Global Information Assurance Certification Paper

Auditing Linux Systems with SNARE

Miles D. Stevenson

SANS GIAC GSEC Practical v1.4b

Auditing Linux Systems with SNARE

Introduction

With the astounding growth in recent years of the use and popularity of Linux, one of the largest gaps inherent in many Unix and Linux Operating Systems is the lack of a good auditing system. While companies such as Sun and Microsoft have been providing powerful auditing capabilities in their Operating Systems for some time now, most of the Unix/Linux variants have remained outdated in regards to security auditing. Security auditing is an essential technology that should be implemented by any system, especially when used in a networked environment by multiple users. Until the Unix/Linux variants recognize this and include adequate auditing capabilities in their systems, it is left up to the administrators to provide this important feature to protect their systems.

This paper describes the use of SNARE and other supporting tools to implement a strong auditing policy on Linux systems. A RedHat Linux 7.2 Firewall will be used to demonstrate this process. In addition to covering auditing techniques, I will demonstrate the use of additional tools to protect the audit trail and will briefly describe theoretical approaches to managing the audit trails of many systems in a larger environment. A basic understanding of Linux administration is assumed, and will not be provided by this paper.

System Security Auditing

Audit trails maintain a record of system activity by system or application processes and by user activity (Swanson & Guttman, p. 53). The goal of security auditing is to ensure that the security measures in place are adequately protecting information resources. There exist many different techniques for security auditing. Some are automated by computer systems, while others are performed manually by trained security professionals. There exists no single technique that is effective in every type of environment. This means that it is very important for Information Technology (IT) personnel to know which techniques work best for the environment that is used.

When dealing with IT systems such as servers, workstations, routers, etc., it is important to distinguish between auditing and logging. The two are often combined to be much more effective, but it must be understood that auditing and logging are not the same, as some administrators believe. Auditing is the act of looking for the security related information that helps us to determine what actions are taking place on the system, how the system is functioning, and how the security controls on the system are enforcing our security policy. Logging is simply the act of recording these data. Auditing and logging are usually combined into what most of the security community refer to as the audit trail.

Security auditing is a very important practice, which must be done by all organizations to protect their systems. The audit trail is one of the first places to look when a security incident is suspected. Audit trails are often one of the only sources of information that can tell us what is happening or has already

happened on the system. If auditing is implemented correctly, it is usually the best way to provide accountability for actions taken on the system. Without auditing, it is almost impossible for an organization to determine what actions are being taken on the system, and who is taking them.

This paper concentrates on automated security auditing of Linux systems. Specifically, on the Operating System, rather than on the applications that are used on top (although this technique may reveal valuable information relating to these applications). Generally, I am interested in knowing what actions are being taken on the system, when they are taken, and who is taking them. As such, auditing often relies on other security practices and technologies to be effective, such as good security policy, strong authentication, reliable and accurate system clocks, and data integrity of the audit trail. Keep these supporting security practices in mind throughout this paper, as each will be discussed.

Linux Security Auditing

All Operating Systems address security auditing in their own way. This paper focuses on RedHat Linux, and will not be applicable to other Unix environments such as OpenBSD, FreeBSD, AIX, HP-UX, etc. In contrast to Sun Solaris and other Operating Systems such as Microsoft Windows NT, 2000, and XP, most of the Unix and Linux environments come with poor auditing capabilities. Even worse, many of the Unix and Linux Operating Systems do not provide any built-in means for C2-style security auditing. Again, do not confuse auditing with logging. The syslog facility was not built with security in mind, and only provides logging capabilities. All audit trails that are found on these systems are generated from other applications such as login, telnet, SSH, cron, etc., but lack a strong auditing mechanism built-in to audit system-related events such as process execution, file access, file modification, and file deletion. It is also important to point out that system security auditing is something that should be taking place in real-time. File Integrity Checkers, such as Tripwire, do in fact help to implement security auditing of the file system, however, the audit trail that is generated from Tripwire is very limited because it is not generated in real-time, or as the events occur. This does not help us to determine when the action was taken, and who took the action.

