SIP proxy with a user location database and authentication of registration and call setup requests

Emin Gabrielyan
2007-04-24
Switzernet Sàrl

The demonstrations presented in this document are based on OpenSER proxy server running with MySQL database. We discuss a reboot safe configuration saving user locations in a database instead of keeping this information in memory. We examine authentication of the registration process and of the call setup process.

1. The experimental configuration

We use an OpenSER proxy server located at IP address 192.168.1.15 and two Budge Tone-100 SIP phones located at IP addresses 192.168.1.10 and at 192.168.1.11 respectively. We run OpenSER with MySQL database system on a Debian

GNU/Linux. We use version 1.2.0 of OpenSER allowing manipulations with user defined pseudo variables.

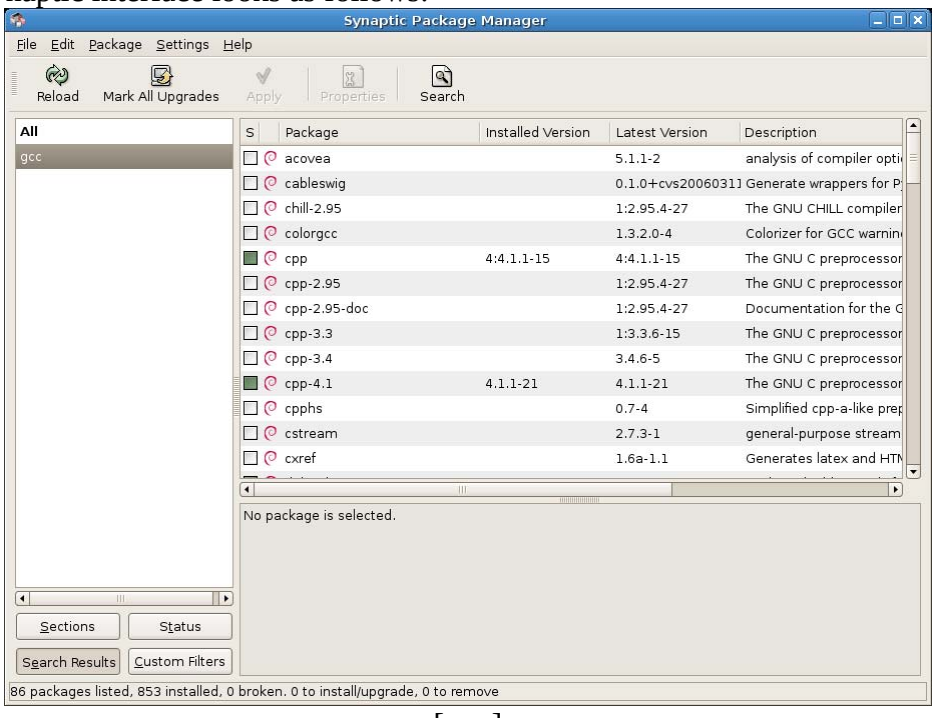
2. OpenSER installation details

You can skip this section, if your OpenSER version 1.2.0 already properly operates with MySQL database.

In order to install the latest version (1.2.0) of OpenSER with mysql, it is required to download the source codes and compile the sip server. For this purpose several pre-required packages need to be installed. Use Synaptic Package Manager to install the following pre-required packages: subversion (svn client), gcc (gcc compiler), flex, bison, libmysqlclient15-dev. The synaptic package manager can be called from the Applications menu of the Gnome desktop environment of Debian:

```
Applications
  Debian
    Apps
      System
        Synaptic Package Manager
```

The Synaptic interface looks as follows:



[png]

User the [Search] icon to list all related components of the packages you need to install (i.e. use search for each of the following packages: subversion, gcc, flex, bison, and libmysqlclient15-dev). Within each list of displayed components, mark for installation the required package name (mark only the package you need, the dependent packages will be added automatically) and click on the [Apply] icon.

Follow the instructions of "Install and Maintain OpenSER from SVN" starting from section 2 (since all pre-required packages are already installed with Synaptic Package Manager) up to section 7 (the configuration files we will discussed directly in this

document). In section 7 of "[Install and Maintain OpenSER from SVN]" you should create the MySQL database:

```
/usr/local/sbin/openser_mysql.sh create
```

The openser_mysql.sh script will ask for the password of the root user of mysql (not the same as the root user of your Unix system):

```
SER02:/usr/local/src/openser-1.2.0/sip-server# openser_mysql.sh create
MySQL password for root:
Enter password:
Enter password:
creating database openser ...
Core OpenSER tables succesfully created.
Install presence related tables ?(y/n):y
creating presence tables into openser ...
Presence tables succesfully created.
Install extra tables - imc,cpl,siptrace,domainpolicy ?(y/n):y
creating extra tables into openser ...
Extra tables succesfully created.
Install SERWEB related tables ?(y/n):y
Domain (realm) for the default user 'admin':

creating serweb tables into openser ...
SERWEB tables succesfully created.

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!                                                 !
!                  WARNING                        !
!                                                 !
! There was a default admin user created:         !
!    username: admin@
!    password: openserrw
!                                                 !
! Please change this password or remove this user !
! from the subscriber and admin_privileges table. !
!                                                 !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
SER02:/usr/local/src/openser-1.2.0/sip-server#
SER02:/usr/local/src/openser-1.2.0/sip-server#
```

