Global Information Assurance Certification Paper

Linux Security Auditing

Paul Whelan
GIAC ID: paul.wh001
GIAC Security Essentials Certification (GSEC)
Version 1.2e
Original submission – on extension
Course taken at Portsmouth, NH March, 2001

Introduction

One of the most important tools for an administrator is the audit trail. When something goes wrong or something seems amiss this is the first place the trained administrator will look. This is true not only for support personnel troubleshooting issues but especially for security administrators who are investigating possible system breach. The significance of the audit trail should never be underestimated when attempting to maintain a reasonably secure system. This means that auditing should be taken into consideration in the overall system design when determining what is needed for system capacity and the ability of selected applications to log meaningful data.

Effective and successful auditing involves more than relying on the log files of applications and services. It involves deciding where to physically log the data (to local disk or to a log server), using utilities to monitor log files and alerting upon certain events as they happen, deploying a network sniffer or IDS to audit activity, using file integrity checkers, and other methods. The strategy is the old "defense in depth" concept where reliance is not placed on a single method but on several methods working in tandem.

The focus in this document is on the Linux operating system and is concerned primarily with concept and not attempting to be a "Howto" as there is sufficient documentation for the individual utilities and programs covered here.

Planning

When designing a system, take system auditing into account when determining the capacity that is needed. Far too often, systems are not built with this in mind. For example, say I want to deploy a web application server for a high volume site and I planned for enough disk space for my apps, databases, and system files. When I see how much space the logs are taking up due to the shear volume of requests my tendency will be to sacrifice the logs over the application. After all, what do I have the box for in the first place? So, I implement a poor log rotations scheme in order save on my disk space (they're only logs, right?). Now I've lost my ability to know what's happening on my system beyond my last log rotation. I've lost the ability to establish any trends or patterns that would help me in the investigative process.

The problem with not anticipating or projecting how much space or power you will need in the auditing process is that when it comes down to running my services or keeping my logs – the logs always lose. The best thing a cracker can hope for is that you have poor logging and auditing processes because if it's not in the logs – then it didn't happen.

Some recommendations are as follows:

- Use a separate physical disk and, if possible, a separate controller. Logging for high volume sites can prove disk intensive and it helps to have a dedicated disk and controller to increase performance.
- Use a dedicated partition if the above is not possible or if you are using a RAID array. The problem with this approach is in knowing how much space to allocate to the partition. Too little space and you will run out of disk (and logging) in little time. Too much and you've begun to waste space that could be used for other things. If available, check to see how much data is already being logged and use that as a guideline for a minimum starting point. Also, on Linux systems, you

should put your logs on a separate partition to prevent a DOS attack by filling up your file system where your system binaries and files reside.

- Memory and CPU. You don't necessarily need more memory just to log data, but if you are going to deploy sniffers, IDS, log monitors, etc., then you have to consider how much memory and CPU cycles those services are going to take up.
- Log not only to the local disk but also log to a protected syslog server. Also, rotate the logs off line to some form of backup media.

What to audit

Three basics are: application, file system, and network. First of all, by application I mean services/daemons as well, or basically anything software that is executed interactively by a user or in the background by the system. Most decently written applications have some type of logging capabilities some are descriptive and verbose and some are as useless as a three-legged chair (that is, if the chair is designed to have four, of course). We need to pay attention to what written the logs. This is where you'll find some of your most descriptive information as to what's going or has gone on in the system.

Audit your file system as well. What files have been added or deleted. What files have been changed either in content or permissions? It's interesting to find out that /etc/passwd has changed in the middle of the night on Saturday. Or to find out why /bin/login is giving you different output or behavior than you are used to seeing. Or why is there a symlink from /.bash_history to /dev/null?

Network auditing can be the most interesting. The other types of logging prove interesting when somebody has already gotten on to your system. But network auditing can show what was happening on the network that led to the break in. Was it a buffer overflow, was it a dos attack? The network auditing will show you that including giving an indication of the attempted break-ins or probes to your system.

