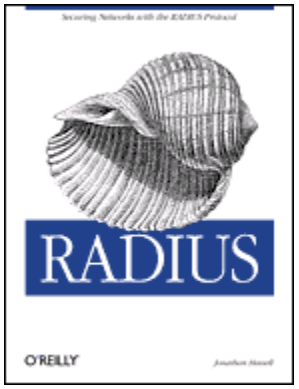


RADIUS

Securing Public Access to Private Resources

By Jonathan Hassell
October 2002
ISBN 0-596-00322-6,
206 pages

Chapter 5
Getting Started with FreeRADIUS

Up to this point, I've talked about the theoretical underpinnings of both the authentication-authorization-accounting (AAA) architecture as well as the specific implementation of AAA characteristics that is the RADIUS protocol. I will now focus on practical applications of RADIUS: implementing it, customizing it for your specific needs, and extending its capabilities to meet other needs in your business. First, though, I need a product that talks RADIUS.

Enter FreeRADIUS.

Introduction to FreeRADIUS

The developers of FreeRADIUS speak on their product and its development, from the FreeRADIUS web site:

> FreeRADIUS is one of the most modular and featureful [sic] RADIUS servers available today. It has been written by a team of developers who have more than a decade of collective experience in implementing and deploying RADIUS software, in software engineering, and in Unix package management. The product is the result of synergy between many of the best-known names in free software-based RADIUS implementations, including several developers of the Debian GNU/Linux operating system, and is distributed under the GNU GPL (version 2).

FreeRADIUS is a complete rewrite, ground-up compilation of a RADIUS server. The configuration files exhibit many similarities to the old Livingston RADIUS server. The product includes support for:

- Limiting the maximum number of simultaneous logons, even on a per-user basis
- More than one DEFAULT entry, with each being capable of "falling through" to the next
- Permitting and denying access to users based on the *huntgroup* to which they are connected
- Setting certain parameters to be huntgroup specific
- Intelligent "hints" files that select authentication protocols based on the syntax of the username
- Executing external programs upon successful login
- Using the $INCLUDE filename format with configuration, users, and dictionary files
- Vendor-specific attributes
- Acting as a proxy RADIUS server

FreeRADIUS supports the following popular NAS equipment:

- 3Com/USR Hiper Arc Total Control
- 3Com/USR NetServer
- 3Com/USR TotalControl
- Ascend Max 4000 family
- Cisco Access Server family
- Cistron PortSlave
- Computone PowerRack
- Cyclades PathRAS
- Livingston PortMaster
- Multitech CommPlete Server
- Patton 2800 family

FreeRADIUS is available for a wide range of platforms, including Linux, FreeBSD, OpenBSD, OSF/Unix, and Solaris. For the purposes of this book, I will focus on FreeRADIUS running under Linux. Also, as of this printing, a stable Version 1.0 of the product had not been released. However, development of the server is very stable, careful, and somewhat slow, so changes to the procedures mentioned are unlikely. In the event a procedure does change, it's likely to be a relatively small modification. Always check the FreeRADIUS web site for up-to-date details.

Installing FreeRADIUS

At present, the FreeRADIUS team doesn't offer precompiled binaries. The best way to start off is to grab the latest source code, compressed using tar and *gzip*, from the FreeRADIUS web site at http://www.freeradius.org. Once the file is on your computer, execute the following command to uncompress the file:

```
tar -zxvf freeradius.tar.gz
```

Next, you'll need to compile FreeRADIUS. Make sure your system at least has *gcc*, *glibc*, *binutils*, and *gmake* installed before trying to compile. To begin compiling, change to the directory where your uncompressed source code lies and execute *./configure* from the command line. You can also run *./configure -flags* and customize the settings for the flags in Table 5-1.

Table 5-1:Optional configuration flags for FreeRADIUS

Flag	Purpose	Default
--enable-shared[=PKGS]	Builds shared libraries.	Yes
--enable-static[=PKGS]	Builds static libraries.	Yes
--enable-fast-install[=PKGS]	Optimizes the resulting files for fastest installation.	Yes
--with-gnu-ld	Makes the procedure assume the C compiler uses *GNU lD*.	No
--disable-libtool-lock	Avoids locking problems. This may break parallel builds.	Not applicable
--with-logdir=DIR	Specifies the directory for log files.	LOCALSTATEDIR/log
--with-radacctdir=DIR	Specifies the directory for detail files.	LOGDIR/radacct
--with-raddbdir=DIR	Specifies the directory for configuration files.	SYSCONFDIR/raddb
--with-dict-nocase	Makes the dictionary case insensitive.	Yes
--with-ascend-binary	Includes support for attributes provided with the Ascend binary filter.	Yes
--with-threads	Uses threads if they're supported and available.	Yes

--with-snmp	Compiles SNMP support into the binaries.	Yes
--with-mysql-include-dir=DIR	Specifies where the include files for MySQL can be found.	Not applicable
--with-mysql-lib-dur=DIR	Specifies where the dictionary files for MySQL can be found.	Not applicable
--with-mysql-dir-DIR	Specifies where MySQL is installed on the local system.	Not applicable
--disable-ltdl-install	Does not install *libltdl*.	Not applicable
--with-static-modules=QUOTED-MODULE-LIST	Compiles the list of modules statically.	Not applicable
--enable-developer	Turns on extra developer warnings in the compiler.	Not applicable

Commonly, the following locations are used when installing a RADIUS product (these practices go back to the Cistron RADIUS server):

Binaries
: */usr/local/bin* and */usr/local/sbin*

Manual (man) pages
: */usr/local/man*

Configuration files
: */etc/raddb*

Log files
: */var/log* and */var/log/radacct*

To make the compiler use these locations automatically, execute:

```
./configure --localstatedir=/var --sysconfdir=/etc
```

The programs will then be configured to compile. The rest of this chapter will assume that you installed FreeRADIUS in these locations.

