# Global Information Assurance Certification Paper

# Building a Secure Nagios Server
# GIAC GCUX Assignment – Option 1
# Securely Administering UNIX

**Prepared By: Chris Dahlke**
**Submitted: March 21, 2005**

Table of Contents

## Abstract

The objective of this paper is to document a secure installation and deployment strategy for Nagios, which is a very comprehensive and flexible network monitoring application. One of the reasons that I chose this particular package for my assignment is that the consulting firm that I work for has many clients with Nagios monitoring systems and I felt that it was important for our clients that I look at the way we are deploying this software in their environments and determine for myself whether we are deploying this software in a secure and effective way for our clients. I have, during this process, seen some security flaws in the way we are deploying this systems and I am using this opportunity to create a template for future deployments as well as to go back and adjust the systems which are currently in production.

The approach that I took, was to recreate a typical customer environment in my test environment and then look at it from a security perspective rather than from a network management perspective. As is typical with "new solutions", we began implementing these systems to fill a hole for a single point of monitoring of network resources. Many of our customers had server monitoring packages and/or network monitoring packages, but these systems were very different from each other and did not give a comprehensive view of the state of critical resources on the network. In rare cases where the software they had was correctly implemented and/or actually used, it required that the system administrators manage two or more packages and was almost never kept current. In most cases monitoring was the only consideration and security was not even an after thought.

The Nagios solution fit the bill quite nicely for providing a central point for all network system monitoring. Once we deployed a few of these systems, despite a fair amount of server hardening, we began to realize that there could be more potential for security concern than just on the Nagios server itself.

On the plus side, since we look at so many aspects of the system when deploying software of this nature, we often find many other weak points in the networks which we can fix during the deployment of these monitoring solutions. It is quite usual for us to find network devices with "public" as a Read Write community string or to find network devices which have default passwords and code older than my children. We also found that it is extremely rare for small to medium sized customers to do any sort of logging of network activity. In all cases where we have deployed these servers we have been able to increase the level of logging significantly and also synchronize log data in time on a central syslog daemon. This of course has it's good and bad because we have central logging but we also have more sensitive data to protect on our monitoring server.

## Define the Environment

### Physical/Environmental Issues -

In any server deployment, if an “evil” person can gain physical access to your equipment they own your server. In my test environment I have the “server” in my locked office tucked away in the basement. This is not ideal because my wife and kids could compromise the server since they often have physical access to the equipment. In any environment it is important to know who has physical access to the machines and to assess the threat level of each person. Sometimes the threat is more of accidental downtime than one of security but all of this should be documented. I know it is slightly silly but I am going to document the threat level of each person in my test environment as an example of how I ask my customers to do the same in their environments. I feel that this exercise will help to define the issues just as I would want to do at our theoretical GIAC Enterprises.

People with Access to the Servers

Myself (Chris)
Technical Ability to compromise server – Fairly high
Access to system console – yes
Knows at least one password to log onto on system – yes
Knows root password of system – yes
Has access to install daemon's/software on system - yes
Performs updates on system – yes
Security training level - moderate
Risk to compromise security of system – low based on no motive, any compromise would be inadvertent
Steps taken to mitigate risk from this person – Externally scan server for vulnerabilities with Nessus, no services on this network are published to the Internet.

(Wife)
Technical Ability to compromise server – Low
Access to system console – yes
Knows at least one password to log onto on system – no
Knows root password of system – no
Has access to install daemon's/software on system – no
Performs updates on system – no
Security training level - none
Risk to compromise security of system – low based on technical ability and lack of motive
Risk to bring system down accidentally – low based on frequency of physical interaction with the server.
Steps taken to mitigate risk from this person – Risk low no steps taken

(Son)

Technical Ability to compromise server – low but getting there
Access to system console – yes
Knows at least one password to log onto on system – no
Knows root password of system – no
Has access to install daemon's/software on system - no
Performs updates on system – no
Security training level - none
Risk to compromise security of system – low due to lack of motive
Risk to bring system down accidentally – High based on past experience with random button pushing.
Steps taken to mitigate risk from this person – User training, threat of punishment (no video games).
Perceived effectiveness of risk mitigation – good, no downtime or compromise of this server related to this person.

Hopefully this gives a fair idea of the goal here; assess the risks, assess the threat level, take appropriate steps to lessen the risk to an acceptable level. It is also important to have a plan for recovery in the event of downtime. There should always be a business continuity plan in place and a procedure for recovery from failures or security compromises. For each site where a server is deployed, an analysis of this type should be done.

Once this test server is prepared and tested, it will be moved into the production GIAC Enterprises network where there will be a whole new set of security parameters. The risk in my non-production test network, as you can see is pretty low, but once the server or preferably a near clone of the test server moves on site, the physical security situation will change. Because of this, it is important that I look at not just the threat assessment for my test lab but also a the on site threats at each deployment location.

Additional precautions which I have taken on my system to help ensure the security of my server were to set a strong grub password and a strong bios password and also to set the boot order to go to the hard drive first in the boot order. This will at least slow down anyone who is interested in gaining access to the server so I have a better chance of catching them in the act. It would be preferable to have a locking case on the system as well to prevent reset of the bios password although my current test system does not have this. Most of my "deployed" servers do have this ability so this is to be done for the production boxes.

**Server Specifications**
1Ghz Compaq Deskpro EN
768M Ram, 20Gig HD, 10/100 intel ethernet adapter
w/ cd burner
External 80Gig USB HD (backup system)

**Operating system**
Gentoo Linux 2004.3 kernel version 2.6.10

The reason that Gentoo was chosen for this deployment is that it is the primary Linux flavor that we use in our client server deployments. It has many nice features from a security perspective as well as some concerns as do all operating systems.

On the positive side, Gentoo deployments are not “frilly”. When I set up a Gentoo system I know exactly what is on it and where it came from because it comes with pretty much nothing by default. The stage3 distribution does include a small number of executables to make the system functional but there are no services listening on the network when the system is built. Since it is more challenging to compromise a system which has no network services listening on it I feel that this is a good place to start when building a server. Another positive feature of Gentoo which is also used in other Linux varieties is Portage. Many common and some not so common applications can be compiled, installed, updated, and removed through a series of emerge commands. This is a good thing for keeping the systems up to date, which is also critical when maintaining a secure server.

On the negative side, because the system comes with very little by default, it is very often the case that system administrators will connect these servers to the Internet to install software before the systems have been hardened at all. Of course since there are no services running this is not always a huge problem initially. One of my concerns with this has always been that services will be installed and turned on in their default configurations. This means, to me at least, that we trust the application developers to use secure defaults in their configuration files. While many developers try to do this, I prefer to carefully test application configurations before deploying the applications where they could be potentially compromised. Another concern that I have related to this practice is that if the applications are being installed and compiled dynamically, there obviously is a compiler on the system. For a protected environment, this is not generally considered good practice.

My suggestion when using Gentoo or any similar Linux distribution in a secure environment would be to pre-download the distribution files for the required applications and any supporting software required to secure the system. Then the system can be built, secured, and tested “off-line” before it is connected to the Internet for the first time. It is also important to have a similar system which can be used as a test environment (on a separate network) and to compile and test applications for deployment to the “secure system”. In the real world, I feel that this is practical if some planning is done, unfortunately I seldom see this done in practice because of the inconvenience. Laziness is the key to intruder happiness. :-)

Once the system is deemed secure and is connected to the Internet there is still security controversy regarding the deployment of updates to the system. While emerge provides a nice mechanism for deploying software updates, many security professionals are leery of applying updates to secure systems “on the fly”. Much of this concern comes from the fact that you need to have a compiler on your “secure system”. While this is a valid concern, I think that there is a position of compromise. For secure servers, as I suggested earlier, a duplicate of the system should be built in a test environment. This way, automated updates can be tested first in a non-production environment and then deployed in production once the server has been thoroughly tested with the latest updates.

At this point, the patches can be quickly deployed to the production environment or all of the update packages can be individually downloaded and installed. In the case of an emerge type install, the “emerge -b” option can be used in the test environment when installing a package. This will allow for us to install the package on the test system, check it for issues or vulnerabilities, then move it to our production system where it can be safely installed without having to install a compiler on the secure host. If we also want to duplicate our configuration files to the production system we can use the quickpkg utility which is part of the gentoolkit. This will capture the installation as a package including the modified configuration files and place the prebuilt packages in a directory specified by the PKGDIR variable in the /etc/make.conf file. I have found that an easy and pretty secure way to transfer files to the secure server is to set the PKGDIR variable to point to a removable disk such as a USB flash drive. Then the drive can be mounted on the production system as read only and used to deploy the updates.