Auditing on Linux

Syslog

Syslog is the standard logging facility for Unix/Linux operating systems. On a RedHat Linux system it stores the syslog files in /var/log and comes with "logrotate" pre-configured to rotate the logs once a week and to keep the logs for 4 rotation cycles (1 month). You can change these default settings by editing the /etc/logrotate.conf file. For instance you tell it to keep the logs for 52 rotations cycles (1 year based on weekly rotations) and compress the files that were rotated. The logrotate program is an excellent utility to maintain your log files. It basically takes the current log file, renames it *logfilename.1*, if *logfilename.1* already exists then it moves the existing one to *logfilename.2*, it then creates a new, empty *logfilename*. An easy way to look at is to remember that *logfilename* is always the current log and *logfilename.1* is the previous.

But syslog is not without it's shortcomings. There is a syslog replacement called syslog-ng (syslog new generation) that was written to pick up in functionality where the standard syslog leaves off. You can download syslog-ng from http://www.balabit.hu/en/products/syslog-ng/ and there's also a good article on it published on Linux Gazette (http://www.linuxgazette.com/issue43/scheidler.html).

Standard syslog puts data into certain files on the basis of which facility the program has called syslog. You can customize which facilities log to which files by editing the

/etc/syslogd.conf file. But with syslog-ng you can log based on regular expressions. This enables you to detail where syslog-ng will log what data, like, having all of the ftpd log data written to /var/log/ftpd, or having all occurrences of "failed login" be written to /var/log/badlogins. There is flexibility here.

Swatch/Logwatch

Most of us do not have the ability to sit down in front of a console and tail the logs all day long. Enter Swatch and Logwatch. Swatch and Logwatch is just a couple of the many utilities out there that can help you keep an eye on your logs. Swatch is a fairly powerful utility in that it has the ability execute a program or script when a particular pattern is matched. It runs as a deamon (service) tailing the log file your specify looking for given patterns as they are written to the log file. You can configure swatch to sound a beep, send an email, or execute a program (such as a script that will page you) when a pattern is found. You can get real detailed with what you tell Swatch to alert you on, thereby reducing the possible false-positives you may encounter (not eliminating – reducing). Lance Spitzner's Armoring Linux document recommends the use of swatch (http://www.enteract.com/~lspitz/linux.html). You can get the latest code and documents for swatch from http://www.oit.ucsb.edu/~eta/swatch/.

Logwatch, on the other hand, runs as a cron job that will tail your log file email you a report (ftp://ftp.kaybee.org/pub/linux/logwatch-2.1.1.tar.gz).

The usefulness of programs like these is that you a chance of getting informed about an event before a hacker can cover his tracks.

Tripwire/AIDE

Tripwire and AIDE are used to verify the integrity of the files on your system. It's a great way to know if someone has replaced /bin/login or modified /etc/passwd. But there is a caveat to it; you have to store the database offline, preferably on a floppy or other removable media. Why? If you leave the database on the disk then that is open game for a hacker. What's to stop him from updating the database with the new signatures of his modified files? Of course, this poses a problem in administration. Most security administrators do most of their administering remotely. How are they going to pop in a floppy disk from miles away? Or how can the program run from a cron job if the database is unavailable?

One way to help this problem would be to have a baseline database of your systems binaries and configuration files (after everything has been properly configured, of course) and store the database on read-only media, like a CD. This way the database cannot be modified. Update the database on a regular basis (monthly, quarterly, etc.) to allow for changes in a system (the password file may grow or shrink).

The bottom line is that these should be considered essential tools that can only be trusted as long as the database is trusted. If the database is left on the disk – it can't be trusted.

You can get the open source Linux edition of Tripwire from http://www.tripwire.com/products/linux/. AIDE is available from http://www.cs.tut.fi/~rammer/aide.html. I would recommend trying them both out. AIDE hasn't had the press that Tripwire has but AIDE is no slouch – it's a very powerful alternative to Tripwire.