Next, type `make`. This will compile the binaries. Finally, type `make install`. This will place all of the files in the appropriate locations. It will also install configuration files if this server has not had a RADIUS server installed before. Otherwise, the procedure will not overwrite your existing configuration and will report to you on what files it did not install.

At this point, your base FreeRADIUS software is installed. Before you begin, though, you'll need to customize some of the configuration files so that they point to machines and networks specific to your configuration. Most of these files are located in */etc/raddb*. The following files are contained by default:

```
radius:/etc/raddb # ls -al
total 396
drwxr-xr-x    2 root     root         4096 Apr 10 10:39 .
drwxr-xr-x    3 root     root         4096 Apr 10 10:18 ..
-rw-r--r--    1 root     root          635 Apr 10 10:18 acct_users
-rw-r--r--    1 root     root         3431 Apr 10 10:18 attrs
-rw-r--r--    1 root     root          595 Apr 10 11:02 clients
-rw-r--r--    1 root     root         2235 Apr 10 10:39 clients.conf
-rw-r--r--    1 root     root        12041 Apr 10 10:18 dictionary
-rw-r--r--    1 root     root        10046 Apr 10 10:39 dictionary.acc
-rw-r--r--    1 root     root         1320 Apr 10 10:39 dictionary.aptis
-rw-r--r--    1 root     root        54018 Apr 10 10:39 dictionary.ascend
-rw-r--r--    1 root     root        11051 Apr 10 10:39 dictionary.bay
-rw-r--r--    1 root     root         4763 Apr 10 10:39 dictionary.cisco
-rw-r--r--    1 root     root         1575 Apr 10 10:39 dictionary.compat
-rw-r--r--    1 root     root         1576 Apr 10 10:39 dictionary.erx
-rw-r--r--    1 root     root          375 Apr 10 10:39 dictionary.foundry
-rw-r--r--    1 root     root          279 Apr 10 10:39 dictionary.freeradius
-rw-r--r--    1 root     root         2326 Apr 10 10:39 dictionary.livingston
-rw-r--r--    1 root     root         2396 Apr 10 10:39 dictionary.microsoft
-rw-r--r--    1 root     root          190 Apr 10 10:39 dictionary.nomadix
-rw-r--r--    1 root     root         1537 Apr 10 10:39 dictionary.quintum
-rw-r--r--    1 root     root         8563 Apr 10 10:39 dictionary.redback
-rw-r--r--    1 root     root          457 Apr 10 10:39 dictionary.shasta
-rw-r--r--    1 root     root         2958 Apr 10 10:39 dictionary.shiva
-rw-r--r--    1 root     root         1274 Apr 10 10:39 dictionary.tunnel
-rw-r--r--    1 root     root        63265 Apr 10 10:39 dictionary.usr
-rw-r--r--    1 root     root         2199 Apr 10 10:39 dictionary.versanet
-rw-r--r--    1 root     root         1767 Apr 10 10:18 hints
-rw-r--r--    1 root     root         1603 Apr 10 10:18 huntgroups
-rw-r--r--    1 root     root         2289 Apr 10 10:39 ldap.attrmap
-rw-r--r--    1 root     root          830 Apr 10 10:18 naslist
-rw-r--r--    1 root     root          856 Apr 10 10:18 naspasswd
-rw-r--r--    1 root     root         9533 Apr 10 10:39 postgresql.conf
-rw-r--r--    1 root     root         4607 Apr 10 10:39 proxy.conf
-rw-r--r--    1 root     root        27266 Apr 10 10:57 radiusd.conf
-rw-r--r--    1 root     root        27232 Apr 10 10:39 radiusd.conf.in
-rw-r--r--    1 root     root         1175 Apr 10 10:18 realms
-rw-r--r--    1 root     root         1405 Apr 10 10:39 snmp.conf
-rw-r--r--    1 root     root         9089 Apr 10 10:39 sql.conf
-rw-r--r--    1 root     root         6941 Apr 10 10:18 users
-rw-r--r--    1 root     root         6702 Apr 10 10:39 x99.conf
-rw-r--r--    1 root     root         3918 Apr 10 10:39 x99passwd.sample
```

The clients File

First, take a look at the */etc/raddb/clients* file. This file lists the hosts authorized to hit the FreeRADIUS server with requests and the secret key those hosts will use in their requests. Some common entries are already included in the */etc/raddb/clients* file, so you may wish to simply uncomment the appropriate lines. Make sure the secret key that is listed in the *clients* file is the same as that programmed into your RADIUS client equipment. Also, add the IP address of a desktop console machine with which you can test your setup using a RADIUS ping utility. A sample clients file looks like this:

```
# Client Name           Key
#----------------       ----------
#portmaster1.isp.com    testing123
#portmaster2.isp.com    testing123
#proxyradius.isp2.com   TheirKey
localhost               testing123
192.168.1.100           testing123
tc-clt.hasselltech.net  oreilly
```

> **TIP:** It's recommended by the FreeRADIUS developers that users move from the clients file to the *clients.conf* file. The *clients.conf* file will be addressed later in Chapter 6, but for the sake of simplicity and startup testing, I will continue using the plain clients file in this introduction.

