pam-5.html#ss5.1. "The first thing you have to realize is that this happens to 50% of users who ever do anything with PAM. It happened here, not once, not twice, but three times, all different, and in the end, the solution was the same every time." Most users can and do lock themselves out of their own systems when reconfiguring PAM.

The diagram below is one of the best diagrams I've seen to demonstrate very basically what PAM does. Taken from the article Pluggable Authentication Modules for Windows NT by Ann Arbor University of Michiagan [6]http://www.citi.umich.edu/u/itoi/ni_pam_usenix.pdf page 5 Figure 2.4. This diagram flows from left to right. On the left are services/applications asking to be authenticated to the system. On the right are different types of authentication available to the system. PAM intercepts the application and looks in the Configuration Table for the type or types of authentication allowed for that

application and how the session is to be controlled. It then uses one or more PAM_*modules* to communicate with the authentication mechanism called for in the specific application's Configuration Table. If there is a match the application will then use that method to authenticate to the system before granting access.

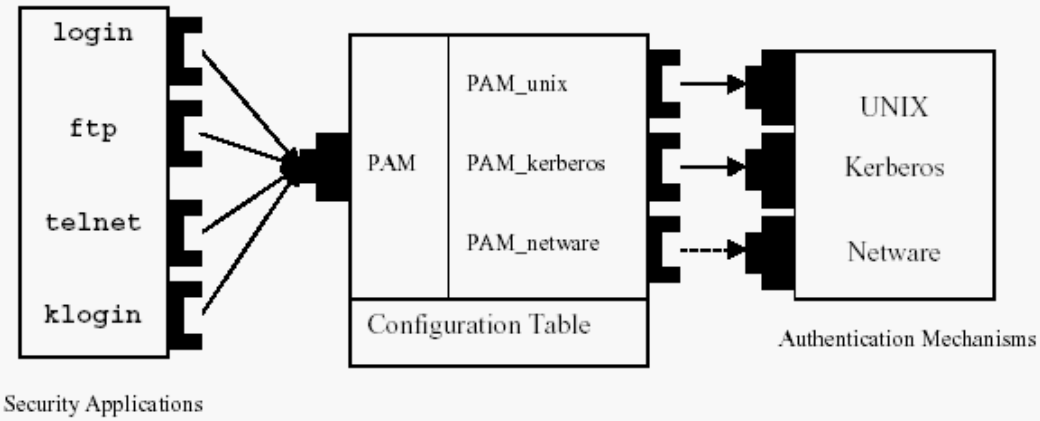

Figure 2.4: UNIX authentication with PAM

4.0 Basic How-To of PAM

To administrate PAM one needs to know the basic building blocks of PAM. These are defined in many How-To documents including those mentioned in the reference section. I have used all these references and tied their definitions together to get a plainer understanding of the parts PAM uses.

There are four basic types of management that PAM takes care of, which are authentication certification, account management, session management, and password management. "This reasoning leads to a partitioning of the entire set of interfaces into four areas of functionality: (1) authentication, (2) account, (3) session, and (4) password." [1]OAF-RFC 86.0 p. 6 These four management task types, sometimes called modules, are referred to in the configuration files respectively as auth, account, session, and password (see section 4.3 for further definition). The question I kept asking was, "But how does PAM take care of these four different management tasks?". I needed a greater understanding of what PAM was and how it worked.

Besides the four basic management tasks that PAM does, I have determined there are four building blocks that work together to be PAM. To begin with this is where I had the most difficulty in understanding PAM. As a systems administrator I felt I needed to understand more than just how to modify the configuration files. I needed to know what drove PAM and how it worked on a system. When I first started researching this subject I thought PAM was just another daemon working on the system using configuration files to make its decisions, however it is not a daemon or any other program running on the

system. Then I figured it was part of the kernel and went looking for it in the message logs and proc files, but to no avail. What I didn't know was that PAM is based on a functional programming scheme. I now understand that PAM is an intricate part of the system using mostly dynamically linked libraries that get called when a service or application has been programmed to use them. This understanding made all the pieces fall together for me, and hopefully will shed some light for someone else on how and why PAM is configured the way it is. I will address PAM from the following main building blocks that I have discovered instead of just showing how to set up the configuration files. In general this way of explaining PAM differs from the norm, which tends to just explain PAM by showing you how to set up the configuration files.

The main building blocks of PAM are:

- **PAM aware applications/services**.
- **PAM libraries** on the system, usually located in /lib/security/pam_*.so
- **PAM configuration file or files**; /etc/pam.conf or /etc/pam.d/*app_name*
- **Information data files** or databases that a library may look for or need to access.