The emerge process does an automatic MD5 checksum of packages against an md5 file in the portage tree when downloading. It is a good start, since the md5 hashes are kept in the portage tree download. This means that we can at least do a gpg check of the portage snapshot since the pgp checksums for these files are readily available for download. I would suggest, for this reason, that at a minimum, the portage updates be done and gpg verified manually. MD5 checking by the emerge process is an important, although not perfect, verification step for software updates. If packages are to be deployed manually, the md5 checksum should be validated manually as well or the distribution files should be brought over and installed with emerge so that the md5 checks occur automatically.

Unfortunately MD5 checksums are not as good of a method as a gpg check because of known vulnerabilities in MD5 . In fact, NIST (the National Institute of Standards and Technology) has published a statement regarding their recommendation to use SHA-1 over MD5. Hopefully in the near future, the portage update process will switch to using gpg checks of all of the packages instead of MD5. I have found that it is not always possible, even when doing manual tarball installations of applications, to find a gpg check file. Hence we

are still forced to fall back on MD5 checksums, and therefore, I do not feel that the security is all that much improved doing manual installations over doing partially automated updates if the proper pre-testing and validation procedures are in place. Hopefully, if the updates are quicker and easier to deploy to the test system, the updates will get applied regularly so that we can keep a step ahead of the exploitation of the vulnerabilities which are constantly being documented in applications.

Another tool included in the gentoolkit which is still not very mature, but which shows promise is glsa-check. (GLSA=Gentoo Linux Security Advisory) This tool allows us to do a quick check to see if our system has any documented security vulnerabilities due to outdated package deployments. Some people will run this utility in a script and then automate package builds and/or deployments to address any issues found. While this idea has merit, I prefer to just run a daily check on my test system in cron and have the results mailed to me so that I can validate the issues manually. I have seen this implemented where the check is done in cron and then an "emerge -B" is run so that the packages are downloaded and compiled but not installed. This might be a good strategy for our test bed server so that we can look over the vulnerabilities and if we decide to deploy the updates, we can then emerge them on any systems we want much more quickly. While this tool is a good baseline check for a Gentoo system, I still feel that the results need to be validated with a more comprehensive auditing tool such as Nessus.

**Network environment**

In a typical deployment scenario the monitoring server is not advertising services directly to the Internet. The server is typically installed on the production network where it can see the most network resources. This is an inherent requirement of the application and also an inherent vulnerability. Since a monitoring application such as Nagios main purpose is to "watch" all of the critical systems on a network, it is a nice roadmap of those resources for an attacker. This makes it highly critical that the server be protected from the Internet or any other un-trusted hosts. Additionally, to do some types of service checks, the Nagios service may need to log into critical network resources.

In my environment, the server is directly on my test network, there is no DMZ and the two other hosts have been hardened and are "trusted". I have set up some non-critical devices to monitor and have the whole setup behind a firewall. There is no connection to my production network. The entire setup consists of the firewall, a manageable switch, a host to monitor, the test server, and a workstation which I will use for assessing the vulnerabilities in the test system and as a sample monitoring system which will represent the admin user workstation that we would have in a typical production environment.

**Define the new Software or Service**

The network monitoring server is being set up and tested for Deployment for a

small branch office of GIAC Enterprises. The server will be used to monitor resources on all of the corporate switch, server, access point, and the firewall. It may be used later to monitor additional network devices or may be rolled into production at the other locations once it has proven itself as a useful to the IT staff. Initial services to be monitored are detailed as follows:

| | |
|---|---|
| Servers | status for intranet web page<br>status of dns/dhcp services<br>cpu utilization<br>memory utilization<br>disk utilization<br>status of print spooler service |
| Layer 3 Switch | link status on server port<br>link status on ap port |
| Access Points | up/down status of interfaces |
| Firewall | up/down status of management interface (ssh)<br>confirm dns lookup to outside source to check connectivity |

The server will be set up, administered, and updated by Chris, an outside consultant, but responsibility for handling of alerts will be given to Bob, the local network Administrator. Chris is doing the initial build of the system in his test lab and then will move the server into the production environment once it is ready.
To monitor all of the required services on the Windows server a Nagios agent called NSClient will be installed on the server.

**What is Nagios to be used for?**
Nagios will be used to monitor all of the systems listed above. This is a small set of services compared to most environments but the concepts for monitoring are the same regardless of the quantity of hosts being monitored. If any of the services being monitored meets the predefined alert threshold, the event will be logged and a notification will be sent out to Bob and/or Chris.

**Why do Nagios with Apache2?**
It is possible to deploy Nagios without using Apache. It can gather events and send alerts without use of Apache as a front end. Unfortunately, without a web server it would be more difficult to provide Bob with a consolidated view of his network resources. Bob doesn't know Linux at all and wants a nice easy to use interface for viewing reports on the status of his network resources. While this could be done in other ways, Bob saw the web interface for Nagios at a conference and now he wants to have that capability in his network.

**Packages required to complete project:**

net-analyzer/nagios-plugins
sys-fs/reiserfsprogs
sys-kernel/development-sources
sys-kernel/Linux-headers
app-editors/vim
net-dns/bind-tools
sys-process/vixie-cron
net-analyzer/nagios-nrpe
app-vim/nagios-syntax
sys-boot/grub
sys-apps/slocate
net-www/apache
net-analyzer/fping
net-analyzer/nagios
net-misc/netkit-telnetd
sys-kernel/genkernel
sys-process/lsof
net-analyzer/net-snmp
sys-libs/glibc
app-admin/syslog-ng
app-editors/nano
net-analyzer/nagios-core
net-analyzer/nagios-imagepack
app-admin/logrotate
dev-php/mod_php
net-firewall/iptables
app-portage/gentoolkit
dev-php/php
app-forensics/aide

## Security issues introduced by the new software

**Need to advertise a web server –** Because our customer has decided that nice looking reports and a unified view of service status is desired, we have had to introduce an apache web server service onto our server. Since this application was not running before there was no risk of exploiting the system through it. The issues with adding this software are as follows.

A port must be listening on the server for the Apache daemon to handle web browser requests. This is a potential risk because there could be a mis-configuration or a bug in Apache or one of it's modules which could allow an attacker to gain access to the system.

The Nagios information is potentially sensitive so even though we want some People to see it we don't want the information to be available to anyone but Bob the network administrator.

To help minimize the risk of this threat, we will take a number of steps to reduce the threat of opening this service on the network.

First, we will install Apache and take steps to ensure that the server daemon is running with appropriate permissions and that the file system permissions have been properly set up to avoid potential issues related to compromise. We also want to delete any potentially risky files from the web directories and make sure that we have a secure method of accessing any required scripts. The Nagios cgi's themselves contain a mechanism for helping to prevent compromise which we will configure here as well. There is a variable which lets you exclude "dangerous" characters from being used when executing these scripts and this is very important to do. We will of course follow this and all of the other security recommendations from the "Securing Nagios" section of the application's own documentation.

Next, we will set up Apache to use authentication for access to any Nagios data. In fact, since Apache is only being used for Nagios in this case and because only Bob needs access to it, we will set up the Apache server so that it will only accept authenticated connections to any of its pages.

Next, we configure Apache for encryption using a self signed SSL certificate. Bob is the only user of this system so it is easy enough for us to validate the authenticity of the certificate so that Bob's workstation will trust the Nagios server. If many users we going to be monitoring the system, it may be more practical to have the web server certificate signed by a trusted root certification authority. By signing the certificate and enabling SSL, we help to prevent sniffing of sensitive reporting data as well as Bob's password off the network. Next we shut off access to the web server using unencrypted http connections. This helps to ensure that Bob will not inadvertently log into the web server improperly (with no SSL) and pass sensitive information over the network.

Finally, we configure an iptables firewall on the server which restricts the server so that only Bob can access it and only on TCP port 443 for his SSL connection.

**Service checks can contain sensitive information –** Some of the services on the network can only be effectively checked through the use of a login process. This means that we need to have an account which can check the service and the login information must be stored on the Nagios server somewhere so that the checking can be automated. In addition to passing login information to servers, the service checks could also return sensitive information. The results of the check could, for example, pull data out of a database to confirm that SQL reads on the database are working correctly. If the data it pulls out is not planned well, you could unintentionally post sensitive database data to your monitoring server logs or web page.