You can check the tables created by openser_mysql.sh script:

```
SER02:/home/emin/Desktop/branch/a2#
SER02:/home/emin/Desktop/branch/a2# mysql
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| openser            |
+--------------------+
3 rows in set (0.00 sec)

mysql> use openser;
mysql> show tables;
+-----------------------+
| Tables_in_openser     |
+-----------------------+
| acc                   |
| active_sessions       |
| active_watchers       |
| address               |
| admin_privileges      |
| aliases               |
| cpl                   |
| dbaliases             |
| domain                |
```

```
| domainpolicy          |
| grp                   |
| gw                    |
| gw_grp                |
| imc_members           |
| imc_rooms             |
| lcr                   |
| location              |
| missed_calls          |
| pdt                   |
| pending               |
| phonebook             |
| presentity            |
| pua                   |
| re_grp                |
| server_monitoring     |
| server_monitoring_agg |
| silo                  |
| sip_trace             |
| speed_dial            |
| subscriber            |
| trusted               |
| uri                   |
| usr_preferences       |
| usr_preferences_types |
| version               |
| watchers              |
| xcap_xml              |
+-----------------------+
37 rows in set (0.00 sec)

mysql> quit
```

In case of error you can delete the OpenSER database and reinstall it again:

```
SER02:/usr/local/src/openser-1.2.0/sip-server#
SER02:/usr/local/src/openser-1.2.0/sip-server# openser_mysql.sh drop
MySQL password for root:
SER02:/usr/local/src/openser-1.2.0/sip-server# openser_mysql.sh create
```

3. Saving locations of User Agents in the database

If user location information is kept in the memory, User Agents will be unable to receive calls after the proxy server is rebooted. The User Agents will be able to receive calls only after they register themselves again (usually after expiration of a certain time from the moment of their previous registration or upon a reboot of UA).

When user locations are saved in database, the system is reboot safe, since the proxy server, once rebooted, can access the location information stored in the database.

We run the OpenSER server in a debug mode as a terminal process. We xlog() function for logging the processing details on the screen. We need mysql module to store user locations in a database. By setting the usrloc's parameter db_mode to 2 we tell OpenSER to use mysql for storing contact information (and not the memory).

```
debug=3
children=4
fork=no
log_stderror=yes
listen=192.168.1.15
port=5060
```

```
mpath="/usr/local/lib/openser/modules/"
loadmodule "mysql.so"
loadmodule "sl.so"
loadmodule "tm.so"
loadmodule "rr.so"
loadmodule "maxfwd.so"
loadmodule "usrloc.so"
loadmodule "registrar.so"
loadmodule "textops.so"
loadmodule "mi_fifo.so"
loadmodule "xlog.so"

modparam("mi_fifo", "fifo_name", "/tmp/openser_fifo")
modparam("usrloc", "db_mode", 2)
modparam("rr", "enable_full_lr", 1)
```

[configuration file]

In our configuration file we display all messages received by the proxy server. The requests are printed with a green background and the replies with a cyan background:

```
route
{
        t_on_reply("1");
        xlog("L_NOTICE","$rm\n$Cbg$mb$Cxx\n");
        ...
}

onreply_route[1]
{
        xlog("L_NOTICE","$rs ($rr) concerning $rm\n$Cbc$mb$Cxx\n");
}
```

Pseudo-variables ($Cxy) for setting colors are described in "OpenSER Pseudo-variables for Version 1.2.x".

When usrloc's parameter db_mode is set to 0, the registration information is kept only in memory:

```
modparam("usrloc", "db_mode", 0)
```

If the proxy server is rebooted the location of the callee phones is lost and an INVITE request cannot be served. In such a scenario, the lookup("location") function fails.

```
if (!lookup("location")) {
     xlog("L_NOTICE","$CrxUser is not found$Cxx\n");
     sl_send_reply("404", "Not Found");
     exit;
};
```

The printout of the OpenSER server is logged in a file [htm], [txt], [doc].

When db_mode parameter is set to 2, user locations are saved on the disk.

```
modparam("usrloc", "db_mode", 2)
```

When invoking save("location") while receiving a REGISTER request, the contact information will be recorded in the openser database table called "location".

```
if (uri==myself) {
  if (method=="REGISTER") {
    save("location");
    exit;
  };
  ...
}
```

The information stored in the location table can be viewed through the mysql interface:

```
SER02:/home/emin/Desktop/branch/a3# mysql
```

```
mysql> use openser;
mysql> select username,contact,user_agent from location;
+----------+---------------------+---------------------------+
| username | contact             | user_agent                |
+----------+---------------------+---------------------------+
| 11       | sip:11@192.168.1.11 | Grandstream BT110 1.0.8.33 |
| 10       | sip:10@192.168.1.10 | Grandstream BT110 1.0.8.33 |
+----------+---------------------+---------------------------+
2 rows in set (0.00 sec)

mysql> quit
```

Using the select SQL statement, we see in the location table two records for registered Budge Tone-100 telephones with phone numbers 10 and 11.

A scenario, where the proxy server is able to process a phone call immediately after a reboot, is logged [htm], [txt], [doc].

Basic SQL statements can be found in a short SQL tutorial.

4. Authentication with origin server (registrar) for registration and for receiving incoming calls

In this section we focus on authentication of User Agent wishing to register itself at SIP server. At this stage we authenticate only the registration process. Unauthenticated user will not be able to register and therefore will not be able to receive calls (since there will be no entry in the location table). However such a User Agent will be still able to make calls, since currently we do not require authentication for processing outgoing calls. In all discussed examples authorization and authentication are handled in SIP on a request-by-request basis [rfc3261, p.17], [rfc3261].