4.1 PAM Aware Applications/Services

"In order to work with PAM, applications are developed to be independent of any particular authentication scheme." [7]Standfield & Smith, p.484. You can use PAM if the application has the PAM functions built into it or you have the source code for the application and can therefore build in the functions yourself.

The PAM aware application will have a call to the PAM library and then in turn the libraries/modules do the work of authentication according to the PAM configuration file specification. These modules are dynamically linked functions that can be called upon by the applications. Anyone wanting to program an application to be PAM aware or just curious on how it works, can get further information from "The Linux-PAM Application Developers' Guide" by Andrew G. Morgan at [8]http://www.kernel.org/pub/linux/libs/pam/Linux-PAM-html/pam_appl.html.

4.2 PAM Libraries

The PAM library modules are at the heart of what makes PAM work. Which are located in /lib/security/pam_*.so on most Linux distributions. These are the dynamically linked functions that are called to do the PAM configured tasks. Different modules are developed to work with one or more of the four basic management task types; *auth*, *account*, *session*, and *password*.

Not being a programmer, but having a basic understanding of programming, I had the most trouble figuring out that PAM was a functional programming scheme and then trying to understand what these modules where doing. This was quite a cognitive shift between sys-admin way of doing things and programming, as in script writing vs. functional programming. I was used to using daemons and their configuration files or kernel controls when utilizing something new. I found that to understand how to use the different modules I needed to understand that the modules where developed for different management task types. In general the module variables that will differ accordingly are: types of tasks, control flags and arguments. You must tailor your functions along with these given variables, according to the desired library/module.

Basically when a *management task* calls a module it will perform the task it was designed to perform according to the type of task, control flag, and any arguments passed to it. What the module is developed for and what arguments it will except are defined in the documentation directory /usr/share/doc/pam-*version#*/txts/README.pam_*module-name,* the man pages, or documentation on the web at several sites including [9]http://www.freebsd.org/doc/en_US.ISO8859-1/articles/pam/pam-freebsd-modules.html and [10]http://www.kernel.org/pub/linux/libs/pam/modules.html. The reason I give so many references is because I couldn't find one central location for this documentation on all the modules available for use. The very first source would be the most logical place to find the desired information. However, not all systems have had all the documentation loaded on to them during install. Linux is very integrated with PAM and therefore it isn't just a package to install on the system, since most Linux distributions now come with PAM already configured.

It is important to read the documentation on any modules you want to use in order to make sure you are using it in the best way for your system. It is the module that does the work of PAM and has the potential to either let you lock down your system so you can't get in or open it up to the world even with kerberos or some other tight security authentication method on your systems.

To learn more about programming *modules* you can read "The Linux-PAM Modules Writers' Guide" at [10]http://www.kernel.org/pub/linux/libs/pam/Linux-PAM-html/pam_modules.html.

4.3 PAM Configuration

There are two ways to configure PAM. Either by a single file, /etc/pam.conf or several files in /etc/pam.d/, these are given the name of the application that is using PAM to authenticate it to the system. Both configuration methods are allowed to use comment statements, these are designated with a proceeding "#" and applies to the end-of-line.

The syntax for the configuration files in **pam.d** has four fields in each file.

type control-flag module-path module-arguments

The **pam.conf** file has five fields, the noticeable difference is that instead of a file for each service, the application name is the first field. It must be noted that on older systems the configuration might be in one file instead of a directory.

application-name type control-flag module-path module-arguments

Using the pam.d directory is now the preferred way of setting up PAM. This actually helps reduce the risk of corrupting the one file and then locking you out of the system. It is also easier to see what services are authenticating through PAM by listing the files in the directory. Those services you don't want to authenticate to the system can be renamed or removed. I will now deal with pam.d directory configuration type files.

4.3.1 Four Fields of pam.d files Defined

4.3.1.1 TYPE: The first field in the configuration file is the TYPE field, sometimes called the module field. As previously stated this field has four types, auth, account, session, and password. The four types of tasks that PAM recognizes and what they do are:

- **auth** This is the *task type* that PAM uses to know how to authenticate a user to the system's authentication method(s). The auth module actually does two tasks. The first is to determine that the user is who they say they are, by passwords or what ever means of authentication called for in the configuration file. Second the module sets up the credentials for the user, such as user ID, group memberships, and resources. [2]http://www.freebsd.org/doc/en_US.ISO8859-1/articles/pam/pam-essentials.html
- **account** This task checks the "non-authentication-related issues of account availability, such as access restrictions based on the time of day or the server's work load." In other words it verifies the accounts availability. [2]http://www.freebsd.org/doc/en_US.ISO8859-1/articles/pam/pam-essentials.html
- **session** The session task handles what is needed to set-up and tear-down a session. Including logging and setting up any mounts. [2]http://www.freebsd.org/doc/en_US.ISO8859-1/articles/pam/pam-essentials.html
- **password** PAM uses this task to "change the authentication token associated with an account, either because it has expired or because the

user wishes to change it." [2]http://www.freebsd.org/doc/en_US.ISO8859-1/articles/pam/pam-essentials.html