This problem requires a combination of good policies as well as some technical

good practices. Depending on the specific implementation of the Nagios server, there may or may not be password information stored in the configuration files. In some cases the cgi.cfg file will contain authentication information for Nagios to store data into a MySQL database, in other cases, web server or other database passwords may be stored in the resource.cfg file. It is important to set good permissions on these files so that the web users cannot find a way to get to the information. Another risk, of course, is if these passwords, or in some cases, community strings, are transmitted across the wire in plaintext to do the service checks. My gut feeling on this is that if I can't connect to it securely I would prefer not to check it, but I certainly do not have a consensus on that. SNMP is widely used for monitoring networks, and the community strings are tossed about networks quite freely in plaintext in many environments. If policy requires a check of this type it must be assumed that this password or community string WILL be compromised. Other steps must be taken to ensure that simply having the password will not get you to the information. It is also vital that community strings not match any other system passwords. Since our example network has two situations like this I have documented the steps that I took to help minimize the risk. Many network services have some mechanism for providing password security. Services which do not should ultimately be redesigned (like SNMPv3) or some mechanism should be used for tunneling the data through a protocol such as SSH whenever possible.

Keep in mind that there are new plug-ins developed for Nagios all the time, and most people write some of their own as well. It is important to look closely at each plug-in which you are using and determine that it is safe to use in your environment. Capture the traffic from the plug-in and see what is being passed so that you can evaluate the risk level of each plug-in.

In our scenario, we want to monitor resources on our server. We are using the NSClient on the server to provide information about the server to our Nagios server. This service uses Borland Delphi 7 to gather information about the system and return it to the Nagios server when queried. The traffic on the wire does not give much obvious information about what is being queried but the service authentication is clear text so potentially anyone could acquire that password off the wire and query the service for information. To minimize the risk of this occurring, I configured the built in firewall service on the target machine to only respond to NSClient requests from our monitoring server.

This may not be practical in all environments, so it may be more desirable in many environments to use a WMI based service such as nagios-wse, a handy add on for Nagios which operates on an IIS server and pulls information from Windows servers on the network through the Windows Management Interface (WMI). This is nice because no agents are required on the servers and you can use SSL and web authentication to get access to the information. You can also restrict access to the website so that only the Nagios server can access the information. It does require setting up a hardened IIS server on the network but

it is worth the effort when you are trying to pass as little information across the wire in clear text as possible.

**Server is a blueprint of the network –** Since it is likely that we are monitoring these services because they are important to us, the server is going to have to hold information about the entire layout of our most important resources. If an attacker gets into this system, they will have a wonderfully documented "picture" of the entire infrastructure. That would be bad.

To help prevent an intruder from gaining access to information about our network from our server we need to ensure that our server in general is secure and stays secure. Hardening of the OS is not a task but an ongoing process. It is very common for a system to be set up very securely initially and then over time or through neglect, applications become outdated, passwords become stale and policies for monitoring log files and updating the system are sometimes put off or forgotten. It is critical that a maintenance schedule be developed and strictly followed.

**Linux systems are unfamiliar to many sys-admins -** Since we are often deploying these systems into small environments with limited IT resources, there is frequently no local user who is capable of securely maintaining this critical server. Obviously if they are not comfortable with the system they are not likely to maintain it properly. Critical security updates may not get loaded if there is no plan in place to ensure that they will.

Because this server will be deployed in an environment where the local network administrator is not comfortable with the operating system, some outside assistance will be required to properly maintain the server. As mentioned in the previous section, a rigorous schedule of procedures must be followed to keep the system secure.

**How to maintain security remotely –** Mostly because of the previous issue, we need to come up with a system for keeping the production Nagios server up to date despite the lack of a local security person. There are many potential options for solving this problem. For this project we are only deploying the one server but we need to be prepared to keep not only this server but also those of other customers up to date as well.

Since Bob can only take care of some of the minor tasks, it will fall to me to ensure that the system is up to date and secure. I have no access to the server from the outside world so I can only do work on the server when I visit the project site. To help with monitoring of the security of the system, I have set up various mechanisms for gathering information from the system. Also, because I maintain many very similar systems at various customer sites, I have developed a secure test lab where I pre-download and compile critical updates and test the security and stability of the updates. I also have all of the systems which I

maintain, sending me copies of key log files and other useful information via email. It is very important that the monitoring systems be set up as close to identical as possible from an application standpoint to help reduce the administrative burden of testing for the systems. It would also be preferable to have identical processors on all of the systems so that the packages only have to be compiled once.

Even though the application binaries are tested, configured and scanned for vulnerabilities in the test lab, it is important to validate the security through regular on-site scans as well. Part of the policy for updates must be to do a vulnerability scan of the server following each update. This is a reasonably easy thing to do and it greatly helps validate that the update procedures have been correctly followed.

**Network device reconfiguration considerations –** In order to implement a monitoring system such as Nagios, you often need to turn on services such as SNMPv2. Turning on SNMP on network devices and servers is a potential hazard again because of the information which could be gathered about devices or servers in the system.

As we discussed earlier, the problem that I see with the very common SNMP service checks is that SNMPv2 passes the community string in the clear. Due to that problem, it is highly advisable to try to use SNMPv3 or at least use access lists on your network devices so that only the monitoring station is allowed to use the community string to pull information from your devices. I would also recommend that you only have a "Read" community string defined. If a Read-Write string is required in your environment, ensure that the strings are different for Read-Only and Read-Write and use different strings on different types of devices.

**Implement your Solution**
**Installation guide**
Preparing the Gentoo system -
Because we deploy many similar systems, the base installation is sometimes done by imaging a new machine with a pre-hardened base configuration in our production environment. After this is done, then the binary packages are updated to the appropriate current versions and configured to meet the needs of the specific environment. The initial installation procedure which was followed closely resembles the procedure documented on the Gentoo website in the "Gentoo Linux x86 Handbook". I will highlight some of the key differences in our procedures which were done to enhance the security of the deployment.

First of all, most security professionals are not at all comfortable with implementing a production server which has a compiler on it. Since Gentoo uses Portage and emerge to do just about everything we had to put some

careful consideration into how to build the server without a compiler. Obviously, the answer is to use binary packages, but since not every package that we might want in the world is precompiled for our system we needed to come up with a procedure where we could build a system in our test bed, compile the applications on it, test them for issues, security or otherwise, and then transfer the binaries to our production systems.

My initial test system build was done on a system virtually identical to the production system that I wanted to deploy. To ensure the integrity of the cd media which I used for the initial setup, I downloaded the cd from one of the Gentoo mirror sites and then downloaded the .asc (PGP checksum) from a different mirror. I verified the image as follows:

```
# Obtained appropriate public key
gpg --keyserver pgp.mit.edu --recv-keys 17072058

# Checked the downloaded files to ensure integrity
gpg --verify install-x86-universal-2004.3-r1.iso.asc  install-x86-universal-2004.3-r1.iso
```

I also downloaded and verified the latest version of the portage package and the latest copy of the distribution files package and checked them as well since the ones on the cd are pretty out of date. I burned those files to another cd for use during the install.

I then followed through the installation process as usual but inserted my checked files in instead of following the download procedures in the manual.

Another key difference from the normal instructions is that I will not connect my machine to the network until it is built and secured. For this to be possible, I need to obtain a number of other current packages including a compiler (since this is my test system we are building) as well as iptables, Aide, and chkrootkit so that I can lock the machine down for when I do connect it to the network.

To get the files that I needed without connecting the system to the network I followed the procedures for “Installing from Stage1 without network access” from the “Gentoo Linux Alternate Installation Method HOWTO”

The key steps were from section 4.2, 4.3, and 4.4 as shown here including the packages that I chose:

On the target system:

```
emerge -fp glibc baselayout texinfo gettext zlib binutils gcc ncurses iptables aide vim grub development-sources reiserfsprogs syslog-ng logrotate vixie-cron genkernel gentoolkit chkrootkit 2> stage1.list
```

Note: When we build the production system we will skip gcc, we will not need a compiler because we are going to pre-compile the binaries on the test server.

cut -f 1 -d ' ' stage1.list > stage1.download

Then take the file to another Gentoo machine and run the following:
wget -N -i stage1.download

Once you have obtained all the files, take them to the target computer and copy them to /mnt/gentoo/usr/portage/distfiles

Then you can run emerge again without the "-fp" and the packages will all compile and install.

Once all of these packages exist, the installation process has been completed, and the system is booting on its own. We can now configure iptables and our network settings and connect to the network.

