Chapter 16. Security

Table of Contents

16.1. Synopsis

Hundreds of standard practices have been authored about how to secure systems and networks, and as a user of FreeBSD, understanding how to protect against attacks and intruders is a must.

In this chapter, several fundamentals and techniques will be discussed. The FreeBSD system comes with multiple layers of security, and many more third party utilities may be added to enhance security.

This chapter covers:

- Basic FreeBSD system security concepts.
- The various crypt mechanisms available in FreeBSD.
- How to configure TCP Wrappers for use with inetd(8).
- How to set up Kerberos on FreeBSD.
- How to configure and use OpenSSH on FreeBSD.
- How to use OpenSSL on FreeBSD.
- How to use file system ACLs.
- How to use pkg to audit third party software packages installed from the Ports Collection.
- How to utilize FreeBSD security advisories.
- What Process Accounting is and how to enable it on FreeBSD.
- How to control user resources using login classes or the resource limits database.
- What is Capsicum and a basic example.

Certain topics due to their complexity are found in dedicated chapters such as Firewalls, Mandatory Access Control and articles like VPN over IPsec.

16.2. Introduction

Security is everyone's responsibility. A weak entry point in any system could allow intruders to gain access to critical information and cause havoc on an entire network. One of the core principles of information security is the CIA triad, which stands for the Confidentiality, Integrity, and Availability of information systems.

The CIA triad is a bedrock concept of computer security as customers and users expect their data to be protected. For example, a customer expects that their credit card information is securely stored (confidentiality), that their orders are not changed behind

the scenes (integrity), and that they have access to their order information at all times (availability).

To provide CIA, security professionals apply a defense in depth strategy. The idea of defense in depth is to add several layers of security to prevent one single layer failing and the entire security system collapsing. For example, a system administrator cannot simply turn on a firewall and consider the network or system secure. One must also audit accounts, check the integrity of binaries, and ensure malicious tools are not installed. To implement an effective security strategy, one must understand threats and how to defend against them.

What is a threat as it pertains to computer security? Threats are not limited to remote attackers who attempt to access a system without permission from a remote location. Threats also include employees, malicious software, unauthorized network devices, natural disasters, security vulnerabilities, and even competing corporations.

Systems and networks can be accessed without permission, sometimes by accident, or by remote attackers, and in some cases, via corporate espionage or former employees. As a user, it is important to prepare for and admit when a mistake has led to a security breach and report possible issues to the security team. As an administrator, it is important to know of the threats and be prepared to mitigate them.

When applying security to systems, it is recommended to start by securing the basic accounts and system configuration, and then to secure the network layer so that it adheres to the system policy and the organization's security procedures. Many organizations already have a security policy that covers the configuration of technology devices. The policy should include the security configuration of workstations, desktops, mobile devices, phones, production servers, and development servers. In many cases, standard operating procedures (SOPs) already exist. When in doubt, ask the security team.

16.3. Securing Accounts

Maintaining secure accounts in FreeBSD is crucial for data confidentiality, system integrity, and privilege separation, as it prevents unauthorized access, malware, and data breaches while ensuring compliance and protecting an organization's reputation.

16.3.1. Preventing Logins

In securing a system, a good starting point is an audit of accounts. Disable any accounts

that do not need login access.

> **Tip**
>
> Ensure that `root` has a strong password and that this password is not shared.

To deny login access to accounts, two methods exist.

The first is to lock the account, this example shows how to lock the `imani` account:

```
# pw lock imani
```

The second method is to prevent login access by changing the shell to **/usr/sbin/nologin**. The nologin(8) shell prevents the system from assigning a shell to the user when they attempt to login.

Only the superuser can change the shell for other users:

```
# chsh -s /usr/sbin/nologin imani
```

16.3.2. Password Hashes

Passwords are a necessary evil of technology. When they must be used, they should be complex and a powerful hash mechanism should be used to encrypt the version that is stored in the password database. FreeBSD supports several algorithms, including SHA256, SHA512 and Blowfish hash algorithms in its `crypt()` library, see crypt(3) for details.

The default of SHA512 should not be changed to a less secure hashing algorithm, but can be changed to the more secure Blowfish algorithm.

> **Note**
>
> Blowfish is not part of AES and is not considered compliant with any Federal Information Processing Standards (FIPS). Its use may not be permitted in some environments.

To determine which hash algorithm is used to encrypt a user's password, the superuser can view the hash for the user in the FreeBSD password database. Each hash starts with a symbol which indicates the type of hash mechanism used to encrypt the password.

If DES is used, there is no beginning symbol. For MD5, the symbol is `$`. For SHA256 and SHA512, the symbol is `$6$`. For Blowfish, the symbol is `$2a$`. In this example, the password for `imani` is hashed using the default SHA512 algorithm as the hash starts with `$6$`. Note that the encrypted hash, not the password itself, is stored in the password database:

```
# grep imani /etc/master.passwd
```

The output should be similar to the following:

```
imani:$6$pzIjSvCAn.PBYQBA$PXpSeWPx3g5kscj3IMiM7tUEUSPmGexxta.8Lt9TGS
```

The hash mechanism is set in the user's login class.

The following command can be run to check which hash mechanism is currently being used:

```
% grep passwd_format /etc/login.conf
```

The output should be similar to the following:

```
:passwd_format=sha512:\
```

For example, to change the algorithm to Blowfish, modify that line to look like this:

```
:passwd_format=blf:\
```

Then, cap_mkdb(1) must be executed to upgrade the login.conf database:

```
# cap_mkdb /etc/login.conf
```

Note that this change will not affect any existing password hashes. This means that all passwords should be re-hashed by asking users to run `passwd` in order to change their password.

16.3.3. Password Policy Enforcement

Enforcing a strong password policy for local accounts is a fundamental aspect of system security. In FreeBSD, password length, password strength, and password complexity can be implemented using built-in Pluggable Authentication Modules (PAM).

This section demonstrates how to configure the minimum and maximum password length and the enforcement of mixed characters using the pam_passwdqc(8) module. This module is enforced when a user changes their password.

To configure this module, become the superuser and uncomment the line containing `pam_passwdqc.so` in **/etc/pam.d/passwd**.

Then, edit that line to match the password policy:

```
password        requisite       pam_passwdqc.so         min=disabled
```

The explanation of the parameters can be found in pam_passwdqc(8).

Once this file is saved, a user changing their password will see a message similar to the following:

```
% passwd
```

The output should be similar to the following:

```
Changing local password for user
Old Password:

You can now choose the new password.
A valid password should be a mix of upper and lower case letters,
digits and other characters.  You can use a 12 character long
password with characters from at least 3 of these 4 classes, or
a 10 character long password containing characters from all the
classes.  Characters that form a common pattern are discarded by
the check.
Alternatively, if no one else can see your terminal now, you can
pick this as your password: "trait-useful&knob".
Enter new password:
```

If a password that does not match the policy is entered, it will be rejected with a warning and the user will have an opportunity to try again, up to the configured number of retries.

If your organization's policy requires passwords to expire, FreeBSD supports the `passwordtime` in the user's login class in **/etc/login.conf**

The `default` login class contains an example:

```
#       :passwordtime=90d:\
```

So, to set an expiry of 90 days for this login class, remove the comment symbol (#), save the edit, and execute the following command:

```
# cap_mkdb /etc/login.conf
```

To set the expiration on individual users, pass an expiration date or the number of days to expiry and a username to `pw`:

```
# pw usermod -p 30-apr-2025 -n user
```

As seen here, an expiration date is set in the form of day, month, and year. For more information, see pw(8).

16.3.4. Shared Administration with sudo

System administrators often need the ability to grant enhanced permissions to users so they may perform privileged tasks. The idea that team members are provided access to a FreeBSD system to perform their specific tasks opens up unique challenges to every administrator. These team members only need a subset of access beyond normal end user levels; however, they almost always tell management they are unable to perform their tasks without superuser access. Thankfully, there is no reason to provide such access to end users because tools exist to manage this exact requirement.

> **Tip**
>
> Even administrators should limit their privileges when not needed.

Up to this point, the security chapter has covered permitting access to authorized users and attempting to prevent unauthorized access. Another problem arises once authorized users have access to the system resources. In many cases, some users may need access to application startup scripts, or a team of administrators need to maintain the system. Traditionally, the standard users and groups, file permissions, and even the su(1) command would manage this access. And as applications required more access, as more users needed to use system resources, a better solution was required. The most used application is currently Sudo.

Sudo allows administrators to configure more rigid access to system commands and provide for some advanced logging features. As a tool, it is available from the Ports Collection as security/sudo or by use of the pkg(8) utility.

Execute the following command to install it:

```
# pkg install sudo
```

After the installation is complete, the installed `visudo` will open the configuration file with a text editor. Using `visudo` is highly recommended as it comes with a built in syntax checker to verify there are no errors before the file is saved.

