



Global Information Assurance Certification Paper

Copyright SANS Institute
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permitted without express written permission.

Interested in learning more?

Check out the list of upcoming events offering
"Security Essentials: Network, Endpoint, and Cloud (Security 401)"
at <http://www.giac.org/registration/gsec>

Security Features Overview of FreeBSD

Introduction

Building secure systems requires an understanding of what tools and features are available and how to use them. Without this knowledge we severely limit the effectiveness of our security solutions. The goal of this paper is to provide an overview of the FreeBSD security features and its tools. While we could write an entire document on each of the tools and features discussed here we felt it was important to provide the reader with an overview that would enable him to focus on issues relating to their environment. The reader is encouraged to dig deeper should he or she require more information.

Although there are many additional pieces of software, both open source and commercial, that may enhance the security of a FreeBSD system we will be focusing only on what FreeBSD comes with “out of the box”. We feel that it is important to understand what each OS can and can’t do on it’s own before adding additional and sometimes unnecessary software into the mix.

FreeBSD provides us with a set of tools to achieve a reasonable level of security. We will examine these tools along with FreeBSD’s security features that will assist the reader in attaining the level of security required by their organization.

Sysctl

Administrators looking to enhance their systems security, retrieve kernel parameters, alter the system’s network performance or read one of the over 500 system configuration values need to look no further than Sysctl. Sysctl is the mechanism by which FreeBSD reads or writes information inside the running kernel. This allows administrators to make changes to the many subsystems of the operating system without recompiling a kernel and sometimes without a reboot.

Sysctl is composed of two components. The first is the information, which is stored inside the running kernel in a management information base or MIB style tree. The second is the system calls to read or write to this tree. Thankfully administrators don’t have to write code to access this information because FreeBSD provides a nice sysctl binary that does most of the work for us. Using the various options and switches we are able to retrieve and change over 500 values. Let’s take a brief look at how the sysctl tool works.

Each element or variable inside the sysctl hierarchy can be accessed by its name. Because sysctl is hierarchical obtaining access to the variables must be done in a fashion similar to

a directory structure. To do this we use dots to separate the different elements. See below for a few examples.

Example 1.

View the contents of the entire tree.

```
#sysctl -a
```

Example 2.

View the hostname variable.

```
#sysctl kern.hostname
```

Example 3.

Set the tcp blackhole variable to one.

```
#sysctl net.inet.tcp.blackhole=1
```

Sysctl provides us with the tools to enhance the security of our systems. We will examine some of the security related variables in detail.

- `Net.inet.(tcp|udp).blackhole`
When set to 1, packets sent to non-listening ports are dropped on the floor. The typical response is for the host to reply with a RST packet thereby letting the client know there is no service available.
- `Net.inet.(tcp|udp).log_in_vain`
When set to 1, packets sent to non-listening ports are logged to syslog. This helps alert us to some potential problems.
- `Net.inet.ip.check_interface`
When set to 1, verifies that an incoming packet arrives on the interface that has an address matching the packet's destination address.
- `Net.inet.tcp.strict_rfc1948`
When set to 1, initial sequence numbering (ISN) will be randomized. This will provide a defense against sequence number attacks.
- `Net.inet.ip.sourceroute`
When set to 1, will enable source routed packets. This will allow packets to create their own routes through a network. This is disabled by default.
- `Kern.coredump`
When set to 1, will allow the creation of a core file after a coredump on non-privileged programs.
- `Kern.sugid.coredump`
When set to 1, will allow the creation of a core file after a coredump on privileged programs.
- `Kern.securelevel`
Five settings which determine the system's level of security.
- `Kern.consmute`
When set to 1, disable messages to the console.

There are indeed many more settings than have an effect on the systems security. The reader is encouraged to browse the entire tree and find those settings that are appropriate for their environment. Sysctl is a key component in the FreeBSD architecture. Many of the other systems and features that we will examine will have very close ties to this mechanism.

Secure Level

FreeBSD provides additional security functionality to the system in the form of securelevels. Securelevels are operating levels that restrict what actions a system can successfully perform. There are currently five levels available to the system ranging from permanently insecure to network secure mode. Any superuser can raise secure levels but they cannot be lowered. The only method to lower the security level is to make the appropriate changes in /etc/rc.conf and reboot.

