Chapter 7. SER - Adding Authentication and MySQL

What this ser.cfg does:

1. Adds authentication of IP phones by using credentials stored in MySQL
2. Contact information is stored persistently in MySQL

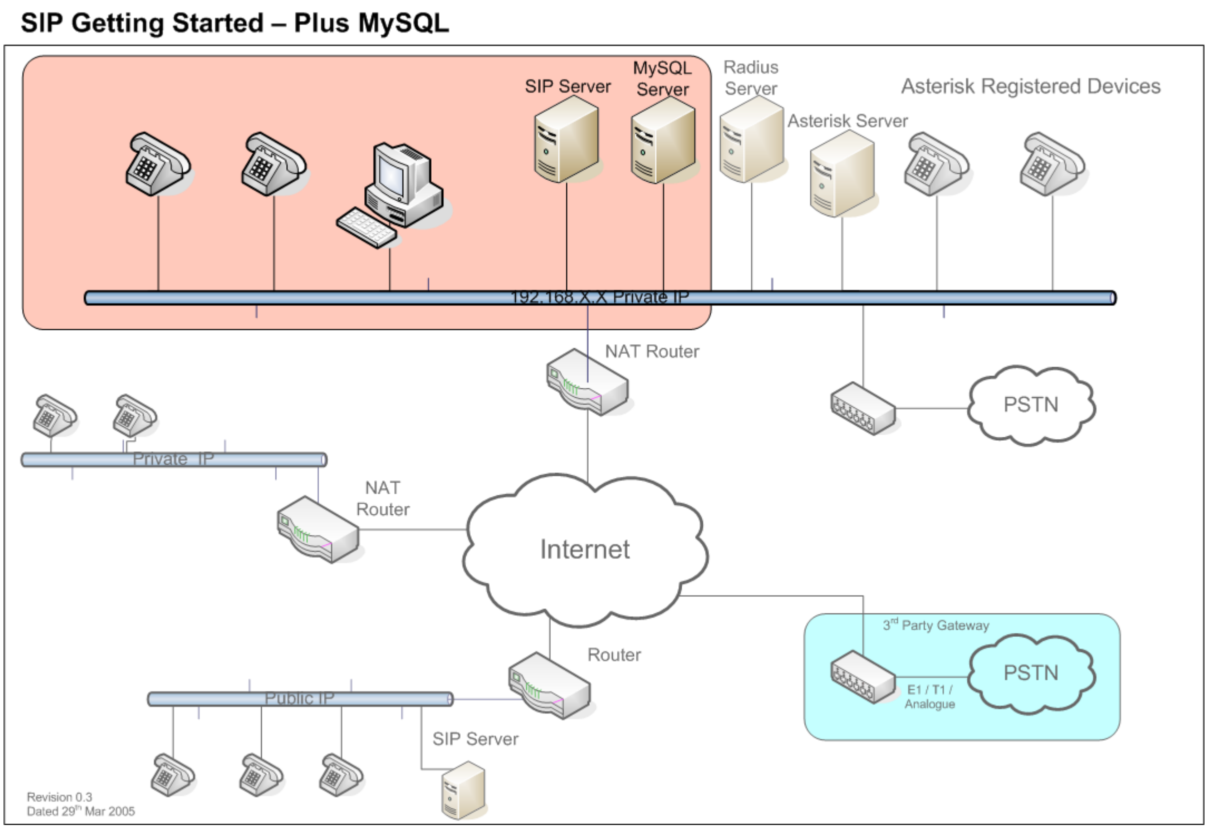
Now that you have tested a basic SIP environment, we need to add more functionality. This section we talk about authentication.

In normal circumstances, we must restrict the use of the SIP server to those telephones (i.e., users) that we want. Authentication is about ensuring only those telephones that we have given a password to are allowed to use our sip services.

To support authentication, we need a way to store information that does not get lost when we stop the sip server. We need a database and the most popular is MySQL as it comes with all Linux configurations. Support is provided for other databases, such as PostgreSQL, but in this Quick Start guide we focus on MySQL.