The configuration file is made up of several small sections which allow for extensive configuration. In the following example, web application maintainer, user1, needs to start,

stop, and restart the web application known as *webservice*. To grant this user permission to perform these tasks, add this line to the end of **/usr/local/etc/sudoers**:

```
user1   ALL=(ALL)       /usr/sbin/service webservice *
```

The user may now start *webservice* using this command:

```
% sudo /usr/sbin/service webservice start
```

While this configuration allows a single user access to the webservice service; however, in most organizations, there is an entire web team in charge of managing the service. A single line can also give access to an entire group. These steps will create a web group, add a user to this group, and allow all members of the group to manage the service:

```
# pw groupadd -g 6001 -n webteam
```

Using the same pw(8) command, the user is added to the webteam group:

```
# pw groupmod -m user1 -n webteam
```

Finally, this line in **/usr/local/etc/sudoers** allows any member of the webteam group to manage *webservice*:

```
%webteam   ALL=(ALL)       /usr/sbin/service webservice *
```

Unlike su(1), sudo(8) only requires the end user password. This avoids sharing passwords, which is a poor practice.

Users permitted to run applications with sudo(8) only enter their own passwords. This is more secure and gives better control than su(1), where the `root` password is entered and the user acquires all `root` permissions.

> **Tip**
>
> Most organizations are moving or have moved toward a two factor authentication

model. In these cases, the user may not have a password to enter.

sudo(8) can be configured to permit two factor authentication model by using the `NOPASSWD` variable. Adding it to the configuration above will allow all members of the *webteam* group to manage the service without the password requirement:

```
%webteam   ALL=(ALL)       NOPASSWD: /usr/sbin/service webservic
```

16.3.5. Shared Administration with Doas

doas(1) is a command-line utility ported from OpenBSD. It serves as an alternative to the widely used sudo(8) command in Unix-like systems.

With doas, users can execute commands with elevated privileges, typically as the root user, while maintaining a simplified and security-conscious approach. Unlike sudo(8), doas emphasizes simplicity and minimalism, focusing on streamlined privilege delegation without an overwhelming array of configuration options.

Execute the following command to install it:

```
# pkg install doas
```

After the installation **/usr/local/etc/doas.conf** must be configured to grant access for users for specific commands, or roles.

The simplest entry could be the following, which grants the user `local_user` with `root` permissions without asking for its password when executing the doas command.

```
permit nopass local_user as root
```

After the installation and configuration of the `doas` utility, a command can now be executed with enhanced privileges, for example:

```
$ doas vi /etc/rc.conf
```

For more configuration examples, please read doas.conf(5).

16.4. Intrusion Detection System (IDS)

Verification of system files and binaries is important because it provides the system administration and security teams information about system changes. A software application that monitors the system for changes is called an Intrusion Detection System (IDS).

FreeBSD provides native support for a basic IDS system called mtree(8). While the nightly security emails will notify an administrator of changes, the information is stored locally and there is a chance that a malicious user could modify this information in order to hide their changes to the system. As such, it is recommended to create a separate set of binary signatures and store them on a read-only, root-owned directory or, preferably, on a removable USB disk or remote server.

It is also recommended to run `freebsd-update IDS` after each update.

16.4.1. Generating the Specification File

The built-in mtree(8) utility can be used to generate a specification of the contents of a directory. A seed, or a numeric constant, is used to generate the specification and is required to check that the specification has not changed. This makes it possible to determine if a file or binary has been modified. Since the seed value is unknown by an attacker, faking or checking the checksum values of files will be difficult to impossible.

> **Tip**
>
> It is recommended to create specifications for the directories which contain binaries and configuration files, as well as any directories containing sensitive data. Typically, specifications are created for **/bin**, **/sbin**, **/usr/bin**, **/usr/sbin**, **/usr/local/bin**, **/etc**, and **/usr/local/etc**.

The following example generates a set of `sha512` hashes, one for each system binary in **/bin**, and saves those values to a hidden file in user's home directory, **/home/**

user/.bin_chksum_mtree:

```
# mtree -s 123456789 -c -K cksum,sha512 -p /bin > /home/user/.bin_chksum_mtree
```

The output should be similar to the following:

```
mtree: /bin checksum: 3427012225
```

> **Warning**
>
> The `123456789` value represents the seed, and should be chosen randomly. This value should be remembered, **but not shared**.
>
> It is important to keep the seed value and the checksum output hidden from malicious users.

16.4.2. The Specification File Structure

The mtree format is a textual format that describes a collection of filesystem objects. Such files are typically used to create or verify directory hierarchies.

An mtree file consists of a series of lines, each providing information about a single filesystem object. Leading whitespace is always ignored.

The specification file created above will be used to explain the format and content:

```
#          user: root  1
#       machine: machinename  2
#          tree: /bin  3
#          date: Thu Aug  24 21:58:37 2023  4

# .
/set type=file uid=0 gid=0 mode=0555 nlink=1 flags=uarch  5
.               type=dir mode=0755 nlink=2 time=1681388848.239523000
    \133        nlink=2 size=12520 time=1685991378.688509000 \
                cksum=520880818 \
```

```
                sha512=5c1374ce0e2ba1b3bc5a41b23f4bbdc1ec89ae82fa012
    cat         size=14600 time=1685991378.694601000 cksum=367253184
                sha512=b30b96d155fdc4795432b523989a6581d71cdf69ba5f0
    chflags     size=8920 time=1685991378.700385000 cksum=1629328991
                sha512=289a088cbbcbeb436dd9c1f74521a89b66643976abda6
    chio        size=20720 time=1685991378.706095000 cksum=194875160
                sha512=46f58277ff16c3495ea51e74129c73617f31351e25031
    chmod       size=9616 time=1685991378.712546000 cksum=4244658911
                sha512=1769313ce08cba84ecdc2b9c07ef86d2b70a4206420dd
```

1. User who created the specification.
2. Machine's hostname.
3. Directory path.
4. The Date and time when the specification was created.
5. `/set` special commands, defines some settings obtained from the files analyzed.
6. Refers to the parsed directory and indicates things like what type it is, its mode, the number of hard links, and the time in UNIX format since it was modified.
7. Refers to the file and shows the size, time and a list of hashes to verify the integrity.

16.4.3. Verify the Specification file

To verify that the binary signatures have not changed, compare the current contents of the directory to the previously generated specification, and save the results to a file.

This command requires the seed that was used to generate the original specification:

```
# mtree -s 123456789 -p /bin < /home/user/.bin_chksum_mtree >> /home/user/.bin_chksum_out
```

This should produce the same checksum for **/bin** that was produced when the specification was created. If no changes have occurred to the binaries in this directory, the **/home/user/.bin_chksum_output** output file will be empty.

To simulate a change, change the date on **/bin/cat** using touch(1) and run the verification command again:

```
# touch /bin/cat
```

Run the verification command again:

```
# mtree -s 123456789 -p /bin < /home/user/.bin_chksum_mtree >> /home/user/.bin_chksum_out
```

And then check the content of the output file:

```
# cat /root/.bin_chksum_output
```

The output should be similar to the following:

```
cat:    modification time (Fri Aug 25 13:30:17 2023, Fri Aug 25 13:3
```

> **Warning**
> This is just an example of what would be displayed when executing the command, to show the changes that would occur in the metadata.

16.5. Secure levels

securelevel is a security mechanism implemented in the kernel. When the securelevel is positive, the kernel restricts certain tasks; not even the superuser (root) is allowed to do them.

The securelevel mechanism limits the ability to:

- Unset certain file flags, such as `schg` (the system immutable flag).
- Write to kernel memory via **/dev/mem** and **/dev/kmem**.
- Load kernel modules.
- Alter firewall rules.

16.5.1. Secure Levels Definitions

The kernel runs with five different security levels. Any super-user process can raise the level, but no process can lower it.

The security definitions are:

-1

Permanently insecure mode - always run the system in insecure mode. This is the default initial value.

0

Insecure mode - immutable and append-only flags may be turned off. All devices may be read or written subject to their permissions.

1

Secure mode - the system immutable and system append-only flags may not be turned off; disks for mounted file systems, **/dev/mem** and **/dev/kmem** may not be opened for writing; **/dev/io** (if your platform has it) may not be opened at all; kernel modules (see kld(4)) may not be loaded or unloaded. The kernel debugger may not be entered using the debug.kdb.enter sysctl. A panic or trap cannot be forced using the debug.kdb.panic, debug.kdb.panic_str and other sysctl's.

2

Highly secure mode - 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 file systems by unmounting them, but also inhibits running newfs(8) while the system is multiuser.

3

Network secure mode - same as highly secure mode, plus IP packet filter rules (see ipfw(8), ipfirewall(4) and pfctl(8)) cannot be changed and dummynet(4) or pf(4) configuration cannot be adjusted.

> **Tip**
>
> In summary, the key difference between `Permanently Insecure Mode` and `Insecure Mode` in FreeBSD secure levels is the degree of security they provide.

> `Permanently Insecure Mode` completely lifts all security restrictions, while `Insecure Mode` relaxes some restrictions but still maintains a level of control and security.

16.5.2. Modify Secure Levels

In order to change the securelevel of the system it is necessary to activate `kern_securelevel_enable` by executing the following command:

```
# sysrc kern_securelevel_enable="YES"
```

And set the value of `kern_securelevel` to the desired security level:

```
# sysrc kern_securelevel=2
```

To check the status of the securelevel on a running system execute the following command:

```
# sysctl -n kern.securelevel
```

The output contains the current value of the securelevel. If it is greater than 0, at least some of the securelevel's protections are enabled.