The only audit mechanism that is generally found in Linux systems (including RedHat) is process accounting. This suite of utilities provides a means to track process execution and logon/logoff events. To use process accounting, it must be enabled with the accton command. Once enabled, process accounting logs its audit trail to /var/log/psacct in its own binary format. There are several commands that can be used to retrieve information from the audit trail, such as the lastcomm command. The lastcomm command prints out information about previously executed commands. A sample of this output is provided below.

```
[root@server log]# lastcomm
```

clear		root	stdout	0.01 secs Thu Nov 14 07:20
man	S	root	stdout	0.00 secs Thu Nov 14 07:19
sh		root	stdout	0.01 secs Thu Nov 14 07:19
sh	F	root	stdout	0.00 secs Thu Nov 14 07:19
less		root	stdout	0.00 secs Thu Nov 14 07:19
crond	F	root	??	0.00 secs Thu Nov 14 07:20
mrtg	S	root	??	1.02 secs Thu Nov 14 07:20
crond	F	root	??	0.00 secs Thu Nov 14 07:20
sadc	S	root	??	0.02 secs Thu Nov 14 07:20

The first row of the output shows the process that was executed, followed by a flag, which gives information about how the process was executed, the user who executed the process, the terminal type that was used, and the timestamp of when the process ended. The 'S' flag denotes that this command was executed by the superuser (root), and the 'F' flag shows that the process was 'forked' from another process. More information on the various flags and usages can be obtained from the manpage for the lastcomm command.

As you can probably tell, process accounting only audits information about the process when that process has finished execution. This can be bad for security, because many malicious processes such as viruses, trojans, and rootkit utilities are designed to keep running. This means that there is a good chance that process accounting will not help you to determine if a malicious process is currently running. Also note that process accounting does not keep track of any arguments that were passed to the command. This is important because the same command can have different implications depending on its arguments. Consider the two following commands:

```
[root@server log]# chmod o+rw /etc/shadow
[root@server log]# chmod o-w /home/mstevens/mydoc
```

..and now the resulting audit trail from those two commands:

chmod	root	??	0.00 secs Thu Nov 14 04:02
chmod	root	??	0.01 secs Thu Nov 14 04:37

As you can see, the first command sets the permissions on the /etc/shadow file so everyone can read and write to it. This has bad security implications as the /etc/shadow file stores the hashed passwords for all users on the system, whereas the second command is benign and simply removes write access for group owners to a personal document. When we go back to review the audit trail after an intruder has made off with our passwords, the only way we can correlate which entry in the audit trail corresponds to our event is by looking at last access times on the /etc/shadow file. This is not good because access times can very easily be changed. Also, imagine if we were not aware that the permissions on

the /etc/shadow file had been changed. Regularly reviewing the audit trail would not give us any clues that malicious actions have taken place on this system!

Process accounting only audits the execution of commands. While very important for security auditing, this is not enough. There are several other events which are very important to pay attention to, such as the use of objects on the file system. Process accounting will not tell us that 'disgruntled user' had a vi session opened on our httpd.conf file, or that the process '/bin/sh .hiddenvirus' just deleted everything under /boot. These are examples of other types of events that should be a part of every good audit trail.

Process accounting also has a few other shortcomings that are worth mentioning if I am to convince you that it does not meet our requirements of a good auditing system. One, is that process accounting logs its audit trail to a file in its own binary format. You cannot simply read or parse this file to see the audit trail, you have to use the provided utilities such as lastcomm. While this makes it a bit more difficult for an attacker to forge entries in your audit trail (but certainly does NOT make it impossible), it makes it difficult to work with the data using other utilities such as log analysis programs (think SWATCH and Logsurfer). It is also more of a pain to parse the raw data for analysis. Another setback, is that there is no good way to send the audit trail to another system because is does not offer such a feature by itself, nor does it log to another facility that provides this feature, such as syslog. Sending logs to a separate system is a very important step to ensure your audit trail's integrity and security. For more information on Linux process accounting, see "Linux Security Administrator's Guide" at: http://www.linuxsecurity.com/docs/SecurityAdminGuide/SecurityAdminGuide.html

Now that I have shown the complete lack of auditing provided by most Unix platforms and severe security problems of the auditing system that is provided by some, let us look at a tool that is at the center of our preferred auditing implementation: SNARE.

