Global Information Assurance Certification Paper

Password Complexity Enforcement and Auditing Enhancement in Red Hat Linux 7.3

GIAC Security Essentials
Certification (GSEC)
Practical Assignment
Version 1.4c

Option 2 - Case Study in
Information Security

Submitted by: George Anderson
Location: Washington, D.C.
Date: 17 December 2004

This paper describes my work to enhance password policy enforcement and implement C2 style logging/auditing capabilities on a network of Red Hat Linux 7.3 systems running a custom-developed, multi-user simulation.

Table of Contents

List of Figures

Abstract

This paper details the steps I took to overcome deficiencies of a customized Red Hat Linux 7.3 installation in the areas of password policy enforcement and audit trail generation. I was solely responsible for designing and implementing a solution to satisfy the requirements for this system.
The specific deficiencies were that password complexity enforcement was not verifiable and there was no facility for logging attempts to access arbitrarily specified file system objects. I was able to completely solve the password complexity issue, but I was able to only partially implement a solution for the logging issue. Logging of failed file system access attempts was incomplete and inconsistent, but given knowledge of its limitations, the additional logging was better than nothing. Both solutions are presented, along with some thoughts on what to be aware of regarding the partial logging solution.

Before

The system consists of a small local area network (LAN) of Intel Pentium 4 PCs running Red Hat Linux 7.3. Even though this LAN was built recently (August 2004), the target application, a custom-developed multi-user simulation (not developed by us), required the use of the older version of Linux.
The PCs are connected through a single 100 Megabit switch by Cat5e cabling. One of the PCs has extra disk space and acts as a Network File System (NFS) server and provides the home directories for normal users. The LAN is located in a secured room requiring specially coded badges for entry. The LAN is never connected to the Internet or any other network.

Current Security Posture

All systems require BIOS passwords and are configured to boot from the hard drive only. All of the systems on the LAN are running Red Hat Linux 7.3. The Grand Unified Boot Loader (GRUB) is used to control kernel booting parameters. Editing of boot options is protected by an MD5 password hash. User authentication consists of username/password pairs using MD5 password hashes and Pluggable Authentication Module (PAM) stacks. The /etc/login.defs and /etc/default/useradd files are configured such that passwords expire after 75 days with a 15-day grace period. New passwords must be kept at least a week before they can be changed. User accounts are locked after five consecutive failed login attempts. Account lockout is provided by the pam_tally module, which is configured in the /etc/pam.d/system_auth stack. System administrator intervention is required to re-enable locked accounts.
System and security events are logged to the files /var/log/messages and /var/log/secure. All password changes and change attempts are logged. Logons and logoffs – both successes and failures – are logged. All privilege escalations and escalation attempts are logged (that is, use of the su command is logged). System administrators are required to log in to unprivileged accounts and then use su to perform maintenance and upgrade tasks.

The system is configured for graphical login using the Gnome Display Manager, with a warning banner substituted for the default Red Hat login background. The telnet service is turned off and ssh is used to provide remote login services. The standard ftp service is enabled, but is configured to prevent both root and anonymous access. All unnecessary services are turned off and uninstalled. Sophos Anti-Virus for Linux is installed on all systems.

Problem Description

After reviewing our current security measures, we were comfortable with all but two conditions: we needed to be able to enforce a strong password composition policy and to log all unsuccessful attempts to access particular file system objects.

The pam_cracklib module distributed by Red Hat uses a scoring algorithm to determine the strength of a particular password's composition. The algorithm supports a minimum length for passwords, but does not allow for specific composition requirements. Each password characteristic, such as uppercase alphabetic characters or numeric digits, is assigned a weighting that is used to calculate the password's "goodness," but there is no way to require a specific characteristic. In fact, the algorithm used actually allows a reduction in the minimum required length of the password, depending on the parameters used! A sufficiently long password could be scored highly enough to avoid the requirement to contain a numeric character. A password with several numeric characters could be scored highly enough to avoid the requirement for mixed-case alphabetic characters. A password with mixed case and numeric digits could be scored highly enough to be accepted even if it contained fewer than the minimum required number of characters. Although the latest version of PAM provides support for enforcing a requirement for an arbitrary minimum number of uppercase and lowercase alphabetic characters, numeric digits, and special characters, this support has not been migrated to Red Hat Linux as of version 9.