While it may seem obvious, *change the shared secrets* from the defaults in the file or the samples listed previously. Failing to do so presents a significant security risk to your implementation and network.

The naslist File

Next, open the */etc/raddb/naslist* file. Inside this file, you should list the full canonical name of every NAS that will hit this server, its nickname, and the type of NAS. For your test console, you can simply use the "portslave" type. Table 5-2 lists the FreeRADIUS-supported NAS equipment and the type identifier needed for the *naslist* file.

Table 5-2:Supported NAS equipment and its type identifier

NAS equipment	Type identifier
3Com/USR Hiper Arc Total Control	usrhiper
3Com/USR NetServer	netserver
3Com/USR TotalControl	tc
Ascend Max 4000 family	max40xx
Cisco Access Server family	cisco
Cistron PortSlave	portslave
Computone PowerRack	computone
Cyclades PathRAS	pathras
Livingston PortMaster	livingston
Multitech CommPlete Server	multitech
Patton 2800 family	patton

A sample */etc/raddb/naslist* file looks like this:

```
# NAS Name              Short Name      Type
#----------------       ----------      ----
#portmaster1.isp.com    pm1.NY          livingston
localhost               local           portslave
192.168.1.100           local           portslave
tc-clt.hasselltech.net  tc.char         tc
```

The naspasswd File

If you have 3Com/USR Total Control, NetServer, or Cyclades PathRAS equipment, you may need to edit the */etc/raddb/naspasswd* file. This lets the *checkrad* utility log onto your NAS machine and check to see who is logged on at what port--which is commonly used to detect multiple logins. Normally, the SNMP protocol can do this, but the equipment listed previously needs a helping hand from the *checkrad* utility. A sample */etc/raddb/naspasswd* file looks like this:

```
206.229.254.15 !root JoNAThaNHasSELl
206.229.254.5  !root FoOBaR
```

The hints File

Progressing along with the FreeRADIUS setup you will come to the */etc/raddb/hints* file. This file can be used to provide "hints" to the RADIUS server about how to provision services for a specific user based on how his login name is constructed. For example, when you've configured your default service to be a SLIP connection, then a SLIP connection will be set up if a user logs in with her standard username (e.g., *meis*). However, if that same user wanted a PPP connection, she could alter her username to be *Prneis*, and the RADIUS server (knowing about that convention from the */etc/raddb/hints* file) would set up a PPP connection for her. Suffixes on the end of the username work in the same way. More on the hints file will be provided later in the chapter. You shouldn't need to edit this file initially since we're just testing, but if you'd like to check it out, a sample */etc/raddb/hints* file looks like this:

```
DEFAULT Prefix = "P", Strip-User-Name = Yes
        Hint = "PPP",
        Service-Type = Framed-User,
        Framed-Protocol = PPP

DEFAULT Prefix = "S", Strip-User-Name = Yes
        Hint = "SLIP",
        Service-Type = Framed-User,
        Framed-Protocol = SLIP

DEFAULT Suffix = "P", Strip-User-Name = Yes
        Hint = "PPP",
        Service-Type = Framed-User,
        Framed-Protocol = PPP

DEFAULT Suffix = "S", Strip-User-Name = Yes
        Hint = "SLIP",
        Service-Type = Framed-User,
        Framed-Protocol = SLIP
```

The huntgroups File

Let's move on to the */etc/raddb/huntgroups* file, where you define certain huntgroups. *Huntgroups* are sets of ports or other communication outlets on RADIUS client equipment. In the case of FreeRADIUS, a huntgroup can be a set of ports, a specific piece of RADIUS client equipment, or a set of calling station IDs that you want to separate from other ports.

You can filter these defined huntgroups to restrict their access to certain users and groups and match a username/password to a specific huntgroup, possibly to assign a static IP address. You define huntgroups based on the IP address of the NAS and a port range. (Keep in mind that a range can be anywhere from 1 to the maximum number of ports you have.) To configure this file, you first specify the terminal servers in each POP. Then, you configure a stanza that defines the restriction and the criteria that a potential user must satisfy to pass the restriction. That criteria is most likely a Unix username or groupname.

Again, you shouldn't have to configure this file to get basic functionality enabled for testing; if you would like to peruse the file and its features, however, I've provided a sample */etc/raddb/huntgroups* file. It's for an ISP with a POP in Raleigh, North Carolina that wants to restrict the first five ports on its second of three terminal servers in that POP to only premium customers:

```
raleigh         NAS-IP-Address == 192.168.1.101
raleigh         NAS-IP-Address == 192.168.1.102
raleigh         NAS-IP-Address == 192.168.1.103
premium         NAS-IP-Address == 192.168.1.101, NAS-Port-Id == 0-4
                Group = premium,
                Group = staff
```

The users File

FreeRADIUS allows several modifications to the original RADIUS server's style of treating users unknown to the *users* file. In the past, if a user wasn't configured in the *users* file, the server would look in the Unix password file, and then deny him access if he didn't have an account on the machine. There was only one default entry permitted. In contrast, FreeRADIUS allows multiple default entries and can "fall through" each of them to find an optimal match. The entries are processed in the order they appear in the users file, and once a match is found, RADIUS stops processing it. The `Fall-Through = Yes` attribute can be set to instruct the server to keep processing, even upon a match. The new FreeRADIUS *users* file can also accept spaces in the username attributes, either by escaping the space with a backslash (\) or putting the entire username inside quotation marks. Additionally, FreeRADIUS will not strip out spaces in usernames received from PortMaster equipment.