16.6. File flags

File flags allow users to attach additional metadata or attributes to files and directories beyond basic permissions and ownership. These flags provide a way to control various behaviors and properties of files without needing to resort to creating special directories or using extended attributes.

File flags can be used to achieve different goals, such as preventing file deletion, making files append-only, synchronizing file updates, and more. Some commonly used file flags in FreeBSD include the "immutable" flag, which prevents modification or deletion of a file, and the "append-only" flag, which allows only data to be added to the end of a file

but not modified or removed.

These flags can be managed using the chflags(1) command in FreeBSD, providing administrators and users with greater control over the behavior and characteristics of their files and directories. It is important to note that file flags are typically managed by root or users with appropriate privileges, as they can influence how files are accessed and manipulated. Some flags are available for the use of the file's owner, as described in chflags(1).

16.6.1. Work with File Flags

In this example, a file named **~/important.txt** in user's home directory want to be protected against deletions.

Execute the following command to set the `schg` file flag:

```
# chflags schg ~/important.txt
```

When any user, including the `root` user, tries to delete the file, the system will display the message:

```
rm: important.txt: Operation not permitted
```

To delete the file, it will be necessary to delete the file flags of that file by executing the following command:

```
# chflags noschg ~/important.txt
```

A list of supported file flags and their functionality can be found in chflags(1).

16.7. OpenSSH

OpenSSH is a set of network connectivity tools used to provide secure access to remote machines. Additionally, TCP/IP connections can be tunneled or forwarded securely through SSH connections. OpenSSH encrypts all traffic to eliminate eavesdropping, connection hijacking, and other network-level attacks.

OpenSSH is maintained by the OpenBSD project and is installed by default in FreeBSD.

When data is sent over the network in an unencrypted form, network sniffers anywhere in between the client and server can steal user/password information or data transferred during the session. OpenSSH offers a variety of authentication and encryption methods to prevent this from happening.

More information about OpenSSH is available in the web page.

This section provides an overview of the built-in client utilities to securely access other systems and securely transfer files from a FreeBSD system. It then describes how to configure a SSH server on a FreeBSD system.

> **Tip**
>
> As stated, this chapter will cover the base system version of OpenSSH. A version of OpenSSH is also available in the security/openssh-portable, which provides additional configuration options and is updated with new features more regularly.

16.7.1. Using the SSH Client Utilities

To log into a SSH server, use ssh(1) and specify a username that exists on that server and the IP address or hostname of the server. If this is the first time a connection has been made to the specified server, the user will be prompted to first verify the server's fingerprint:

```
# ssh user@example.com
```

The output should be similar to the following:

```
The authenticity of host 'example.com (10.0.0.1)' can't be establish
ECDSA key fingerprint is 25:cc:73:b5:b3:96:75:3d:56:19:49:d2:5c:1f:9
Are you sure you want to continue connecting (yes/no)? yes
Permanently added 'example.com' (ECDSA) to the list of known hosts.
Password for user@example.com: user_password
```

SSH utilizes a key fingerprint system to verify the authenticity of the server when the

client connects. When the user accepts the key's fingerprint by typing `yes` when connecting for the first time, a copy of the key is saved to **~/.ssh/known_hosts** in the user's home directory. Future attempts to login are verified against the saved key and ssh(1) will display an alert if the server's key does not match the saved key. If this occurs, the user should first verify why the key has changed before continuing with the connection.

> **Note**
>
> How to perform this check is outside the scope of this chapter.

Use scp(1) to securely copy a file to or from a remote machine.

This example copies `COPYRIGHT` on the remote system to a file of the same name in the current directory of the local system:

```
# scp user@example.com:/COPYRIGHT COPYRIGHT
```

The output should be similar to the following:

```
Password for user@example.com: *******
COPYRIGHT              100% |*****************************|  4735
```

Since the fingerprint was already verified for this host, the server's key is automatically checked before prompting for the user's password.

The arguments passed to scp(1) are similar to cp(1). The file or files to copy is the first argument and the destination to copy to is the second. Since the file is fetched over the network, one or more of the file arguments takes the form `user@host:<path_to_remote_file>`. Be aware when copying directories recursively that scp(1) uses `-r`, whereas cp(1) uses `-R`.

To open an interactive session for copying files, use sftp(1).

Refer to sftp(1) for a list of available commands while in an sftp(1) session.

16.7.2. Key-based Authentication

Instead of using passwords, a client can be configured to connect to the remote machine using keys. For security reasons, this is the preferred method.

ssh-keygen(1) can be used to generate the authentication keys. To generate a public and private key pair, specify the type of key and follow the prompts. It is recommended to protect the keys with a memorable, but hard to guess passphrase.

```
% ssh-keygen -t rsa -b 4096
```

The output should be similar to the following:

```
Generating public/private rsa key pair.
Enter file in which to save the key (/home/user/.ssh/id_rsa):
Created directory '/home/user/.ssh/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/user/.ssh/id_rsa.
Your public key has been saved in /home/user/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:54Xm9Uvtv6H4NOo6yjP/YCfODryvUU7yWHzMqeXwhq8 user@host.example
The key's randomart image is:
+---[RSA 2048]----+
|                 |
|                 |
|                 |
|        . o..    |
|       .S*+*o    |
|      . O=Oo . . |
|       = Oo= oo..|
|      .oB.* +.oo.|
|       =OE**.o..=|
+----[SHA256]-----+
```

The private key is stored in **~/.ssh/id_rsa** and the public key is stored in **~/.ssh/id_rsa.pub**. The *public* key must be copied to **~/.ssh/authorized_keys** on the remote machine for key-based authentication to work.

> **Warning**
>
> Utilizing a passphrase for OpenSSH keys is a key security practice, providing an

> extra layer of protection against unauthorized access and enhancing overall cybersecurity.
>
> In case of loss or theft, this adds another layer of security.

16.7.3. SSH Tunneling

OpenSSH has the ability to create a tunnel to encapsulate another protocol in an encrypted session.

The following command tells ssh(1) to create a tunnel:

```
% ssh -D 8080 user@example.com
```

This example uses the following options:

-D

Specifies a local "dynamic" application-level port forwarding.

user@foo.example.com

The login name to use on the specified remote SSH server.

An SSH tunnel works by creating a listen socket on `localhost` on the specified `localport`.

This method can be used to wrap any number of insecure TCP protocols such as SMTP, POP3, and FTP.

16.7.4. Enabling the SSH Server

In addition to providing built-in SSH client utilities, a FreeBSD system can be configured as an SSH server, accepting connections from other SSH clients.

> **Tip**
>
> As stated, this chapter will cover the base system version of OpenSSH. Please **not**

> confuse with security/openssh-portable, the version of OpenSSH that ships with the FreeBSD ports.

In order to have the SSH Server enabled across reboots execute the following command:

```
# sysrc sshd_enable="YES"
```

Then execute the following command to enable the service:

```
# service sshd start
```

The first time sshd starts on a FreeBSD system, the system's host keys will be automatically created and the fingerprint will be displayed on the console. Provide users with the fingerprint so that they can verify it the first time they connect to the server.

Refer to sshd(8) for the list of available options when starting sshd and a complete discussion about authentication, the login process, and the various configuration files.

At this point, the sshd should be available to all users with a username and password on the system.

16.7.5. Configuring publickey auth method

Configuring OpenSSH to use public key authentication enhances security by leveraging asymmetric cryptography for authentication. This method eliminates password-related risks, such as weak passwords or interception during transmission, while thwarting various password-based attacks. However, it's vital to ensure the private keys are well-protected to prevent unauthorized access.

The first step will be to configure sshd(8) to use the required authentication method.

Edit **/etc/ssh/sshd_config** and uncomment the following configuration:

```
PubkeyAuthentication yes
```

Once the configuration is done, the users will have to send the system administrator

their **public key** and these keys will be added in **.ssh/authorized_keys**. The process for generating the keys is described in Key-based Authentication.

Then restart the server executing the following command:

```
# service sshd reload
```

It is strongly recommended to follow the security improvements indicated in SSH Server Security Options.

16.7.6. SSH Server Security Options

While sshd is the most widely used remote administration facility for FreeBSD, brute force and drive by attacks are common to any system exposed to public networks.

Several additional parameters are available to prevent the success of these attacks and will be described in this section. All configurations will be done in **/etc/ssh/sshd_config**

> **Tip**
>
> Do not confuse **/etc/ssh/sshd_config** with **/etc/ssh/ssh_config** (note the extra d in the first filename). The first file configures the server and the second file configures the client. Refer to ssh_config(5) for a listing of the available client settings.

By default, authentication can be done with both pubkey and password. To allow **only** pubkey authentication, **which is strongly recommended**, change the variable:

```
PasswordAuthentication no
```

It is a good idea to limit which users can log into the SSH server and from where using the `AllowUsers` keyword in the OpenSSH server configuration file. For example, to only allow `user` to log in from `192.168.1.32`, add this line to **/etc/ssh/sshd_config**:

```
AllowUsers user@192.168.1.32
```

To allow `user` to log in from anywhere, list that user without specifying an IP address:

```
AllowUsers user
```

Multiple users should be listed on the same line, like so:

```
AllowUsers root@192.168.1.32 user
```

After making all the changes, and before restarting the service, it is recommended to verify that the configuration made is correct by executing the following command:

```
# sshd -t
```

If the configuration file is correct, no output will be shown. In case the configuration file is incorrect, it will show something like this:

```
/etc/ssh/sshd_config: line 3: Bad configuration option: sdadasdasdas
/etc/ssh/sshd_config: terminating, 1 bad configuration options
```

After making the changes and checking that the configuration file is correct, tell sshd to reload its configuration file by running:

```
# service sshd reload
```

16.8. OpenSSL

OpenSSL is a cryptography toolkit implementing the Secure Sockets Layer (SSL) and Transport Layer Security (TLS) network protocols and many cryptography routines.