4.3.1.2 CONTROL FLAG: The second field in the configuration file "is used to indicate how the PAM library will react to the success or failure of the module it is associated with… the *control flags* determine the relative importance of each module (*task*)." [4]http://www.kernel.org/pub/linux/libs/pam/Linux-PAM-html/pam-5.html#ss4.2 . The power of the *control flag* is that tasks can be *stacked*. That is for example there can be several *auth* tasks referred to in a configuration file, executed in series according to the order they are listed. Depending on if the module passes or fails the control flag then determines what PAM will tell the application. Being able to stack the *type* is where PAM gets its versatility. The *control flag* will determine the behavior of the stack. There are two methods of using *control flags* I will just explain the traditional and simpler method, which uses a single keyword. The more complex method of the *control flag* is delimited with square brackets and consists of a series of *value=action* tokens. The newer method has more control and is therefor more complicated. If you wish to use it or have an understanding of how it works read section 4.1 Configuration file syntax in "The Linux-PAM system Administrators' Guide" at [4]http://www.kernel.org/pub/linux/libs/pam/Linux-PAM-html/pam-5.html#ss4.2. Understanding the logic behind the *control flags* is imperative to understanding the configuration files. There are four keywords:

- **required** "Success of the of the module is required for the module-type facility to succeed. Failure of this module will not be apparent to the user until all of the remaining modules (of the same module (*task)*-type) have been executed." [4]http://www.kernel.org/pub/linux/libs/pam/Linux-PAM-html/pam-5.html#ss4.2
- **requisite** "Like required, however, in the case that such a module returns a failure, control is directly returned to the application. The return value is that associated with the *first* required or requisite module to fail. Note, this flag can be used to protect against the possibility of a user getting the opportunity to enter a password over an unsafe medium. It is conceivable that such behavior might inform an attacker of valid accounts on a system. This possibility should be weighed against the not insignificant concerns of exposing a sensitive password in a hostile environment." [4]http://www.kernel.org/pub/linux/libs/pam/Linux-PAM-html/pam-5.html#ss4.2
- **sufficient** "Success of this module is deemed `*sufficient*' to satisfy the Linux-PAM library that this module-type has succeeded in its purpose. In the event that no previous required module has failed, no more `*stacked*' modules of this type are invoked. (Note, in this case subsequent required modules are not invoked.). A failure of this module is not deemed as fatal to satisfying the application that this module-type has succeeded."

[4]http://www.kernel.org/pub/linux/libs/pam/Linux-PAM-html/pam-5.html#ss4.2

- **optional** "as its name suggests, this control-flag marks the module as not being critical to the success or failure of the user's application for service. In general, Linux-PAM ignores such a module when determining if the module stack will succeed or fail. However, in the absence of any definite successes or failures of previous or subsequent stacked modules this module will determine the nature of the response to the application." [4]http://www.kernel.org/pub/linux/libs/pam/Linux-PAM-html/pam-5.html#ss4.2

It must be strongly noted that the order of the *task types* is of utmost importance. You need to list the *modules* that are *required* to be tested first. If a *sufficient* module is placed in the stack before a module that is *required* for security then the application could be accepted without the proper security being applied.

4.3.1.3 MODULE-PATH The *module path* is the actual path to the library *module* you want to use for a specific *task type*. Remember what was said earlier, not all library *modules* can be used with all the *task types.* If a *task* calls a *modules* and that *module* is not programmed for that *task*, then that line in the configuration file is ignored and PAM moves on to the next line. Most of the library *modules* are located in /lib/security/pam_*.so on Linux systems. It is very important to understand just what the *module* is doing and what *arguments* can be passed to it. The documentation, on a properly installed Linux system, is located in /usr/share/doc/pam-*version#*/txts/README.pam_*module-name* this document defines which *task type* and which *arguments* are acceptable.

4.3.1.4 MODULE-ARGUMENTS As the name implies these are the *arguments* that the *modules* will accept. Again these are defined in the documentation for the *modules.* Take special care in assigning *arguments*, for example the *argument* nullok used with an authentication *module* means that the *module* will pass with no password in the passwd or shadow file.

4.4 Information Files

The information files that PAM uses are the same files that are used by the applications/services. There are a few files specific to some of the modules but these will be referred to in the documentation of that module/library. This is another good reason for understanding the libraries and what any changes to the PAM configuration files will do.