Since we won't add any users to the *users* file for our testing purposes, FreeRADIUS will fall back to accounts configured locally on the Unix machine. However, if you want to add a user to the *users* file to test that functionality, a sample */etc/raddb/users* file looks like this:

```
steve  Auth-Type := Local, User-Password == "testing"
       Service-Type = Framed-User,
       Framed-Protocol = PPP,
       Framed-IP-Address = 172.16.3.33,
       Framed-IP-Netmask = 255.255.255.0,
       Framed-Routing = Broadcast-Listen,
       Framed-Filter-Id = "std.ppp",
       Framed-MTU = 1500,
       Framed-Compression = Van-Jacobsen-TCP-IP
DEFAULT Service-Type == Framed-User
       Framed-IP-Address = 255.255.255.254,
       Framed-MTU = 576,
       Service-Type = Framed-User,
       Fall-Through = Yes
DEFAULT Framed-Protocol == PPP
       Framed-Protocol = PPP,
       Framed-Compression = Van-Jacobson-TCP-IP
```

There will be much more about the *users* file later in this chapter.

The radiusd.conf File

This file is much like Apache's *httpd.conf* file in that it lists nearly every directive and option for the basic functionality of the FreeRADIUS product. You will need to edit the Unix section of this file to make sure that the locations of the *passwd*, *shadow*, and *group* files are not commented out and are correct. FreeRADIUS needs these locations to start up. The appropriate section looks like this:

```
        unix {
         (some content removed)
                #  Define the locations of the normal passwd, shadow, and
                #  group files.
                #
                #  'shadow' is commented out by default, because not all
                #  systems have shadow passwords.
                #
                #  To force the module to use the system passwd fnctns,
                #  instead of reading the files, comment out the 'passwd'
                #  and 'shadow' configuration entries.  This is required
                #  for some systems, like FreeBSD.
                #
                passwd = /etc/passwd
                shadow = /etc/shadow
                group = /etc/group
         (some content removed)
}
```

I will cover the *radiusd.conf* file in more detail later in this chapter.

With that done, it's now time to launch the *radiusd* daemon and test your setup. Execute *radiusd* from the command line; it should look similar to this:

```
radius:/etc/raddb # radiusd
radiusd: Starting - reading configuration files ...
radius:/etc/raddb #
```

If you receive no error messages, you now have a functional FreeRADIUS server. Congratulations!

Testing the Initial Setup

Once you have FreeRADIUS running, you need to test the configuration to make sure it is responding to requests. FreeRADIUS starts up listening, by default, on the port specified either in the local */etc/services* file or in the port directive in *radiusd.conf*. While RFC 2138 defines the standard RADIUS port to be 1812, historically RADIUS client equipment has used port 1645. Communicating via two different ports is obviously troublesome, so many users start the FreeRADIUS daemon with the `-p` flag, which overrides the setting in both the */etc/services* file and anything set in *radiusd.conf*. To do this, run the following from the command line:

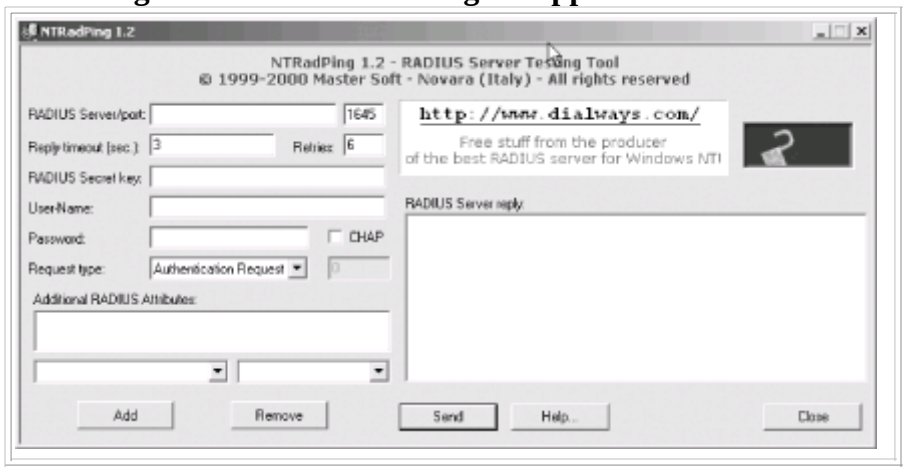
```
radius:/etc/raddb # radiusd -p 1645
radiusd: Starting - reading configuration files ...
```

```
radius:/etc/raddb #
```

The server is now running; it is listening for and accepting requests on port 1645.

So, what is an easy way to test your configuration to see if it functions properly? It's easier than you might think, in fact. MasterSoft, Inc. has released a Windows desktop RADIUS server testing tool called NTRadPing, available at http://www.dialways.com. The latest version as of this writing is 1.2, and it's a freeware tool. Download and install this utility on a Windows machine, and then run it. The initial application window should look much like Figure 5-1.

Figure 5-1.The NTRadPing 1.2 application window



To do a quick test, follow these steps:

1. Enter the IP address of your FreeRADIUS machine in the RADIUS Server/port box, and then the port number in the adjacent box. For this example, I've used IP address `192.168.1.103` and port 1645.

2. Type in the secret key you added in */etc/raddb/clients* for this Windows console machine. For this example, I used the key "testing123."