4.1. Challenge-response access authentication mechanism

The SIP authentication mechanism is the same as that of HTTP which supports two authentication schemes: **Basic** and **Digest** [rfc2617]. With the **Basic** method the username and password are passed over network as clear text [rfc2617, p.1], [rfc2617, p.6]. The **Digest** Access Authentication scheme is based on encryption [rfc2617, p.1], [rfc2617, p.2], [rfc2617, p.6]. Usually the Digest scheme is used in SIP.

The Digest scheme is based on a challenge-response mechanism. According to the Digest scheme, the server **challenges** the user using a server generated **nonce** value. A valid **response** of the user is a checksum of the server's nonce value concatenated with the username and the password. In this way, the password is never sent in the clear [rfc2617, p.6-7]. The **nonce** value is a data string generated by the server each time a challenge is sent. This string is a base64 or hexadecimal data [rfc2617, p.9]. The **response** is a string of 32 hex digits. Since the response is a checksum of the

password and the nonce value sent by the server, the user can prove that the correct password is known without a need to send the password itself [rfc2617, p.11]. The checksum is computed by the MD5 algorithm.

When the server challenges the original request of the user it ignores the original request and waits for a new request with a valid response. The server challenges the user by sending a reply to the original request requiring an authentication. The user must re-send its request again, with a header field containing the required authorization [rfc2617, p.3], [rfc2617, p.6-7].

One of the parameters of the challenge message is the nonce string. Another parameter is the **realm** string. It is a string for identification of the server. Usually the realm is simply the name of the server. Realm can contain also a name of a virtual server in case the same server operates different databases for different types of users. When the user receives the realm string, it knows which username and password to use (many users may have several pairs of usernames and passwords for working with different servers) [rfc2617, p.4], [rfc2617. p.5], [rfc2617, p.8].

Depending on the type of the SIP request, the challenge message sent by the server and the correspondingly retransmitted SIP request sent by the user may belong to one of the two types: authentication with **origin server** and authentication with **proxy**.

Authentication of **registration** requests belongs to the type of authentication with the **origin server**. Authentication of call setup requests belongs to the type of authentication with proxy and will be discussed in section 5. In this section we discuss authentication of registration requests.

The SIP server wishing to authenticate the user upon registration (i.e. the SIP server acts as registrar or as an origin server), uses the **401 (Unauthorized) response** message in order to challenge the authorization of a user agent. This response includes a **WWW-Authenticate** header field containing at least one challenge applicable to the requested resource (typically Digest) and the server's realm [rfc2617, p.3].

A user agent that wishes to authenticate itself with an origin server, after receiving a 401 (Unauthorized) reply, includes an **Authorization** header field in the request and re-sends the request to the server [rfc2617, p.4].

4.2. Experimenting with OpenSER and logging the authentication messages

4.2.1. Creating user records in the OpenSER database for authentication

In our current scenario we have two SIP phones with phone numbers 10 and 11. The SIP phones are located at static IP addresses 192.168.1.10 and 192.168.1.11 respectively. In order to tell the SIP server about these two phones we must creates two records in the subscriber table of the openser database:

```
mysql> use openser;
Database changed
mysql> describe subscriber;
+------------------+------------------+------+-----+
| Field            | Type             | Null | Key |
+------------------+------------------+------+-----+
| id               | int(10) unsigned | NO   | PRI |
| username         | varchar(64)      | NO   | MUL |
| domain           | varchar(128)     | NO   |     |
| password         | varchar(25)      | NO   |     |
| first_name       | varchar(25)      | NO   |     |
| last_name        | varchar(45)      | NO   |     |
| email_address    | varchar(50)      | NO   |     |
| datetime_created | datetime         | NO   |     |
| ha1              | varchar(128)     | NO   |     |
| ha1b             | varchar(128)     | NO   |     |
| timezone         | varchar(128)     | YES  |     |
| rpid             | varchar(128)     | YES  |     |
| phplib_id        | varchar(32)      | NO   | UNI |
| phone            | varchar(15)      | NO   |     |
| datetime_modified | datetime        | NO   |     |
| confirmation     | varchar(64)      | NO   |     |
| flag             | char(1)          | NO   |     |
| sendnotification | varchar(50)      | NO   |     |
| greeting         | varchar(50)      | NO   |     |
| allow_find       | char(1)          | NO   |     |
+------------------+------------------+------+-----+
20 rows in set (0.00 sec)
mysql>
```

Records can be inserted into a table by using the insert into SQL statement (see the SQL insert into tutorial page)

```
mysql> insert into subscriber (username,password) VALUES ('10','abc10');
Query OK, 1 row affected (0.00 sec)
mysql> select id,username,password from subscriber;
+----+----------+-----------+
| id | username | password  |
+----+----------+-----------+
|  1 | admin    | openserrw |
|  2 | 10       | abc10     |
+----+----------+-----------+
2 rows in set (0.00 sec)
```