Some examples of data files on a Linux system and what modules might use these particular data files are listed below. All the below information is from http://www.alphacentauri.demon.nl/linux/basic/pam

- The module pam_securetty.so check the /etc/securetty file to see if the terminal being used to log into as root is in the file. If not the connection is rejected.

- The pam_rhosts_auth.so module uses the ~/.rhosts file to see if access is granted.

- The pam_pwdb.so module is used with the account action to write accounting information to syslog and to also update /etc/utmp and /etc/wtmp files.

- The pam_unix.so uses the /etc/passwd and /etc/shadow files or a password database.

- "The /etc/security directory contains control files that describe PAM's environment and operations." One of the files, limits.conf "shows limits that PAM should put on groups of users when they finish authentication. In this file you can limit the number of processes, amount of memory, and other resources that users will have available." [11] http://www.alphacentauri.demon.nl/linux/basic/pam

5.0 An example of PAM

It was not my intention to give a how-to on using PAM but rather an understanding of how PAM is integrated into a system and therefore give a better understanding of what the configuration files are really doing. There are many sources on how to set up configuration files I will just quote from one to show an example of PAM at work.

The following example is taken from [12] http://www.redhat.com/docs/manuals/linux/RHL-7.3-Manual/ref-guide/s1-pam-samples.html

```
#%PAM-1.0
auth       required   /lib/security/pam_securetty.so
auth       required   /lib/security/pam_unix.so   shadow nullok
auth       required   /lib/security/pam_nologin.so
account    required   /lib/security/pam_unix.so
password   required   /lib/security/pam_cracklib.so retry=3
password   required   /lib/security/pam_unix.so   shadow nullok use_authtok
session    required   /lib/security/pam_unix.so
```

The first line is a comment as is any line starting with a # character. Lines two through four stacks three modules for login authentication.

```
auth       required   /lib/security/pam_securetty.so
```

This line makes sure that if the user is trying to log in as root, the tty on which they are logging in is listed in the /etc/securetty file, if that file exists.

auth required /lib/security/pam_unix.so shadow nullok

This line causes the user to be asked for a password and then checks the password using the information stored in /etc/passwd and, if it exists, /etc/shadow. The pam_unix.so module automatically detects and utilizes shadow passwords stored in /etc/shadow to authenticate users. Please refer to the the Section called *Shadow Utilities* in Chapter 6 for more information on shadow passwords. The argument nullok instructs the pam_unix.so module to allow a blank password.

auth required /lib/security/pam_nologin.so

This is the final authentication step. It checks to see if the file /etc/nologin exists. If nologin does exist and the user is not root, authentication fails. Note: In this example, all three auth modules are checked, even if the first auth module fails. This prevents the user from knowing at what stage their authentication failed. Such knowledge in the hands of an attacker could allow them to more easily deduce how to crack the system.

account required /lib/security/pam_unix.so

This line causes any necessary account verification to be done. For example, if shadow passwords have been enabled, the account component of the pam_unix.so module will check to see if the account has expired or if the user has not changed their password within the grace period allowed.

password required /lib/security/pam_cracklib.so retry=3

If a password has expired, the password component of the pam_cracklib.so module prompts for a new password. It then tests the newly created password to see whether it can easily be determined by a dictionary-based password cracking program. If it fails this test the first time, it gives the user two more chances to create a strong password, due to the retry=3 argument.

password required /lib/security/pam_unix.so shadow nullok use_authtok

This line specifies that if the program changes the user's password, it should use the password component of the pam_unix.so module to do so. This will happen only if the account portion of the pam_unix.so module has determined that the password needs to be changed — for example, if a shadow password has expired. The argument shadow tells the module to create shadow passwords when updating a user's password. The argument nullok instructs the module to allow the user to change their password from a blank password, otherwise a null password is treated as an account lock. The final

argument on this line, use_authtok, provides a good example of how one can stack PAM modules. This argument tells the module not to prompt the user for a new password. Instead it is to accept any password that passes through the previous password module. This way all new passwords must pass the pam_cracklib.so test for secure passwords before being accepted.

```
session    required   /lib/security/pam_unix.so
```

The final line specifies that the session component of the pam_unix.so module will manage the session. This module logs the username and the service type to /var/log/messages at the beginning and end of each session. It can be supplemented by stacking it with other session modules if you need more functionality.