3. In the User-Name field, enter root, and in the Password field, enter the root password for your FreeRADIUS machine.

4. Select *Authentication Request* from the Request Type drop-down list box.

5. Click Send.

If your server is working properly, and you entered a valid root password, you should see the reply in the RADIUS Server reply box to the right of the NTRadPing window. You should see something like:

```
Sending authentication request to server 192.168.1.103:1645
Transmitting packet, code=1 id=1 length=47
Received response from the server in 15 milliseconds
Reply packet code=2 id=1 length=20
Response: Access-Accept
------------------attribute dump----------------------
```

Now, change the password for root inside NTRadPing to something incorrect, and resend the request. You should get an `Access-Reject` message much like the one shown here:

```
Sending authentication request to server 192.168.1.103:1645
Transmitting packet, code=1 id=3 length=47
No response from server (timed out), new attempt (#1)
Received response from the server in 3516 milliseconds
Reply packet code=3 id=3 length=20
Response: Access-Reject
------------------attribute dump----------------------
```

Next, you'll need to test accounting packets. The old standard for RADIUS accounting used port 1646. Change the port number in NTRadPing accordingly, and select *Accounting Start* from the Request Type drop-down list box. Make sure the root password is correct again, and send your request along. The response should be similar to the following:

```
Sending authentication request to server 192.168.1.103:1646
Transmitting packet, code=4 id=5 length=38
Received response from the server in 15 milliseconds
Reply packet code=5 id=5 length=20
Response: Accounting-Response
------------------attribute dump----------------------
```

Finally, stop that accounting process by changing the Request Type box selection to *Accounting Stop* and resending the request. You should receive a response like this:

```
Sending authentication request to server 192.168.1.103:1645
Transmitting packet, code=4 id=6 length=38
Received response from the server in 16 milliseconds
Reply packet code=5 id=6 length=20
Response: Accounting-Response
------------------attribute dump----------------------
```

If you received successful responses to all four ping tests, then FreeRADIUS is working properly. If you haven't, here's a quick list of things to check:

- Is FreeRADIUS running? Use

  ```
  ps -aux | grep radiusd
  ```

to determine whether the process is active or not.

- Is FreeRADIUS listening on the port you're pinging? If necessary, start *radiusd* with an explicit port, i.e.,

  ```
  radiusd -p 1645
  ```

- Have you added your Windows console machine to the list of authorized clients that can hit the RADIUS server? Do this in the */etc/raddb/clients* file.

- Are you using the correct secret key? This as well is configured in the */etc/raddb/clients* file.

- Have you double-checked the locations of the *group*, *passwd*, and *shadow* files inside the *radiusd.conf* file? These locations are specified in the Unix section. Make sure they're not commented out and that the locations are correct.

- Can FreeRADIUS read the *group*, *passwd*, and *shadow* files? If you're running FreeRADIUS as root, this shouldn't be a problem, but check the permissions on these files to make sure the user/group combination under which *radiusd* is running can access those files.

- Is there any port filtering or firewalling between your console machine and the RADIUS server that is blocking communications on the ping port?

- Is the daemon taking a long time to actually start up and print a ready message (if you're running in debugging mode)? If so, your DNS configuration is broken.

To assist in diagnosing your problem, you may want to try running the server in debugging mode. While operating in this mode, FreeRADIUS outputs just about everything it does, and by simply sifting through all of the messages it prints while running, you can identify most problems.

To run the server in debugging mode, enter the following on the command line to start *radiusd*:

```
radiusd -sfxxyz -l stdout
```

It should respond with a ready message if all is well. If it doesn't, then look at the error (or errors as the case may be) and run through the checklist above.

You can also check the configuration of FreeRADIUS using the following command:

```
radiusd -c
```

This command checks the configuration of the RADIUS server and alerts you to any syntax errors in the files. It prints the status and exits with either a zero, if everything is correct, or a one if errors were present. This command is also useful when you're updating a

production server that cannot be down: if there were a syntax error in the files, *radiusd* would fail to load correctly, and downtime would obviously ensue. With the check capability, this situation can be avoided.

In-depth Configuration

At this point, you've compiled, installed, configured, started, and tested a simple FreeRADIUS implementation that is functional. However, 99.5% of the RADIUS/AAA implementations around the world are just not that simple. In this section, I'll delve into the two major configuration files and discuss how to tweak, tune, customize, and effect change to the default FreeRADIUS installation. In Chapter 6, I'll discuss advanced topics, such as pluggable authentication module (PAM) support, integration with MySQL, LDAP usage, and other topics.

Configuring radiusd.conf

radiusd.conf file is the central location to configure most aspects of the FreeRADIUS product. It includes configuration directives as well as pointers and two other configuration files that may be located elsewhere on the machine. There are also general configuration options for the multitude of modules available now and in the future for FreeRADIUS. The modules can request generic options, and FreeRADIUS will pass those defined options to the module through its API.

Before we begin, some explanation is needed of the operators used in the statements and directives found in these configuration files. The = operator, as you might imagine, sets the value of an attribute. The := operator sets the value of an attribute and overwrites any previous value that was set for that attribute. The == operator compares a state with a set value. It's critical to understand how these operators work in order to obtain your desired configuration.

In this chapter, I'll look at several of the general configuration options inside *radiusd.conf*. Some of the more advanced directives in this file will be covered in Chapter 6.

pidfile

This file contains the process identification number for the *radiusd* daemon. You can use this file from the command line to perform any action to a running instance of FreeRADIUS. For example, to shut FreeRADIUS down without any protests, issue:

```
kill -9 `cat /var/run/radiusd.pid'
```

Usage:

```
pidfile = [path]
```

Suggestion:

```
pidfile = ${run_dir}/radiusd.pid
```

user and group

These options dictate under what user and group *radiusd* runs. It is not prudent to allow FreeRADIUS to run under a user and group with excessive permissions. In fact, to minimize the permissions granted to FreeRADIUS, use the user and group "nobody." However, on systems configured to use shadow passwords, you may need to set the user to "nobody" and the group to "shadow" so that *radiusd* can read the *shadow* file. This is not a desirable idea. On some systems, you may need to set both the user and group to "root," although it's clear why that is an even worse idea.