The phplib_id field of the table is marked to be unique. Therefore for each record we must assign a unique value to this field. An old value of a record can be changed by update SQL statement. We replace the old empty value with a string.

```
mysql> update subscriber set phplib_id='070418-1235-aa' where id='2';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
mysql> select id,username,password,phplib_id from subscriber;
+----+----------+-----------+----------------------------------+
| id | username | password  | phplib_id                        |
+----+----------+-----------+----------------------------------+
|  1 | admin    | openserrw | cd78c85cf1641e8e71375e1f7984207f |
|  2 | 10       | abc10     | 070418-1235-aa                   |
+----+----------+-----------+----------------------------------+
2 rows in set (0.00 sec)
```

We can add the second subscriber, with another value for phplib_id (the value for our example is a concatenation string of the date and time in the following format YYMMDD-HHMM-aa):

```
mysql> insert into subscriber (username,password,phplib_id) values
('11','abc11','070418-1256-aa');
Query OK, 1 row affected (0.00 sec)
```

```
mysql> select id,username,password,phplib_id from subscriber;
+----+----------+-----------+----------------------------------+
| id | username | password  | phplib_id                        |
+----+----------+-----------+----------------------------------+
|  1 | admin    | openserrw | cd78c85cf1641e8e71375e1f7984207f |
|  2 | 10       | abc10     | 070418-1235-aa                   |
|  3 | 11       | abc11     | 070418-1256-aa                   |
+----+----------+-----------+----------------------------------+
3 rows in set (0.01 sec)
```

We configure our phone 10 with an Authenticate Password equal to “abc10” and phone 11 we deliberately configure with a wrong password.

4.2.2. Passing the replies of the SIP server through the SIP server itself (for examining their contents)

We are currently ready to use an OpenSER configuration file for authentication of registration requests. When OpenSER processes the requests in stateful mode (e.g. calls are relayed using t_relay() function instead of forward() function) the SIP script is able to catch the replies and associate them with the corresponding requests. It works in case when the replies are generated by a third party, for example by another UA and they pass through the proxy server. In case when the replies are generated by the server itself (this is precisely the case with the replies to REGISTER requests), the SIP script cannot catch and display such replies.

For examining the exchange of challenge-response messages we need to see the replies of our own SIP server. It will be possible if the server sends the replies back to itself and only then to the correct destination (i.e. to the UA which transmitted the request). In SIP, the replies always follow the path of request in the reverse direction. It is ensured by a stack of “Via” header fields collected within the request message while it travels to its destination. The stack keeps the track of the path by storing the IP addresses of all intermediary nodes. The stack is further copied in reply messages and the “Via” fields are removed at each intermediary node when the reply travels back. The top most “Via” field instruct each intermediary node (a proxy server) where is the next hop of the message. Therefore it is sufficient to loop the arrival path of requests in order to have a looped return path of replies. There is only one iteration of looping through our server (i.e. the first arrival of the message is sent to the proxy itself, and the message is processed and is forwarded further at its second arrival).

In the following fragment of our configuration file in case we deal with REGISTER request and if it is the first arrival of this request, we transmit it to ourselves without processing.

```
        if(method=="REGISTER")
        {
          if(!search("P-hint: [Ss]elf-[Ll]ooped"))
          {
            append_hf("P-hint: Self-Looped\r\n");
            t_relay("192.168.1.15");   # the IP address of this proxy
            exit;
          }
        }
```

We display the content of REGISTER messages only for their second arrival (i.e. each REGISTER message will be displayed only once):

```
       # Only the second arrival of REGISTER will be displayed.
```

```
      xlog("L_NOTICE","$rm\n$Cbg$mb$Cxx\n");
```

Thanks to the looping of all register messages, we will be able to see once all replies of our own proxy server to REGISTER requests.

4.2.3. OpenSER configuration script for authentication of registration requests

Our current configuration file now does not grant registrations and does not save locations of User Agents without controlling their usernames and passwords. If the request arrives the first time, the server will not process it, but will send a challenge message: a 401 (Unauthorized) reply. This is done with www_challenge("","0") function:

```
if (uri==myself) {
  if (method=="REGISTER") {
    if (!www_authorize("", "subscriber")) {
      xlog("L_NOTICE","Unable to verify the credentials\n");
      www_challenge("", "0");
      exit;
    };
    save("location");
    exit;
  }
  ...
}
```

When the second request is received, the www_authorize("","subscriber") function checks if the new REGISTER request contains the response parameter, if it corresponds to the previously sent challenge (i.e. to the nonce parameter transmitted in the previous 401 (Unauthorized) challenge), and if the response corresponds to the correct password of the user stored in the subscriber table of the openser database.

The first argument of both functions (which is an empty string for both functions of the example) is the realm parameter. If an empty string "" is used then the server will generate the realm string from the request. In case of REGISTER requests “To” header field domain will be used, because this header field represents a user being registered. The server will use the domain part of the “To” header field as its realm parameter.

Location of the user will be stored in location table of the database only if www_authorize("","subscriber") function returns true.

Therefore, according to the current configuration, without being registered the SIP proxy will not save the location information and therefore the SIP phone will not be able to receive incoming calls via the SIP proxy server.

Note that since in our configuration file, the outgoing calls are processed without authentication, the user will be able to make outgoing calls even if its registration fails. For making an outgoing call without registration, the phone must only internally “*Allow outgoing call without Registration*” (see the phone’s configuration).

4.2.4. Request-challenge-response authentication messages of the registration process

Recall that we configured phone 10 with a correct password and phone 11 with an incorrect password. Our configuration script is designed so as to display messages arriving from User Agents as well as responses of the proxy server itself. The printouts of the SIP server are logged [htm], [txt], [doc].