**Iptables**
Since there are no network services running on this system yet it may be paranoid to set up a firewall at this point but I did anyway, so call me paranoid, its a compliment anyway. The following lines are my initial iptables configuration for the test system. For this system, I will be allowing outbound access and permitting established traffic back into my test server but no other traffic will be allowed. Once I have the remainder of my applications installed and configured I will allow access from one other machine for testing of the Nagios and Apache configurations and for vulnerability assessment with Nessus. I have been experimenting with blocking all outbound access other that http and ftp and then setting the wget proxy settings in wget.conf. This seems to be working very well and is much more secure but it requires a bit more equipment in the lab to do the proxy so it may not be ideal for everyone.

I found a couple of sites which were helpful in building my iptables configuration; I borrowed a few lines of config and some good ideas from them.

http://www.faqs.org/docs/iptables/targets.html#LOGTARGET
http://www.siliconvalleyccie.com/Linux-hn/iptables-intro.htm#_Toc92808865

Here are the rules from iptables-save at this point (without proxy) in the process:

```
*filter
#Drop everything by default
:INPUT DROP [0:0]
:FORWARD DROP [0:0]
:OUTPUT DROP [0:0]
```

```
#Allow return traffic from established connections
-A INPUT -i eth0 -m state --state RELATED,ESTABLISHED -j LOG --log-prefix
"ACCEPT-IN:" --log-level 7
-A INPUT -i eth0 -m state --state RELATED,ESTABLISHED -j ACCEPT

#Allow local loopback connections
-A INPUT -i lo -j ACCEPT

#Log inbound drops
-A INPUT -j LOG --log-prefix "DROP-IN:" --log-level 7
-A INPUT -j DROP

# Allow outbound connections to anywhere.
-A OUTPUT -s 192.168.0.103 -j LOG --log-prefix "ACCEPT-OUT:" --log-level 7
-A OUTPUT -s 192.168.0.103 -j ACCEPT

# Accept local loopback connections
-A OUTPUT -o lo -j ACCEPT

#Log outbound drops
-A OUTPUT -j LOG --log-prefix "DROP-OUT:" --log-level 7
-A OUTPUT -j DROP

COMMIT
```

Use the following to make sure the firewall is running all the time
rc-update add iptables default
/etc/init.d/iptables restart

Also make sure that the /etc/conf.d/iptables file has a path set to the iptables-save location:

```
IPTABLES_SAVE="/etc/sysconfig/iptables"
```

I also made a few changes to the sysctl.conf file to help things a bit more:

```
# Disables packet forwarding
net.ipv4.ip_forward = 0
# Log malformed IP's
net.ipv4.conf.all.log_martians = 1
# Disable redirects
net.ipv4.conf.all.send_redirects = 0
# Disable Source Routed Packets
net.ipv4.conf.all.accept_source_route = 0
# Disable ICMP redirects
net.ipv4.conf.all.accept_redirects = 0
```

```
# Disable ping broadcast response
net.ipv4.icmp_echo_ignore_broadcasts = 1
# Disable ECN
net.ipv4.tcp_ecn = 0
# Enables source route verification
net.ipv4.conf.default.rp_filter = 1
```

Now that the test system has a firewall installed I feel a bit better connecting it to the network so I give it an IP address and connect it to my test network.

Now that it is connected, I can run “emerge sync” ,“emerge portage”, and emerge world” to make sure I am current with everything I have on the system so far.

Then I can install the remainder of the packages that I want. Here is a list of the other packages that I installed at this point.

```
nagios
apache     # Make sure that the ssl flag is set in /etc/make.conf
net-snmp
fping
netkit-telnetd
sharutils
ntpd
```

After I have all of the packages that I need installed and configured on my system I can capture the binary files using the following process:

Set the PKGDIR variable in make.conf. In my case I set it to “/mnt/flash/distfiles” then run “quickpkg <packagename>” for each application to be moved to the production system. You can get a list of everything that is on your system by doing an “emerge world -ep”

Other system hardening steps
Set a BIOS password
-I'm hoping everyone knows how to do this because there is no way I could document the procedure for this on every motherboard out there :-)

Set a GRUB password
Here is a quick walkthrough of the process for Gentoo
http://www.gentoo.org/doc/en/gentoo-security.xml#passwording_GRUB

There are several variations of those directions but I found this to be pretty friendly so you should have no excuse to not do it.

I also like to set a timeout on the logins to the server. Bash has a swell mechanism for doing this, you can just set the TMOUT environment variable to a

number of seconds. I added the following line to my /etc/profile.

export TMOUT=300

While I was in there I also set:
ulimit -S -c 0 > /dev/null 2>&1

This prevents core files which we shouldn't need on a working production system anyway. The reason I do this is that sometimes core dumps contain sensitive information and there are plenty of exploits for this type of issue. Here is an example of an exploit in imapd from a few years back to illustrate my point.

http://www.insecure.org/sploits/imapd.core.html

I also set /etc/motd and /etc/issue files with a warning message.

Warning!
This network and associated computer and information systems are the property of COMPANY. The use of these systems is exclusively restricted to those who have received proper authorization.
Any other use is a violation of Title 18 United States Code Section 1030, and is subject to criminal penalties and civil damages. All use of this network and the associated computer and information systems is subject to monitoring at all times. Any use of this system constitutes explicit consent to such monitoring. All information obtained from such monitoring will be provided to the proper investigative authorities.
Warning!

This came from a firewall template that my buddy Mike Gauthier put together a while ago. I suspect that it originally came from a SANS document at some point but I don't know for sure. There are some other nice sample warning documents for various purposes on the SANS website here.

Set up your logfile rotation
I use logrotate to rotate my logfiles, you may prefer a different way but it is certainly a good idea regardless of how you do it. My strategy here is to break things up a bit to make them more manageable. Gentoo likes to dump EVERYTHING into /var/log/messages. Certainly grep is a swell tool but I like my logfiles to be split up slightly more than that. The way that I am doing this on my system is to use syslog-ng to parse some of the data and then dump it into other files for my perusal. Since the things I am most interested in are the firewall logs and the Nagios logs I have focused my efforts there. I am also grabbing some stuff from a switch to illustrate how that is done. In some environments I also grab event log messages from Windows machines or abend logs from Novell servers. We can then (by parsing those files) include more detailed information with alerts when desired in Nagios. Think of the joy of getting a text

message on your phone from your Nagios server that says not only that the Print Spooler service stopped but also what event id number was logged when it happened. Surely you can see the value here. I use “Snare” to do that on Windows by the way. There are certainly other ways as well....

Nagios has it's own built in log rotation which occurs daily. Iptables just dumps a bunch of junk into the /var/log/messages file. You will have noticed earlier that I tagged a log prefix onto my firewall log messages, those are what I will use here to parse out the data. Here is a copy of my syslog-ng config file so you can see how I did this.

```
##########################################
###syslog-ng.conf
##########################################
options {
        long_hostnames(off);
        sync(0);
        stats(43200);
};

source src { unix-stream("/dev/log"); internal(); file("/proc/kmsg"); };
source net { udp(ip(192.168.0.103) port(514) ); };

# Network Devices
filter switch { facility(local7) and host("nt1100"); }; # My Switch

# local firewall
filter acceptin         {match(^ACCEPT-IN:); };  # iptables LOG prefix
filter acceptout        {match(^ACCEPT-OUT:); };  # iptables LOG prefix
filter acceptfw         {match(^ACCEPT-FW:); };  # iptables LOG prefix
filter dropin           {match(^DROP-IN:); };  # iptables LOG prefix
filter dropout          {match(^DROP-OUT:); };  # iptables LOG prefix
filter dropfw           {match(^DROP-FW:); };  # iptables LOG prefix

# Default locatio for non-console messages
destination messages { file("/var/log/messages"); };

# Destination for network device messages
destination netlog { file("/var/log/net/net.log"); };

# local firewall logfile locations
destination acceptin { file("/var/log/firewall/acceptin.log"); };
destination acceptout { file("/var/log/firewall/acceptout.log"); };
destination acceptfw { file("/var/log/firewall/acceptfw.log"); };
destination dropin { file("/var/log/firewall/dropin.log"); };
destination dropout { file("/var/log/firewall/dropout.log"); };
```

```
destination dropfw { file("/var/log/firewall/dropfw.log"); };

# By default messages are logged to tty12...
destination console_all { file("/dev/tty12"); };

# Send network device messages to /var/net/net.log
log { source(net); filter(switch); destination(netlog); };

# Send firewall messages to specified locations
log { source(src); filter(acceptin); destination(acceptin); };
log { source(src); filter(acceptout); destination(acceptout); };
log { source(src); filter(acceptfw); destination(acceptfw); };
log { source(src); filter(dropin); destination(dropin); };
log { source(src); filter(dropout); destination(dropout); };
log { source(src); filter(dropfw); destination(dropout); };

# Send console messages to tty12
log { source(src); destination(console_all); };

# Send all uncategorized messages to /var/log/messages
log { source(src); destination(messages); };

################################
```

There are a couple of other things that I would like to keep track of on the box to help keep me felling warm and cozy. I am going to set up “aide” to monitor my system binaries for changes and am also going to run a rootkit check every day. Since I am too lazy to go visit every one of my customers every day to check the results I will have the results emailed to me. In real life I like to send these to a group so that if I die someone will still check the files for me. I am also going to use glsa-check daily to let me know if I am missing any patches to address known Gentoo security vulnerabilities.