There are two methods to modify the securelevel of the system. One of them is at boot using the rc.conf (example 1) mechanism. The other is on a running system using the sysctl tool (example 2).

Example 1.

View of /etc/rc.conf

```
kern_securelevel_enable="YES"
kern_securelevel="2"
```

Example 2.

View the current settings.

```
#sysctl kern.securelevel
kern.securelevel: 2
```

Change the securelevel to highly secure mode or level 2

```
#sysctl kern.securelevel=2
```

Let's see what happens when we try to lower the securelevel

```
#sysctl kern.securelevel=1
kern.securelevel: 2
sysctl: kern.securelevel: Operation not permitted
```

The following is a description of the five security levels available and the impact to the system.

From the init(8) man page:

- Permanently Insecure Mode (-1)
- Insecure Mode (0)
Immutable and append-only flags may be turned off. All devices may be read or written subject to their permissions.
- Secure Mode (1)

The system immutable and system append-only flags may not be turned off; disks for mounted filesystems, /dev/mem, and /dev/kmem may not be opened for writing; kernel modules (see kld(4)) may not be loaded or unloaded.

- **Highly Secure Mode (2)**
Same as secure mode, plus disks may not be opened for writing (except by mount(2)) whether mounted or not. This level precludes tampering with filesystems by unmounting them, but also inhibits running newfs(8) while the system is multi-user.
In addition, kernel time changes are restricted to less than or equal to one second. Attempts to change the time by more than this will log the message ``Time adjustment clamped to +1 second".
- **Network Secure Mode (3)**
Same as highly secure mode, plus IP packet filter rules (see ipfw(8) and ipfirewall(4)) cannot be changed and dummynet(4) configuration cannot be adjusted.

Secure level provides another layer of protection to the hosts in an enterprise. Each level provides a higher degree of protection at some expense of usability. The reader is encouraged to evaluate which level is appropriate for their systems.

Jail

In the Unix world there is a never-ending quest to obtain root or superuser access. That is, there are many individuals out there searching for systems which they might compromise thereby obtaining root or superuser access. Why is root important? In the Unix world a user on the system with UID 0 is known as root and with that title comes all power over the system. Root is a privileged user that is able to perform any and all tasks on the system without restriction. Individuals intent on using systems for their own ends constantly exploit this weakness. There is more to the story.

You may be in an environment where individuals have specialized requirements. Take for example a DNS administrator, many times he needs root privileges to start and stop services, make changes and perform other types of maintenance. In a typical Unix environment you have two choices, give the user root access or use some form of sudo program which gives the user temporary root privileges to perform a specific function. While the later is more desirable than the former there is one thing that has been overlooked, the processes themselves. Typically, attackers will target often used, well known processes as their entry point into a system. If successful, the attacker uses these privileged process to gain root access. Is there an answer? What are we to do? The answer is Jail. No, not the attacker, we're going to send the process to jail.

Jail is a method to limit the scope of the super-user account as well as processes than run with varying degrees of privileges. If we take a look at our DNS example again, FreeBSD's Jail would allow us to place the entire DNS system inside a jailed environment and give the DNS administrator privileged access to the jailed environment. The administrator's ability to perform his tasks is limited in scope to what is inside the

jail. He is limited to a specific file namespace similar to the chroot, limited to bind network resources only to a specific IP address and the ability to interact with other processes is limited to only processes that are within the jailed environment. Furthermore, he is unable to use raw sockets, divert and sockets. Modification to the running kernel, including loading kernel modules is also prohibited. Many system calls have also been prohibited within the jail to avoid the jail communicating with the hosting system.

Jail doesn't take everything away. You are still able to perform most file operations, signal processes within the jailed environment and bind to reserved ports on the jail's IP address.

Setting up a jailed environment involves creating a miniature installation within a directory on the host system, setting up the appropriate IP aliases and running the jail command. At this point the jail system is functioning, however OS configuration is still required. It is recommended that the sysinstall be used to assist in this process.