The default syslog facility did not provide a way to capture failed attempts to access arbitrarily specified file system objects. Specifically, we needed to log any failed attempts by a user to access data in another user's home directory, as well as any failed attempts to access or modify files and directories relevant to security. This requirement, while very simple to state, turned out to be the most difficult to implement.

Current Risks

One of the things that the SANS Security Essentials track emphasizes is that risk equals vulnerability times threat. Since this network is never connected to the Internet or any other network, the only threat is local users. With an otherwise reasonably secure setup, the major vulnerabilities are weak passwords and lack of adequate logging.

The inability to strictly enforce password complexity meant that a malicious user might be able to successfully crack another user's password if he were able to gain access to the /etc/shadow file. Inadequate logging would allow this malicious user to attempt to read or copy the /etc/shadow file with impunity.

The inability to log attempted accesses to file system objects also meant that a malicious user could try to read data that he was not authorized to see, or to modify system security parameters without fear of discovery. Surreptitious modification of security settings could give the user unlimited system access.

During

The two requirements that were not satisfied by built-in Red Hat Linux facilities were password complexity enforcement and logging of failed attempts to access certain file system objects. Since these are only marginally related to each other, I'll discuss them separately in the following sections.

Password Complexity Enforcement

My SANS training has instilled in me an appreciation for how easily weak passwords can be cracked, and, more importantly, it's given me a good understanding of what constitutes a strong password. My password policy for this system requires a minimum password length of eight characters, with mixed case (at least one uppercase and at least one lowercase character) and at least one numeric digit (0-9).

Proposed Solution

As stated in the Problem Description section, Red Hat Linux 7.3 does not support the pam_cracklib parameters that allow specific composition requirements. These parameters – dcredit=N, ocredit=N, ucredit=N and lcredit=N where N < 0 – are supported in the latest version of PAM, version 0.77, released in September 2002.
Since I wasn't sure how well the latest version of PAM would work in place of the version (0.75-46, released in early April 2001) included with Red Hat 7.3, I was wary of installing it. It would be much better if a patch for the version I had running on the systems were available. I put a message on the Washington, DC, Area Linux Users Group's mailing list asking if there were a patch; it turns out that there is. The patch, which was created by Werner Puschitz, is available on his web page devoted to securing Red Hat Linux.

Solution Implementation

I downloaded the patch file and printed the instructions, but did not use the patch facility because I wanted to see what source code changes were being made. I opened the patch file and a copy of the original pam_cracklib.c file in separate xterm windows running vi and patched the source code by hand.
I followed the rest of the instructions from the web site to compile and install the new PAM module. I then edited the /etc/pam.d/system_auth file and modified the pam_cracklib line as follows (of course, in the actual file each of the following takes only one line, with no newline between the "retry" parameter and the "minlen" parameter):

Before: password required /lib/security/pam_cracklib.so retry=3 minlen=8

After: password required /lib/security/pam_cracklib.so retry=3 minlen=8 ucredit=-1 lcredit=-1 dcredit=-1 difignore=8

The "ucredit," "lcredit," and "dcredit" values force a minimum of at least one uppercase alphabetic character, one lowercase alphabetic character, and one numeric digit, respectively. The minlen value requires the new password to be at least eight characters long. The retry value gives the user three tries to create a password that meets the password policy before forcing him to start over. The difignore value causes pam_cracklib to ignore the default difok parameter if the length of the new password is at least eight characters. This essentially eliminates the difok parameter from affecting the acceptability of the new password. The default difok value is either 10 or one-half of the length of the new password, whichever is smaller. This forces the user to create a password that does not contain half of the characters in the old password, up to a maximum of ten. With longer passwords, it becomes increasingly difficult to create a new password that meets this requirement, and difignore is used to overcome this difficulty. Since we're using MD5 password hashes, our passwords can be very long. With the ability to require an 8-character, mixed case password with at least one digit, I believe that our passwords will be strong enough without the similarity checking enforced by the default difok value.