Auditing with SNARE

System iNtrusion Analysis and Reporting Environment (SNARE) is a freely available tool from InterSect Alliance (http://www.intersectalliance.com). SNARE is a very simple and powerful tool that was built for Linux to provide system auditing capabilities. It works by inserting a kernel module that monitors system calls to the kernel, and logs the events to /proc/audit. SNARE also provides a daemon that runs in user-space called auditd. This daemon reads the data from /proc/audit and sends relevant security information to a log file. Additionally, SNARE also comes with a GUI tool to configure auditd and read the audit log in much the same way the Event Viewer is used on Windows systems.

You can configure SNARE to define exactly what types of events to audit, such as the execution of a process, the modification of a file, system reboots, and even processes that are trying to use the network interface. SNARE also provides a much more detailed audit trail than process accounting. Here is an example of a single event taken from the InterSect Alliance website at http://www.intersectalliance.com/projects/Snare/Documentation/index.html :

```
testsnare.intersectalliance.com    LinuxAudit
objective,clear,Mon Aug  6 19:43:25 2001,The program
/usr/bin/gimp has been executed by the user leigh
event,execve(),Mon Aug  6 19:43:25 2001
user,leigh(500),users(500),leigh(500),users(500)
process,1651,sh    path,/usr/bin/gimp    arguments,gimp
return,0    sequence,12937
```

Don't worry if the above output looks a bit difficult to read. There are tools available which format the output in a much more user-friendly fashion. This example is showing the execution of a process. One of the nice things about SNARE is that events are audited as soon as they happen, so we do not have to worry about those pesky trojan's avoiding our detection simply because they continue running. Also note that any arguments given to command execution are logged. Now we can determine which use of the chmod command given in the example above was malicious. SNARE also logs the fully qualified domain name of the host which generated the event. This may seem trivial now, but it will be a big help later on when we have multiple machines sending their audit trail to one centralized system. Other very useful pieces of information include the PID of the running process, the real UID/GID of who took the action and the effective UID/GID, the actual system calls that were made, and the return code of the process. Now let's look at getting SNARE up and running.

Installation

SNARE is provided in both source and RPM formats and can be obtained from the InterSect Alliance website at: http://www.intersectalliance.com/projects/Snare/index.html Before moving any further, please be aware that at the time of this writing, SNARE does not work with the latest RedHat Linux kernels. Specifically, RedHat 7.3 using kernel 2.4.18 or above, or RedHat 8.0. This is because of a change made in most of the latest kernels that block processes from having access to hooks in the system call table of the kernel, which SNARE relies on to monitor system calls. On the bright side however, InterSect Alliance is working with the RedHat kernel team to integrate SNARE into the Linux kernel, which if successful, may pave the way for SNARE to be part of all future Linux kernels by default! Now that you have been warned, let's move on to specifics.

For this example, I will be installing SNARE from RPM on a RedHat 7.2 firewall using the default 2.4.7-10 kernel. If you are not using the default kernel that is

installed with your version of RedHat, make sure that you install SNARE either from source RPM or tarball, as the RPM's are designed only for the default RedHat kernels. The installation steps I follow here are specific to my system and are not as detailed as the SNARE documentation, so I encourage you to read through the SNARE documentation which can be found at: http://www.intersectalliance.com/projects/Snare/Documentation/index.html

You will also want to first decide whether or not you want to install the GUI utility that can be used to configure SNARE and analyze the audit trail. I will not be installing the GUI in this example because I am running SNARE on a server. It is generally a good practice not to run the X Window system on servers if you do not need to. This means that I will only need to download the snare-core RPM. As of this writing, the current version of SNARE is version 0.9.1-1.