Although it does require some planning, thought and proper implementation, the Jail facility provides a great deal of protection and task delegation. Most applications require no modification to run within a Jailed environment, making this an attractive option for enhancing the security of your FreeBSD systems within your organization.

Security Profiles

Having a unified and standard way of setting a system's overall security policy after an installation has many advantages. With it we have a consistent and known method of applying security settings to a system. When all parties adopt this system they can be confident of the changes and implementation of security settings. There will be no more questions regarding which sysctl variables have been modified or what options have been added or modified in rc.conf. FreeBSD has provided us with a valuable feature known as Security Profiles.

Security Profiles provide a standard method for defining and implementing your system's security policy. It allows us to define our policy by selecting one of the two security templates. Using these templates we can choose a standard setting that leaves most services accessible to the outside world or we can choose an extreme setting that shuts down most services and communication with the outside world. Through the familiar sysinstall tool we are able to apply the template of our choice. We will take a closer look at each of the settings and what they do for our systems.

Below is a brief synopsis of the two security templates. In each section you will see a bulleted list which contains some rather strange looking text. This text is the actual source taken from the sysinstall tool. We thought it appropriate to go straight to the source for this information, literally.

The medium template disables the securelevel feature of FreeBSD and enables the sendmail and ssh daemon.

- `variable_set2("nfs_reserved_port_only", "YES", 1);`
- `variable_set2("sendmail_enable", "YES", 1);`
- `variable_set2("sshd_enable", "YES", 1);`
- `variable_set2("kern_securelevel_enable", "NO", 1);`

The extreme templates shuts down most services including ssh, sendmail and nfs. The securelevel feature is enabled and set to highly secure mode.

- `variable_set2("nfs_server_enable", "NO", 1);`
- `variable_set2("sendmail_enable", "NO", 1);`
- `variable_set2("sshd_enable", "NO", 1);`
- `variable_set2("kern_securelevel_enable", "YES", 1);`
- `variable_set2("kern_securelevel", "2", 1);`

These two templates provide the protection necessary in the two most common environments. Of course even these templates are not enough for a total solution to securing your system, but they do provide a convenient mechanism for consistently applying a security profile to your system.

Firewall

A firewall is a valuable part of any host-based security solution. FreeBSD has provided us with two firewalls solutions to choose from. IPFW and IPFilter, both capable stateful packet filters are fully supported out of the box. The reader is encouraged to see the index for more information about properly implementing each of these solutions.

Misc

FreeBSD shares many other security related features found on most Unix systems and the reader is encouraged to investigate each further. Because they are not strictly FreeBSD features will outline them here as a reference.

- Syslog
- Tcpwrappers
- SSH
- Quotas

Conclusion

We hope this introduction to the security features of FreeBSD has better equipped the reader to make educated decisions about how and what to implement in each system

deployment. The reader is encouraged to browse the references section for more information on any of the topics of interest.

References

The FreeBSD Documentation Project, “FreeBSD Handbook”

URL: <http://www.freebsd.org/handbook>

Darren Reed, “IP Filter”

URL: <http://coombs.anu.edu.au/~avalon/ip-filter.html>

Hoang Q. Tran, “FreeBSD Firewall Using IP Filter”

URL: <http://www.muine.org/~hoang/freenat.html>

Zhihui Zhang, “FreeBSD 4.0 Sysctl Mechanism”

URL: <http://www.daemonnews.org/200104/sysctl.html>

Jason Taylor, “Sysctl – What is it and what can I do with it?”

URL: <http://www.bsdtoday.com/2001/November/Features591.html>

Brendan Conoboy and Eric Fichtner, “IP Filter Based Firewalls HOWTO”

URL: <http://www.obfuscation.org/ipf/ipf-howto.txt>

Poul-Henning Kamp and Robert N. M. Watson, “Jails: Confining the Omnipotent Root”

URL: <http://docs.freebsd.org/44doc/papers/jail/jail.html>

Evan Sarmiento, “Inside Jail”

URL: <http://www.daemonnews.org/200109/jailint.html>

© SANS Institute 2000 - 2002
Author retains full rights.