Usage:

```
user = [username]; group = [groupname]
```

Suggestion:

```
user = nobody; group = nobody
```

max_request_time

This option specifies the maximum number of seconds a request will be processed by FreeRADIUS. If the handling of a request takes longer than this threshold, the process can be killed off and an `Access-Reject` message returned. This value can range from 5 to 120 seconds.

Usage:

```
max_request_time = 30
```

Suggestion:

```
max_request_time = 60
```

delete_blocked_requests

This directive is paired with the `max_request_time` directive in that it controls when requests that exceed the time threshold should be killed. Most of the time, this value should be set to "no."

Usage:

```
delete_blocked_requests = [yes/no]
```

Suggestion:

```
delete_blocked_requests = no
```

cleanup_delay

When FreeRADIUS sends a reply to RADIUS client equipment, it generally caches that request internally for a few seconds to ensure that the RADIUS client will receive the message (sometimes network problems, offline servers, and large traffic loads might prevent the client from picking up the packet). The client receives a quick reply on its prompting for a second copy of the packet, since the internal cache mechanism for FreeRADIUS is much quicker than processing the request again. This value should be set between 2 and 10: this range is the happy medium between treating every request as a new request and caching so many processed requests that some new requests are turned away.

Usage:

```
cleanup_delay = [value]
```

Suggestion:

```
cleanup_delay = 6
```

max_requests

This directive specifies the maximum number of requests FreeRADIUS will keep tabs on during operation. The value starts at 256 and scales with no upper limit, and ideally this is set at the number of RADIUS clients you have multiplied by 256. Setting this value too high causes the server to eat up more system memory, while setting it too low causes a delay in processing new requests once this threshold has been met. New requests must wait for the cleanup delay period to finish before they can be serviced.

Usage:

```
max_requests = [value]
```

Suggestion:

```
max_requests = [256 * x number of clients]
```

bind_address

This directive specifies the address under which *radiusd* will accept requests and reply to them. The "address" can be an IP address, fully qualified domain name, or the `*` wildcard character (to instruct the daemon to listen on all interfaces).

Usage:

```
bind_address = [value]
```

Suggestion:

```
bind_address = *
```

port

This setting instructs FreeRADIUS to listen on a specific port. While the RADIUS RFC specifies that the official RADIUS port is 1812, historically NAS equipment and some RADIUS servers have used port 1645. You should be aware of the port your implementation uses. While you can specify a certain port here, you can also instruct *radiusd* to use the machine's */etc/services* file to find the port to use. Additionally, using the `-p` switch when executing *radiusd* will override any port setting provided here.

Usage:

```
port = [value]
```

Suggestion:

```
port = 1645
```

hostname_lookups

This directive tells FreeRADIUS whether to look up the canonical names of the requesting clients or simply log their IP address and move on. Much like with Apache, DNS queries take a long time and, especially on highly loaded servers, can be a detriment to performance. Turning this option on also causes *radiusd* to block the request for 30 seconds while it determines the CNAME associates with that IP address. Only turn this option on if you are sure you need it.

Usage:

```
hostname_lookups = [yes/no]
```

Suggestion:

```
hostname_lookups = no
```

allow_core_dumps

This directive determines whether FreeRADIUS should dump to core when it encounters an error or simply silently quit with the

error. Only enable this option if you're developing for FreeRADIUS or attempting to debug a problem with the code.

Usage:

```
allow_core_dumps = [yes/no]
```

Suggestion:

```
allow_core_dumps = no
```

regular and extended expressions

This set of controls configures regular and extended expression support. Realistically, you shouldn't need to alter these as they're set when running the *./configure* command upon initial install.

Usage:

```
regular_expressions = [yes/no]; extended_expressions = [yes/no]
```

Suggestion:

```
regular_expressions = yes; extended expressions = yes
```

log

These directives control how access to and requests of the FreeRADIUS server are logged. The `log_stripped_names` control instructs FreeRADIUS whether to include the full `User-Name` attribute as it appeared in the packet. The `log_auth` directive specifies whether to log authentication requests or simply carry them out without logging. The `log_auth_badpass` control, when set to yes, causes *radiusd* to log the bad password that was attempted, while the `log_auth_goodpass` logs the password if it's correct.

Usage:

```
log_stripped_names = [yes/no]; log_auth = [yes/no];
log_auth_badpass = [yes/no]; log_auth_goodpass = [yes/no]
```

Suggestion:

```
log_stripped_names = no; log_auth = yes;
log_auth_badpass = yes; log_auth_goodpass = no
```

lower_user and lower_pass

To eliminate case problems that often plague authentication methods such as RADIUS, the FreeRADIUS developers have included a feature that will attempt to modify the `User-Name` and `User-Password` attributes to make them all lowercase; this is done either before an authentication request, after a failed authentication request using the values of the attributes as they came, or not at all.

Clearly setting the `lower_user` directive to `after` makes the most sense: it adds processing time to each request, but unless this particular machine normally carries a high load, the reduced troubleshooting time is worth the extra performance cost. However, a secure password often makes use of a combination of uppercase and lowercase letters, so security dictates leaving the password attribute alone.