First, I download snare-core-0.9-1.i386.rpm into my home directory:
/home/mstevens/snare-core-0.9-1.i386.rpm

Next, I verify the file with md5:
```
[root@firewall mstevens]# md5sum snare-core-0.9-1.i386.rpm
98f8cc78639686e836953cf1fa5bbbae  snare-core-0.9-1.i386.rpm
```

Install the package:
```
[root@firewall mstevens]# rpm -ivh snare-core-0.9-1.i386.rpm
Preparing...                ########################################### [100%]
   1:snare-core             ########################################### [100%]
```

Start up SNARE:
```
[root@firewall mstevens]# /sbin/service audit start
Installing Audit Module: Using /lib/modules/2.4.7-10/audit/auditmodule.o

Starting /usr/sbin/auditd:
SNARE audit daemon: version 0.90 starting up
InterSect Alliance Pty Ltd
http://www.intersectalliance.com/
SNARE audit daemon: driver open, starting audit
```

Check the log file:
```
[root@firewall audit]# tail /var/log/audit/audit.log
firewall.private.com       LinuxAudit      objective,priority,Thu Nov 14 10:28:16
2002,The file /var/log/audit/audit.log has been opened (read only) by the user root
event,open(O_RDONLY),Thu Nov 14 10:28:16 2002
user,root(0),root(0),root(0),root(0)    process,22776,tail       path,/var/log/audit/audit.log
return,3        sequence,1030
```

And we are up and running. There is still more work to be done however. Next, SNARE must be configured to fit your security policy. All of SNARE can be

configured from a single configuration file in /etc/audit/audit.conf. The authors recommend that you do not edit the configuration file manually, but rather use the GUI configuration utility unless you really know what you are doing. If you are the cautious type then this is a good practice, however, audit.conf is not very difficult to work with, and I have yet to have any problems, so I will be manually editing my configuration file. The contents of the default audit.conf file are below.

```
# WARNING: DO NOT MANUALLY EDIT, UNLESS YOU KNOW WHAT YOU
ARE DOING
[AuditType]
      type=Objective

[HostID]

[Output]
      file=/var/log/audit/audit.log

[Objectives]
      criticality=4   event=open(.*),creat,mkdir,mknod,link,symlink   return=Success
user!=root     match=^/etc/shadow$
      criticality=2   event=open(.*),creat,mkdir,mknod,link,symlink   return=Failure
user!=root     match=^/etc/shadow$
      criticality=4
event=open(O_WRONLY|O_RDWR|O_CREAT|O_TRUNC|O_APPEND),creat,mkdir,
mknod,link,symlink     return=Success  user!=root
 match=^/etc/passwd$
      criticality=2
event=open(O_WRONLY|O_RDWR|O_CREAT|O_TRUNC|O_APPEND),creat,mkdir,
mknod,link,symlink     return=Failure  user!=root
 match=^/etc/passwd$
      criticality=2   event=open(.*),creat,mkdir,mknod,link,symlink   return=Failure
user=.* match=^(/var/log|/etc)/audit.*
      criticality=3   event=open(.*),creat,mkdir,mknod,link,symlink   return=Success
user=.* match=^(/var/log|/etc)/audit.*
      criticality=4
event=open(O_WRONLY|O_RDWR|O_CREAT|O_TRUNC|O_APPEND),creat,mkdir,
mknod,link,symlink     return=Success  user!=root
 match=^/(sbin|usr/sbin|bin|usr/bin|usr/X11R6/bin|usr/bin/X11)/.*
      criticality=1   event=execve,exit       return=Success  user=.* match=^/bin/su$
      criticality=2   event=execve,exit       return=Failure  user=.* match=^/bin/su$
      criticality=1   event=socketcall(ACCEPT)       return=*       user!=root     match=.*
      criticality=3
event=chmod,rename,reboot,truncate,truncate64,chown,lchown,chown32,lchown32
return=*       user=.* match=^/etc/.*
      criticality=2   event=execve,exit       return=*       user=.* match=.*newgrp.*
      criticality=3   event=rmdir,unlink      return=*       user=.* match=^/etc/.*
```

```
        criticality=2   event=rmdir,unlink      return=*        user=.* match=^/var/log/.*
        criticality=0   event=socketcall(CONNECT)       return=*        user=.* match=.*
        criticality=0   event=execve,exit       return=*        user!=root      match=.*

[Events]
        open=0
        creat=0
        execve=0
        exit=0
        mkdir=0
        unlink=0
        mknod=0
        rmdir=0
        chown=0
        lchown=0
        chown32=0
        lchown32=0
        chmod=0
        symlink=0
        link=0
        rename=0
        reboot=0
        truncate=0
        chroot=0
        setuid=0
        setreuid=0
        setresuid=0
        setuid32=0
        setreuid32=0
        setresuid32=0
        setgid=0
        setregid=0
        setresgid=0
        setgid32=0
        setregid32=0
        setresgid32=0
        truncate64=0
        socketcall=0
        create_module=0
```