Logging Attempted Accesses to File System Objects

Logging attempted accesses to file system objects proved to be a somewhat less tractable problem. Actually, we needed to log only failed access attempts, but even that wasn't possible with standard Red Hat Linux facilities.
Since I'd had good luck getting help from the local Linux Users Group on the password complexity enforcement problem, I posted a message to the list asking about logging and auditing software. This time, however, the list was uncharacteristically quiet. So I did several Google searches with different combinations of keywords and found several web sites that seemed to offer possible solutions.

Proposed Solution

One possible solution was the Secure Auditing for Linux (SAL) project. This project is an initiative to "develop a kernel level auditing package for Red Hat Linux that is compliant with the Common Criteria specifications (C2 level equivalency)" (Godinez, p.1). Unfortunately, a review of the project documentation revealed that setup and configuration were quite involved, requiring at least a kernel rebuild. The very tight integration between our application and the operating system suggested that solutions involving kernel rebuilds should be avoided.
Security-Enhanced Linux, or SELinux, was another possibility that I investigated. However, since it is a standalone Linux distribution, not an add-on for Red Hat, it would not work for us. Additionally, it is more of a reference implementation than a production version. It also included much more capability than we

needed – capability that would have to be managed, and that would add to the workload with no perceived gain in functionality or security.
After looking at a few other possible solutions, I followed a link to the web site for the Intersect Alliance, the creators of the System iNtrusion Analysis and Reporting Environment (SNARE). A review of the web site and SNARE documentation convinced me that this was a workable solution.

Solution Implementation

Obtaining the correct binaries for SNARE was the only significant hurdle to implementing this solution. The latest open source version, 0.9.6, is currently available for Red Hat releases 9 and later, with planned binaries for older Red Hat releases. Thankfully, Intersect Alliance has kept their older web pages online, and even provided a link to them. The binaries for SNARE v0.9.1, which was the version concurrent with Red Hat 7.3, also were still available in their archives.
Getting SNARE up and running was straightforward. After downloading the appropriate files from the Intersect Alliance web site, I followed the simple installation instructions and had the audit daemon, which collects the event data, and the GUI front end, which is used to examine the logged events, operational in less than 10 minutes.
The GUI front end also includes the configuration tool. This tool is used to specify the data to collect, how they are to be collected, and where they are to be stored. Configuring and testing the data collection parameters, and creating the filtering scripts were the major focus of this part of the task. These processes are detailed in the remainder of this section.
Data can be collected based on raw kernel events or based on higher level file system types of events called "objectives." Raw kernel event data collection is specified by low-level system calls, and data on the selected calls are collected for all users and all files and folders. This approach does not offer fine control and can result in very large log files. Objective data collection is specified by type of file system operation, filter expressions for both objects and users, and access attempt results (success, failure, or both). This approach also allows the assignment of an alert level to each event. Since we needed to monitor only specific file system objects, I chose to use the objective approach.
The collected data can be stored in a local file, sent to another SNARE daemon over the network for remote storage, or both. Because our application uses quite a bit of network bandwidth when it is running, I opted to store the data locally on each host. I configured all hosts to store the collected data in /var/log/audit/audit.log.
An objective must be created for each event that must be logged. An event is any attempt by a user to access a file system object (strictly speaking, SNARE can also log attempts to change user or group identification, network connection attempts, and attempts to reboot or create system modules, but we are concerned only with file system objects). These attempts can either succeed or fail. Logged events are assigned one of five alert levels, which are color coded in the GUI display.