Here are the cron jobs that I used to take care of those tasks. I placed these in /etc/cron.daily. I'm using vixie-cron, personal choice, pick what you like... The rootkit check is pretty painless, the aide check keeps the box busy for a while.

```
#! /bin/sh
touch /var/log/rootkit.log
chkrootkit > /var/log/rootkit.log
uuencode /var/log/rootkit.log rootkit.log | mail -s nagios1rootkitchk virgil@foo.bar
```

```
#! /bin/sh
touch /var/log/aide.log
aide --update > /var/log/aide.log
uuencode /var/log/aide.log aide.log | mail -s nagios1aidechk virgil@foo.bar
```

```
#! /bin/sh
touch /var/log/glsa.log
emerge sync
glsa-check  -l > /var/log/glsa.log
uuencode /var/log/glsa.log glsa.log | mail -s nagios1glsachk virgil@foo.bar
```

Now that we have some nice log files, we want to help make sure that we keep them safe. I like to email them off the box every day so I can put them back and see if they were messed with. You can see that I already sent the aide and chkrootkit results off the box, now I want to get a copy of the other important logfiles as well. The files that I am interested in are the firewall logs, the message logs, and the Nagios logs. To get those files, I made a modification to the script which runs logrotate each day. Now once it completes the rotation, it emails me the files as illustrated here:

```
#! /bin/sh

/usr/sbin/logrotate /etc/logrotate.conf
uuencode /var/log/messages.1.gz messages.gz | mail -s nagios1messages
virgil@foo.bar
uuencode /var/log/firewall/acceptin.log.1.gz acceptin.gz | mail -s
nagios1acceptinvirgil@foo.bar
uuencode /var/log/firewall/acceptout.log.1.gz acceptout.gz | mail -s
nagios1acceptout virgil@foo.bar
uuencode /var/log/firewall/dropfw.log.1.gz dropfw.gz | mail -s nagios1dropfw
virgil@foo.bar
uuencode /var/log/firewall/dropin.log.1.gz dropin.gz | mail -s nagios1dropin
virgil@foo.bar
uuencode /var/log/firewall/dropout.log.1.gz dropout.gz | mail -s nagios1dropout
virgil@foo.bar
uuencode /var/log/nagios/nagios.log nagios.log | mail -s nagios1nagioslog
virgil@foo.bar
```

**Installation/Config of Nagios**

Apache configuration – Since Nagios needs Apache for our scenario, we need to get that going on the box, we already emerged it and Nagios so what we need is already on the box, now we need to configure it. I would like at this point to give credit to Mike Gauthier who I work with. He figured out a great deal regarding the Nagios installation process and documented it all very well in a series of scripts. Since we work together to deploy these servers, the next part of this document will have stuff from the scripts that he worked out for some of our initial deployments. None of these scripts are published, but I wanted to give credit where it is due and his advice and guidance have been invaluable to me while compiling this document.

I found a nice article on secure Apache installation on a Berkley “Computer

Protection Program" site that I liked and I would recommend that you read that before turning on Apache.

Gentoo nicely runs apache as an apache user by default so we don't have to deal with that but please don't run it as root.

First we need to clean out the default junk from the Apache document directory and create a link to the Nagios web stuff.

```
cd /var/www/localhost/htdocs/
rm index.html.*
rm apache_pb*
rm manual
ln -s /usr/nagios/share monitor
```

Then we need to modify /etc/apache2/conf/commonapache2.conf to set up our authentication.

```
<Directory "/usr/nagios/">
        AllowOverride None
        order deny,allow
        allow from all
        AuthUserFile /web/access/user
        AuthGroupFile /web/access/group
        AuthName "Monitor Page"
        AuthType Basic
        require group admin
</Directory>

<Directory "/var/www/localhost/securehtdocs/">
        Options ExecCGI FollowSymLinks
        AllowOverride None
        order deny,allow
        allow from all
        AuthUserFile /web/access/user
        AuthGroupFile /web/access/group
        AuthName "Monitor Page"
        AuthType Basic
        require group admin
</Directory>
```

Then we need to create our user(s) and assign them to a group.

```
cd /
ln -s /etc/apache2 web  #this makes life a bit easier getting to stuff
mkdir /web/access
```

#notice that the access directory is not in the directory tree of the web server. I
#never get why people do that but they do so please don't, that is bad.
cd /web/access
htpasswd2 -cb user chris <PASSWORD>
htpasswd2 -cb user bob <PASSWORD>
echo 'admin: chris,bob' > group

**Then we need to configure SSL on Apache**

**SSL configuration:**
On Gentoo you will need to uncomment the following line in /etc/conf.d/apache2
APACHE2_OPTS="-D SSL"

Then you will need to edit /etc/apache2/conf/apache2.conf and comment out the following line to shut off “regular” http:

#Listen 80

Also make sure these are in there:

# Hide server advertisements
ServerSignature Off
ServerTokens Prod

While in the same file, add a virtual host section similar to this:

```
<VirtualHost 192.168.0.103:443>
        DocumentRoot /var/www/localhost/securehtdocs
        DirectoryIndex index.htm index.html
        ServerName nagios.foo.bar
        ServerAdmin virgil@foo.bar
        SSLEngine on
        SSLCipherSuite HIGH:MEDIUM
        SSLCertificateKeyFile conf/ssl/server.key
        SSLCertificateFile conf/ssl/server.crt
        ErrorLog /var/log/apache2/error_log
        TransferLog /var/log/apache2/transfer_log
        ScriptAlias /nagios/cgi-bin/ /usr/nagios/sbin/
        RewriteEngine on
        RewriteCond %{REQUEST_METHOD} ^(TRACE|TRACK)
        RewriteRule .* - [F]
</VirtualHost>
```

Some people prefer to put that in vhosts.conf so feel free to do so if you wish but then you will have to have a line in apache2.conf that looks like this:

Include conf/vhosts/vhosts.conf

Obviously you need to generate a key to put in those SSL directories. This can be either a self signed key as I did or you can go buy one which is signed by a trusted root certifications authority if it makes sense for your environment.

This is how you create the self signed cert according to the Gentoo Linux Handbook. It worked out fine for me.

```
#Step one - create the key and request:
openssl req -new > new.cert.csr

#Step two - remove the passphrase from the key (optional):
openssl rsa -in privkey.pem -out new.cert.key

#Step three - convert request into signed cert:
openssl x509 -in new.cert.csr -out new.cert.cert -req -signkey new.cert.key -days
1825

#Step four, copy the proper files to the apache directories so you can use them
cp new.cert.cert /etc/apache2/conf/ssl/Server.crt
cp new.cert.key /etc/apache2/conf/ssl/Server.key
```

Thats about all we need for apache so then do this:

```
/etc/init.d/apache2 start
rc-update add apache2 default
```

That will start Apache and set it to run on startup.

Now lets work on Nagios a bit. At this point, Nagios really has no configuration files so you can either snag them from another box to get started as I often like to do or you can do this to get the sample ones.

```
for f in services contactgroups hosts cgi resource contacts \
        hostgroups checkcommands escalations timeperiods nagios \
        misccommands dependencies; do
 cp /usr/share/doc/nagios-core-1.2-r2/sample-configs/$f.cfg-sample.gz \
    /etc/nagios
 gunzip /etc/nagios/$f.cfg-sample.gz
 mv /etc/nagios/$f.cfg-sample /etc/nagios/$f.cfg
done
```

Obviously, the version number has to be accurate for that to work. There is a lot of sample stuff in those files and you will probably not use much of it and will probably customize the files a great deal as you get going but this is better than

starting with nothing. I will include the critical portions of the configuration files as I go along here.