One last example of a module used with Linux-PAM should be mentioned. That is the module /lib/security/pam_stack.so. This module is unique to Linux-PAM and is used to further stack any *task type* with the *task type* stack of another PAM configuration file; typically the system-auth service file is used. The following is an example of how it might be used in the previous example.

```
#%PAM-1.0
auth        required   /lib/security/pam_securetty.so
auth        sufficient /lib/security/pam_stack.so service=system-auth
auth        required   /lib/security/pam_unix.so   shadow nullok
auth        required   /lib/security/pam_nologin.so
account     requisite  /lib/security/pam_stack.so service=system-auth
account     required   /lib/security/pam_unix.so
password    required   /lib/security/pam_cracklib.so retry=3
password    required   /lib/security/pam_unix.so   shadow nullok use_authtok
session     required   /lib/security/pam_unix.so
session     optional   /lib/security/pam_stack.so service=system-auth
```

In this example line two will now stack the auth lines from the file system-auth in up to this point. If it passes all the requirements from system_auth's auth stack then none of the other auth lines following this point will be run, if it fails then the rest of this stack will continue. Now the account stack is run first from the file system-auth. If any failures in that file's stack from the account lines are found then the rest of the account stack in this file will not run. The password stack in the system-auth file does not affect the password stack here because it was not included with the pam_stack.so. The session stack will run both the required session here and the one in the system-auth file as an optional setting; no failures will effect the session.

It should be noted that for consistent authentication of several services the system-auth file could be used. Just be careful that it works the way you want it to and not as a catch all or you might be opening your system up where you

didn't mean. Included in the List of References are several good links to learn more about configuring the PAM files.

6.0 List of References:

1. RFC 86.0, PAM standard, http://www.opengroup.org/tech/rfc/rfc86.0.html
2. Pluggable Authentication Modules, Dag-Erling Smørgrav http://www.freebsd.org/doc/en_US.ISO8859-1/articles/pam/pam-essentials.html
3. http://lwn.net/2001/0927/history.php3
4. Linux-PAM Users http://www.kernel.org/pub/linux/libs/pam/whereislinuxpam.html
5. http://www.kernel.org/pub/linux/libs/pam/Linux-PAM-html/pam-5.html#ss5.1
6. Pluggable Authentication Modules for Window NT http://www.citi.umich.edu/u/itoi/ni_pam_usenix.pdf
7. Vicki Stanfield & Roderick Smith, "Linux System Administration 2nd edition" , pp 484
8. The Linux-PAM Application Developers' Guide http://www.kernel.org/pub/linux/libs/pam/Linux-PAM-html/pam_appl.html
9. http://www.freebsd.org/doc/en_US.ISO8859-1/articles/pam/pam-freebsd-modules.html
10. Modules/Applications available or in progress http://www.kernel.org/pub/linux/libs/pam/modules.html .
11. http://www.alphacentauri.demon.nl/linux/basic/pam
12. http://www.redhat.com/docs/manuals/linux/RHL-7.3-Manual/ref-guide/s1-pam-samples.html

Additional Resources:

13. M-009: Red Hat Linux PAM Vulnerability http://www.ciac.org/ciac/bulletins/m-009.shtml
14. Linux-PAM http://www.kernel.org/pub/linux/libs/pam/
15. The Linux-PAM System Administrators' Guide http://www.us.kernel.org/pub/linux/libs/pam/Linux-PAM-html/pam.html#toc3
16. User Authentication HOWTO http://www.tldp.org/HOWTO/User-Authentication-HOWTO/index.html
17. http://www.alphacentauri.demon.nl/linux/basic/pam

Auditing Linux Workstation befind Firewall

GIAC IT Security Audit Kickstart (GSAE)
Assignment 2 version 1.1

Lila Zurzolo

February 2003

Table of Contents

1 Objectives

The overall goal of this document is to provide guidance for the reader in the secure configuration of the Linux RedHat 7.0-7.3 operating systems' installation. It is to assist in removing common and known security vulnerabilities. This document was derived in part from the Computer Security Resource Center (CSRC) document "UNIX Security Checklist (version 1.1)" available from http://csrc.nist.gov/secpubs/unix_security_checklist.txt.

This document may be considered a formal Corporation configuration guide to ensure that Linux workstations are secured to a level that supports Corporation requirements. However, any vendor supplied Security Administrator's Guide should be used in conjunction with this document to identify optimal processes for securing a particular Linux Workstation.

The main concern is the outsider threat of access through the firewall by using insecure services. It has been determined that the probability of theft and/or access to the console of the Linux Workstation is low and therefore no power-on password or bios password is required. It is the objective of this audit to determine if any services are running in an insecure mode.

2 Scope

This document is for Linux workstations in access-controlled areas behind the corporate firewall with access to the network. This document should be used to test the Linux Workstations independently; this report will cover only one station.