The mechanics of configuring an objective are straightforward. Click on the configuration tool icon, then on the "Objectives" tab, then on the "Add Objective" button. A window appears with five sections, one for each of the necessary parameters: high-level event, filter expression for file system object, event success and/or failure, user filter, and alert level. Most of the parameters are specified by radio button selection, with text boxes for input of the file system and user filter expressions.

As previously stated, we required logging of all failed attempts to access particular file system objects. Specifically, the file system objects we needed to monitor are defined as any files or subdirectories anywhere in the following directory trees: /bin, /sbin, /usr/bin, /usr/sbin, /etc, /var/log, /usr/local, and /home. It might seem that logging accesses to files in the /home directory tree would cause an overwhelming number of uninteresting events, but since we were interested only in failed access attempts, this was not a problem – any failed attempt is an interesting event. As long as a user stays in his own home directory, or in directories for which he has group permissions, he should not have any failed access attempts, with the exception of attempting to read files that do not exist. Access attempts are defined as any attempt to open a file for reading or writing, create or remove a file or directory, open a directory (change the current working directory or list the contents of a directory), or change the attributes (owner, permissions, or name) of a file or directory.

As an example, here is the process for creating the objectives for the /etc directory tree. Note that there are four objectives necessary to cover all possible access attempts – "open a file for reading only," "write or create a file or directory," "remove a file or directory," and "modify system, file or directory attributes." The first two could have been combined since there is a "read, write, or create a file or directory" event type. However, this would mean that read only failures would have to be set at the same alert level as write or create failures; I wanted to make them different. So, to add the first objective for /etc, do the following:

- Click on the "Add Objective" button
- Select the radio button for "Open a file for reading only"
- Select the radio button for "Partial" in the filter expression section
- Enter the string "/etc/" in the expression text box (note that both the leading and trailing slashes must be included to distinguish it from objects with names that include the string "etc")
- Select the radio button for "Failures"
- Select the radio button for "All Users"
- Select the radio button for "Priority," which is alert level 4, color-coded orange in the GUI display
- Click on the "Save Objective" button
- Click on the "Save and Apply" button to save the configuration file and restart the audit daemon

The other three objectives for /etc are added similarly, with appropriate changes for the event type, filter expression, and alert level. I chose to set the alert level for "read only" failures to "Priority," which is level 4, and all of the other events to alert level 5, "Critical." The reason is that, with the exception of the /etc/shadow file, I believe attempts to write or delete security or configuration files are more serious than attempts to read them. Note that "read only" failures are still assigned a reasonably high alert level.
Objectives for the remaining directory trees are duplicates of those for the /etc tree. They are created by iteratively applying the process detailed above.
After adding all of the objectives for the relevant directory trees, the log entries can be viewed using the GUI. Upon inspection, it is apparent that there are many entries that are misleading. Two causes of "read only" access failures bear mentioning here. The first, and most common, is files that do not exist. The other is that the xscreensaver tries to open the /etc/shadow file using the uid of the user (the xscreensaver is configured to require a password in order to return to the desktop). Because these entries are benign, I wrote the following shell script to filter them out. This script is run by the auditor (as root) just prior to auditing the logs.

```
#!/bin/sh
#
# stop the audit daemon...
service audit stop
#
# rename the log file with today's date
# (this file becomes the backup/archive file)
mv audit.log audit.log.`date -I`-all
#
# filter out the records for files that don't exist...
egrep -i -v "return,-2" audit.log.`date -I`-all > audit.log
#
# restart the audit daemon with the new,
# filtered log file
service audit start
```

After

Once the solutions were implemented, we were confident that the password policy would be enforced and that failed attempts to access critical system and security files would be logged. Before turning the system over to the users, however, we tested it to ensure that it was performing correctly and consistently.