The next step is to configure the cgi.cfg file which controls the behavior of the cgi's. You will want to set the permissions in these lines:

authorized_for_configuration_information=chris,bob
authorized_for_system_commands=chris,bob
authorized_for_all_services=chris,bob
authorized_for_all_hosts=chris,bob
authorized_for_all_service_commands=chris,bob
authorized_for_all_host_commands=chris,bob

You will want to add

Then make sure that this is set in resource.cfg or you will get an annoying message about Nagios not being able to tell what it's own state is.
$USER1$=/usr/nagios/libexec

Then you will want to add the nagios user to the apache group

usermod -G nagios apache

Then you will want to check some permissions or just set them to make sure they are right. I got the information on how this is supposed to be set up from the Nagios security documentation:
http://nagios.sourceforge.net/docs/2_0/commandfile.html

chmod 2770 /var/nagios/rw
“nagios” is the owner of this folder and “apache” is the group. This is required for Nagios to work properly for running external commands.
You also need to modify the following lines in nagios.cfg
check_external_commands=1 (is 0 by default)
command_file=/var/nagios/rw/nagios.cmd
illegal_macro_output_chars=`~$^&”|'<>
illegal_object_name_chars=`~!$%^&*”|'<>?,()=

This is slightly scary because Nagios will execute any commands it finds in the file specified here. Thats why we want the permissions right and why we don't run Nagios as a privileged user. This would be VERY BAD if Nagios were installed on a multi-user system so make sure this is not on that sort of a box.

I also make sure that all of the files in /etc/nagios are set with owner=root, group=apache and have -rw-r----- permissions. The resource.cfg file gets -rw------ permissions because it can contain service check password information and is designed fo use in this manner. The cgi's do not need to directly read this file.

Avoid putting sensitive information in the other config files.

Then you need to set up all of the hosts, hostgroups, service checks, services, timeperiods, dependencies, and escalations. This is the time consuming part. At a minimum, you will want to make sure that you have your contacts.cfg set up with sections such as this:

```
# 'nagios' contact definition
define contact{
        contact_name                    chris
        alias                           Chris Dahlke
        service_notification_period     24x7
        host_notification_period        24x7
        service_notification_options    w,u,c,r
        host_notification_options       d,u,r
        service_notification_commands   notify-by-email,notify-by-epager
        host_notification_commands      host-notify-by-email,host-notify-by-epager
        email                           virgil@foo.bar
        pager                           5555551212@message.phoneco.com
        }
```

Then you will want to define one or more contact groups as appropriate.
#contactgroups.cfg

```
# 'GIAC' contact group definition
define contactgroup{
        contactgroup_name  GIAC
        alias              GIAC
        members            chris,bob
        }
```

Then you will want to determine what service checks you need and configure them:

Here are some samples from checkcommands.cfg:

```
# 'check_telnet' command definition
define command{
        command_name    check_telnet
        command_line    $USER1$/check_tcp -H $HOSTADDRESS$ -p 23
        }

# 'check_dns' command definition
define command{
        command_name    check_dns
        command_line    $USER1$/check_dns -H www.yahoo.com -s
$HOSTADDRESS$
```

```
# 'check_nt_C_disk' command definition
define command{
        command_name    check_nt
        command_line    $USER1$/check_nt -H $HOSTADDRESS$ -p 1248 -v
USEDDISKSPACE -l C -w 80 -c 90 -s $USER5$
#Note that $USER5$ is defined in resource.cfg and is the password for the
service check on this one.
```

There are pretty much unlimited check commands available so this can take some time to decide what to check.

Then you will want to define the actual check parameters in services.cfg

```
# Generic service definition template.  You can have multiple templates to call
from and tell each service which one to use.

define service{
        name                            generic-service ; The 'name' of this service
        active_checks_enabled           1       ; Active service checks are enabled
        passive_checks_enabled          1       ; Pas serv checks are
enabled/accepted
        parallelize_check               1       ; Active serv checks should be parallel
        obsess_over_service             1       ; We should obsess over this service
        check_freshness                 0       ; Default is to NOT check service '
        notifications_enabled           1       ; Service notifications are enabled
        event_handler_enabled           1       ; Service event handler is enabled
        flap_detection_enabled          1       ; Flap detection is enabled
        process_perf_data               1       ; Process performance data
        retain_status_information       1       ; Retain status information across
        retain_nonstatus_information    1       ; Retain non-status information a
        register                        0       ; DONT REGISTER THIS DEFINITION -
        is_volatile                     0
        check_period                    24x7
        max_check_attempts              5
        normal_check_interval           3
        retry_check_interval            1
        notification_interval           0
        notification_period             24x7
        notification_options            w,u,c,r
        }
```

A typical service definition will look like this:

```
define service{
        use                             generic-service              ;Name of service template
```

```
        host_name               switch          ;Host name as defined in dns
        service_description     Telnet          ;Description of check
        contact_groups          GIAC            ;as defined in contactgroups.cfg
        check_command           check_telnet    ;as defined in
checkcommands.cfg
        }
```

If you copied the default config files and dont want to use escalations.cfg or dependencies.cfg you will need to clean up those files as well as the misccommands.cfg file. All of those files contain a bunch of junk examples that will break Nagios when you try to start it otherwise.

Now that the server is set up the way we want, we can configure what we want to watch with aide.
To set up aide to monitor our important system binaries:

```
#initialize database
aide –init

# Then copy /etc/aide/aide.conf and aide.db to RO media (cd, net, or ro flash?)
cd /etc
mkisofs -f -R -r -l -J -VSAFE -oAIDESAFE.iso aide/
cdrecord -v -pad speed=1 dev=/dev/hdc AIDESAFE.iso
mount /mnt/cdrom

#regularly update and compare aide.db.out database against cd
aide –update > /var/log/aide.log # We will also send this off the box as shown
earlier.
```

This check requires that the initial and new database locations be set properly in aide.conf.

Here are the changes that I made to the aide configuration file for my project:

```
#########################################################################
#Make the following changes to this file - add more checks if desired

@@define CONFDIR /etc/aide
@@define SAFE /mnt/cdrom
database=file:@@{SAFE}/aide.db #create manually after update (check log
first)
database_new=file:@@{CONFDIR}/aide.db.out #used for compare option
database_out=file:@@{CONFDIR}/aide.db.out #used in init and update option
report_url=file:/etc/aide/report.txt #report of changes to /usr
MYCheck=s+n+b+md5+sha1+rmd160 # Set parameters for check
/usr MYCheck  #Check all of /usr to catch binaries
/etc MYCheck  #Check all of /etc to catch config changes
```

```
#Some people like to check logs with this as well, I prefer to send critical logs o
ff the box, if you want to do that, set a rule definition something like "LogRule=>
" and assign it to your logfile locations ie: /var/log LogRule

# Note: There are a number of default rules (mostly exceptions) in aide.conf
which I commented out as well

@@{TOPDIR}/.* Norm

#!@@{TOPDIR}/.*~
#!@@{TOPDIR}/src/.*\.o
#!@@{TOPDIR}/src/(aide|core)$ L
#!@@{TOPDIR}/.*RCS
#!@@{TOPDIR}/.*CVS
#!@@{TOPDIR}/.*aide\.db.*
#!@@{TOPDIR}/.*\.cvsignore.*
```

You could also copy aide.db.out to aide.db each time your run this to see only current changes but that really should only be necessary when you update the binaries.

Make sure to compare against a known good aide.db (hence the cd)
Look for unexpected changes to binaries

**Preparing Servers**

Preparing the servers will depend of course on the type of servers and the type of checks you are doing. You don't really need to prepare much for a DNS check or a ping check but if you want to check cpu utilization or disk space usage you will need to do a bit of work. In our case, we need to check a Windows server for various things and we have decided that we will use the Nsclient to do this. This means that on our server we need to take a couple of steps. First we need to download and install the Nsclient from http://nsclient.ready2run.nl/download.htm

Then we need to copy it to a folder, install it, set a password, and start the service as is well documented in the readme.html file which comes down with the install zip file.

The service checks for this using “check_nt” are very well documented in that file as well.

I mentioned earlier that the password for this check is in clear text so if you use this tool please also set up a firewall on the server that only allows your management server to connect to port TCP/1248. If you can't do that, then I would suggest using a different method for gathering the information from the server.