The first register message of SIP phone 10 does not contain authorization header field (some insignificant header fields are removed from the examples; consult the full printout to view the complete messages):

```
0(26142) REGISTER
REGISTER sip:192.168.1.15 SIP/2.0
Via: SIP/2.0/UDP 192.168.1.15;branch=z9hG4bK55d.4261f4d1.0
Via: SIP/2.0/UDP 192.168.1.10;branch=z9hG4bK1cfcbbd452dd007a
From: <sip:10@192.168.1.15>;tag=20321951e0685e4d
To: <sip:10@192.168.1.15>
Contact: <sip:10@192.168.1.10>
Call-ID: 33706713ea3b83fd@192.168.1.10
CSeq: 100 REGISTER
Expires: 150
User-Agent: Grandstream BT110 1.0.8.33
Max-Forwards: 68
Content-Length: 0
P-hint: Self-Looped
```

The proxy does therefore not process the register request and sends a challenge with a 401 (Unauthorized) reply:

```
0(26142) 401 (Unauthorized) concerning REGISTER
SIP/2.0 401 Unauthorized
Via: SIP/2.0/UDP 192.168.1.15;branch=z9hG4bK55d.4261f4d1.0
Via: SIP/2.0/UDP 192.168.1.10;branch=z9hG4bK1cfcbbd452dd007a
From: <sip:10@192.168.1.15>;tag=20321951e0685e4d
To: <sip:10@192.168.1.15>;tag=4f4b3fc66ce1848604aacd9b43692540.ca9b
Call-ID: 33706713ea3b83fd@192.168.1.10
CSeq: 100 REGISTER
WWW-Authenticate: Digest realm="192.168.1.15",
nonce="462791f06aa9b37a34fd11b1adc58c5a9df7c95b"
Server: OpenSER (1.2.0-notls (i386/linux))
Content-Length: 0
```

A 401 (Unauthorized) response message is used by an origin server to challenge the authorization of a user agent. For REGISTER requests, the SIP server is considered as an origin server. When processing a call, the SIP server is considered as a proxy server and Proxy Authentication Required response is sent instead of 401 (Unauthorized) reply (see section 5). The 401 (Unauthorized) message of the origin server contains **WWW-Authenticate** header field [rfc2617, p.3].

As a response to the challenge of the origin server, the UA retransmits the initial request, but this time with an **Authorization** header field with parameters corresponding to those of **WWW-Authenticate** header field of the server's challenge [rfc2617, p.4]:

```
0(26142) REGISTER
REGISTER sip:192.168.1.15 SIP/2.0
Via: SIP/2.0/UDP 192.168.1.15;branch=z9hG4bK65d.33f1abb2.0
Via: SIP/2.0/UDP 192.168.1.10;branch=z9hG4bKa468c8b72fe7a3a7
From: <sip:10@192.168.1.15>;tag=20321951e0685e4d
To: <sip:10@192.168.1.15>
```

```
Contact: <sip:10@192.168.1.10>
Authorization: Digest username="10", realm="192.168.1.15", algorithm=MD5,
uri="sip:192.168.1.15", nonce="462791f06aa9b37a34fd11b1adc58c5a9df7c95b",
response="be9e8d46b8dcd522f22301e1f2b3287c"
Call-ID: 33706713ea3b83fd@192.168.1.10
CSeq: 101 REGISTER
Expires: 150
User-Agent: Grandstream BT110 1.0.8.33
Max-Forwards: 68
Content-Length: 0
P-hint: Self-Looped
```

Both messages, the WWW-Authenticate header field of the server's challenge and the Authorization header field of user's re-transmitted request (i.e. the response to the challenge), contain the authentication scheme "Digest" and the same realm parameter "192.168.1.15" identifying the registrar SIP server. According to "Digest" authentication scheme, in the WWW-Authenticate header field the server provided a **nonce** parameter (generated on the fly for every challenge), and the user provided a **response** parameter which is the MD5 checksum of a string which contains server's **nonce** and the user password. In such a way the user proves that it knows the correct password without a need to transmit the password itself.

The server creates the same MD5 checksum locally and compares the checksum computed locally with that of received from the user (in the **response** parameter of **Authorization** header field). If the server side computed checksum matches with the user side computed checksum, the server replies 200 (OK) and processes the REGISTER request of the user (i.e. saves its location in the database):

```
0(26142) 200 (OK) concerning REGISTER
SIP/2.0 200 OK
Via: SIP/2.0/UDP 192.168.1.15;branch=z9hG4bK65d.33f1abb2.0
Via: SIP/2.0/UDP 192.168.1.10;branch=z9hG4bKa468c8b72fe7a3a7
From: <sip:10@192.168.1.15>;tag=20321951e0685e4d
To: <sip:10@192.168.1.15>;tag=4f4b3fc66ce1848604aacd9b43692540.6cb7
Call-ID: 33706713ea3b83fd@192.168.1.10
CSeq: 101 REGISTER
Contact: <sip:10@192.168.1.10>;expires=150
Server: OpenSER (1.2.0-notls (i386/linux))
Content-Length: 0
```

Note that all requests contain two "Via" header fields. The first one is added by the User agent before the departure of the request:

```
Via: SIP/2.0/UDP 192.168.1.10;branch=z9hG4bKa468c8b72fe7a3a7
```

The second one (the top most "Via" header field) is added by the proxy server after the first arrival of the request to the server:

```
Via: SIP/2.0/UDP 192.168.1.15;branch=z9hG4bK65d.33f1abb2.0
Via: SIP/2.0/UDP 192.168.1.10;branch=z9hG4bKa468c8b72fe7a3a7
```

The message is displayed upon the second arrival of the request to the server (with the server's first "Via" stamp).