Solution Testing and Validation

When implementing or modifying a security measure, testing and validation are paramount. In fact, about 80% of the total time expended on this project was devoted to testing and validating the solutions. Testing is the process of ensuring that the system works like it's designed to work. Validation is the process of ensuring that the design correctly expresses the intent. A properly designed test matrix can address both issues simultaneously.

Password Policy

A matrix of test cases was developed to ensure that the modified pam_cracklib module was working correctly. The following table shows the test cases, the expected results, the actual results, and whether or not the required password characteristic was validated. All cases were run from the account "audie," which is the account that the auditor uses.

Table 1. Test Cases for Password Policy Enforcement

Test Case	Expected Result	Actual Result	Validated?
set password i4iatFByh	accepted	accepted	Yes
set password w3kooatt	rejected – no uppercase	rejected – needs uppercase	Yes
set password WwyamCaa	rejected – no digit	rejected – needs numeric	Yes
set password W2YAMCAA	rejected – no lowercase	rejected – needs lowercase	Yes
Did previous passwd command exit with error status?	yes – token manipulation error	yes – token manipulation error	Yes
set password I4IATfbYH	rejected – change of case only	rejected – change of case only	Yes
set password zc3ZX	rejected – too short	rejected – too short	Yes
set password yhi4iatFB	rejected – rotated	rejected – rotated	Yes
set password w2yamc&ahNYYNha&cmay2w	rejected – palindrome	rejected – palindrome	Yes

With all tests producing the expected results, the pam_cracklib patch and configuration is properly enforcing the password policy. Users' passwords will now be safer because password cracking attacks will be impractical using current mainstream processors.

Logging Requirements

As with the testing of password policy enforcement, a matrix of test cases was developed to determine whether the appropriate events were being logged. Of somewhat less importance, but still an issue, the test matrix also included cases to determine whether uninteresting events were being logged, and, if so, to ensure that the number of uninteresting events was manageable.
The first set of test cases was performed while logged in as user audie, a

normal, unprivileged account used for auditing the system. It is useful to create a matrix that defines the file system access privileges for a normal user so that no case is overlooked. The following table shows this matrix for the normal users on this system.

Table 2. Matrix of File System Access Privileges for Normal User

	Read/execute	Write/create file/dir	Remove file/dir	Chown	Chmod
/bin, /usr/bin, /sbin, /usr/sbin	permit – no log	deny – log	deny – log	deny – log	deny – log
/etc	permit most – no log; deny shadow file	deny – log	deny – log	deny – log	deny – log
/var/log	permit most – no log; deny files messages and secure	deny – log	deny – log	deny – log	deny – log
/usr/local	permit – no log	deny – log	deny – log	deny – log	deny – log
/home (own)	permit – no log	permit – no log	permit – no log	permit – no log	permit – no log
/home (others')	deny – log	deny – log	deny – log	deny – log	deny – log

The following table details the normal user test cases. The touch command is used to test create/write access. The rm command is used to test remove access. The chown and chmod commands are used to test file attribute and permission modification access. The cat command is used to test read only access. All test case attempts should fail and be logged. Note that for file and directory removal attempts, a file named audit_del_test was created by root in the affected directories with permissions appropriately set. This file was also used for the chown and chmod test cases.

Table 3. File System Access Logging Test Cases for Normal User

Test Case	Succeed?	Logged?	Pass/Fail
touch /bin/audit_test	no	yes	pass
cd /bin; touch audit_test	no	no	fail
touch /usr/bin/audit_test	no	yes	pass
cd /usr/bin; touch audit_test	no	no	fail
touch /sbin/audit_test	no	yes	pass
cd /sbin; touch audit_test	no	no	fail
touch /usr/sbin/audit_test	no	yes	pass
cd /usr/sbin; touch audit_test	no	no	fail
touch /etc/audit_test	no	yes	pass