The openssl program is a command line tool for using the various cryptography functions of OpenSSL's crypto library from the shell. It can be used for

- Creation and management of private keys, public keys and parameters
- Public key cryptographic operations

- Creation of X.509 certificates, CSRs and CRLs
- Calculation of Message Digests
- Encryption and Decryption with Ciphers
- SSL/TLS Client and Server Tests
- Handling of S/MIME signed or encrypted mail
- Time Stamp requests, generation and verification
- Benchmarking the crypto routines

For more information about OpenSSL, read the free OpenSSL Cookbook.

16.8.1. Generating Certificates

OpenSSL supports the generation of certificates both to be validated by a CA and for own use.

Run the command openssl(1) to generate a valid certificate for a CA with the following arguments. This command will create two files in the current directory. The certificate request, **req.pem**, can be sent to a CA which, will validate the entered credentials, sign the request, and return the signed certificate. The second file, **cert.key**, is the private key for the certificate and should be stored in a secure location. If this falls in the hands of others, it can be used to impersonate the user or the server.

Execute the following command to generate the certificate:

```
# openssl req -new -nodes -out req.pem -keyout cert.key -sha3-512 -newkey rsa:4096
```

The output should be similar to the following:

```
Generating a RSA private key
.....................................................................
.....................................+++++
writing new private key to 'cert.key'
-----
You are about to be asked to enter information that will be incor
into your certificate request.
```

```
What you are about to enter is what is called a Distinguished Name o
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:ES
State or Province Name (full name) [Some-State]:Valencian Community
Locality Name (eg, city) []:Valencia
Organization Name (eg, company) [Internet Widgits Pty Ltd]:My Compan
Organizational Unit Name (eg, section) []:Systems Administrator
Common Name (e.g. server FQDN or YOUR name) []:localhost.example.org
Email Address []:user@FreeBSD.org

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:123456789
An optional company name []:Another name
```

Alternately, if a signature from a CA is not required, a self-signed certificate can be created. This will create two new files in the current directory: a private key file **cert.key**, and the certificate itself, **cert.crt**. These should be placed in a directory, preferably under **/etc/ssl/**, which is readable only by `root`. Permissions of `0700` are appropriate for these files and can be set using `chmod`.

Execute the following command to generate the certificate:

```
# openssl req -new -x509 -days 365 -sha3-512 -keyout /etc/ssl/private/cert.key -out /etc/
```

The output should be similar to the following:

```
Generating a RSA private key
.....................................+++++
...........+++++
writing new private key to '/etc/ssl/private/cert.key'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be incorpor
into your certificate request.
What you are about to enter is what is called a Distinguished Name o
```

```
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:ES
State or Province Name (full name) [Some-State]:Valencian Community
Locality Name (eg, city) []:Valencia
Organization Name (eg, company) [Internet Widgits Pty Ltd]:My Compan
Organizational Unit Name (eg, section) []:Systems Administrator
Common Name (e.g. server FQDN or YOUR name) []:localhost.example.org
Email Address []:user@FreeBSD.org
```

16.8.2. Configuring the FIPS Provider

With the import of OpenSSL 3 into the base system (on FreeBSD 14 and later), its new concept of provider modules was introduced in the system. Besides the default provider module built-in to the library, the *legacy* module implements the now optional deprecated cryptography algorithms, while the *fips* module restricts the OpenSSL implementation to the cryptography algorithms present in the FIPS set of standards. This part of OpenSSL receives particular care, including a list of relevant security issues, and is subject to the FIPS 140 validation process on a regular basis. The list of FIPS validated versions is also available. This allows users to ensure FIPS compliance in their use of OpenSSL.

Importantly, the fips_module(7) is protected by an additional security measure, preventing its use without passing an integrity check. This check can be setup by the local system administrator, allowing every user of OpenSSL 3 to load this module. When not configured correctly, the FIPS module is expected to fail as follows:

```
# echo test | openssl aes-128-cbc -a -provider fips -pbkdf2
```

The output should be similar to the following:

```
aes-128-cbc: unable to load provider fips
Hint: use -provider-path option or OPENSSL_MODULES environment varia
00206124D94D0000:error:1C8000D5:Provider routines:SELF_TEST_post:mis
00206124D94D0000:error:1C8000E0:Provider routines:ossl_set_error_st
00206124D94D0000:error:1C8000D8:Provider routines:OSSL_provider_init
```

```
00206124D94D0000:error:078C0105:common libcrypto routines:provider_i
```

The check can be configured through the creation of a file in **/etc/ssl/fipsmodule.cnf**, which will then be referenced in OpenSSL's main configuration file **/etc/ssl/openssl.cnf**. OpenSSL provides the openssl-fipsinstall(1) utility to help with this process, which can be used as follows:

```
# openssl fipsinstall -module /usr/lib/ossl-modules/fips.so -out /etc/ssl/fipsmodule.cnf
```

The output should be similar to the following:

```
INSTALL PASSED
```

The **/etc/ssl/openssl.cnf** should then be modified, in order to:

- Include the **/etc/ssl/fipsmodule.cnf** file generated above,
- Expose the FIPS module for possible use,
- And explicitly activate the default module.

```
[...]
# For FIPS
# Optionally include a file that is generated by the OpenSSL fipsins
# application. This file contains configuration data required by the
# fips provider. It contains a named section e.g. [fips_sect] which
# referenced from the [provider_sect] below.
# Refer to the OpenSSL security policy for more information.
.include /etc/ssl/fipsmodule.cnf

[...]

# List of providers to load
[provider_sect]
default = default_sect
# The fips section name should match the section name inside the
# included fipsmodule.cnf.
fips = fips_sect
```

```
# If no providers are activated explicitly, the default one is activ
# See man 7 OSSL_PROVIDER-default for more details.
#
# If you add a section explicitly activating any other provider(s),
# probably need to explicitly activate the default provider, otherwi
# becomes unavailable in openssl.  As a consequence applications dep
# OpenSSL may not work correctly which could lead to significant sys
# problems including inability to remotely access the system.
[default_sect]
activate = 1
```

With this done, it should be possible to confirm that the FIPS module is effectively available and working:

```
# echo test | openssl aes-128-cbc -a -provider fips -pbkdf2
```

The output should be similar to the following:

```
enter AES-128-CBC encryption password:
Verifying - enter AES-128-CBC encryption password:
U2FsdGVkX18idooW6e3LqWeeiKP76kufcOUClh57j8U=
```

This procedure has to be repeated every time the FIPS module is modified, e.g., after performing system updates, or after applying security fixes affecting OpenSSL in the base system.

16.9. Kerberos

Kerberos is a network authentication protocol which was originally created by the Massachusetts Institute of Technology (MIT) as a way to securely provide authentication across a potentially hostile network. The Kerberos protocol uses strong cryptography so that both a client and server can prove their identity without sending any unencrypted secrets over the network. Kerberos can be described as an identity-verifying proxy system and as a trusted third-party authentication system. After a user authenticates with Kerberos, their communications can be encrypted to assure privacy and data integrity.

The only function of Kerberos is to provide the secure authentication of users and

servers on the network. It does not provide authorization or auditing functions. It is recommended that Kerberos be used with other security methods which provide authorization and audit services.

The current version of the protocol is version 5, described in RFC 4120. Several free implementations of this protocol are available, covering a wide range of operating systems. MIT continues to develop their Kerberos package. It is commonly used in the US as a cryptography product, and has historically been subject to US export regulations. In FreeBSD, MITKerberos is available as the security/krb5 package or port. The Heimdal Kerberos implementation was explicitly developed outside of the US to avoid export regulations. The Heimdal Kerberos distribution is included in the base FreeBSD installation, and another distribution with more configurable options is available as security/heimdal in the Ports Collection.

In Kerberos users and services are identified as "principals" which are contained within an administrative grouping, called a "realm". A typical user principal would be of the form *`user`*`@`*`REALM` (realms are traditionally uppercase).

This section provides a guide on how to set up Kerberos using the Heimdal distribution included in FreeBSD.

For purposes of demonstrating a Kerberos installation, the name spaces will be as follows:

- The DNS domain (zone) will be `example.org`.
- The Kerberos realm will be `EXAMPLE.ORG`.

> **Note**
>
> Use real domain names when setting up Kerberos, even if it will run internally. This avoids DNS problems and assures inter-operation with other Kerberos realms.

16.9.1. Setting up a Heimdal KDC

The Key Distribution Center (KDC) is the centralized authentication service that Kerberos provides, the "trusted third party" of the system. It is the computer that issues Kerberos tickets, which are used for clients to authenticate to servers. As the KDC is considered trusted by all other computers in the Kerberos realm, it has heightened security concerns. Direct access to the KDC should be limited.