3 Checklist

3.1 Physical Security

The operating system of a computer cannot protect data or availability of the operating system form those who have physical access to the computer system.

- DO verify the workstation is inside the restricted access area.
- DO disable booting from removable media.
- SUGGEST disable the Boot password on systems in locked areas.
- Do enable the Boot password on workstations that are not access controlled.
- Do change the Boot password from the manufacturer's default if it is not disabled.

3.2 Operating System Configuration

The operating system of a workstation must be installed/configured in a secure manner. The firewall does not block a badly configured workstation from being exploited. The workstation needs to have current configurations and patches on

them for optimal security. The services and configurations mentioned in this section are a baseline for system security and not all-inclusive of best practices. If these services are not at least configured accordingly, the system could be vulnerable to exploit. The corporation has only approved RedHat Linux 7.0-7.3 for use on the Linux workstations these guidelines only address that OS.

3.2.1 Security Patches

- ENSURE that the security patches for Linux kernel are up to date.
  Fix the known security problems through installation of vendor-supplied security patches. At the command prompt type *uname –a* to get the kernel version installed on the workstation. Check the rpms, with the *rpm-qa* command, loaded onto the system with the security patches listed for that kernel at http://rhn.redhat.com/errata/rh73-errata-security.html.

3.2.2 /etc/inetd.conf

- ENSURE that the permissions on this file are set to 600.
- ENSURE that the owner is root.
- DO disable any services that you do not require.
  To do this I suggest that you comment out ALL services by placing a "#" at the beginning of each line. Then enable only the ones you NEED by removing the "#" from the beginning of the line. In particular, it is best to avoid "r" commands and tftp, as they have been major sources of insecurities. For changes to take effect, you need to restart the *inetd* process. At the command prompt type */etc/rc.d/init.d/inetd restart*.

3.2.3 Fingerd

- ENSURE that finger is not installed on the workstation.

3.2.4 "r" commands

- DO disable all "r" commands (rlogin, rsh, etc.)
- ENSURE there are no .rhosts files on the workstation.
  These are usually stored in users home directories as ~/.rhosts.

3.2.5 /etc/services

- ENSURE that the permissions on this file are set to 644.
- ENSURE that the owner is root.

3.2.6 Sendmail

- ENSURE that sendmail is off for all run levels.
  This can be checked with the command *chkconfig –list sendmail*. If it is on for any run level disable it with *ckconfig –level 2345 sendmail off* then disable the running sendmail with */etc/rc.d/init.d/sendmail stop*.
- ENSURE that if you have a line starting with :OW" in the /etc/sendmail.cf file, it only has a "*" next to it.

3.2.7 ftp

- ENSURE that ftp is off for all run levels.

3.2.8 tcp_wrapper

- SUGGEST use this package whenever possible.
- Enable PARANOID mode.
- Consider running with the RFC931 option.
- Deny all hosts by putting "all:all" in the /etc/hosts.deny file and explicitly list trusted hosts who are allowed access by listing them in the /etc/hosts.allow file.
- SUGGEST wrap all TCP services that you have enabled in /etc/inetd.conf.
- SUGGEST consider wrapping any udp servies you have enabled in /etc/inetd.conf, you will have to use the *nowait* option.

3.2.9 /etc/pam.d

- ENSURE that *nullok* is not used in any authentication task, in any of the configuration files in /etc/pam.d.

3.2.10 Legal Notice at logon

- ENSURE that the Corporation's legal notice is displayed when users logon to the system.

  If the Corporate legal notice is not displayed at logon add the notice to the /etc/motd file.

3.2.11 Password Policy

Linux workstations must implement a policy to protect and control passwords associated with user accounts.

- DO use password shadowing or a third party product, such as kerberos.
- ENSURE that permissions for /etc/shadow are set for 400.
- ENSURE that the owner is root.
- ENSURE that permissions for /etc/passwd are set for 444.
- ENSURE that the owner is root.
- DO set maximum password age to 365 days.
- SUGGEST set minimum password age to 0 days.
- DO set minimum password length to eight characters.
- SUGGEST configure so that password uniqueness is enforced to the last five passwords.
- DO configure so that Account Lockout will occur after five invalid logon attempts.
- ENSURE the user of the Linux workstation is registered in the Corporate's database as a current active employee and owner of the workstation.
- DO use machine-generated passwords.

- ENSURE all accounts have passwords.
- ENSURE any account without a password is locked out.

3.2.12 Special Accounts

- ENSURE there are no shared accounts other than root with approved z-accounts.
- DISABLE all guest accounts.
- DO assign non-functional shells (such as /bin/false) to system accounts such as bin, daemon and the sync account.

