Chapter 11 - Debugging

This chapter describes debugging FreeRADIUS servers. It discusses:

- Debugging Overview
- General methodology/principles
- Debugging live systems
- NAS issues

11.0 Debugging Overview

This chapter begins with a discussion of the methodology behind a good debugging process, which is followed by a series of common problems and recommended solutions. The information given here applies to most versions of the server, but the message contents are taken from version 2.1.10.

When configuring a server, debugging is the single most difficult task. Most real-world systems involve complex interactions with business rules, databases, proxies, and multiple kinds of NAS equipment. The sheer number of moving pieces involved in a RADIUS system means that configurations can be fragile, poorly understood, and prone to misbehavior. Knowing how to solve these problems is as important as knowing how to create the configuration in the first place.

It is important to keep consistent methodology in mind when reading the solutions given below. In many cases, applying a consistent methodology will lead you to the solution no matter what the problem.

There are a few common mistakes that are made when configuring the server. They are listed below:

- Making a large number of changes to the configuration at once or without testing. The methods outlined in this chapter should be used instead.
- Using `Password = "password"` or `User-Password = "password"` instead of `Cleartext-Password := "password"`. The `User-Password` method of configuring the "known good" password has been deprecated for many years; it should not be used in any production system.
- Setting `Auth-Type` to a specific value is nearly always wrong. Setting the `Auth-Type` is only done when the aim is to prevent other authentication methods from working. Do not set it; simply let the server figure out what to do. The decisions made by the server will be correct for simple configurations.
- Following third-party web sites or documentation. These are often years out of date and give wildly incorrect advice. Administrators should instead use this technical manual, the documentation that ships with the server, the wiki (wiki.freeradius.org), or the answers to questions on the freeradius-users mailing list.
- Failing to use the correct certificates for EAP. The methods outlined above should be used to create and test EAP with certificates.
- Failing to run the server in debugging mode. There are relatively few reasons for not running the server in debugging mode, and many more reasons for doing so. The only reason not to use debugging mode is when employing a production server.

11.1 General Methodology/Principles

The primary tool in debugging is the use of a systematic method. These methods may seem slow and laborious, but experience has shown that they are the fastest and easiest paths to a solution. The alternative method that is commonly used and that does not yield good results is "trial and error", wherein random changes are made in the hope that something will start working.

The first step of a systematic method is to use revision control on the configuration files.

11.1.1 Revision Control

Revision control should always be used on the configuration files. Each "checked in" revision should be a "known working" configuration. There should generally be no difference between the "on disk" configuration and the configuration in the revision control system.

When a change to the configuration is required, the current configuration should be double-checked to ensure it works and is identical to the "checked in" version. Local modifications can then be made with the confidence that if something goes wrong, the modified configuration can be discarded and replaced with a previous one that is known to work.

Each commit into the revision control system should be an independent configuration change that has been proven through testing to work no matter how minor. These commits should be performed even if the configuration has not reached the desired goal. Keeping the commits small yields a "ratchet" effect to configuration changes. Each change moves ahead one small step, and ensures that any backwards steps are also small.

The `git` revision control system is the system used on the main `freeradius.org` site; instructions on its use are thus outlined below. Other revision control systems can also be used.

The first step is to create the `git` repository. This should be done using the default configuration of the server. The default configuration is the one that existed before any local changes were made. If the current system has changes from the default configuration, just follow these steps anyways.

1. Create the local repository:

```
$ cd /etc/raddb
$ git init
```

2. Add all of the files to the repository:

```
$ git add .
$ git commit -a -m "Initial commit"
```

3. Make changes to local files.:

```
$ vi radiusd.conf
```

4. Commit the changes, giving a short but descriptive change message:

```
$ git commit -a "Changed ABC to XYZ"
```

If the changes break something, revert to the previous configuration using the following command:

```
$ git checkout -f .
```

To add a new file that was not previously tracked:

```
$ git add raddb/foo
$ git commit -m "Added file foo" raddb/foo
```

To see what changes have been made over time:

```
$ git log
```

To see what has changed between the last commit and the files in the current directory:

```
$ git diff
```

The few commands shown above do not do justice to all of the capabilities of `git`. They can, however, be used to perform simple tracking and revision control.

11.1.2 Divide and Conquer

After revision control, the next most important debugging method is divide and conquer. This method involves splitting the problem into multiple independent pieces and solving each one individually. When each individual piece has been solved, the partial solutions are gradually combined to obtain a whole.