While running a KDC requires few computing resources, a dedicated machine acting only as a KDC is recommended for security reasons.

To begin, install the security/heimdal package as follows:

```
# pkg install heimdal
```

Next, update **/etc/rc.conf** using `sysrc` as follows:

```
# sysrc kdc_enable=yes
# sysrc kadmind_enable=yes
```

Next, edit **/etc/krb5.conf** as follows:

```
[libdefaults]
    default_realm = EXAMPLE.ORG
[realms]
    EXAMPLE.ORG = {
        kdc = kerberos.example.org
        admin_server = kerberos.example.org
    }
[domain_realm]
    .example.org = EXAMPLE.ORG
```

In this example, the KDC will use the fully-qualified hostname `kerberos.example.org`. The hostname of the KDC must be resolvable in the DNS.

Kerberos can also use the DNS to locate KDCs, instead of a `[realms]` section in **/etc/krb5.conf**. For large organizations that have their own DNS servers, the above example could be trimmed to:

```
[libdefaults]
      default_realm = EXAMPLE.ORG
[domain_realm]
    .example.org = EXAMPLE.ORG
```

With the following lines being included in the `example.org` zone file:

```
_kerberos._udp      IN  SRV     01 00 88 kerberos.example.org.
_kerberos._tcp      IN  SRV     01 00 88 kerberos.example.org.
_kpasswd._udp       IN  SRV     01 00 464 kerberos.example.org.
_kerberos-adm._tcp  IN  SRV     01 00 749 kerberos.example.org.
_kerberos           IN  TXT     EXAMPLE.ORG
```

> **Note**
>
> In order for clients to be able to find the Kerberos services, they *must* have either a fully configured **/etc/krb5.conf** or a minimally configured **/etc/krb5.conf** *and* a properly configured DNS server.

Next, create the Kerberos database which contains the keys of all principals (users and hosts) encrypted with a master password. It is not required to remember this password as it will be stored in **/var/heimdal/m-key**; it would be reasonable to use a 45-character random password for this purpose. To create the master key, run `kstash` and enter a password:

```
# kstash
```

The output should be similar to the following:

```
Master key: xxxxxxxxxxxxxxxxxxxxxxx
Verifying password - Master key: xxxxxxxxxxxxxxxxxxxxxxx
```

Once the master key has been created, the database should be initialized. The Kerberos administrative tool kadmin(8) can be used on the KDC in a mode that operates directly on the database, without using the kadmind(8) network service, as `kadmin -l`. This resolves the chicken-and-egg problem of trying to connect to the database before it is created. At the `kadmin` prompt, use `init` to create the realm's initial database:

```
# kadmin -l
kadmin> init EXAMPLE.ORG
Realm max ticket life [unlimited]:
```

Lastly, while still in `kadmin`, create the first principal using `add`. Stick to the default options for the principal for now, as these can be changed later with `modify`. Type `?` at the prompt to see the available options.

```
kadmin> add tillman
```

The output should be similar to the following:

```
Max ticket life [unlimited]:
Max renewable life [unlimited]:
Principal expiration time [never]:
Password expiration time [never]:
Attributes []:
Password: xxxxxxxx
Verifying password - Password: xxxxxxxx
```

Next, start the KDC services by running:

```
# service kdc start
# service kadmind start
```

While there will not be any kerberized daemons running at this point, it is possible to confirm that the KDC is functioning by obtaining a ticket for the principal that was just created:

```
% kinit tillman
```

The output should be similar to the following:

```
tillman@EXAMPLE.ORG's Password:
```

Confirm that a ticket was successfully obtained using `klist`:

```
% klist
```

The output should be similar to the following:

```
Credentials cache: FILE:/tmp/krb5cc_1001
        Principal: tillman@EXAMPLE.ORG

  Issued                Expires               Principal
Aug 27 15:37:58 2013  Aug 28 01:37:58 2013  krbtgt/EXAMPLE.ORG@EXAMP
```

The temporary ticket can be destroyed when the test is finished:

```
% kdestroy
```

16.9.2. Configuring a Server to Use Kerberos

The first step in configuring a server to use Kerberos authentication is to ensure that it has the correct configuration in **/etc/krb5.conf**. The version from the KDC can be used as-is, or it can be regenerated on the new system.

Next, create **/etc/krb5.keytab** on the server. This is the main part of "Kerberizing" a service - it corresponds to generating a secret shared between the service and the KDC. The secret is a cryptographic key, stored in a "keytab". The keytab contains the server's host key, which allows it and the KDC to verify each others' identity. It must be transmitted to the server in a secure fashion, as the security of the server can be broken if the key is made public. Typically, the **keytab** is generated on an administrator's trusted machine using `kadmin`, then securely transferred to the server, e.g., with scp(1); it can also be created directly on the server if that is consistent with the desired security policy. It is very important that the keytab is transmitted to the server in a secure fashion: if the key is known by some other party, that party can impersonate any user to the server! Using `kadmin` on the server directly is convenient, because the entry for the host principal in the KDC database is also created using `kadmin`.

Of course, `kadmin` is a kerberized service; a Kerberos ticket is needed to authenticate to the network service, but to ensure that the user running `kadmin` is actually present (and their session has not been hijacked), `kadmin` will prompt for the password to get a

fresh ticket. The principal authenticating to the kadmin service must be permitted to use the `kadmin` interface, as specified in **/var/heimdal/kadmind.acl**. See the section titled "Remote administration" in `info heimdal` for details on designing access control lists. Instead of enabling remote `kadmin` access, the administrator could securely connect to the KDC via the local console or ssh(1), and perform administration locally using `kadmin -l`.

After installing **/etc/krb5.conf**, use `add --random-key` in `kadmin`. This adds the server's host principal to the database, but does not extract a copy of the host principal key to a keytab. To generate the keytab, use `ext` to extract the server's host principal key to its own keytab:

```
# kadmin
```

The output should be similar to the following:

```
kadmin> add --random-key host/myserver.example.org
Max ticket life [unlimited]:
Max renewable life [unlimited]:
Principal expiration time [never]:
Password expiration time [never]:
Attributes []:
kadmin> ext_keytab host/myserver.example.org
kadmin> exit
```

Note that `ext_keytab` stores the extracted key in **/etc/krb5.keytab** by default. This is good when being run on the server being kerberized, but the `--keytab` *`path/to/file`* argument should be used when the keytab is being extracted elsewhere:

```
# kadmin
```

The output should be similar to the following:

```
kadmin> ext_keytab --keytab=/tmp/example.keytab host/myserver.exampl
kadmin> exit
```

The keytab can then be securely copied to the server using scp(1) or a removable media.

Be sure to specify a non-default keytab name to avoid inserting unneeded keys into the system's keytab.

At this point, the server can read encrypted messages from the KDC using its shared key, stored in **krb5.keytab**. It is now ready for the Kerberos-using services to be enabled. One of the most common such services is sshd(8), which supports Kerberos via the GSS-API. In **/etc/ssh/sshd_config**, add the line:

```
GSSAPIAuthentication yes
```

After making this change, sshd(8) must be restarted for the new configuration to take effect: `service sshd restart`.

16.9.3. Configuring a Client to Use Kerberos

As it was for the server, the client requires configuration in **/etc/krb5.conf**. Copy the file in place (securely) or re-enter it as needed.

Test the client by using `kinit`, `klist`, and `kdestroy` from the client to obtain, show, and then delete a ticket for an existing principal. Kerberos applications should also be able to connect to Kerberos enabled servers. If that does not work but obtaining a ticket does, the problem is likely with the server and not with the client or the KDC. In the case of kerberized ssh(1), GSS-API is disabled by default, so test using `ssh -o GSSAPIAuthentication=yes` *`hostname`*.

When testing a Kerberized application, try using a packet sniffer such as `tcpdump` to confirm that no sensitive information is sent in the clear.

Various Kerberos client applications are available. With the advent of a bridge so that applications using SASL for authentication can use GSS-API mechanisms as well, large classes of client applications can use Kerberos for authentication, from Jabber clients to IMAP clients.

Users within a realm typically have their Kerberos principal mapped to a local user account. Occasionally, one needs to grant access to a local user account to someone who does not have a matching Kerberos principal. For example, `tillman@EXAMPLE.ORG` may need access to the local user account `webdevelopers`. Other principals may also need access to that local account.

The **.k5login** and **.k5users** files, placed in a user's home directory, can be used to solve

this problem. For example, if the following **.k5login** is placed in the home directory of `webdevelopers`, both principals listed will have access to that account without requiring a shared password:

```
tillman@example.org
jdoe@example.org
```

Refer to ksu(1) for more information about **.k5users**.

16.9.4. MIT Differences

The major difference between the MIT and Heimdal implementations is that `kadmin` has a different, but equivalent, set of commands and uses a different protocol. If the KDC is MIT, the Heimdal version of `kadmin` cannot be used to administer the KDC remotely, and vice versa.

Client applications may also use slightly different command line options to accomplish the same tasks. Following the instructions at http://web.mit.edu/Kerberos/www/ is recommended. Be careful of path issues: the MIT port installs into **/usr/local/** by default, and the FreeBSD system applications run instead of the MIT versions if `PATH` lists the system directories first.