The replies of the server contain the same stack of "Via" fields:

```
Via: SIP/2.0/UDP 192.168.1.15;branch=z9hG4bK65d.33f1abb2.0
Via: SIP/2.0/UDP 192.168.1.10;branch=z9hG4bKa468c8b72fe7a3a7
```

It ensures that the message will follow the request's arrival path in the reverse direction. It means that the reply will be first routed to 192.168.1.15 (so to the proxy itself), then at that hop (i.e. in the proxy), the top most "Via" header field will be removed and the request will be routed to the IP address indicated by the next "Via"

header field: 192.168.1.10 (i.e. to the final destination). We see the reply message upon its arrival to the proxy, when the header field of the proxy is not yet removed.

Note also that the first exchange of REGISTER and 401 (Unauthorized) constitutes one transaction (see the branch number):

```
REGISTER sip:192.168.1.15 SIP/2.0
Via: SIP/2.0/UDP 192.168.1.10;branch=z9hG4bK1cfcbbd452dd007a
SIP/2.0 401 Unauthorized
Via: SIP/2.0/UDP 192.168.1.10;branch=z9hG4bK1cfcbbd452dd007a
```

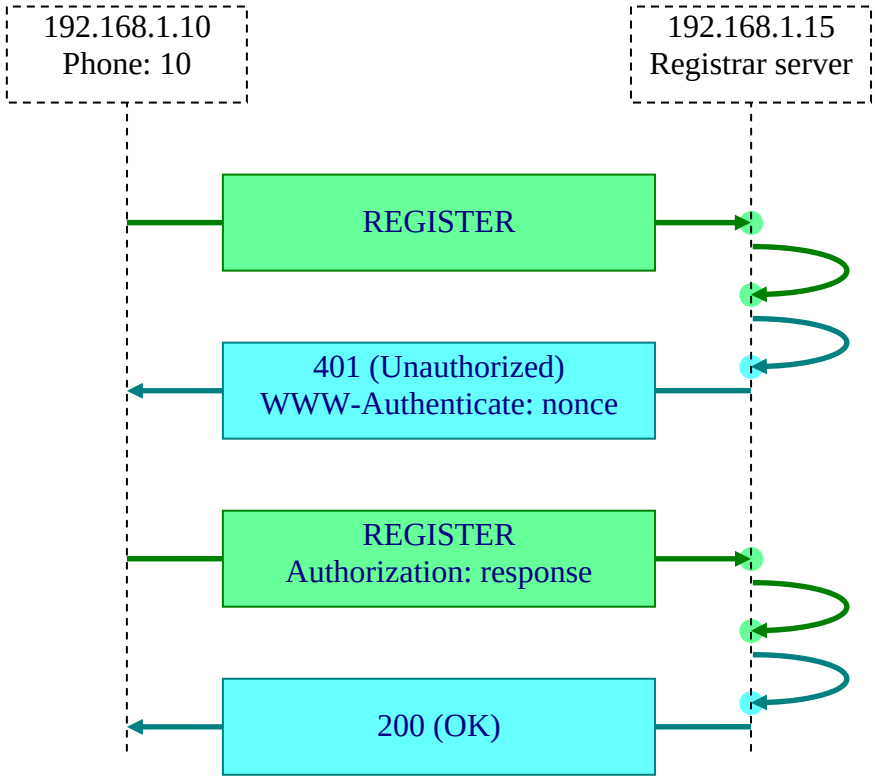
The second exchange of REGISTER and 200 (OK) constitutes a different transaction (a different branch number):

```
REGISTER sip:192.168.1.15 SIP/2.0
Via: SIP/2.0/UDP 192.168.1.10;branch=z9hG4bKa468c8b72fe7a3a7
SIP/2.0 200 OK
Via: SIP/2.0/UDP 192.168.1.10;branch=z9hG4bKa468c8b72fe7a3a7
```

The Call-ID (dedicated for identification of dialogs) is the same in all messages throughout the authentication process:

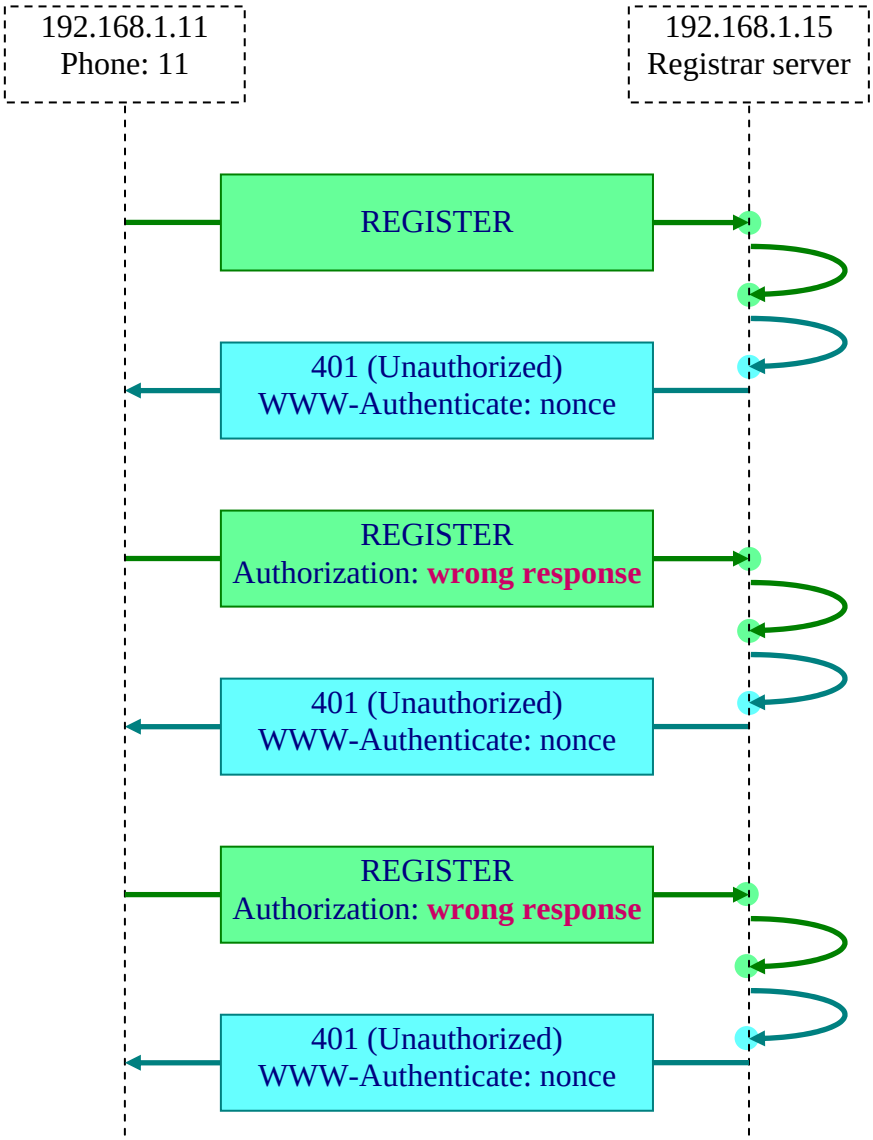
```
REGISTER sip:192.168.1.15 SIP/2.0
Call-ID: 33706713ea3b83fd@192.168.1.10
SIP/2.0 401 Unauthorized
Call-ID: 33706713ea3b83fd@192.168.1.10
REGISTER sip:192.168.1.15 SIP/2.0
Call-ID: 33706713ea3b83fd@192.168.1.10
SIP/2.0 200 OK
Call-ID: 33706713ea3b83fd@192.168.1.10
```

The diagram of the two transactions of the authentication dialog between UA 192.168.1.10 and registrar origin server 192.168.1.15 is shown below:



Concerning the second phone 11, it is configured with a wrong password. It will therefore re-transmit the request with wrong responses. The server will send again

401 (Unauthorized) messages to the user. The UA abandons its registration attempts after two failed Authorization responses (see the full log for the contents of the messages):



Since the authentication of phone 11 is failed, its contact information will not be stored in the location table:

```
mysql> use openser;
mysql> select id,username,contact,user_agent from location;
+----+----------+---------------------+----------------------------+
| id | username | contact             | user_agent                 |
+----+----------+---------------------+----------------------------+
| 12 | 10       | sip:10@192.168.1.10 | Grandstream BT110 1.0.8.33 |
+----+----------+---------------------+----------------------------+
1 row in set (0.00 sec)
```

Since location table contains no record about user **11**, it will be not possible to reach user **11** via our proxy server:

```
if (!lookup("location")) {
  xlog("L_NOTICE","$Crx User is not found$Cxx\n");
  sl_send_reply("404", "Not Found");
  exit;
```