To add support for MySQL you need to go back to the source code and modify a few parameters. In Chapter 11 - Supporting MySQL describes how to do this and re-install the binaries. Once you have updated your SER environment you need to modify your ser.cfg file as described below.

Figure 7.1. Reference Design Plus MySQL



MySQL ser.cfg Listing

Listed below is the SIP proxy configuration to which builds upon subject matter covered in the Hello World section.

```
debug=3
fork=no
log_stderror=yes

listen=192.0.2.13   # INSERT YOUR IP ADDRESS HERE
port=5060
children=4

dns=no
rev_dns=no
fifo="/tmp/ser_fifo"
fifo_db_url="mysql://ser:heslo@localhost/ser"❶

loadmodule "/usr/local/lib/ser/modules/mysql.so"❷
loadmodule "/usr/local/lib/ser/modules/sl.so"
loadmodule "/usr/local/lib/ser/modules/tm.so"
loadmodule "/usr/local/lib/ser/modules/rr.so"
loadmodule "/usr/local/lib/ser/modules/maxfwd.so"
loadmodule "/usr/local/lib/ser/modules/usrloc.so"
loadmodule "/usr/local/lib/ser/modules/registrar.so"
loadmodule "/usr/local/lib/ser/modules/auth.so"❸
loadmodule "/usr/local/lib/ser/modules/auth_db.so"❹
loadmodule "/usr/local/lib/ser/modules/uri_db.so"❺

modparam("auth_db|uri_db|usrloc", "db_url", "mysql://ser:heslo@localhost/ser")❻
modparam("auth_db", "calculate_ha1", 1)❼
modparam("auth_db", "password_column", "password")❽
modparam("usrloc", "db_mode", 2)❾
modparam("rr", "enable_full_lr", 1)

route {

  # -----------------------------------------------------------------
  # Sanity Check Section
  # -----------------------------------------------------------------
  if (!mf_process_maxfwd_header("10")) {
    sl_send_reply("483", "Too Many Hops");
    break;
  };

  if (msg:len > max_len) {
    sl_send_reply("513", "Message Overflow");
    break;
  };

  # -----------------------------------------------------------------
  # Record Route Section
  # -----------------------------------------------------------------
  if (method!="REGISTER") {
    record_route();
  };
  # -----------------------------------------------------------------
  # Loose Route Section
  # -----------------------------------------------------------------
  if (loose_route()) {
    route(1);
    break;
  };

  # -----------------------------------------------------------------
  # Call Type Processing Section
  # -----------------------------------------------------------------
  if (uri!=myself) {
    route(1);
    break;
  };

  if (method=="ACK") {
    route(1);
    break;
  } if (method=="INVITE") { ❿
    route(3);[11]
    break;
  } else  if (method=="REGISTER") {
    route(2);
    break;
  };

  lookup("aliases");
  if (uri!=myself) {
    route(1);
    break;
  };

  if (!lookup("location")) {
    sl_send_reply("404", "User Not Found");
    break;
  };

  route(1);
}

route[1] {

  # -----------------------------------------------------------------
  # Default Message Handler
  # -----------------------------------------------------------------
  if (!t_relay()) {
    sl_reply_error();
  };
}

route[2] {

  # -----------------------------------------------------------------
  # REGISTER Message Handler
  # ----------------------------------------------------------------
  sl_send_reply("100", "Trying");[12]
```

```
    if (!www_authorize("","subscriber")) { 13
      www_challenge("","0");14
      break;15
    };

    if (!check_to()) { 16
      sl_send_reply("401", "Unauthorized");17
      break;
    };

    consume_credentials();18

    if (!save("location")) { 19
      sl_reply_error();
    };
  }

20route[3] {
    # -----------------------------------------------------------------
    # INVITE Message Handler
    # -----------------------------------------------------------------
    if (!proxy_authorize("","subscriber")) { 21
      proxy_challenge("","0");22
      break;
    } else if (!check_from()) { 23
      sl_send_reply("403", "Use From=ID");24
      break;
    };

    consume_credentials();25

    26lookup("aliases");
    if (uri!=myself) {
      route(1);
      break;
    };

    27if (!lookup("location")) {
      sl_send_reply("404", "User Not Found");
      break;
    };

    route(1);28
  }
```