You can configure SNARE to use one of two different auditing types: Objective or Event. Toward the beginning of audit.conf, under the [Audit Type] section, you have to tell SNARE which type you want to use. Objective is recommended and is the default. Event auditing is the simpler of the two. With Event auditing, you simply tell SNARE which system calls you want audited and which you do not. This is done with a simple on or off setting for each system call under the

[Events] section. Setting a system call to 1 turns auditing on, while 0 turns auditing off. For example, if I wanted to audit every time a process gets executed, I would set execve=1. If you use Event auditing, make sure you know what each system call is used for. Also, be careful not to overwhelm your system with audit logs! If you use Event auditing, you run the risk of generating a lot of logs, which could crash your system. This is the main reason that I do not recommend using Event auditing. I can promise you that if you try and audit every system call, you will be in trouble.

Objective auditing is a bit more complex, but much more powerful than Event auditing. It is with Objective auditing that you can tell SNARE exactly what you want audited, down to the file and user who accesses it. Objective auditing is configured as a rule set. Each rule has several properties that you can define, such as the user who creates the event, the system calls related to the event, a return value to indicate success or failure, and assign a criticality level to the event. When an action performed on the system matches one of the rules it is logged. Even with Objective auditing, you have to be careful not to create a rule set that will generate too many logs. Let's look at one of the rules that are present in the default configuration.

```
criticality=4   event=open(.*),creat,mkdir,mknod,link,symlink   return=Success
user!=root      match=^/etc/shadow$
```

This rule tells SNARE to audit any time a non-root user successfully performs almost any operations on the /etc/shadow file. This is good, because we know that if a non-root user can perform any of these actions, then the /etc/shadow file has incorrect permissions and there is a very good possibility that we have been compromised. Notice that for this rule, the criticality level is set to 4, which is very high. If we were sorting through a large audit trail, we could easily sort by the most critical events, and this event would be at the top. This is a very useful feature when dealing with large audit trails.

The next property is the event property. This tells SNARE which types of system calls to look for. The open(.*) is directing SNARE to watch any type of open event regardless of whether it is read-only, write-mode, append-mode, etc. SNARE also provides several built-in groupings of events called 'classes' which help make writing your rule set easier. Consult the SNARE documentation for more on classes and related events. Next, we have the return property. Here you tell SNARE whether to look for a successful event, a failure event, or both. This is very similar to Windows auditing. Next is the user property. It tells SNARE to look for any user that does not equal root. Finally, there is the match property. This is a very powerful feature of SNARE which can be used to build regular expressions that match against the rule target. In this case, it is the /etc/shadow file. If you are unfamiliar with the use of regular expressions, consult the man page for egrep.

After you are finished building your rule set, it is a good idea to test it and ensure that your policy works. Again, make sure you are not auditing so many events that your system fills up with logs. Deciding on what you want to audit is always the most painstaking part of implementing your auditing policy. If you audit too little, you may miss valuable information that could be useful in the future. If you audit too much, then you end up with so much data to analyze that you end up missing the important events. It is a good idea to start with the default policy, and then add additional events in accordance with your organizations security policy. One rule that I would think about adding is auditing the creation or modification of .rhosts files. Your organizations security policy may state that all user actions must be tracked. In such case it may be a good idea to put all such user accounts in a 'personnel' group, and audit all events related to that group. In any case, keep in mind that you will never get it right the first time. Your auditing policy is probably something that will change relatively often, so it is simply a matter of adding more rules when you need them, and tuning them down when they are either generating too much data to process or affecting system performance.