cd /etc; touch audit_test	no	no	fail
touch /var/log/audit_test	no	yes	pass
cd /var/log; touch audit_test	no	no	fail
touch /usr/local/audit_test	no	yes	pass
cd /usr/local; touch audit_test	no	no	fail
touch /home/andy/audit_test	no	yes	pass
cd /home/andy	no	no	fail
rm /bin/audit_del_test	no	yes	pass
cd /bin; rm audit_del_test	no	yes	pass
rm /usr/bin/audit_del_test	no	yes	pass
cd /usr/bin; rm audit_del_test	no	yes	pass
rm /sbin/audit_del_test	no	yes	pass
cd /sbin; rm audit_del_test	no	yes	pass
rm /usr/sbin/audit_del_test	no	yes	pass
cd /usr/sbin; rm audit_del_test	no	yes	pass
rm /etc/audit_del_test	no	yes	pass
cd /etc; rm audit_del_test	no	yes	pass
rm /var/log/audit_del_test	no	yes	pass
cd /var/log; rm audit_del_test	no	no	fail
rm /usr/local/audit_del_test	no	yes	pass
cd /usr/local; rm audit_del_test	no	yes	pass
rm /home/andy/audit_del_test	no	no	fail
chown audie:audie /bin/audit_del_test	no	no	fail
cd /bin; chown audie:audie audit_del_test	no	no	fail
chown audie:audie /usr/bin/audit_del_test	no	no	fail
cd /usr/bin; chown audie:audie audit_del_test	no	no	fail
chown audie:audie /sbin/audit_del_test	no	no	fail
cd /sbin; chown audie:audie audit_del_test	no	no	fail
chown audie:audie /usr/sbin/audit_del_test	no	no	fail
cd /usr/sbin; chown audie:audie audit_del_test	no	no	fail
chown audie:audie /etc/audit_del_test	no	no	fail
cd /etc; chown audie:audie audit_del_test	no	no	fail
chown audie:audie /var/log/audit_del_test	no	no	fail
cd /var/log; chown audie:audie audie_del_test	no	no	fail
chown audie:audie /usr/local/audit_del_test	no	no	fail
cd /usr/local; chown audie:audie audit_del_test	no	no	fail
chown audie:audie /home/andy/audit_del_test	no	no	fail
chmod o+rwx /bin/audit_del_test	no	yes	pass
cd /bin; chmod o+rwx audit_del_test	no	yes	pass
chmod o+rwx /usr/bin/audit_del_test	no	yes	pass

cd /usr/bin; chmod o+rwx audit_del_test	no	yes	pass
chmod o+rwx /sbin/audit_del_test	no	yes	pass
cd /sbin; chmod o+rwx audit_del_test	no	yes	pass
chmod o+rwx /usr/sbin/audit_del_test	no	yes	pass
cd /usr/sbin; chmod o+rwx audit_del_test	no	yes	pass
chmod o+rwx /etc/audit_del_test	no	yes	pass
cd /etc; chmod o+rwx audit_del_test	no	yes	pass
chmod o+rwx /var/log/audit_del_test	no	yes	pass
cd /var/log; chmod o+rwx audit_del_test	no	no	fail
chmod o+rwx /usr/local/audit_del_test	no	yes	pass
cd /usr/local; chmod o+rwx audit_del_test	no	yes	pass
chmod o+rwx /home/andy/audit_del_test	no	no	fail
cat /etc/shadow	no	yes	pass
cd /etc/; cat shadow	no	yes	pass
cat /var/log/messages	no	yes	pass
cd /var/log; cat messages	no	no	fail
cat /var/log/secure	no	yes	pass
cd /var/log; cat secure	no	no	fail

These results are a bit discouraging. The first thing to notice is that the chown command isn't trapped at all. It is supposed to be trapped by the "modify system, file, or directory attributes" objective type, like the chmod command, but it evidently isn't.