Usage:

```
lower_user = [before/after/no]; lower_pass = [before/after/no]
```

Suggestion:

```
lower_user = after; lower_pass = no
```

nospace_user and nospace_pass

Much like the `lower_user` and `lower_pass` controls, these directives preprocess an `Access-Request` packet and ensure that no spaces are included. The available options are the same: `before`, `after`, or `no`. Again, the most obvious choice is to set `nospace_user` to `after` to save helpdesk time. Some administrators have a tendency to not allow spaces in passwords; if this is the case, set `nospace_pass` to `before` (since there is a system-wide policy against spaces in passwords, testing a request as-is is not required).

Usage:

```
nospace_user = [before/after/no]; nospace_password = [before/after/no]
```

Suggestion:

```
nospace_user = after; nospace_password = before
```

Configuring the users File

The *users* file, located at */etc/raddb/users*, is the home of all authentication security information for each user configured to access the system. Each user has an individual stanza, or entry. The file has a standard format for each stanza:

1. The first field is the username for each user, up to 253 characters.

2. On the same line, the next criteria are a list of required authentication attributes such as protocol type, password, and port

number.

3. Following the first line, each user has a set of defined characteristics that allow FreeRADIUS to provision a service best for that user. These characteristics are indented under the first line and separated into one characteristic per line. For example, you might find a Login-Host entry, a dial-back configuration, or perhaps PPP configuration information.

The *users* file also comes with a default username of--you guessed it--DEFAULT, which is generally the catchall configuration. That is to say, if there is no explicit match for a particular user, or perhaps the attribute information for a user is incomplete, *radiusd* will configure the session based on the information in the DEFAULT entry.

FreeRADIUS processes this file in the order in which the entries are listed. When information received from the RADIUS client equipment matches an entry in the *users* file, FreeRADIUS stops processing and sets the service up based on that *users* file entry. However, you can alter this behavior by setting the `Fall-Through` attribute to yes in an entry. When *radiusd* encounters a positive fall-through entry, it will continue processing the *users* file and then select the best match for the particular session. The DEFAULT user can also have a `Fall-Through` attribute, which means you can have multiple DEFAULT entries for various connection scenarios.

If you don't want to issue a password for each user via their entry in the *users* file, then simply set `Auth-Type := System` on the first line for each user. FreeRADIUS will then query the system password database for the correct password, which saves some administrative headache.

A sample complete entry

The following is a complete entry for the user *jhassell*, dialing into a NAS server using PPP. Note that (a) there is no `Fall-Through` attribute set, so FreeRADIUS will stop processing when it encounters this entry, and (b) no DEFAULT entry will be used to add attribute information to this connection:

```
jhassell    Auth-Type := System
            Service-Type = Framed-User,
            Framed-Protocol = PPP,
            Framed-IP-Address = 192.168.1.152,
            Framed-IP-Netmask = 255.255.255.0,
            Framed-Routing = Broadcast-Listen,
            Framed-Filter-Id = "20modun",
            Framed-MTU = 1500,
            Framed-Compression = Van-Jacobsen-TCP-IP
```

Next, here's a complete entry for the user Anna Watson. She has a space in her user-name and she also has a password specified in her entry. She also gets a positive fall-through so that she can use some of the DEFAULT user's attributes with her connection:

```
"Anna Watson"    Auth-Type := Local, User-Password == "yes123"
                 Reply-Message = "Hello, %u"
                 Service-Type = Framed-User,
```

```
                Framed-Routing = Broadcast-Listen,
                Framed-Filter-Id = "20modun",
                Fall-Through = Yes
```

DEFAULT entries

These DEFAULT user configurations match with all usernames that can get to them (i.e., the individual users must have a positive `Fall-Through` attribute). Recall from the earlier discussion that DEFAULT entries may also have `Fall-Through` attributes.

First, let's make sure that all users are checked against the system password file unless they have a password explicitly assigned in the entry.

```
DEFAULT     Auth-Type := System
            Fall-Through = Yes
```

Now, include a DEFAULT entry for all users connecting via a framed protocol, such as PPP or SLIP. Note that I tell the RADIUS client to assign the IP address via the `Framed-IP-Address` attribute's value (see Chapter 3 for details).

```
DEFAULT     Service-Type = Framed-User
            Framed-IP-Address = 255.255.255.254,
            Framed-MTU = 576,
            Service-Type = Framed-User,
            Fall-Through = Yes
```

Finally, set the DEFAULT entry for PPP users. I've already told FreeRADIUS to assign framed protocol users with a dynamic IP address, so all I need to do is set the compression method and explicitly designate PPP as the framed protocol for this default.

```
DEFAULT     Framed-Protocol == PPP
            Framed-Protocol = PPP,
            Framed-Compression = Van-Jacobsen-TCP-IP
```

If a user attempts to connect and matches neither any of the explicit user entries nor any of the DEFAULT entries, then he will be denied access. Notice that with the last DEFAULT entry, `Fall-Through` isn't set: this ensures the user is kicked off if he doesn't match any of the scenarios.