**Preparing Network Devices**
Again, the amount of work to do here depends on what checks you are planning to do. Since I am usually connecting to all of the devices anyway I like to take this opportunity to set up some improved security. For devices that support it, I shut of RW SNMP and set a long nasty RO community string such as "eico2aoshuqueeteeRooneeteiphoogiekahfeib". I also usually reset the switch password to something good like "875h3lhfjfh4rysdf" which I'm pretty sure is not in the dictionary. Also, if the device supports a separate privileged password, use a different one for that.
Please choose your own passwords, I just whacked away on the keyboard a bit for those but there are some swell random password generators around if you are not feeling too creative. pwgen seems pretty ok or some people prefer to make their own random generator. Feel free to decide yourself but please do not use "m0n3y" if you work at a bank or "$ch00l" if you work at a school. Those are better than many I have seen. You get the idea....

In addition to making the passwords better, I like to turn off telnet and any sort of web administration on network devices. Telnet is just silly to use so use SSH instead. If you like web interfaces for some reason, please at least turn on SSL for them. At least then while you are inefficiently clicking away in your slow java interface, you won't be dumping your password on the wire. :-)

You should also set up either a management VLAN or an authorized managers access list or both so that when your long nasty community string gets compromised they still won't be able to use it unless they spoof your IP address. The more work for the bad guys the better.

**The Production Server**
Well, now our test server is pretty much set up and ready to go. I want to check it over for problems, fix them and then start setting up my production box.
Here are the things I checked on my server before preparing to transfer the setup to my production box.

**Audit Procedure**

- Ran chkrootkit – ok
- Sniffed traffic from the box for several hours and checked for discrepancies against my emailed firewall logs or anything else odd. - ok (Could improve part this with a more automated process using IDS like Snort)
- Performed a nessus scan from my management station and corrected the issues that I found with a couple of modules in Apache. - ok
- Performed a manual glsa-check and fixed an openssl vulnerability that I found. - ok

**Update Procedure – for production system**

Now that I am reasonably sure that the system is running the way I want, I am going to run quickpkg on all of the packages on this system. I have set my PKGDIR to /mnt/flash. To automate this task somewhat, I pulled a list of current packages on the system with “emerge -ep world | cut -f2 -d'/' > packagelist”. This gives me a nice list of packages. After a tiny bit of cleanup to that file, I can do this:

for f in $(cat packagelist); do quickpkg =$f; done

which creates nice packages of all of my applications with the config files that I have changed.

This process assumes that you want to have the most current versions of everything on your systems. If that is not the case for some reason you will need to adjust your process a bit in order to get the older files to the production system. If it is just a package or two, I would recommend manually emerging the appropriate versions and deleting the newer version ebuild files from the appropriate directories in /usr/portage/<package>.

Now that we have the files we need on our large flash drive, we can move it over to the new system and build it using those.

Here we pretty much go back and start over with our Gentoo installation process except that now we will set our /etc/make.conf PKGDIR variable early on in the process and we will mount our flash device so that we can very quickly install all of the same packages. This system can be built completely off-line because we don't need to download any packages. Also remember the VERY important step of not loading the gcc package. We don't want a compiler on this system.

Once the system has been built we can verify all of the config files and set unique system settings like IP address. We also need to make the appropriate adjustments to out iptables configuration for our production environment. This system does not really need outbound access to the Internet but it does need access to pretty much everything on our local network (so we can actually monitor). So we will at least need to make a change like this to our config.

Change from something like this:
```
# Allow outbound connections to anywhere.
-A OUTPUT -s 192.168.0.103 -j LOG --log-prefix "ACCEPT-OUT:" --log-level 7
-A OUTPUT -s 192.168.0.103 -j ACCEPT
```

To something like this:
```
# Allow outbound connections to anywhere.
-A OUTPUT -s 192.168.0.103 -d 192.168.0.0/24 -j LOG --log-prefix "ACCEPT-
OUT:" --log-level 7
-A OUTPUT -s 192.168.0.103 -d 192.168.0.0/24 -j ACCEPT
```

You will probably be using a different IP scheme as well obviously....

You will also need to change passwords, hostnames, domainnames, and a lot of stuff in the nagios config. I would also generate new crypto keys for apache and ssh if you decide to use it for management of the server. (If you do that you will need to allow it in the firewall config.)

One other check that will need to be made is that the glsa-check should be removed on the production system. Since that system does not communicate with the Internet, it can't do the emerge sync and the results will quickly become outdated. We will run this check only in the test environment to help validate our choice of updates.

**Remote management considerations**

As much as I would love to do it, because it would make my life much easier, I can't recommend allowing remote access of any kind into these servers. Sure there are ways to make it fairly secure. I could set up a VPN tunnel to my office or open ssh up from the outside to my server but I really don't need it enough to justify the risk. I have decided that it is better to leave the box nice and disconnected from the Internet.

Because the box is not connected directly to the Internet we need to plan out an update schedule. As much as I would like to do daily updates on the servers I can't justify that to my customers, nor could I physically accomplish it for that matter.

Since I will have a solid test environment to work from, I want to ensure that I have a good testing procedure for that box. The production systems will be reasonably easy to maintain once we have a good procedure in place.

**Update Procedures**

**Daily Tasks:**

- Review logs – especially aide and chkrootkit. Backup systems may vary a bit but daily backup logs should be reviewed as well.
- emerge sync test system – we handle this in the glsa check script to ensure that our package list is current.
- If anything questionable is noted in the logs or in the aide or glsa check investigate immediately. Notify customers if any significant issue is discovered.
- Review any security updates to determine the threat level of not applying the patch.
- If a significant threat is noted, update the test system immediately and check the system for security and stability issues. Follow Audit Procedure from above. Update aide database and burn a new cd.

- If the threat is deemed significant enough, arrange for onsite visits to update customer servers ASAP. Generate deployment packages to RO media following Update Procedure as documented earlier.
- If the threat is deemed minimal, or if there are other updates to be done, follow the usual weekly procedure for updates.

**Weekly tasks**

- Apply all desired updates to the test system
- Review SecurityFocus websites for bugs and other issues that may affect security. Also subscribe to mailing lists for bugs on packages in your deployment. Once done, review those emails regularly.
- Test system security and stability. Follow Audit Procedures from above.
- Generate update packages for client deployment on RO media. Follow Update Procedure as documented earlier.
- Visit each customer site to deploy updates and deploy them.
- Update the aide database on the server
- Burn a new updated database cd for aide and mount it on the system.
- Perform a nessus scan of the server and note any problems to be corrected first in the test bed.

An interesting thing of note is that as I followed my own procedures here over a several week timeframe, I discovered and corrected some vulnerabilities in my systems while going through the process. Here is a list of the most recent vulnerabilities that my process found and which I corrected on my servers.

**Nessus Scan Found:**
Vulnerability found on port https (443/tcp) :

The remote host is using a version of mod_ssl which is
older than 2.8.18.

This version is vulnerable to a flaw which may allow an attacker to disable the remote web site remotely, or to execute arbitrary code on the remote host.

*** Note that several Linux distributions patched the old version of
*** this module. Therefore, this alert might be a false positive. Please
*** check with your vendor to determine if you really are vulnerable to
*** this flaw

Solution : Upgrade to version 2.8.18 or newer
Risk factor : Low
CVE : CAN-2004-0488
BID : 10355

**(This was a false positive.)**

. Warning found on port https (443/tcp)

Your webserver supports the TRACE and/or TRACK methods. TRACE and TRACK
are HTTP methods which are used to debug web server connections.

It has been shown that servers supporting this method are subject
to cross-site-scripting attacks, dubbed XST for
"Cross-Site-Tracing", when used in conjunction with
various weaknesses in browsers.

An attacker may use this flaw to trick your
legitimate web users to give him their
credentials.

Solution: Disable these methods.

If you are using Apache, add the following lines for each virtual
host in your configuration file :

```
RewriteEngine on
RewriteCond %{REQUEST_METHOD} ^(TRACE|TRACK)
RewriteRule .* - [F]
```

If you are using Microsoft IIS, use the URLScan tool to deny HTTP TRACE
requests or to permit only the methods needed to meet site requirements
and policy.