Analysis of the remaining test cases leads to a likely possible reason for such inconsistent results. The filename buffer passed to SNARE might be a simple copy of the buffer that was passed to the system function that performs the requested action rather than the fully qualified path/filename. Because we're using objectives that filter on partial path names, SNARE won't trap failed access attempts if this is the case. This is a reasonable hypothesis since some of the commands work most of the time, while others work only when the full path/filename is specified on the command line. Those that are logged correctly are probably expanding the filename before calling the underlying system function, while those that aren't are probably just passing whatever they get from the shell. The only evidence contradictory to this are the cases involving the /var/log and /home directories, and the single case involving the /etc/shadow file. Absolutely no failed access attempts were logged when the command was executed while /var/log was the current working directory. This suggests that there is something special about this directory, but I don't know what it might be. On the other hand, all failed attempts to read the /etc/shadow file were correctly logged, whether or not /etc was the current working directory. Again, this suggests that there is something special about the /etc/shadow file – which, of course, there is.

The other contradictory test cases involved the /home directory. The only failed access attempt logged in these cases was the "write or create a file or directory" attempt. Although all of the cases specified the complete path/filename, and all

of the other attempts failed, none were logged. I suspect that the permission setting on the /home/andy directory, which denies access to "other" users, is part of the answer, but I can't prove it.
Finally, note that, in every case, the attempted access failed, which is the correct, expected result. This provides some confidence that the Linux permissions system is working correctly.
There is one interesting side note. I noticed during testing that the sgi_fam service, which scans every 6 seconds, was configured to check the /home/~/.gnome/nautilus-scripts file. Because this file did not exist for audie, and SNARE has an objective with /home as the target, an event record for this file was created every 6 seconds. For any practical user session, this would generate an overwhelming number of uninteresting records. They'd be removed by the filter.sh script, but the backup files would still be huge. I solved this problem by disabling sgi_fam, which we don't need anyway. And here I thought that all unnecessary services were turned off. It's a never ending process.
The second set of test cases was performed while su'ed to root. This set was designed to determine which, if any, uninteresting events were being logged. Root should have full access to any object in the file system. Failures are expected only when root tries to read a nonexistent file. These failures should be logged initially, but then eliminated by the filter.sh shell script.

Table 4. File System Access Logging Test Cases for root

Test Case	Logged?	Removed by filter.sh?
cat /bin/non_existent_file	yes	yes
cat /usr/bin/non_existent_file	yes	yes
cat /sbin/non_existent_file	yes	yes
cat /usr/sbin/non_existent_file	yes	yes
cat /etc/non_existent_file	yes	yes
cat /var/log/non_existent_file	yes	yes
cat /usr/local/non_existent_file	yes	yes
cat /home/andy/non_existent_file	yes	yes
cat /home/non_existent_file	yes	yes
cat /root/non_existent_file	no	n/a

The results were viewed while logged in as user audie using su to become root to run the filter.sh script and to run the SNARE GUI.

Risk Assessment

With an enforceable, strong password policy and the addition of a facility to log failed attempts to access file system objects, the major vulnerabilities of the system have been greatly reduced. Strong passwords reduce the vulnerability of the authentication system by rendering password attacks impractical. Logging of failed file access attempts both alerts system administrators to possibly nefarious activity and acts as a deterrent to casual file system browsing. These reductions in vulnerability directly result in a reduction of risk