This method seems simple when the steps are given as in **Chapter 5 - Basic Authentication Methods** on page 23, where we tested PAP authentication before proceeding to the MS-CHAP or EAP methods. The hard part is knowing how to divide the problem into independent pieces. Following the questions below in order is a good starting point to determining the nature of the problem and its solution. The questions are ordered by importance, in that earlier questions must be answered before later questions are asked.

Network Issues

Is the server receiving packets?

If the server is not receiving packets:

- Check that the NAS configuration has the correct IP address and port for the server.
- Check that the server is listening on the correct IP address and port.
- Check that packets (ICMP "pings" or similar) can be sent from the NAS to the server and that the server receives them.
- Use wireshark to check that the machine running the server is receiving packets.

Request Packets

Is the NAS sending the expected data in the packets?

If the NAS is not sending the expected data in the packets:

- Update or change the NAS configuration to send the expected data. This suggestion applies to both accounting and authentication packets but is generally more applicable to accounting. If the server is not logging the correct accounting information, the reason is usually that the NAS is not sending that information to the server.

Shared Secret

Is the shared secret correct?

If the shared secret is not correct, the server will usually print a message saying so:

- Reenter the secret on the NAS and the server.

Databases

Can the server read the user's "known good" password from a database (e.g. flat files, /etc/passwd, LDAP, SQL, etc.)

If the server cannot read the user's "known good" password:

- Use `radtest` to send test packets to debug the problem. There is no need to use a real NAS for debugging database issues.

Is the database slow?

In normal circumstances, database ''SELECT''s and ''INSERT''s should be nearly instantaneous. If they take seconds or minutes, then the database needs to be optimized:

- Add indexes or remove unnecessary data from the database.

Certificates

For EAP methods, does the client finish the EAP conversation? That is, does the server send an `Access-Accept`?

If the server does not send an `Access-Accept`, then the client is rejecting the servers certificate:

- The server certificate or CA certificate needs to be added to the client machine.

Reply Packet

Is the server sending back the correct attributes in the `Access-Accept`?

If the server is not sending back the correct attributes to the NAS:

- Change the server configuration until it sends back an `Access-Accept` containing the correct attributes.

If the server's reply does contain the desired attributes, then:

- Examine the NAS configuration and documentation to see why the NAS is ignoring the servers response.

Read the Debug Log

Is the server's response wrong or unexpected?

If the servers response is wrong or unexpected:

- Run it in debugging mode to see what it is doing.
- Try to narrow down the problem to one user or switch port.
- Try to define a minimal test case (packets, attribute contents) that will reproduce the problem.
- Use `radclient` and `eapol_test` to test the problem instead of using a real NAS.

Summary

In summary, there are only a few moving pieces in RADIUS. The user (supplicant, etc.), the NAS (or AP), the RADIUS server, and the databases. When each piece is checked independently, the source of the problem can be quickly identified.

11.1.3 Test Systems

Every non-trivial RADIUS installation should have a test system where changes are tested before being deployed in the production network.

The test system can be a VMWare image, or it can be separate machine. It is best if the test system is not one of the "live" production servers. The potential for catastrophic errors is just too large.

Debugging a live system is a recipe for disaster: users logging in at the same time as the debugging packets are being sent make it difficult to tell what the server is doing, and changes to the system made during the debugging process will affect users logging in and may result in users being denied access. In addition, very little is worse than having your production servers go down due to a configuration mistake.

The test system should be set up to be nearly identical to the real system. That is, the test system should have the same operating system, the same OS version, libraries, and packages as the live system. If the systems are not the same, then the test system will not match the production system; the tests may pass on the test system but fail on the production system.

11.1.4 Test Packets

Test packets are used to generate traffic, which looks "real" but doesn't involve an actual NAS. They let the system be tested as if the NAS was there, but without requiring the actual NAS.

The `radsniff` program may be used to grab "live" packet samples from the existing network. These packets can be used to generate test packets for the server.

The main benefit of `radsniff` is that it is aware of the RADIUS protocol and can thus perform filtering on the RADIUS packets. Even if the packets have been captured by another tool, `radsniff` should be used to filter through the capture file, to print out only the relevant requests and responses. Otherwise,

Tools such as `tcpdump` or `wireshark` can also be used if `radsniff` is not installed on the system.

11.1.5 EAP Testing

If you are using EAP methods, the best test tool is `eapol_test`. Complete instructions can be found in Section **6.2 Testing With eapol_test** on page 46.