When using MIT Kerberos as a KDC on FreeBSD, execute the following commands to add the required configurations to **/etc/rc.conf**:

```
# sysrc kdc_program="/usr/local/sbin/krb5kdc"
# sysrc kadmind_program="/usr/local/sbin/kadmind"
# sysrc kdc_flags=""
# sysrc kdc_enable="YES"
# sysrc kadmind_enable="YES"
```

16.9.5. Kerberos Tips, Tricks, and Troubleshooting

When configuring and troubleshooting Kerberos, keep the following points in mind:

- When using either Heimdal or MITKerberos from ports, ensure that the `PATH` lists the port's versions of the client applications before the system versions.

- If all the computers in the realm do not have synchronized time settings, authentication may fail. "Clock Synchronization with NTP" describes how to synchronize clocks using NTP.

- If the hostname is changed, the `host/` principal must be changed and the keytab updated. This also applies to special keytab entries like the `HTTP/` principal used for Apache's www/mod_auth_kerb.

- All hosts in the realm must be both forward and reverse resolvable in DNS or, at a minimum, exist in **/etc/hosts**. CNAMEs will work, but the A and PTR records must be correct and in place. The error message for unresolvable hosts is not intuitive: `Kerberos5 refuses authentication because Read req failed: Key table entry not found.`

- Some operating systems that act as clients to the KDC do not set the permissions for `ksu` to be setuid `root`. This means that `ksu` does not work. This is a permissions problem, not a KDC error.

- With MITKerberos, to allow a principal to have a ticket life longer than the default lifetime of ten hours, use `modify_principal` at the kadmin(8) prompt to change the `maxlife` of both the principal in question and the `krbtgt` principal. The principal can then use `kinit -l` to request a ticket with a longer lifetime.

- When running a packet sniffer on the KDC to aid in troubleshooting while running `kinit` from a workstation, the Ticket Granting Ticket (TGT) is sent immediately, even before the password is typed. This is because the Kerberos server freely transmits a TGT to any unauthorized request. However, every TGT is encrypted in a key derived from the user's password. When a user types their password, it is not sent to the KDC, it is instead used to decrypt the TGT that `kinit` already obtained. If the decryption process results in a valid ticket with a valid time stamp, the user has valid Kerberos credentials. These credentials include a session key for establishing secure communications with the Kerberos server in the future, as well as the actual TGT, which is encrypted with the Kerberos server's own key. This second layer of encryption allows the Kerberos server to verify the authenticity of each TGT.

- Host principals can have a longer ticket lifetime. If the user principal has a lifetime of a week but the host being connected to has a lifetime of nine hours, the user cache will have an expired host principal and the ticket cache will not work as expected.

- When setting up **krb5.dict** to prevent specific bad passwords from being used as

described in kadmind(8), remember that it only applies to principals that have a password policy assigned to them. The format used in **krb5.dict** is one string per line. Creating a symbolic link to **/usr/share/dict/words** might be useful.

16.9.6. Mitigating Kerberos Limitations

Since Kerberos is an all or nothing approach, every service enabled on the network must either be modified to work with Kerberos or be otherwise secured against network attacks. This is to prevent user credentials from being stolen and re-used. An example is when Kerberos is enabled on all remote shells but the non-Kerberized POP3 mail server sends passwords in plain text.

The KDC is a single point of failure. By design, the KDC must be as secure as its master password database. The KDC should have absolutely no other services running on it and should be physically secure. The danger is high because Kerberos stores all passwords encrypted with the same master key which is stored as a file on the KDC.

A compromised master key is not quite as bad as one might fear. The master key is only used to encrypt the Kerberos database and as a seed for the random number generator. As long as access to the KDC is secure, an attacker cannot do much with the master key.

If the KDC is unavailable, network services are unusable as authentication cannot be performed. This can be alleviated with a single master KDC and one or more slaves, and with careful implementation of secondary or fall-back authentication using PAM.

Kerberos allows users, hosts and services to authenticate between themselves. It does not have a mechanism to authenticate the KDC to the users, hosts, or services. This means that a trojaned `kinit` could record all user names and passwords. File system integrity checking tools like security/tripwire can alleviate this.

16.9.7. Resources and Further Information

- The Kerberos FAQ
- Designing an Authentication System: a Dialog in Four Scenes
- RFC 4120, The Kerberos Network Authentication Service (V5)
- MIT Kerberos home page

- Heimdal Kerberos project wiki page

16.10. TCP Wrappers

TCP Wrappers is a host-based network access control system. By intercepting incoming network requests before they reach the actual network service, TCP Wrappers assess whether the source IP address is permitted or denied access based on predefined rules in configuration files.

However, while TCP Wrappers provide basic access control, they should not be considered a substitute for more robust security measures. For comprehensive protection, it's recommended to use advanced technologies like firewalls, along with proper user authentication practices and intrusion detection systems.

16.10.1. Initial Configuration

TCP Wrappers are enabled by default in inetd(8). So the first step will be to enable inetd(8) executing the following commands:

```
# sysrc inetd_enable="YES"
# service inetd start
```

Then, properly configure **/etc/hosts.allow**.

> **Warning**
> Unlike other implementations of TCP Wrappers, the use of **hosts.deny** is deprecated in FreeBSD. All configuration options should be placed in **/etc/hosts.allow**.

In the simplest configuration, daemon connection policies are set to either permit or block, depending on the options in **/etc/hosts.allow**. The default configuration in FreeBSD is to allow all connections to the daemons started with inetd.

Basic configuration usually takes the form of `daemon : address : action`, where `daemon` is the daemon which inetd started, `address` is a valid hostname, IP address, or an IPv6 address enclosed in brackets ([ ]), and `action` is either `allow` or `deny`. TCP

Wrappers uses a first rule match semantic, meaning that the configuration file is scanned from the beginning for a matching rule. When a match is found, the rule is applied and the search process stops.

For example, to allow POP3 connections via the mail/qpopper daemon, the following lines should be appended to **/etc/hosts.allow**:

```
# This line is required for POP3 connections:
qpopper : ALL : allow
```

Whenever this file is edited, restart inetd:

```
# service inetd restart
```

16.10.2. Advanced Configuration

TCP Wrappers provides advanced options to allow more control over the way connections are handled. In some cases, it may be appropriate to return a comment to certain hosts or daemon connections. In other cases, a log entry should be recorded or an email sent to the administrator. Other situations may require the use of a service for local connections only. This is all possible through the use of configuration options known as wildcards, expansion characters, and external command execution. To learn more about wildcards and their associated functionality, refer to hosts_access(5).

16.11. Access Control Lists

Access Control Lists (ACLs) extend traditional UNIX® file permissions by allowing fine-grained access control for users and groups on a per-file or per-directory basis. Each ACL entry defines a user or group and the associated permissions, such as read, write, and execute. FreeBSD provides commands like getfacl(1) and setfacl(1) to manage ACLs.

ACLs are useful in scenarios requiring more specific access control than standard permissions, commonly used in multi-user environments or shared hosting. However, complexity may be unavoidable, but careful planning is required to ensure that the desired security properties are being provided

> **Note**
>
> FreeBSD supports the implementation of NFSv4 ACLs in both UFS and OpenZFS. Please note that some arguments to the setfacl(1) command only work with POSIX ACLs and others in NFSv4 ACLs.

16.11.1. Enabling ACL Support in UFS

ACLs are enabled by the mount-time administrative flag, `acls`, which may be added to **/etc/fstab**.

Therefore it will be necessary to access **/etc/fstab** and in the options section add the `acls` flag as follows:

```
# Device        Mountpoint        FStype    Options     Dump      Pa
/dev/ada0s1a    /                 ufs       rw,acls     1         1
```

16.11.2. Get ACLs information

It is possible to check the ACLs of a file or a directory using getfacl(1).

For example, to view the ACL settings on **~/test** file execute the following command:

```
% getfacl test
```

The output should be similar to the following in case of using NFSv4 ACLs:

```
# file: test
# owner: freebsduser
# group: freebsduser
           owner@:rw-p--aARWcCos:-------:allow
           group@:r-----a-R-c--s:-------:allow
        everyone@:r-----a-R-c--s:-------:allow
```

And the output should be similar to the following in case of using POSIX.1e ACLs:

```
# file: test
# owner: freebsduser
# group: freebsduser
user::rw-
group::r--
other::r--
```

16.11.3. Working with ACLs

setfacl(1) can be used to add, modify or remove ACLs from a file or directory.

As noted above, some arguments to setfacl(1) do not work with NFSv4 ACLs, and vice versa. This section covers how to execute the commands for POSIX ACLs and for NFSv4 ACLs and shows examples of both.

For example, to set the mandatory elements of the POSIX.1e default ACL:

```
% setfacl -d -m u::rwx,g::rx,o::rx,mask::rwx directory
```

This other example sets read, write, and execute permissions for the file owner's POSIX.1e ACL entry and read and write permissions for group mail on file:

```
% setfacl -m u::rwx,g:mail:rw file
```

To do the same as in the previous example but in NFSv4 ACL:

```
% setfacl -m owner@:rwxp::allow,g:mail:rwp::allow file
```

To remove all ACL entries except for the three required from file in POSIX.1e ACL:

```
% setfacl -bn file
```

To remove all ACL entries in NFSv4 ACL:

```
% setfacl -b file
```

Refer to getfacl(1) and setfacl(1) for more information about the options available for these commands.