Prefixes and suffixes

You can use prefixes and suffixes appended to the user name to determine what kind of service to provision for that particular connection. For example, if a user adds *.shell* to their username, you add the following DEFAULT entry to the users file to provision a shell service for her. FreeRADIUS authenticates her against the system password file, telnets to your shell account machine, and logs her in.

```
DEFAULT    Suffix == ".shell", Auth-Type := System
           Service-Type = Login-User,
           Login-Service = Telnet,
           Login-IP-Host = shellacct1.rduinternet.com
```

Similarly, you can set up an entry in the users file where if a user connects with a prefix of "s.", then you can provision SLIP service for him. FreeRADIUS can authenticate him against the system passwords, and then fall through to pick up the SLIP attributes from another DEFAULT entry. Here is an example:

```
DEFAULT    Prefix == "s.", Auth-Type := System
           Service-Type = Framed-User,
           Framed-Protocol = SLIP,
           Fall-Through = Yes
```

Using RADIUS callback

The callback feature of the RADIUS protocol is one of the most interesting and useful security measures that you, as an administrator, can enforce. You can configure FreeRADIUS to call a specific user back via his individual entry in the users file. (Of course, you could make a DEFAULT entry that calls every user back, but the application of that technique is more limited and requires many more resources than a standard implementation.) The following is an example of a callback configuration for user *rneis*: she dials in, is then called back, is authenticated, and then given a session on the shell account machine.

```
rneis    Auth-Type := System
         Service-Type = Callback-Login-User,
         Login-Service = Telnet,
         Login-IP-Host = shellacct1.rduinternet.com,
         Callback-Number = "9,1-919-555-1212"
```

Completely denying access to users

You can set up a specific user entry to deny access to him. For example, you may have an automated script that takes input from your billing system (a list of usernames that have not paid their bills, possibly) and re-writes user entries to deny access. They would write something like the following, for the user *aslyter*:

```
aslyter    Auth-Type := Reject
           Reply-Message = "Account disabled for nonpayment."
```

Alternatively, you could also set up a group on your system called "suspended," and FreeRADIUS could detect whether an individual username was contained within that group and reject access as necessary. To do this, create a DEFAULT entry much like the following:

```
DEFAULT    Group == "suspended", Auth-Type := Reject
           Reply-Message = "Account suspended for late payment."
```

Troubleshooting Common Problems

In this section, I'll take a look at some of the most frequently occurring problems with a new FreeRADIUS setup and how to fix them.

Linking Errors When Starting FreeRADIUS

If you receive an error similar to the following:

```
Module: Loaded SQL
rlm_sql: Could not link driver rlm_sql_mysql: file not found
rlm_sql: Make sure it (and all its depend libraries!) are in the search path
radiusd.conf[50]: sql: Module instantiation failed.
```

It means that some shared libraries on the server are not available. There are a couple of possible causes from this.

First, the libraries that are needed by the module listed in the error messages couldn't be found when FreeRADIUS was being compiled. However, if a static version of the module was available, it was built at compile time. This would have been indicated with very prominent messages at compile time.

The other cause is that the dynamic linker on your server is not configured correctly. This would result in the libraries that are required being found at compile time, but not run time. FreeRADIUS makes use of standard calls to link to these shared libraries, so if these calls fail, the system is misconfigured. This can be fixed by telling the linker where these libraries are on your system, which can be done in one of the following ways:

- Write a script that starts FreeRADIUS and includes the variable *LD_LIBRARY_PATH*. This sets the paths where these libraries can be found.
- If your system allows it, edit the */etc/ld.so.conf* file and add the directory containing the shared libraries to the list.
- Set the path to these libraries inside *radiusd.conf* using the *libdir* configuration directive. The *radiusd.conf* file has more details on this.

Incoming Request Passwords Are Gibberish

Gibberish is usually indicative of an incorrectly formed or mismatched shared secret, the phrase shared between the server and the RADIUS client machine and used to perform secure encryption on packets. To identify the problem, run the server in debugging mode, as described previously. The first password printed to the console screen will be inside a RADIUS attribute (e.g., `Password = "rneis\dfkjdf7482odf"`) and the second will be in a logged message (e.g., `Login failed [rneis/dfkjdf7482odf]`). If the data after the slash is gibberish--ensure it's not just a really secure password--then the shared secret is not consistent between the server and

the RADIUS client. This may even be due to hidden characters, so to be completely sure both are the same, delete and re-enter the secret on both machines.

The gibberish may also result from a shared secret that is too long. FreeRADIUS limits the secret length to 16 characters, since some NAS equipment has limitations on the length of the secret yet don't make it evident in error logs or the documentation.

NAS Machine Ignores a RADIUS Reply

You may be seeing duplicate accounting or authentication requests without accompanying successful user logins. In this case, it's likely that you have a multi-homed RADIUS server, or at least a server with multiple IP addresses. If the server receives a request on one IP address, but responds with a different one, even if the reply comes from the machine for which the original packet was destined, the NAS machine will not accept it. To rectify this, launch FreeRADIUS with the `-i` command-line switch, which binds the daemon to one specific IP address.

CHAP Authentication Doesn't Work Correctly

If PAP authentication works normally, but users authenticating with the CHAP protocol receive errors and denials, you do not have plain text passwords in the users file. CHAP requires this, while PAP can take passwords from the system or from any other source. For each user who needs CHAP authentication, you must add the `Password = changeme` check item to his individual entry, of course changing the value of the password as appropriate.

Some people may say using CHAP is much more secure, since the user passwords are not transmitted in plain text over the connection between the user and the NAS. This is simply not true in practice. While hiding the password during transmission is beneficial, the CHAP protocol requires you to leave plain text passwords sitting in a file on a server, completely unencrypted. Obviously, it's much more likely that a cracker will gain access to your RADIUS server, grab the *users* file with all of these plainly available passwords, and wreak havoc and harm on your network than it is that the same cracker would intercept *one* user's password during the establishment of the connection.

Back to: Sample Chapter Index

Back to: RADIUS

O'Reilly Home | O'Reilly-Partnerbuchhandlungen | Bestellinformationen | Kontakt
Datenschutz | Über O'Reilly