FireBrick RADIUS servers

This document describes the standard generic RADIUS servers provided for FireBrick LNS operation. These provide RADIUS authentication for L2TP (e.g. broadband) connections, and RADIUS accounting for L2TP.

# Installation / build / environment

The files are provided as C source code, designed to run on a linux install (typically Debian) and work with MariaDB or mysql back end SQL servers.

You will need to ensure you have a suitable build environment, which may be different to the running systems if you wish. Typically, for build, you need to apt install a number of packages: make, gcc, libmariadbclient-dev, libssl-dev, libpopt-dev

We will normally provide a Makefile for you as well, but you will typically need to fine tune it with addition -D settings for defaults. These are described below, and can normally be applied using command line arguments instead if you prefer.

# Tools provided

The key tools provided are :-

| C code provided | |
| --- | --- |
| sqllib.c/h | A library of wrappers for mysql/MariaDB SQL client access  © Andrews & Arnold Ltd |
| radius-defs.h | An include file of common RADIUS AVP IDs as used by the following |
| fb-radius-msg.c | A tool to send CoA messages to FireBrick (e.g. disconnect, change speed, etc, on line connections) |
| fb-radius-auth.c | RADIUS Authentication server, answering authentication queries from FireBrick and providing IP assignments, etc. |
| fb-radius-acct.c | RADIUS Accounting server, handling account from FireBrick recording usage per line, logging each session, historical sessions, daily and hourly usage stats, and blip graph. This also updates authentication table with details of last session (times, speeds, LNS, etc). |

All of the command include the —help option.

# Database

The RADIUS authentication and accounting functions work with a database back end.

## Database access

It is recommended that database access is controlled by use of a .my.cnf file. The tools will, by default, attempt to access one in the users home directory. Note that these tools do not need root access to run. The .my.cnf file is also used by the mysql command line. The file contains a [client] section with keywords user, host, password, database, ssl-ca, ssl-cert, ssl-key which are simple *name*=*value* pairs. The value can be in single quotes if required (may be for passwords).

Note that .my.cnf must be set to only be accessible by the owner, for security reasons.

The default location for the .my.cnf file can be set using -DSQLCONF at build time or —sql-conf command line option.

You can also override the .my.cnf file, or avoid the need for it using command line options —sql-host, —sql-user, —sql-pass, —sql-database. To set up ssl you need to use .my.cnf.

We recommend use of .my.cnf, set using -DSQLCONF if not in the default location as that means the commands can be run with no need to reference any SQL options.

## Database tables

Database table names are up to you, and can be set at compile time, or by command line arguments. These are specified below for each table.

## Database fields

The design allows for you to create a database table that has any additional fields you require - these will not impact the operation of the tools.

The main accounting tables for session, history, daily stats, and blip are created automatically when the accounting server starts if they do not exist, so you do not have to define these tables.

The authentication table, however, needs creating. In most cases the fields in this table are optional. If not specified then they are simply not used. This allows things like the exact method of identifying an authentication entry to depend on the fields present in the table.

Note that presence of fields in the authentication table are checked at start of running the server.

## Authentication table

The authentication table is used by fb-radius-auth to identify attributes to be sent back to the FireBrick, including IP addresses, routing table, graph name (Chargeable User Identity) which is used for logging and accounting, etc.

Fields include those to identity the specific entry; fields to provide response data such as IP addresses; and fields for information on last login/logout. These last fields can be in a separate table if necessary.

For fb-radius-auth this can be preset at compile time as -DTABLE, or at runtime as —sql-table

For fb-radius-acct this can be preset at compile time as -DSTATUS, or at tune time as —status

### Selecting the specific authentication entry

The incoming connection provides a number of key attributes which can be used to identify the authentication entry in the table.

| Information from incoming connection | |
| --- | --- |
| USER\_NAME | This is the login used on the PPP connection, and is normally in the form of a username, an @, and a realm. This is is called *login* and is then broken down in to several components, e.g. if it is fred-1@myrealm   * *login\_prefix* is the initial alphanumeric part, e.g. fred * *login\_suffix* is any part after the prefix and a hyphen, e.g. 1 * *username* is the part before the @, e.g. fred-1 * *realm* is the part after the @, e.g. myrealm |
| TUNNEL\_CLIENT\_ENDPOINT | This is the name from the L2TP incoming config, and is treated as the *carrier* name, e.g. Zen, BT, TT, whatever. |
| CALLING\_STATION\_ID | This is the carrier specific circuit ID provided on the connection, called the *carrier\_circuit\_id* |

How these are used depends on the fields you have defined in the authentication table. The following are checked in order until one yields a result.

1. If there is a *carrier\_circuit\_id* field defined, then this is checked against the calling station ID provided. If *carrier* field is defined, and a tunnel client endpoint exists (and non empty) then this is also checked.
2. If there is a *login\_prefix* field defined then this is checked against the login prefix. If a *login\_suffix* field is defined and there a login suffix in the login, then this is also checked. If there is a *realm* field defined, and a realm is in in the login then this is also checked. If *carrier* field is defined, and a tunnel client endpoint exists (and non empty) then this is also checked.
3. If there is a *username* field defined, this is checked against the login user name. If there is a *realm* field defined, and a realm is in in the login then this is also checked. If *carrier* field is defined, and a tunnel client endpoint exists (and non empty) then this is also checked.
4. If there is a *login* field defined, this is checked against the whole login provided. If *carrier* field is defined, and a tunnel client endpoint exists (and non empty) then this is also checked.