Snort/Tcpdump

Snort is a network sniffer. Actually, that's an understatement. Snort has the ability to be a powerful network intrusion detection system, completely customizable. You can write your own rules for snort or you can use the provided rules. Snort, depending on the rule set, can detect attacks by looking for certain signatures and behaviors in the packets that it looks at. More than just telling you if somebody tried to connect to UDP port 31337 on your system Snort alert on the content or payload of the packets if it knows what to look for (it knows what to look based on what is in the rule set).

A good thing to do to get started is to establish baselines. For a period of time take a look at ALL the traffic. This will help you to know what can be considered "normal" traffic patterns for the subnet you are on. You can then adjust your rule set based on these baselines. If you alert and log everything in a production environment, it will get old fast and you won't want to look at any of the alerts or logs because you know 99 percent of it is nothing you should worry about so you end up not worrying about it period.

Know your systems and know your network. Find out what's normal traffic looks like. Find out what unknown traffic is. For instance, on this system you're only offering http and ftp services – so why are there a multitude of connection attempts on port 6667?

Tcpdump is another valuable utility. It's built using the pcap (packet capture) libraries to give you the ability to filter output. It doesn't come with a ready set of abilities that snort has to offer. But you can use it to roll a little something of your own. For instance, take a look at the Shadow Project. With Shadow, you take a tcpdump log file (a file that's created by running tcpdump with the –w *filename* switch) and parse through it with an advanced set of predefined filters.

The benefit of snort is that it's all "real time". You don't have to write all the snort data to a database and then later on parse through it later.

At http://www.snort.org you can find binaries, source code, pre-written rule sets, add-on/companion utilities and ample documentation. There's even a Win32 port that I've personal tested on NT/W2K with great results.

Tcpdump is pretty standard on Linux distributions but in case you don't have it you can get it at http://www.tcpdump.org. You can also find a link to the pcap library, which is necessary for compiling tcpdump.

System utilities

Make use of standard system utilities such as ps, netstat, ifconfig, lsmod, w, who, last, and find (to name a few). Use ps often to find out what's running on your system. Is there anything unknown or suspicious? Use netstat to find out what ports you're listening on and what processes have those ports open (netstat –anp | grep LISTEN). Use ifconfig to see if any adapters have put into promiscuous mode. If you run snort then your interface will be in promiscuous mode unless you run it with the –p switch. Use lsmod to find out what modules are loaded on your system. Use w to find out who is currently on the system and what processes they are running. Use "who" to find out who is currently on the system. Use last to find out when the last logins occurred by whom and from where.

You can also use find to discover suid files and new files or directories.

The problem with using these utilities is determining what level of trust you can place in them. We'll look at that issue next.

Integrity
The thing we are most concerned with on this matter is integrity. What good is all the auditing that we can do if we can't trust the data? There are measures you can take to add layers of security to your implementation. Remember, security is more of a journey than a destination (I would give credit to the source of that if I could remember where I read it).
I want to make sure that I can trust the data I am seeing. A good rule of thumb is, a system cannot be trusted after it's been compromised. Don't trust the logs; don't trust the file checkers or even trust system utilities – any of them. You have to be sure you can trust them.

With syslog it's best to not only log locally but to log off the system. This includes logging to a remote, locked down syslog server, or logging to a line printer via lpd, or logging to another system via a serial line. LPD may be the most impractical of those for the average site due to it's large use of ink and paper and the ease of which someone can merely delete the lpd daemon (which I've seen done in one of the hacks I've investigated). If you log remotely to another syslog server then make sure that server is inaccessible to the rest of the Internet. When you invoke syslogd from your log server (the machine dedicated to receive log messages from the other machines) use the "-r" switch so it is enabled to receive messages from other machines. If it's on the DMZ that make sure firewall rules are appropriately setup to make it unreachable from the Internet. Set up the remote syslog server so that the only service that it is running is syslog (and maybe ssh for remote administration). Additionally, if it's a Linux box, setup ipchains or iptables rules only allowing connections to your syslog daemon from the other system (/sbin/ipchains –A input –p udp –s ip_of_sender ! –d ip_of_this_system 514 –j DENY).