The next step is to configure SNARE to send its logs to a separate machine. This is easily done by adding the line:
network=hostname:port
under the [Output] section of the configuration file and restarting SNARE. When sending logs over the network, SNARE works much like syslog. It uses UDP to send the data, but it does let you specify which port you wish to send the data to. Pick a higher port number above 1024 that you can dedicate for the use of SNARE throughout your network. You also need to have SNARE running on the remote system to receive the messages. Sending your logs to a separate system goes a long way in protecting the integrity of your logs and is highly recommended.

Additional Considerations

I hope that by now you are convinced of the importance of strong auditing and that SNARE can help you achieve it. At the beginning of this paper I asked you to keep in mind some additional supporting security practices which can help make security auditing a lot more effective. I want to mention many of those points here so that if you do decide to implement a good auditing system you have additional hints to help make your implementation much more effective.

One of the most valuable things that auditing can provide is accountability. It can tell you what actions took place, when they were taken, and who took them. The integrity of these three sources of information depend on much more than your auditing tools. One very important thing to remember is that the integrity of your audit trail heavily relies on how strong your authentication mechanisms are. If you are using a weak authentication scheme, such as relying on IP address authentication for remote hosts, or sending cleartext usernames and passwords

over telnet sessions, then how can you trust the usernames that are referenced in your logs? If all of the administrators log into the system as root, then how do you know who actually executed those commands when they all show up as root? The simple answer is that you can't. If you do not implement strong authentication mechanisms, and good security policy to protect them, then you can not rely on your audit trail to tell you the truth.

Just as important as authentication, is the accuracy of your system clocks. It is a very common mistake to have several systems on the same network with out of sync clocks. This makes analyzing audit trails almost impossible. Use tools such as NTP to automatically keep your system clocks in-sync with each other. Protect your systems with a firewall so outsiders cannot easily reset your system time and play tricks with your audit trail. If you do not protect it, it will be exploited.

Unfortunately, SNARE does not provide any built-in mechanisms to ensure the integrity of your audit logs. This is left up to you. Sending your logs to a remote system is a good start. The challenging part is ensuring message integrity with UDP. Experiment with other tools such as IPSec, netcat, and SSH. One very useful trick is to use netcat to tunnel UDP traffic through SSH, improving the integrity of your data. A good example of how to do this can be obtained from http://www.netsys.com/cgi-bin/display_article.cgi?1136 . Keep in mind that encrypting the traffic will add overhead, so you may want to select a faster, less secure cipher to encrypt your transmissions. Remember that confidentiality is not nearly as important as integrity and availability when dealing with your audit logs. Using technologies such as IPSec and SSH wil also provide you with authentication for your auditing system. If implemented properly, your central logging server will not blindly accept any traffic it recieves, but will rely on the authentication mechanisms provided by tools such as SSH.

Don't forget to backup your audit logs. I've seen gzip take an 80MB text file down to 5MB so there should be no excuse for not backing up those logs for at least a month. Your organizations security policies will often outline how long audit logs should be retained. It is a very common practice to see attackers use "low and slow" types of attacks to avoid detection. If an incident does happen, you will want to have at least a months worth of logs to look through so you have a better chance to determine when the attacks really started.