Once an entry is found, if any of the fields above are defined and have a value to be checked against, and that value does not match what was sent, then this is included as a note in the syslog for the connection so as to help identify errors.

Finally, having found an entry, if a *password* field is defined and not NULL, then the password provided is checked, and if that does not match the login fails. If no entry is found the login also fails.

By default, a failed login will reject the connection. **This is a bad idea!** Rejecting a connection causes it immediately retry indefinitely which is load on carrier and LNS and RADIUS server. We strongly recommend building with -DREJECTTABLE defined, e.g. 99, to place any failed logins in to a temporary table. This is preset with a one hour timeout, a 10.10.10.10 IP address, and no routing to other L2TP sessions.

### Selecting Chargeable User Identity (CUI)

The FireBrick has the concept of a Chargeable User Identity for each active connection. These have to be unique as they are the basis of pre-empting a new connection when an old one has not closed for some reason. This is used as the name for the graph of the connection and the basis for accounting information.

The simplest way to do this is when you have a carrier that sends a defined unique circuit ID for every connection that relates to the actual line. This works well for BT when send a BBEUxxxxxxx ID. It is not so good for Talk Talk as they will use a complex circuit ID for ADSL (e.g. THBK-C1#GR-07-214-THBK-5 atm 0/15/0/6:0.38) but a simple FTTCxxxxxx for FTTC lines.

A simple solution, which we recommend, is to define your own circuit IDs. Set the assigned CUI prefix using -DASSIGNCUI at build time or —assign-cui on the command line (e.g. XY). You then have a field called ID in your authentication table, typically an auto incrementing number with leading zeros or starting from a high number, e.g. 123456 which would result in a CUI of XY123456.

It is also possible to include a *cui* field in the database. If this is set then that is what is used as the CUI (taking precedence over the assigned CUI and ID field).

### Configuring the connection

The following fields are applied if defined in the table and non NULL.

| Configuration control fields | |
| --- | --- |
| session\_timeout | Defines the number of seconds before the session is automatically terminated |
| lcp\_rate | Defines the number of seconds between LCP echos (1 is recommended) |
| lcp\_timeout | Defines the number of seconds before LCP timeout |
| class | Defines an additional graph name allowing groups of lines to have additional graphing as an aggregate and aggregate shaping if the graph has a pre-set speed defined in the FireBrick config |
| table\_number | Defines the routing table for payload traffic 0-99 |
| speed | Defines the bits per second line rate to use, otherwise the line speed is set by the incoming L2TP connection (possibly with speed adjust) |
| speed\_adjust | This is a percentage speed adjust, typically used to allow for overhead for ATM or Ethernet headers, and can be useful to set slightly lower still to allow for VoIP to work when a link is full. |
| ip4\_wan | Defines a single IPv4 address as WAN. This needs setting for all line as the FireBrick does not do pool allocations. Set the same for each line where bonding lines to a FireBrick at the end user. |
| ip4 | Can contain one of more IPv4/bits values, space separated. These are routed to the line as well as the WAN. Set the same for each line where bonding lines to a FireBrick at the end user. |
| ip4\_ppp | New IPv4 PPP endpoint address (i.e. LNS IP on PPP link) |
| ip4\_dns | One or two DNS IPv4 servers (space separated) |
| ip6\_wan | Defines a single IPv6 address as WAN. This is technically not necessary as devices should use a LAN address if they have no WAN, but not all devices play nicely, so you may want to define these from a separate block. One trick is make a block and set these based on the IPv4 WAN. Also, set the same for each line where bonding lines to a FireBrick at the end user. |
| ip6 | Can contain one of more IPv6/bits values, space separated. These are routed to the line as well as the WAN. These are sent for prefix delegation. Typically a single /64 is needed. Set the same for each line where bonding lines to a FireBrick at the end user. |
| filter\_x | This allows a filter to be set (x being any letter). See the FireBrick manual for the full set. Some can be very useful for some use cases such as TCP fix, ignoring MRU, restarting LCP, etc. If the value is defined and either Y or true, then the upper case filter is sent. If the value is defined and either N or false then the lower case filter is sent. If the value is a number, it is sent as upper case letter and the number specified. |

### Relaying connections

In addition to termination a connection on IP using the above fields, it is also possible to defined a relayed connection. If the relay field is defined and not NULL, the following apply instead of IP address assignments, and DNS. Other settings such as LCP, speed, etc, still apply.

| Relay connections | |
| --- | --- |
| relay | A host name or IP address to which to relay via L2TP  Prefix with a \* (star) to cause relay via RADIUS session steering at that host/IP |
| login | Overrides the login when passed on to relay |
| calling | Overrides the calling number when passed on to relay |
| called | Overrides the called number when passed on to relay |
| relay\_hostname | L2TP hostname to use (like a username/login at the L2TP level) |
| relay\_graph | The graph to use for the relay |
| relay\_password | L2TP level password / secret to use, or RADIUS shared secret for steering |

### Last connection details

The authentication table is an obvious place to store some details of the last connection, but this can be separate, as explained above, for fb-radius-auth and fb-radius-acct.