With tripwire and AIDE, store the databases on read-only media. Also, you may want to store the actual tripwire binaries on the CD as well to prevent tampering. Or at least have a location of where you store a known good copy of the binaries so they can be compared and restored to the system.

With the system utilities (ps, netstat, etc.) you have know way of knowing if you are using trojaned binaries (unless you use tripwire or you happen to know the md5sum by memory or from a known good copy of the binary). If your system is penetrated then these utilities cannot be trusted. They can all be built to lie to you about the output that it shows you. Even a quick and dirty perl or shell script can be written to accept your command line input then run the real program and filter the data that is seen on output (as I've seen in one of my investigations).
It's best, as well, to keep a known, good copy of these utilities on a CD. This way you can be sure that the binaries are good. You can also you use an encrypted loop back file system. You can download BestCrypt, which is a tool to create loop back file systems that you can encrypt with blowfish. Mount it and copy your tools there. Now you can mount it when you need it and dismount it when you're done. It would require a passphrase to be able to mount it. You can get BestCrypt from

http://www.jetico.com/linux.html. CD is still better because someone can delete the loop back file containing your data (unless of course you store the loop back file on a cd – then you have an encrypted file system that can't be deleted).
Does that mean that you still can't be lied to? Nope. Someone can install a loadable kernel module on your system that modifies the /proc file system. Kernel module root kits raised the bar in system compromise. The most popular have been Adore (originally available from http://www.kalug.lug.net - although you can't find it there anymore) and LRKM (the Linux Root-kit Kernel Module). The /proc file system is used by just about all these system utilities to get their information. You may have good binaries but what if where they are getting their information from can't be trusted? There are measures you can put in place that can detect suspicious modules be loaded into your kernel but there is only one sure way to make sure that won't happen (such as St. Jude http://www.sourceforge.net/projects/stjude) – remove loadable module support from your kernel.
On a secure system you don't want to give anyone the possibility of loading a root-kit module onto your system. This means that when configuring your kernel – first remove everything you don't need (if you don't need isdn, usb, sound, etc. don't put it in there – you will need the space). Then add everything you do need as directly in the kernel and stead of modules (remember say "No" on the questions that ask you if you want loadable module support). Some avoid this route because they think the kernel will be too big or that they will lose the flexibility of being able to add and remove modules on the fly. If you get rid of everything in the kernel that is not needed for your system (if you don't know what you need or not - read the kernel Howto) then you should be fine. Is it inconvenient? Sure it is. But what is when it comes to security?
So, now you have your trusted set of databases and system utilities on read-only media and you've disabled loadable kernel modules. We have just made the hackers job that much harder and you can place a little more confidence in the data you are looking at.

Running network intrusion detection software is a perfect edition your audit toolkit. Even if someone compromises your system and covers their tracks you can still look at the network audit trail to determine where they were coming from. That statement must qualified by saying: only if you're network intrusion detection system can be trusted. There's nothing to prevent you from running snort on the very system you are building the audit scheme for, but the best place to put it is on a dedicated system with plenty of disk space. You will need to put this node in a place where it can "hear" all of the traffic going to and from the system or network you want to audit. If you are placing it on a switched segment then you need to use a switch that has the capability of mirroring all the traffic to the port your intrusion detection node is on. One method of building a network intrusion node is to bring up the Ethernet adapter in promiscuous mode without an IP address and use a second Ethernet adapter that is connected to the private segment unreachable from the Internet. This makes the system unreachable from the Internet while still able to "listen" in on the traffic going by.
Deciding on how to write your policy will be the biggest challenge in setting up a network intrusion detection node. Do you log everything? Well, depending on your organization that may be an imperative. Do you just log known badness? In other words, create your rule set looking for known attack signatures. That's the common approach and works so long as you keep up to date with the latest attacks. Or do you make decide

on what's good traffic and log everything else? That is decidedly a more difficult approach since a lot of products don't really allow one to create rule sets based on 'here's what's good – log everything else'. But that may yield some of the most interesting data.