(risk = threat * vulnerability).
The pam_cracklib patch has not caused any complications so far. There are a few issues related to the use of SNARE.
The first SNARE-related issue is that the binary distribution of SNARE for Red Hat 7.3 works only with the original, stock kernel (2.4.18-3), which means that we have no real recourse to any vulnerabilities found in that kernel. Although I've tried compiling SNARE from the sources so that it works with a more recent kernel, I have not been successful. I suspect that, when it was installed, the application made modifications to the system files that are preventing building of a working executable. The risk associated with this is mitigated somewhat by the fact that the LAN is never connected to another network, and is isolated in a secure room.
Another issue related to SNARE is that the auditor needs to su to root in order to run the filter.sh script. This script has to stop the audit daemon in order to back up the audit.log file and then create a filtered version. After the filtered version of the audit.log is created, the script then has to restart the daemon. Root privileges are required to stop and start the audit daemon. This doesn't pose a significant threat for us, however, since the auditor is also the system administrator in this case. It would be better if SNARE could be installed such that it was owned by the auditor's account.
The major issue with SNARE is the fact that it doesn't always work as desired. This can lead to a false sense of security, which can lead to a relaxed security posture, which is a major vulnerability. Knowing that SNARE potentially misses many of the events we'd like to capture keeps me vigilant, and thus helps to mitigate the risk associated with this vulnerability.
Even with these shortcomings, I believe that there is a benefit gained from running SNARE. As Eric Cole says in the audio recordings for the Security Essentials track, "prevention is ideal but detection is a must" (Cole, timecode 22:03 – 22:05). Logging as many interesting events as possible increases the chance that unauthorized activity will be detected.
The only remaining significant vulnerability is social engineering. It is still possible for a user to be deceived, finessed, or coerced into disclosing his password. In the same way, he might also be persuaded to provide unauthorized access to data. More extensive or more frequent background checks combined with better user education could mitigate the risk associated with this vulnerability.

Conclusion

Default Red Hat installations lean more to the "ease of use" end of the spectrum than to the "so secure you can't use it" end. The facilities provided with their distributions, however, do provide most of what is necessary to configure a secure system.
A strong password policy is meaningless if it cannot be strictly enforced. Why hasn't Red Hat implemented the latest version of pam_cracklib in its distributions?
Secure systems require proper auditing. Auditing requires proper logging. Red

Hat provides the capability to log just about everything except for the C2 style logging of accesses to file system objects. This deficiency is not unique to Red Hat, but is endemic to the design of both the 2.4 and 2.6 Linux kernels. That's unfortunate, because built-in C2-style auditing support would make Linux more accessible in highly sensitive environments.

SNARE is an effort to provide C2-style logging. However, beware! It is possible to install SNARE, configure a few objectives, and settle back with a cup of coffee and a false sense of security. Of course, this is true with any improperly configured security product. Tools that hook into a running kernel to extract data require thorough post-installation testing and validation. SNARE is better than the default logging facilities included with Linux, but it isn't the last word in event data collection. Better kernel support for event logging would go a long way toward making SNARE's job easier. This would give the developers more time to concentrate on event filtering instead of being bothered with kernel data extraction.

The network described in this paper has been operational for more than four months now, so all users have had to change their passwords at least once. Although they grumble about the composition requirements (they'd complain no matter how lax the requirements were), their protestations serve as constant validation that the password policy is being enforced. And so far, no nefarious activity has shown up in the SNARE logs – but we keep looking.

References

Cole, Eric. Track 1 – SANS Security Essentials. Volume 1.6 Audio: SECBK_66_1203.mp3. SANS Press, January 2004.

Godinez, Javier. Secure Auditing for Linux Home Page. February 2003. <http://secureaudit.sourceforge.net/index.html>.

Google Home Page. December 2004. Google. <http://www.google.com.>

Intersect Alliance Open Source Projects Home Page. Intersect Alliance. November 2004. <http://www.intersectalliance.com/projects/Snare/index.html>.

Morgan, Andrew G. The Linux-PAM System Administrator's Guide. 26 June 2002. <http://www.kernel.org/pub/linux/libs/pam/Linux-PAM-html/pam-6.html#ss6.3>.

novalug – Washington DC Area Linux Users Group Home Page. August 2004. <http://www.tux.org/mailman/listinfo/novalug>.

Puschitz, Werner. "Procedure for Patching pam_cracklib.c." 17 December 2002. <http://www.puschitz.com/pam_cracklib_patch.shtml.>.

Security-Enhanced Linux Home Page. National Security Agency. 8 December 2004. <http://www.nsa.gov/selinux>.