3.3 Root Account

The root account is the most powerful account on the workstations. With root privileges the system is wide open, therefore this account must be the most secure.

- DO restrict the number of people who know the root password to the LAN Sys-Admins.
- DO restrict the number of people who have z-accounts. Typically this is limited to at most 3 or 4 people including the Sys-Admins.
- DO *su* from user accounts to z-accounts rather than logging in as root.
- ENSURE root does not have a ~/.rhosts file.
- ENSURE "." is not in root's path.
- ENSURE root's login files do not source any other files not owned by root or which are group or world writable.
- ENSURE root's cron job files do not source any other files not owned by root or which are group or world writable.

3.4 General Services

These are additional services that must be limited or removed completely from the workstations due to the outsider threat of exploitation, denial of service, or/and theft of data.

3.4.1 Remote Access Service

- DISABLE all dial-in ports.

3.4.2 Secure Terminals

- ENSURE that the permissions for the /etc/security file are 644.
- ENSURE that this file is owned by root.
- ENSURE that secure option is removed from all entries that don't need root login capabilities.

3.4.3 Secure Shell Client and Server

- ENSURE that the Secure Shell Client (ssh) and Sever daemon (sshd) are installed.
- ENSURE that kerberos is installed.

3.5 Other Network Security Features

Any availability of shared resources is also a vulnerability to the system if not configured properly; or in some cases just because they are available. These shares must be controlled in order to restrict access to only those with the permission to do so.

3.5.1 World Wide Web (WWW) – httpd

- ENSURE that the httpd daemon is not running.
  No Linux workstation is approved to run a web server.

3.5.2 NFS

- DO use /etc/exports file to export ONLY the file systems you need to export.
- ENSURE that the *exports* file does not contain a "localhost" entry or an entry with the hosts name of the localhost.
- DO export to fully qualified hostnames only.
- ENSURE that export lists do not exceed 256 characters.
- DO export file systems read-only (-ro) whenever possible.
- ENSURE that the permissions of the /etc/exports file are 644.
- ENSURE that the file is owned by root.

4 Report

On February 18, 2003 an audit was performed on the Linux workstation, hostname *nemesis* which was assigned to Mr. Joe Newperson as his Linux workstation. The audit was conducted per the requirements outlined in this document. The findings as well as recommendations are presented here.

4.1 Findings:

The findings will be reported according to the sections outlined in the audit checklist.

4.1.1 Physical Security

The workstation was found to be physically located in Mr. Newperson's office inside the restricted area with a key lock for the only door that opens up into this office. The restricted area has a single person turnstile gate making it next to impossible for someone to tail gate in with someone else. The *nemesis* had the power-on password disabled. This office has the appropriate setup for what is called for in the corporation protocal, however Mr. Newperson did not have a key to the door and would leave it unlocked when he was gone. A power on password was added to *nemesis* by the sys-admin for the time being until Mr. Newperson can obtain a key to the office. Although this situation is unfavorable, it is however common due to the fact of the corporations

large size. Mr. Newperson was instructed in how his computer is not the only piece of equipment that should be locked down due to security reasons. Also that this is a prime example of how writing down security passwords could be compromised in this given situation and that ultimately it is his responsibility to make sure this does not happen. The boot order was found to be in correct configuration by not allowing removable media to boot first.

4.1.2 Operating System Configuration

The security patches where checked against the latest security rpm releases on the RedHat site. These where found to be up to date according to the RedHat security rpms.

The /etc/inetd.conf was not on the system, however /etc/xinetd.conf was and had the correct permissions and owner. The file contents had the unused servers commented out and only the bare essentials left in.

Finger was not on the workstation.

The r-command server daemons where not on the workstation and there was no .rhosts files on the system. The r-commands to log onto another host where present, but this presents no compromise for *nemesis* because it was not excepting rlogin.

The /etc/services file was checked for permissions and owner, both where set correctly. However there was no check called for in this document on seeing which ports are open on the system.

Sendmail was not running the daemon on the system. The /etc/sendmail.cf file was configured as outlined in this document.

There where no ftp daemons running on the system.

The tcp_wrapper is not running on this system. The daemon tcpd is used with inetd in order to check the /etc/hosts.allow and /etc/hosts.deny files. On this system the inetd service is replaced with xinetd which does the work of inetd using tcpd. Therefore tcp_wrappers are not needed on this sytem in its present configuration state. The /etc/hosts.allow and /etc/hosts.deny files where configured according to Corporation's standards.

In checking the /etc/pam.d configuration files it was discovered that there where some auth tasks set up with nullok arguments allowed. These configuration files where for services that where not running on the system. This is not an immediate security vulnerability because the

services are not currently on, however, if in the future they are turned on for any desired reason this could lead to an exploit.