Test regularly

OK, so you have everything set up and working correctly, right? How do you know it's working correctly? Just because you haven't seen anything in the logs doesn't necessarily mean nothing has happened. Maybe the logging is failing somewhere in the process. Maybe snort was invoked on the loop back adapter instead of the Ethernet (but that would never happen). Maybe I'm running tripwire against a bogus database. The point is – you have to make sure it's working. Do not just assume. This means, yes, hacking your own system. Port scanning, brute force attacks, dos attacks, etc. Find out if the auditing scheme you have put into place will actually tell you what's going on. If not, it should be fixed because that means it could have been going on all along without your knowledge.

There are good commercial tools available to do this (which I have used) but I have to admit I am partial to Nessus (http://www.nessus.org) and Nmap (http://www.insecure.org).

The beauty in them is not just because they are free – it's because they are good, dam good. Nessus is comes in 2 pieces: a server and a client. The server runs on Unix and there are clients that run on Unix and windows. Strong crypto between the client and the server has also been added. Nessus seems to always be update to date with the latest attack modules (called plugins). It also has a powerful scripting language, which you can use to write your own attack plugins. Fire up Nessus and let it loose on your system (of course, at an approved time window so as not to disrupt any business activity). Look at the report that Nessus gives to see if there are any holes on the system you need to tighten up, but also, look at your logs on the attacked host. Did your logs record any unusual activity? Did your network intrusion node pick any of this up?

Nmap has a plethora of options to use when scanning a remote system. One of the most interesting is the OS detection option. It attempts to make a guess at what the operating system is based on the structure of the packets sent from the remote host (what is the value of the ttl field, what is the window size, how does it respond when it's sent a packet with the FIN and SYN bit set, etc.) Most operating system vendors are somewhat unique in the way the do things with the IP stack they implement and Nmap can take advantage of these nuances.

Analyze (conclusion)

Once you have auditing set up properly, and you can trust the data, and you have tested it to see that it works you have to use it. Simple enough. Don't let the logs sit there unlooked at. That's why you have to determine baselines to find out what you should be looking at. There is a danger in too much logging. It's only a danger because of our human nature to be uninterested in looking at the same old stuff everyday. We scan through thousands of lines of logs knowing that we don't need to be seeing all of it and the danger is that we miss that one entry where there was a root login from a dial up account from some obscure ISP in Europe. Make your auditing scheme useful and useable. Otherwise it's ineffective.

Resources and References

Online Articles and Documentation

Auditing your firewall. http://www.linuxvoodoo.com/security/audit.html
Armoring Linux. http://www.enteract.com/~lspitz/linux.html
Linux Security Howto. http://www.secinf.net/info/unix/linhowto/Security-HOWTO.html
Watching your log files with Swatch. http://www.secinf.net/info/unix/lance/swatch.html
Linux Administrator's Security Guide (The LASG). http://securityportal.com/lasg/
Know your enemy series. http://project.honeynet.org/papers/

Books

Hacking Exposed. McClure, Scambray, Kurtz, Prosise.
Hack Proofing Your Network. Ryan Russel. Syngress Media.
Maximum Linux Security. Anonymous. SAMS
Practical Unix and Internet Security. Garfinkel, Spafford. O'Reilly.

Tools

Snort. http://www.snort.org
Nessus. http://www.nessus.org
Nmap. http://www.insecure.org
Tcpdump. http://www.tcpdump.org
Syslog-ng. http://www.balabit.hu/en/products/syslog-ng
Tripwire. http://www.tripwire.com
AIDE. http://www.cs.tut.fi/~rammer/aide.html
BestCrypt. http://www.jetico.com/linux.html