If you are using Sun ONE Web Server releases 6.0 SP2 and later, add the
following to the default object section in obj.conf:

```
<Client method="TRACE">
 AuthTrans fn="set-variable"
 remove-headers="transfer-encoding"
 set-headers="content-length: -1"
 error="501"
</Client>
```

If you are using Sun ONE Web Server releases 6.0 SP2 or below, compile
the NSAPI plugin located at:
http://sunsolve.sun.com/pub-cgi/retrieve.pl?doc=fsalert%2F50603

See http://www.whitehatsec.com/press_releases/WH-PR-20030120.pdf
http://archives.neohapsis.com/archives/vulnwatch/2003-q1/0035.html
http://sunsolve.sun.com/pub-cgi/retrieve.pl?doc=fsalert%2F50603

http://www.kb.cert.org/vuls/id/867593

Risk factor : Medium

**(As you can see, this was a valid configuration flaw which I corrected and fixed in this documentation.)**

**glsa-check found**

GLSA 200411-15
1. Gentoo Linux Security Advisory

Version Information

| | |
|---|---|
| **Advisory Reference** | GLSA 200411-15 / OpenSSL |
| **Release Date** | November 08, 2004 |
| **Latest Revision** | November 08, 2004: 01 |
| **Impact** | Normal |
| **Exploitable** | Local |

| Package | Vulnerable versions | Unaffected versions | Architecture(s) |
|---|---|---|---|
| dev-libs/openssl | < 0.9.7d-r2 | >= 0.9.7d-r2 | All supported architectures |
| sys-apps/groff | < 1.19.1-r2 | >= 1.19.1-r2 | All supported architectures |

**(This turned out to be slightly sneaky because I was at the right version but had not yet run "etc-update" to update my configuration files. This turned out to be needed to fix the issue.)**

You can see, I'm sure, that there is a fair amount of work involved with checking and re-checking these systems but once you have a good base-line for your systems it is really not bad to keep up with.

**Summary**

When I started this document I had some reservations about the feasibility of adequately securing a server that has to talk to so many network devices. While I still feel that there are some areas that could certainly be improved on I think that I have made a good deal of progress towards improving my own approach when deploying these servers. Many of the things that I learned to do while working on this project, such as building a Gentoo box off-line will benefit me greatly in my work. Not talking to the Internet is a good thing from a security

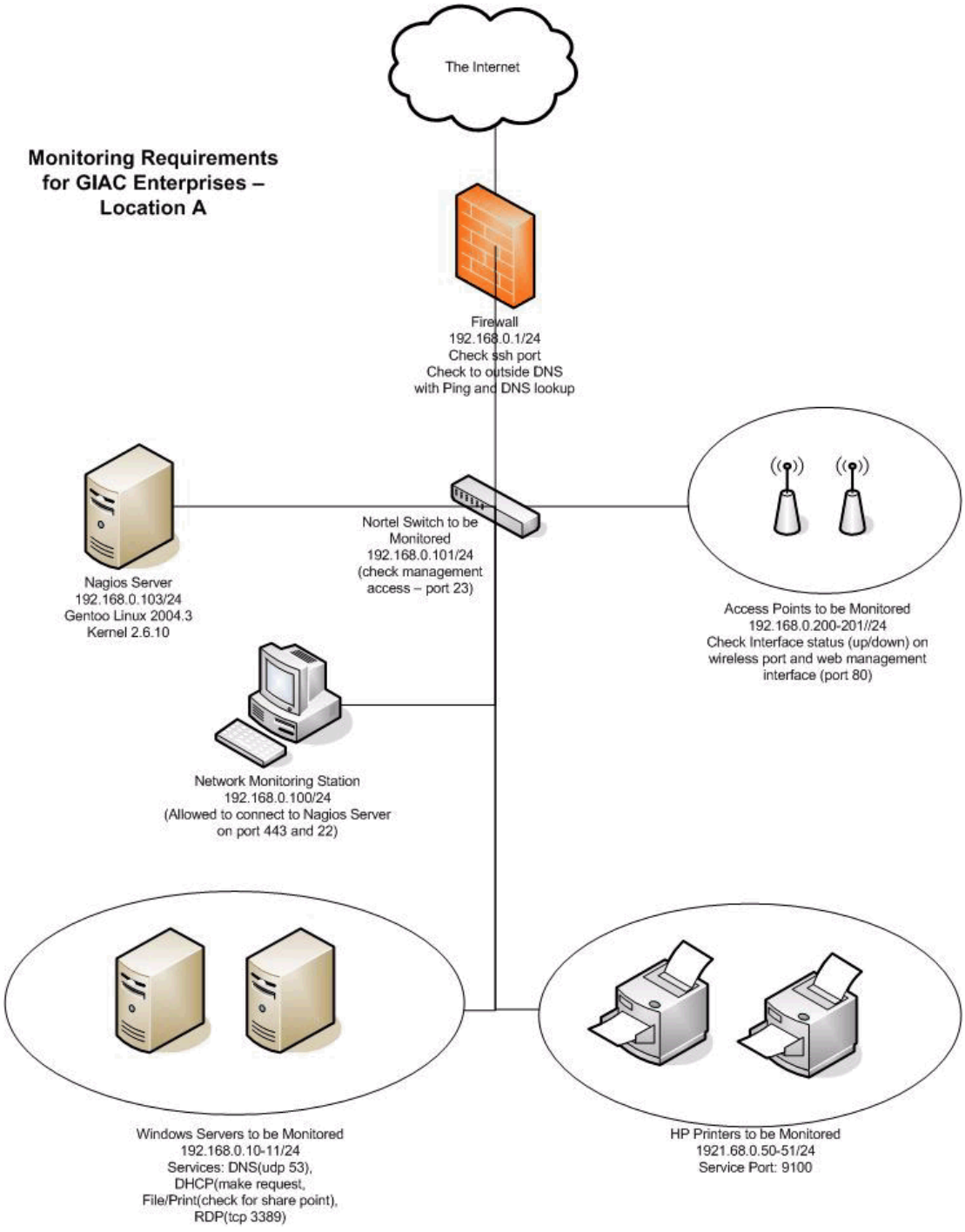
perspective, in my opinion. Some of the issues that I see yet with Nagios are probably obvious to you as well.

My biggest concern is that every plug-in for Nagios, and the list is changing all the time, could have a potential security impact on the system as a whole. One thing that I plan to continue to do as I move forward, is to look at each of the plug-ins one at a time and separately document the potential security issues of all of the ones that I want to use. Since this could turn into a book rather than a paper, I only looked at a few of them in this document. I will continue to pursue this though as I move forward. Perhaps a Nagios plug-in security advisory site may need to turn up in the future.

Another thing that I would like to explore further is the potential for using a different management interface than Apache for looking at my Nagios data. I have seen that there are a number of add-ons which are working towards this goal and I will look into those in more depth as well. If I could be rid of Apache, it would certainly make it easier to maintain these servers. I would also like to look into multi-server Nagios deployments and explore the potential risks of passing data back and forth between multiple servers. I do not have a customer doing this yet but it is likely to come up in the future so I would like to be more prepared there.



**Here is a diagram of the sample environment.**




## References


http://www.nagios.org
Ethan Galstad, March 8, 2005, Official Nagios Website

http://eprint.iacr.org/2004/199.pdf

# Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD

Xiaoyun Wang, Dengguo Feng, Xuejia Lai, Hongbo Yu[1]
The School of Mathematics and System Science, Shandong University, Jinan250100, China[1]
Institute of Software, Chinese Academy of Sciences, Beijing100080, China[2]
Dept. of Computer Science and Engineering, Shanghai Jiaotong University, Shanghai, China[3]
xywang@sdu.edu.cn[1]
revised on August 17, 2004

http://csrc.nist.gov/hash_standards_comments.pdf
NIST Brief Comments on Recent Cryptanalytic Attacks on Secure Hashing Functions And the Continued Security Provided by SHA-1 – 08-25-2004

http://nagios.sourceforge.net/docs/2_0/security.html
Ethan Galstad, March 8, 2005, Official Nagios Website

http://nagios-wsc.sourceforge.net/
Fjthomas, Mar, 21 2005

http://www.gentoo.org/doc/en/handbook/handbook-x86.xml?full=1
Copyright 2001-2005 Gentoo Foundation, Inc.

http://www.gentoo.org/doc/en/altinstall.xml
Copyright 2001-2005 Gentoo Foundation, Inc.

http://www.sans.org/resources/policies/
© 2002-2005 The SANS™ Institute

http://www.intersectalliance.com/projects/SnareWindows/index.html#Download
InterSect Alliance Pty Ltd
PO Box 1210
Belconnen, Canberra ACT 2617
Australia

http://www.lbl.gov/ITSD/Security/systems/apache-server.html
Berkely Lab – Ernest Orlando Lawrence Berkely National Laboratory – Computer Protection Program

http://www.securityfocus.com/archive/1
Copyright © 1999-2005 SecurityFocus