16.12. Capsicum

Capsicum is a lightweight OS capability and sandbox framework implementing a hybrid capability system model. Capabilities are unforgeable tokens of authority that can be delegated and must be presented to perform an action. Capsicum makes file descriptors into capabilities.

Capsicum can be used for application and library compartmentalisation, the decomposition of larger bodies of software into isolated (sandboxed) components in order to implement security policies and limit the impact of software vulnerabilities.

16.13. Process Accounting

Process accounting is a security method in which an administrator may keep track of system resources used and their allocation among users, provide for system monitoring, and minimally track a user's commands.

Process accounting has both positive and negative points. One of the positives is that an intrusion may be narrowed down to the point of entry. A negative is the amount of logs generated by process accounting, and the disk space they may require. This section walks an administrator through the basics of process accounting.

> **Note**
> If more fine-grained accounting is needed, refer to Security Event Auditing.

16.13.1. Enabling and Utilizing Process Accounting

Before using process accounting, it must be enabled using the following commands:

```
# sysrc accounting_enable=yes
```

```
# service accounting start
```

The accounting information is stored in files located in **/var/account**, which is automatically created, if necessary, the first time the accounting service starts. These files contain sensitive information, including all the commands issued by all users. Write access to the files is limited to `root`, and read access is limited to `root` and members of the `wheel` group. To also prevent members of `wheel` from reading the files, change the mode of the **/var/account** directory to allow access only by `root`.

Once enabled, accounting will begin to track information such as CPU statistics and executed commands. All accounting logs are in a non-human readable format which can be viewed using sa(8). If issued without any options, sa(8) prints information relating to the number of per-user calls, the total elapsed time in minutes, total CPU and user time in minutes, and the average number of I/O operations. Refer to sa(8) for the list of available options which control the output.

To display the commands issued by users, use `lastcomm`.

For example, this command prints out all usage of `ls` by `trhodes` on the `ttyp1` terminal:

```
# lastcomm ls trhodes ttyp1
```

Many other useful options exist and are explained in lastcomm(1), acct(5), and sa(8).

16.14. Resource Limits

In FreeBSD, resource limits refer to the mechanisms that control and manage the allocation of various system resources to processes and users. These limits are designed to prevent a single process or user from consuming an excessive amount of resources, which could lead to performance degradation or system instability. Resource limits help ensure fair resource distribution among all active processes and users on the system.

FreeBSD provides several methods for an administrator to limit the amount of system resources an individual may use.

The traditional method defines login classes by editing **/etc/login.conf**. While this method is still supported, any changes require a multi-step process of editing this file,

rebuilding the resource database, making necessary changes to **/etc/master.passwd**, and rebuilding the password database. This can become time consuming, depending upon the number of users to configure.

rctl(8) can be used to provide a more fine-grained method for controlling resource limits. This command supports more than user limits as it can also be used to set resource constraints on processes and jails.

This section demonstrates both methods for controlling resources, beginning with the traditional method.

16.14.1. Types of Resources

FreeBSD provides limits for various types of resources, including:

Table 1. Resource types

Type	Description
CPU Time	Limits the amount of CPU time a process can consume
Memory	Controls the amount of physical memory a process can use
Open Files	Limits the number of files a process can have open simultaneously
Processes	Controls the number of processes a user or a process can create
File Size	Limits the maximum size of files that a process can create

Type	Description
Core Dumps	Controls whether processes are allowed to generate core dump files
Network Resources	Limits the amount of network resources (e.g., sockets) a process can use

For a full listing of types see login.conf(5) and rctl(8).

16.14.2. Configuring Login Classes

In the traditional method, login classes and the resource limits to apply to a login class are defined in **/etc/login.conf**. Each user account can be assigned to a login class, where `default` is the default login class. Each login class has a set of login capabilities associated with it. A login capability is a *`name=value`* pair, where *name* is a well-known identifier and *value* is an arbitrary string which is processed accordingly depending on the *name*.

The first step to configure a resource limit will be to open **/etc/login.conf** by executing the following command:

```
# ee /etc/login.conf
```

Then locate the section for the user class to be modified. In this example, let's assume the user class is named `limited`, create it in case it does not exist.

```
limited:\  1
        :maxproc=50:\  2
        :tc=default:  3
```

1 Name of the user class.

2 Sets the maximum number of processes (maxproc) to 50 for users in the `limited` class.

3 Indicates that this user class inherits the default settings from the "default" class.

After modifying the **/etc/login.conf** file, run cap_mkdb(1) to generate the database that FreeBSD uses to apply these settings:

```
# cap_mkdb /etc/login.conf
```

chpass(1) can be used to change the class to the desired user by executing the following command:

```
# chpass username
```

This will open a text editor, add the new `limited` class there as follows:

```
#Changing user information for username.
Login: username
Password: $6$2H.419USdGaiJeqK$6kgcTnDadasdasd3YnlNZsOni5AMymibkAfRCP
DVsKyXx26daabdfqSdasdsmL/ZMUpdHiO0
Uid [#]: 1001
Gid [# or name]: 1001
Change [month day year]:
Expire [month day year]:
Class: limited
Home directory: /home/username
Shell: /bin/sh
Full Name: User &
Office Location:
Office Phone:
Home Phone:
Other information:
```

Now, the user assigned to the `limited` class will have a maximum process limit of 50. Remember that this is just one example of setting a resource limit using the **/etc/login.conf** file.

Keep in mind that after making changes to the **/etc/login.conf** file, the user needs to log out and log back in for the changes to take effect. Additionally, always exercise caution when editing system configuration files, especially when using privileged access.

16.14.3. Enabling and Configuring Resource Limits

The rctl(8) system provides a more fine-grained way to set and manage resource limits for individual processes and users. It allows you to dynamically assign resource limits to specific processes or users, regardless of their user class.

The first step to use rctl(8) will be to enable it adding the following line to **/boot/loader.conf** and reboot the system:

```
kern.racct.enable=1
```

Then enable and start the rctl(8) service by executing the following commands:

```
# sysrc rctl_enable="YES"
# service rctl start
```

Then rctl(8) may be used to set rules for the system.

Rule syntax (rctl.conf(5)) is controlled through the use of a subject, subject-id, resource, and action, as seen in this example rule:

```
subject:subject-id:resource:action=amount/per
```

For example to constrained the user to add no more than 10 processes execute the following command:

```
# rctl -a user:username:maxproc:deny=10/user
```

To check the applied resource limits the rctl(8) command can be executed:

```
# rctl
```

The output should be similar to the following:

```
user:username:maxproc:deny=10
```

Rules will persist across reboots if they have been added to **/etc/rctl.conf**. The format is a rule, without the preceding command. For example, the previous rule could be added as:

```
user:username:maxproc:deny=10
```

16.15. Monitoring Third Party Security Issues

In recent years, the security world has made many improvements to how vulnerability assessment is handled. The threat of system intrusion increases as third party utilities are installed and configured for virtually any operating system available today.

Vulnerability assessment is a key factor in security. While FreeBSD releases advisories for the base system, doing so for every third party utility is beyond the FreeBSD Project's capability. There is a way to mitigate third party vulnerabilities and warn administrators of known security issues. A FreeBSD add on utility known as pkg includes options explicitly for this purpose.

pkg polls a database for security issues. The database is updated and maintained by the FreeBSD Security Team and ports developers.

Installation provides periodic(8) configuration files for maintaining the pkg audit database, and provides a programmatic method of keeping it updated.

After installation, and to audit third party utilities as part of the Ports Collection at any time, an administrator may choose to update the database and view known vulnerabilities of installed packages by invoking:

```
% pkg audit -F
```

The output should be similar to the following:

```
vulnxml file up-to-date
chromium-116.0.5845.96_1 is vulnerable:
  chromium -- multiple vulnerabilities
  CVE: CVE-2023-4431
```

```
    CVE: CVE-2023-4427
    CVE: CVE-2023-4428
    CVE: CVE-2023-4429
    CVE: CVE-2023-4430
    WWW: https://vuxml.FreeBSD.org/freebsd/5fa332b9-4269-11ee-8290-a8a

  samba413-4.13.17_5 is vulnerable:
    samba -- multiple vulnerabilities
    CVE: CVE-2023-3347
    CVE: CVE-2023-34966
    CVE: CVE-2023-34968
    CVE: CVE-2022-2127
    CVE: CVE-2023-34967
    WWW: https://vuxml.FreeBSD.org/freebsd/441e1e1a-27a5-11ee-a156-080

  2 problem(s) in 2 installed package(s) found.
```

By pointing a web browser to the displayed URL, an administrator may obtain more information about the vulnerability.

This will include the versions affected, by FreeBSD port version, along with other web sites which may contain security advisories.

16.16. FreeBSD Security Advisories

Like many producers of quality operating systems, the FreeBSD Project has a security team which is responsible for determining the End-of-Life (EoL) date for each FreeBSD release and to provide security updates for supported releases which have not yet reached their EoL. More information about the FreeBSD security team and the supported releases is available on the FreeBSD security page.