The corporate legal notice was displayed at logon and had the correct wording.

Checked the /etc/passwd and /etc/shadow files for both permissions and owner and found them to be set correctly. Linux-PAM is setup to authenticate the user for all the login services. The password aging and length is to this documents specification. The 5 incorrect password attempt was tested and found to be working correctly, including logging of all incorrect password attempts. The passwd command is linked to a machine generated password generator. It was verified that Mr. Newperson did indeed use the machine-generated password, the link make it not possible for him to use a self created one. PAM is also set to enforce a new password for the last 5 password changes. All users in the password file where checked against the corporate's database for active employment status, one inactive employee was found in the password file. This was a severe security vulnerability due to the fact that this was a past sys-admin and there was still a z-account associated with the person. The accounts where locked immediately and the corporation's proper supervisors were notified of the finding. All the special accounts had /bin/false shells so that no one could use them for logging into the system.

4.1.3 Root Account

As previously mentioned, there was one z-account left on the system for an employee who was no longer working for the corporation. This was a serious finding. Everything else checked to be in correct order and working properly; therefore concluding that total integration had been achieved and that all security aspects leading up to this point were fulfilled. The crontabs didn't have any cron jobs that shouldn't be on the system or that where owned by anyone that shouldn't be on the system. There where no .rhosts files in ~/root/. All roots private files had proper permissions and owner set.

4.1.4 General Services

All the dial-in ports are inactive and modems are currently not installed on the workstation. The only network card was a GIGA-bit ethernet card and the connection is not hooked up to an ISDN phone only the LAN fiber connection.

The /etc/security is a directory with the following files and directories in it;

```
-rw-r--r--    1 root    root         1969 Nov  9  2001 access.conf
```

```
drwxr-xr-x  2 root   root       4096 Oct 24 15:01 console.apps
-rw-r--r--  1 root   root       2453 Nov  9  2001 console.perms
-rw-r--r--  1 root   root       2146 Nov  9  2001 group.conf
-rw-r--r--  1 root   root       1418 Nov  9  2001 limits.conf
-rw-r--r--  1 root   root       2862 Nov  9  2001 pam_env.conf
-rw-r--r--  1 root   root       2154 Nov  9  2001 time.conf
```

The console.apps directory has the following files;

Bindconf	kbdrate	printtool	sysctlconfig-gtk
Dateconfig	kisdndock	reboot	up2date
firewall-config	kppp	redhat-config-bindconf	up2date-config
gdmconfig	kuser	redhat-config-date	up2date-nox
gnome-lokkit	kwuftpd	redhat-config-printer-gui	v4l-conf
gnorpm-auth	linuxconf-auth	redhat-config-time	xcdroast
gtoaster	locale_config	redhat-config-users	xserver
halt	neat	rhn_register	
hwbrowser	poweroff	rp3-config	
internet-druid	printconf-gui	serviceconf	

All these are owned by root and have the permissions set to 644. The console.perms.file had the correct settings for the tty logins. The other files should be gone through to assure settings don't violate any security settings. They are mainly used for the COE session currently running on the system and are owned by root and only available to the user who is running the COE. Checking these files is beyond the scope of this document.

SSH and the sshd (secure shell client and server) are running on this system. Kerberos is also installed and configured on this system. It these services are set up to except both the system password and the corporate kerberos password for authentication with an ssh logon.

4.1.5 Other Network Security Features

The httpd daemon is not running on this system. There is no web server loaded onto the system.

The nfs service is not running on this system and the /etc/exports file is empty. There is no reason at this time for this system to share out any file systems.

4.2 Recommendations:

Most of the findings were not severe and easily corrected. The most severe vulnerability has its root cause of using a ghost image of an older system to set-up new workstations. It was determined that is the reason for an old sys-admin's account being on the system even though this is a new system. It is recommended that a new ghost image is generated with any corrections added.

Other discrepancies came from this document not being updated with the systems OS. The tcp_wrapper checks, the pam.d directory having files that aren't needed, and the /etc/security directory being a newer way of limiting resources. It is further recommended that the document be updated to reflect the new technology being used and to include a section on what ports should be open or closed.

The last finding had to do with the procedures for bringing in new employees. There should be a checklist to insure that all the resources they need are given to them before they are responsible for what could be corporate sensitive information. Give Mr. Joe Newperson a key.

5 Resources:

5.1 http://csrc.nist.gov/secpubs/unix_security_checklist.txt
5.2 http://www.cert.org
5.3 http://rhn.redhat.com/errata/rh73-errata-security.html
5.4 http://www.sans.org/top20/