Authenticating ser.cfg Analysis

Our new SIP proxy configuration is quickly maturing to meet real world requirements. We now store all user registration data to MySQL which enables us to restart the SIP proxy without affecting user registrations. This means that you can quickly restart SER without the clients knowing about it.

Another very important feature that we have introduced is authentication. Authentication happens during two different times in order to secure our SIP router. The first place we need to look at is the area that handles REGISTER messages because we do not want anonymous users to have the ability to register with our SIP proxy.

The second area we must secure is the handler that processes INVITE messages because we do not want unauthenticated users to make telephone calls. If we allowed this then we would have what is called an open relay and if your SIP proxy is connected to a PSTN gateway you could be responsible for excessive toll charges.

An important thing to note is that comments from the Hello World configuration have been removed for clarity so that you can quickly find new areas in this ser.cfg file, which have been commented.

1. The fifo_db_url directive is also included to suppress start up warnings that would otherwise appear when adding MySQL support. We do not directly use the fifo_db_url from ser.cfg, however, other ancillary tools, such as serctl use it to add users to the database.
2. MySQL support is added easily by including the mysql.so module in the loadmodule section. A very important thing to notice is that mysql.so is loaded before all other modules. The reason is that mysql.so does not have any module dependencies, however, other modules, such as uri_db, do depend on the mysql.so module.
3. The auth.so module is not directly used by ser.cfg, however it is necessary to enable authentication functionality. The authentication functionality in this ser.cfg file is provided by the auth.so and auth_db.so modules.
4. Auth_db.so is the module that we invoke directly. This module interoperates with auth.so to perform its function.
5. Uri_db.so is the module that exposes some authentication functions that we will use in this ser.cfg example, namely check_to().
6. The auth_db module exposes a db_url parameter which is required in order to tell SER where to find a MySQL database for user authentication. If you notice we have included auth_db, uri_db, AND usrloc in a single modparam by using the pipe symbol. We do this as a short cut, however it is perfectly legal in SER syntax to have included these in separate modparam statements.
7. The auth_db parameter calculate_ha1 tells SER whether or not to use encrypted passwords in the MySQL subscriber table. In a production system you will want to set this parameter to zero, but in our development system we use unencrypted passwords so the value is set to one.
8. The auth_db module defaults the password column name to ha1, however the MySQL schema that SER uses calls the password column password. Therefore we must inform SER that the column name has changed.
9. The usrloc parameter db_mode must be changed from zero, used in the Hello World example to 2 in this example to configure SER to use MySQL for storing contact information and authentication data.
10. We now explicitly define an INVITE handle route[3]. This handler is responsible for setting up a call.
11. Pass control to route[3] for all INVITE messages that have not been loose routed. Invite messages hitting this statement are original INIVITEs rather than re-INVITEs. After an INVITE message is processed we exit by calling break.
12. When we receive a REGISTER message, we immediately send a 100 Trying message back to the SIP client to stop it from retransmitting REGISTER messages. Since SER is UDP based there is no guaranteed delivery of SIP messages, so if the sender does not get a reply back quickly then it will retransmit the message.

    By replying with a 100 Trying message we tell the SIP client that we're processing its request.
13. The www_authorize() function is used to check the user's credentials against the values stored in the MySQL subscriber table. If the supplied credentials are correct then this function will return TRUE, otherwise FALSE is returned.

    If credentials were not supplied with the REGISTER message, then this function will return FALSE.

    The first parameter specifies the realm in which to authenticate the user. Since we are not using realms, we can just use an empty string here. You can think of the realm in exactly the same way you think of a web server realm (domain).