11.2 Debugging Live Systems

Running the server in debugging mode is not always an option. As of version 2.1.4, the `raddebug` tool can be used to get debug output from a production server. The prerequisites for the `raddebug` tool to work are the following:

- `radmin` must be available in the `PATH`. This is done as part of the default installation.
- The user running `raddebug` must have permission to connect to the server control socket. This usually means running it as `root` or as user `radiusd`.
- The control socket must be enabled. This is done automatically in version 2.1.4 and later, as `raddb/sites-available/control-socket`.
- The control socket must be marked as `mode = rw`. The `control-socket` file must be edited and the relevant line near the bottom of the file must be un-commented.
- The user running `raddebug` must have permission to read and write files in the `logdir` directory, which is usually `/var/log/radiusd`. This usually means running it as `root`, or as user `radiusd`.

11.2.1 Enabling the Control Socket

A sample control-socket file is reproduced below:

```
listen {
      type = control
      socket = ${run_dir}/${name}.sock
      mode = rw
}
```

This configuration tells the server to listen on a control socket and to create the socket in the `run_dir` directory, which is usually `/var/run/radiusd`. The name of the socket file is taken from the name of the server. When multiple server processes are running on the same machine, this configuration lets each server have their own control socket. The final configuration is to set the access mode to `rw`, for "read and write". This lets "raddebug" make changes to the server to enable debugging for different users.

11.2.2 Using raddebug

The simplest way to use `raddebug` is to show the debug output for a particular user. Start the command before the user logs in:

```
$ raddebug -u bob
```

The familiar debugging output should be printed to the screen. Try logging in with a different user name while `raddebug` is running; no debugging output should appear for that session.

It is also possible to produce debugging output for all traffic originating from an NAS using the `-i` command-line option:

```
$ raddebug -i 192.0.2.16
```

More complex conditions are described in the next section.

11.2.3 Generic Conditions

Complex debugging conditions using the `unlang` syntax can be defined using the `-c` command-line option. For example, the following command produces debugging output only for accounting traffic from a particular NAS:

```
$ raddebug -c '((Packet-Type == Accounting-Request) && (Packet-Src-IP-
Address == 127.0.0.1))'
```

The condition should be enclosed in single quotes (') in order to prevent the Unix shell from interpreting the special characters in the condition. See `man unlang`, the `CONDITIONS` section for documentation on the conditions. See section 16.6 - also the other chapter of the book!

Creating a "good" condition can be difficult. If the condition is too broad, then the result will be a lot of debugging output that will be difficult to understand. If the condition is too narrow, then the output will not contain the information you need to solve the problem. In general, using `-u user` should be sufficient.

In some cases, the user name will be "anonymized". This happens most often with EAP in WiMAX or roaming scenarios. In that case, using `-u` will not be possible, as it will either match all users or no user. The solution here is to key off from another field of the packet, such as `Calling-Station-Id`, which should be the MAC address of the user device:

```
$ raddebug -c 'Calling-Station-Id == "00-01-02-03-04-05"'
```

It is possible, however, that the MAC address of the users machine may not be known and that the `Calling-Station-Id` attribute may be in one of many non-standard formats. The only possible solution is then to have the user try to login multiple times while running `raddebug`. The information obtained in one session can be used to narrow down the debug conditions for a following session

11.2.4 Limitations

While `raddebug` is useful, it has a number of limitations.

For example, only one instance can be running at any given time. If a second instance is started while the first one is running, all of the output will go to the second instance, and the first one will stop producing output.

This limitation occurs because the debugging output is placed in an intermediate file, rather than being sent directly to "standard output". There can only be one of these output files at a time. To prevent the file from growing too large, the `raddebug` program is automatically terminated after 60 seconds. If longer periods of time are required to run the program, use the command-line option `-t`. To let it run for sixty seconds, use `-t 60`, or `-t 600` for ten minutes. In most cases, sixty seconds of output should suffice.

When `raddebug` exits, debug mode in the server is disabled and the intermediate file is deleted. This process helps ensure that the debugging output has minimal impact on the production server.

Another limitation is that the administrator running `raddebug` has "write" permission to the running server via the "radmin" command. This means the administrator can change the running configuration, in addition to using `raddebug`.

When a session is marked for debugging, all packets associated with that session produce debug output. This includes EAP "inner tunnel" requests and all proxied packets and responses.

11.3 NAS Issues

An NAS may not always do what the server says. While the common question may be “how can the server be configured so that the user gets the right access”, a better question is “what is needed to send to the NAS so that it gives the user the correct access”.