Think about taking advantage of freely available log analysis utilities such as the Simple WATCHer (SWATCH), written by Todd Atkins, which can be obtained from http://www.oit.ucsb.edu/~eta/swatch/ , or Logsurfer, developed by Wolfgang Ley and Uwe Ellerman, which can be found at http://www.cert.dfn.de/eng/logsurf/ I personally recommend Logsurfer because it has several advantages over SWATCH, such as the ability to analyze any text file, the ability to specify exceptions, and including timeouts and resource limits. The advantages of Logsurfer as listed on the Logsurfer website:

- Works on any textfile (or text from standard input)
- Matching of lines is done by two regular expression (logline must match the first expression but must not match the optional second regular expression). So you are able to specify exceptions.
- Uses contexts (collection of messages) instead single lines
- Flexible but easy configuration

- Timeouts and resource limits included
- Handles "shifting" of logfiles (just send a -HUP signal to close and reopen the logfile after you have moved the old one to another place and created a new one)
- Dynamic rules can change the actions associated with logmessages (something might happen that makes you interested in messages you would usually drop)
- Multiple reactions on one logline possible
- Portable written C-code (uses GNU regex library and autoconfigure)
- …

Logsurfer can be downloaded directly via FTP from ftp://ftp.cert.dfn.de/pub/tools/audit/logsurfer/ . With the help of a log analysis utility you can be notified immediately via email or pager when critical events take place. I would recommend that at a minimum, you configure your log analysis tool to notify you of any events with a criticality level of 4. With the help of these utilities you can turn your well established auditing system into a full-blown Host Based Intrusion Detection System (HIDS).

If you are implementing an auditing system across many hosts in a large, networked environment, or just simply want to go all out, think about putting your audit logs into a database such as MySQL rather than just flat text files. Initially, it takes more time and effort, but in the long run centralizing all of your data into a managed database will make analyzing and managing your audit logs much easier. I have implemented similar scenarios with Perl that simply reads the incoming data from the network, sorts it, and sends it to specific tables in a database. It is not as complex as you might imagine and if you are not a “perl hacker,” it is not very difficult to find one that is willing to help. If you search hard enough, you may even be able to find pre-written scripts for you on the Internet, but I would seriously recommend customizing it to your situation. Just remember not to abandon your raw audit logs. Raw audit logs are often the only evidence that will help you in legal situations, and are always the best source of information when doing in-depth analysis.

Lastly and probably the most important of all, make sure that you comply with your organization’s security policy. Your security policy protects you more than any technical countermeasure you have. You do not want to lose your job over auditing user activity when it is against your organization’s privacy policies.

Make sure you have the appropriate permission from your managers before auditing your systems. You can make a lot of people unhappy if you do not. If you are not sure, then ask. If you get mixed responses, do not move forward until it is crystal clear that you are complying with your organization's policies.

References

Swanson, Mariann, & Guttman, Barbara. “Generally Accepted Principles and Practices for Securing Information Technology Systems.” NIST Special Publication 800-14 (1996): 50-52.

InterSect Alliance. “Information Technology Security.” SNARE – System iNtrusion Analysis & Reporting Environment.
URL: http://www.intersectalliance.com/projects/Snare/index.html (5 Nov. 2002).

Atkins, Todd. “SWATCH: The Simple WATCHer.” SWATCH: The Simple WATCHer. 8 Nov. 2001. URL: http://www.oit.ucsb.edu/~eta/swatch/ (2 Nov. 2002).

DFN-CERT. “Logsurfer Homepage.” Logsurfer. 1 Dec. 2000.
URL: http://www.cert.dfn.de/eng/logsurf/ (2 Nov. 2002).

Staff Writer. “NETSYS.COM – The Intelligent Hacker’s Choice.” Fun and Games with netcat, ssh, tunneling. URL: http://www.netsys.com/cgi-bin/display_article.cgi?1136 (5 Nov. 2002).

Wreski, Dave. “Linux Security Administrator’s Guide.” User, System, and Process Accounting. 22 Aug. 1998.
URL:
http://www.linuxsecurity.com/docs/SecurityAdminGuide/SecurityAdminGuide.html
(2 Nov. 2002).

Computer Protection Program. “Berkely Lab Computer Protection Program.” Enabling and Configuring System Auditing.
URL: http://www.lbl.gov/ICSD/Security/systems/auditing.html (3 Nov. 2002).

Whelan, Paul. “SANS Information Security Reading Room.” Linux Security Auditing. 1 Jun. 2001. URL: http://rr.sans.org/audit/linux_sec.php (5 Nov. 2002).