One task of the security team is to respond to reported security vulnerabilities in the FreeBSD operating system. Once a vulnerability is confirmed, the security team verifies the steps necessary to fix the vulnerability and updates the source code with the fix. It then publishes the details as a "Security Advisory". Security advisories are published on the FreeBSD website and mailed to the FreeBSD security notifications mailing list, FreeBSD security mailing list, and FreeBSD announcements mailing list.

16.16.1. Format of a Security Advisory

Here is an example of a FreeBSD security advisory:

```
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA512

=============================================================================
FreeBSD-SA-23:07.bhyve                                          Security
                                                               The FreeBS

Topic:          bhyve privileged guest escape via fwctl

Category:       core
Module:         bhyve
Announced:      2023-08-01
Credits:        Omri Ben Bassat and Vladimir Eli Tokarev from Micros
Affects:        FreeBSD 13.1 and 13.2
Corrected:      2023-08-01 19:48:53 UTC (stable/13, 13.2-STABLE)
                2023-08-01 19:50:47 UTC (releng/13.2, 13.2-RELEASE-p
                2023-08-01 19:48:26 UTC (releng/13.1, 13.1-RELEASE-p
CVE Name:       CVE-2023-3494

For general information regarding FreeBSD Security Advisories,
including descriptions of the fields above, security branches, and t
following sections, please visit <URL:https://security.FreeBSD.org/>

I.   Background

bhyve(8)'s fwctl interface provides a mechanism through which guest
firmware can query the hypervisor for information about the virtual
machine.  The fwctl interface is available to guests when bhyve is r
with the "-l bootrom" option, used for example when booting guests i
UEFI mode.

bhyve is currently only supported on the amd64 platform.

II.  Problem Description

The fwctl driver implements a state machine which is executed when t
guest accesses certain x86 I/O ports.  The interface lets the guest
a string into a buffer resident in the bhyve process' memory.  A bug
the state machine implementation can result in a buffer overflowin
copying this string.
```

```
III. Impact

A malicious, privileged software running in a guest VM can exploit t
buffer overflow to achieve code execution on the host in the bhyve
userspace process, which typically runs as root.  Note that bhyve ru
in a Capsicum sandbox, so malicious code is constrained by the
capabilities available to the bhyve process.

IV.  Workaround

No workaround is available.  bhyve guests that are executed without
"-l bootrom" option are unaffected.

V.   Solution

Upgrade your vulnerable system to a supported FreeBSD stable or
release / security branch (releng) dated after the correction date.

Perform one of the following:

1) To update your vulnerable system via a binary patch:

Systems running a RELEASE version of FreeBSD on the amd64, i386, or
(on FreeBSD 13 and later) arm64 platforms can be updated via the
freebsd-update(8) utility:

# freebsd-update fetch
# freebsd-update install

Restart all affected virtual machines.

2) To update your vulnerable system via a source code patch:

The following patches have been verified to apply to the applicable
FreeBSD release branches.

a) Download the relevant patch from the location below, and verify t
detached PGP signature using your PGP utility.

[FreeBSD 13.2]
# fetch https://security.FreeBSD.org/patches/SA-23:07/bhyve.13.2.pat
```

```
# fetch https://security.FreeBSD.org/patches/SA-23:07/bhyve.13.2.pat
# gpg --verify bhyve.13.2.patch.asc

[FreeBSD 13.1]
# fetch https://security.FreeBSD.org/patches/SA-23:07/bhyve.13.1.pat
# fetch https://security.FreeBSD.org/patches/SA-23:07/bhyve.13.1.pat
# gpg --verify bhyve.13.1.patch.asc

b) Apply the patch.  Execute the following commands as root:

# cd /usr/src
# patch < /path/to/patch

c) Recompile the operating system using buildworld and installworld
described in <URL:https://www.FreeBSD.org/handbook/makeworld.html>.

Restart all affected virtual machines.

VI.  Correction details

This issue is corrected by the corresponding Git commit hash or Subv
revision number in the following stable and release branches:

Branch/path                                Hash                     Rev
- -------------------------------------------------------------------
stable/13/                                 9fe302d78109    stable/13-n2
releng/13.2/                               2bae613e0da3  releng/13.2-n2
releng/13.1/                               87702e38a4b4  releng/13.1-n2
- -------------------------------------------------------------------

Run the following command to see which files were modified by a
particular commit:

# git show --stat <commit hash>

Or visit the following URL, replacing NNNNNN with the hash:

<URL:https://cgit.freebsd.org/src/commit/?id=NNNNNN>

To determine the commit count in a working tree (for comparison agai
nNNNNNN in the table above), run:
```

```
# git rev-list --count --first-parent HEAD

VII. References

<URL:https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2023-3494>

The latest revision of this advisory is available at
<URL:https://security.FreeBSD.org/advisories/FreeBSD-SA-23:07.bhyve.
-----BEGIN PGP SIGNATURE-----

iQIzBAEBCgAdFiEEthUnfoEIffdcgYM7bljekB8AGu8FAmTJdsIACgkQbljekB8A
Gu8Q1Q/7BFw5Aa0cFxBzbdz+O5NAImj58MvKS6xw61bXcYr12jchyT6ENC7yiR+K
qCqbe5TssRbtZ1gg/94gSGEXccz5OcJGxW+qozhcdPUh2L2nzBPkMCrclrYJfTtM
cnmQKjg/wFZLUVr71GEM95ZFaktlZdXyXx9Z8eBzow5rXexpl1TTHQQ2kZZ41K4K
KFhup91dzGCIj02cqbl+1h5BrXJe3s/oNJt5JKIh/GBh5THQu9n6AywQYl18HtjV
fMb1qRTAS9WbiEP5QV2eEuOG86ucuhytqnEN5MnXJ2rLSjfb9izs9HzLo3ggy7yb
hN3tlbfIPjMEwYexieuoyP3rzKkLeYfLXqJU4zKCRnIbBIkMRy4mcFkfcYmI+MhF
NPh2R9kccemppKXeDhKJurH0vsetr8ti+AwOZ3pgO21+9w+mjE+EfaedIi+JWhip
hwqeFv03bAQHJdacNYGV47NsJ91CY4ZgWC3ZOzBZ2Y5SDtKFjyc0bf83WTfU9A/0
drC0z3xaJribah9e6k5d7lmZ7L6aHCbQ70+aayuAEZQLr/N1doB0smNi0IHdrtY0
JdIqmVX+d1ihVhJ05prC460AS/Kolqiaysun1igxR+ZnctE9Xdo1BlLEbYu2KjT4
LpWvSuhRMSQaYkJU72SodQc0FM5mqqNN42Vx+X4EutOfvQuRGlI=
=MlAY
-----END PGP SIGNATURE-----
```

Every security advisory uses the following format:

- Each security advisory is signed by the PGP key of the Security Officer. The public key for the Security Officer can be verified at OpenPGP Keys.
- The name of the security advisory always begins with `FreeBSD-SA-` (for FreeBSD Security Advisory), followed by the year in two digit format (`23:`), followed by the advisory number for that year (`07.`), followed by the name of the affected application or subsystem (`bhyve`).
- The `Topic` field summarizes the vulnerability.
- The `Category` refers to the affected part of the system which may be one of `core`, `contrib`, or `ports`. The `core` category means that the vulnerability affects a core component of the FreeBSD operating system. The `contrib` category means that the vulnerability affects software included with FreeBSD, such as BIND. The `ports` category indicates that the vulnerability affects software

available through the Ports Collection.

- The `Module` field refers to the component location. In this example, the `bhyve` module is affected; therefore, this vulnerability affects an application installed with the operating system.
- The `Announced` field reflects the date the security advisory was published. This means that the security team has verified that the problem exists and that a patch has been committed to the FreeBSD source code repository.
- The `Credits` field gives credit to the individual or organization who noticed the vulnerability and reported it.
- The `Affects` field explains which releases of FreeBSD are affected by this vulnerability.
- The `Corrected` field indicates the date, time, time offset, and releases that were corrected. The section in parentheses shows each branch for which the fix has been merged, and the version number of the corresponding release from that branch. The release identifier itself includes the version number and, if appropriate, the patch level. The patch level is the letter `p` followed by a number, indicating the sequence number of the patch, allowing users to track which patches have already been applied to the system.
- The `CVE Name` field lists the advisory number, if one exists, in the public cve.mitre.org security vulnerabilities database.
- The `Background` field provides a description of the affected module.
- The `Problem Description` field explains the vulnerability. This can include information about the flawed code and how the utility could be maliciously used.
- The `Impact` field describes what type of impact the problem could have on a system.
- The `Workaround` field indicates if a workaround is available to system administrators who cannot immediately patch the system.
- The `Solution` field provides the instructions for patching the affected system. This is a step by step tested and verified method for getting a system patched and working securely.
- The `Correction Details` field displays each affected Subversion or Git branch with the revision number that contains the corrected code.

- The `References` field offers sources of additional information regarding the vulnerability.

Last modified on: February 18, 2025 by Fernando Apesteguía

‹ Prev Home Next ›

Community

Get involved

Community forum

Mailing lists

IRC Channels

Bug Tracker

Legal

Donations

Licensing

Privacy Policy

Legal notices

Made with ♥ by the FreeBSD Community