The following fields are updated on login in that table, if present.

| Updated on login | |
| --- | --- |
| last\_login | Date/time of last login |
| last\_tx\_speed | Rate of last login, after any adjustment or setting |
| last\_rx\_speed | Rate of last login - note that not all carriers send this so may be 0 |
| last\_nas | The name of the last NAS |
| last\_lac | The IP address of the LAC |
| last\_lns | The IP address of the LNS |
| last\_table\_number | The routing table of last login |
| last\_tunnel | The tunnel ID of last login |
| last\_tunnel\_graph | The tunnel graph name of last login |
| last\_mru | The MRU of last login |
| last\_cug | The CUG of last login (set by filter\_a or filter\_r, deprecated feature) |
| last\_username | The last full username (with realm) |

| Updated on logout | |
| --- | --- |
| last\_logout | Date/time of last logout |
| last\_cause | Cause code of last logout |

## Session table

The session table is updated by fb-radius-acct. It is created if not defined, so no need to create the table first. It is specified by -DSESSION on build, or —session on command line.

This only contains active sessions. If for some reason a session is no longer active but the RADIUS STOP has not been received (e.g. LNS restarted un-cleanly) then the session is automatically deleted after a few hours.

| Session table | |
| --- | --- |
| id | The 30 character unique ID of the PPP session - globally unique |
| cui | Chargeable User Identity |
| carrier | Carrier name of session |
| calling | Calling line number |
| called | Called line number |
| login | Full login used |
| tunnel | Tunnel ID |
| tunnel\_graph | Tunnel graph name |
| nas | NAS name |
| lac | LAC IP |
| lns | LNS IP |
| start | Start date/time of session |
| last | Last update date/time |
| mru | MRU of session |
| tx\_speed | Tx speed of session as adjusted |
| rx\_speed | Rx speed of session |
| table\_number | Table number of session |
| cug | CUG of session |
| U | Upload bytes so far last update |
| D | Download bytes so far last update |
| UP | Upload packets so far last update |
| DP | Download packets so far at last update |

## History table

The history table records completed sessions. It is created if not defined, so no need to create the table first. It is specified by -DHISTORY on build, or —history on command line.

The fields are the same as the session table except for *finish* instead of *last* which records the end date/time of the session, and *cause* which records the final cause code.

## Daily stats table

The daily stats table records a record per CUI and day which includes usage stats. It is created if not defined, so no need to create the table first. It is specified by -DDAILY on build, or —daily on command line.

| Daily stats per date and CUI | |
| --- | --- |
| dated | The date of this record |
| dow | The day of week (three letter abbreviation) |
| cui | The Chargeable User Identity |
| U | Total upload bytes (so far, if today) |
| D | Total download bytes (so far, if today) |
| PU | Total upload packets (so far, if today) |
| PD | Total download packets (so far, if today) |
| U00 to U23 | Total upload bytes for each hour (so far if current hour) |
| D00 to D23 | Total download bytes for each hour (so far if current hour) |

## Blip table

The blip table is a useful table for spotting problems (unexpected bursts of login or logout). It has a record for each minute of the day so goes back nearly 24 hours. Entries are cleared out ahead for about 5 minutes automatically. Totals are updated on each login and logout. Ideally you want a way to show this data graphically. It is created if not defined, so no need to create the table first. It is specified by -DBLIP on build, or —blip on command line.

| Blip table | |
| --- | --- |
| tod | Time of record, one per minute |
| logins | Total logins |
| logout | Total logouts |

# RADIUS CoA messages

The fb-radius-msg command allows a message to be sent.

The —help provides the options, and the RADIUS secret can be preset using -DSECRET at compile time.

The main options are as follows. The LNS specified with —lns and the session using either —session and the account session ID, or —cui to specific the CUI of a session.

* —disconnect will disconnect a live session
* —connect-info sends a new speed setting (i.,e. tx, or tx/rx bps)
* —filter-id sends a new filter ID. Many filter IDs can be changed on the fly. See FireBrick manual
* —session-timeout sends a new session timeout
* —terminate-action defines if the timeout causes disconnect or RADIUS accounting

This can also be used to check the status of a RADIUS server with —status-server

# Options

The tools provide a number of options which can be seen using —help. Many, as you see above, can be pre-set at compile time.

One of the main ones that is well worth building in at compile time is the RADIUS shared secret, specified using -DSECRET.

The two other main options are:-

* —background which causes the command to fork and go in to background as a daemon. Ensure any other running versions are killed first
* —debug which causes debug output, notably the SQL commands being run, to be written to stderr.

# Syslog

Logs are written to LOCAL7 on local syslog server. It is well worth ensuring you have log files set up for this.