    The second value tells SER which MySQL table to use for finding user account credentials. In this case we specified the subscriber table.
14. Here we actually send back a 401 Unauthorized message to the SIP client which tell it to retransmit the request with digest credentials included.

    www_challenge() takes two arguments. The first is a realm, which will appear in the WWW-Authorize header that SER sends back to the SIP client. If you put a value in here then that realm will appear to the SIP client when it is challenged for credentials.

The second value affects the inclusion of the qop parameter in the challenge request. It is recommended to keep this value set to 1. See RFC2617 for a complete description of digest authentication. Please note though that some IP phones do not support qop authentication. You may try to use 0 if you experience Wrong password problems.

15 Since we sent back a 401 error the SIP client on the previous line we no longer need to service this REGISTER message. Therefore we use the break command to return to the main route block.

16 When operating a SIP proxy you must be certain that valid user accounts that have been successfully registered cannot be used by unauthenticated users. SER includes the check_to() function for this very reason.

We call check_to() prior to honouring the REGISTER message. This causes SER to validate the supplied To: header against the previously validated digest credentials. If they do not match then we must reject the REGISTER message and return an error.

17 If check_to() returned FALSE then we send a 401 Unauthorized message back to the SIP client. Then we call the break command to return to the main route block.

18 We do not want to risk sending digest credentials to upstream or downstream servers, so we remove any WWW-Authorize or Proxy-Authorize headers before relaying further messages.

19 If execution of ser.cfg makes it to this line then the SIP user has successfully been validated against the MySQL subscriber table, so we use the save(location) function to add the user's contact record to the MySQL location table. By saving this contact record to MySQL we can safely restart SER without affecting this registered SIP client.

20 Route[3] is introduced here to handle INVITE messages.

21 We use proxy_authorize() to make sure we are not an open relay. Proxy_authorize() will require INVITE messages to have digest credentials included in the request. If they are included the function will try to validate them against the subscriber table to make sure the caller is valid.

Like www_authorize(), proxy_authorize() also takes two arguments. The first is a realm and the second is the MySQL table in which to look for credentials. In our example this is the MySQL subscriber table.

22 If the user did not properly authenticate then SER must reply with a 401 Unauthorized message. The proxy_challenge() function takes two arguments that are the same as the www_challenge() function, namely a realm and a qop specifier.

Now that we've sent a 401 challenge to the caller we execute a break command to return to the main route block.

23 Here we call check_from() to make sure the INVITE message is not using hi-jacked credentials. This function checks the user name against the digest credentials to verify their authenticity.

24 If check_from() returned false then we reply to the client with a 401 unauthorized message and return to the main route block.

25 We do not want to risk sending digest credentials to upstream or downstream servers, so we remove any WWW-Authorize or Proxy-Authorize headers before relaying further messages.

26 Look up any associated aliases with the dialed number. If there is an aliases and it is not a locally served domain then just relay the message.

27 Now that all request URI transformations have been done, we can attempt to look for the correct contact record in the MySQL location table. If an AOR (aka, address of record) cannot be found then we reply with a 404 error.

28 Finally, if execution has made it here, then the caller has been properly authenticated and we can safely relay the INVITE message.

Using the Authenticating ser.cfg Proxy

Before you can use this new SIP route configuration you must configure SIP user accounts by using the serctl shell script. We'll continue with user 1000 and 1001. In the Hello World example we happily registered any user that sent a REGISTER message.

Now we require authentication. So open up a terminal window and execute the following two commands:

1. serctl add 1000 password1 user1@nowhere.com [mailto:user1@nowhere.com]

2. serctl add 1001 password2 user2@nowhere.com [mailto:user2@nowhere.com]

You will be prompted for a password. This will be your MySQL SER user's password. Refer to the section on setting up MySQL for further instructions on configuring the MySQL user account.

Once the accounts are created you need to update your SIP phones by changing the passwords to whatever you used in the serctl commands. Then reboot your SIP phones and you should be able to complete telephone calls.