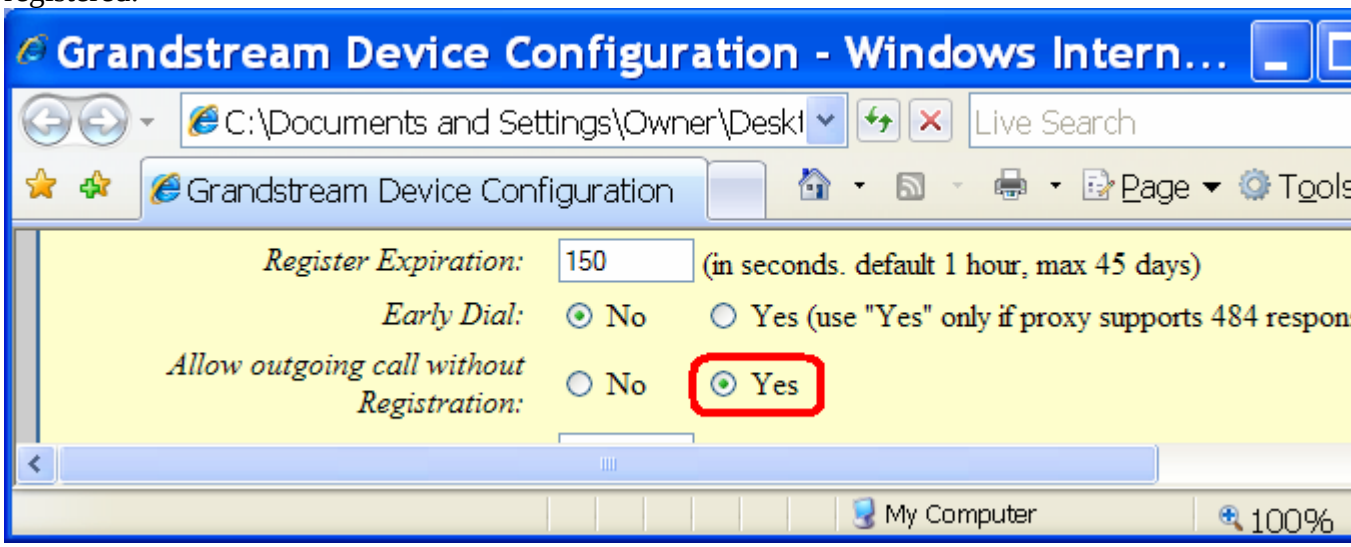
```
};
```

When user **10** tries to call user **11**, the query of the location table fails and 404 (Not Found) is replied to user **10**:

```
0(26142) INVITE
INVITE sip:11@192.168.1.15 SIP/2.0
Via: SIP/2.0/UDP 192.168.1.10;branch=z9hG4bKf31f5ae59154c1ea
From: <sip:10@192.168.1.15>;tag=a3594da4a9837d35
To: <sip:11@192.168.1.15>
0(26142) User is not found
```

For the full content of messages see the complete log [htm], [txt], [doc].

Note that our current configuration script does not require authentication when processing calls. Authentication is actually required only for registration. It means that though the phone 11 is unregistered it is still able to process a call. See the complete log [htm], [txt], [doc] of the current scenario, containing the SIP messages demonstrating a successfully established call from phone **11** to phone **10**. It is only required that the SIP phone itself permit the user to make calls without being registered:



[htm], [bmp]

4.2.5. Checking the “To” header field supplied by user against the previously validated Digest credentials

An additional security check is added in the configuration file described in Authentication and MySQL section of SER Getting Started document (OpenSER is a spin-off of the SER project). We slightly modified the configuration file of the SER tutorial in order to make it compatible with the OpenSER server. After checking user’s credentials (by invoking www_authorize("","subscriber") function) the server performs a checking of the “To” header field by invoking a check_to() function.

```
route
{
  ...
  if (method=="REGISTER") {
    route(2);
    exit;
  }
```

```
  ...
}

route[2]
{
  sl_send_reply("100","Trying to Register");
  if(!www_authorize("","subscriber")) {
    www_challenge("","0");
    exit;
  }
  if(!check_to()) {
    sl_send_reply("401","Unauthorized");
    exit;
  }
  ...
  if(!save("location")) {
    sl_reply_error();
  }
}
```

[configuration file]

A risk exists that a valid user account that has been successfully registered may be used by an unauthorized user. An unauthorized user may intercept the **Authorization** header field of a valid user and use the so obtained correct **response** parameter for authorization of its own register request.

```
REGISTER sip:192.168.1.15 SIP/2.0
To: <sip:10@192.168.1.15>
Contact: <sip:10@192.168.1.10>
Authorization: Digest username="10", realm="192.168.1.15", algorithm=MD5,
uri="sip:192.168.1.15", nonce="462791f06aa9b37a34fd11b1adc58c5a9df7c95b",
response="be9e8d46b8dcd522f22301e1f2b3287c"
```

In register requests the “To” header field represents the user being registered (not the “From” header field). If www_authorize("","subscriber") function ensures in the validity of the **response** parameter the save("location") should save in the location table the coordinates of the user specified in the “To” header field (the actual location being provided in “Contact” header field). A malicious user agent may insert in its register request a good **Authorization response** parameter intercepted from a valid register request, by providing at the same time in the “To” header field its own username. The www_authorize("","subscriber") function will accept the **response** and the save("location") which considers only the “To” field would successfully register the malicious user.

According to RFC2617, the response value is an MD5 checksum of string containing not only the nonce and password parameters (as referred most of the time), but also of the username, the method, and the requested URI [rfc2617, p.6-7]. The check_to() function checks in particular if the username provided in the “To” header field of the request message corresponds to the username encoded in the **response** parameter.

By calling check_to() prior to honoring the REGISTER message we make sure that an imposter cannot use a third-party valid response to register its username in the server. If the server fails validating the supplied “To” header against the previously validated digest credentials, then we reject the REGISTER message and return an error:

```
  if(!check_to()) {
    sl_send_reply("401","Unauthorized");
    exit;
```

```
}
```

5. Authentication for processing outgoing calls

In a secured SIP router, authentication happens during two different times. The first place is the area that handles REGISTER messages because we do not want anonymous users to have the ability to register with our SIP proxy. This type of authentication is already discussed in section 4. 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 the SIP proxy is connected to a PSTN gateway we are then responsible for excessive toll charges.

The call setup authentication process is not exactly the same as the registration authentication (with registrar, namely the origin server). For origin server authentication, the server challenges with the header field **WWW-Authenticate** and the user responses with the header field **Authorization**. For call setup, the proxy server challenges with a different header field **Proxy-Authenticate**, and the user responses with the header field **Proxy-Authorization** [rfc2617, p.4].

Challenge message codes are also different. The SIP server sends its **WWW-Authenticate** challenge for registration requests in a **401 (Unauthorized)** reply, but the **Proxy-Authenticate** challenge for call setup requests is sent in a **407 (Proxy Authentication Required)** reply [rfc2617, p.3].

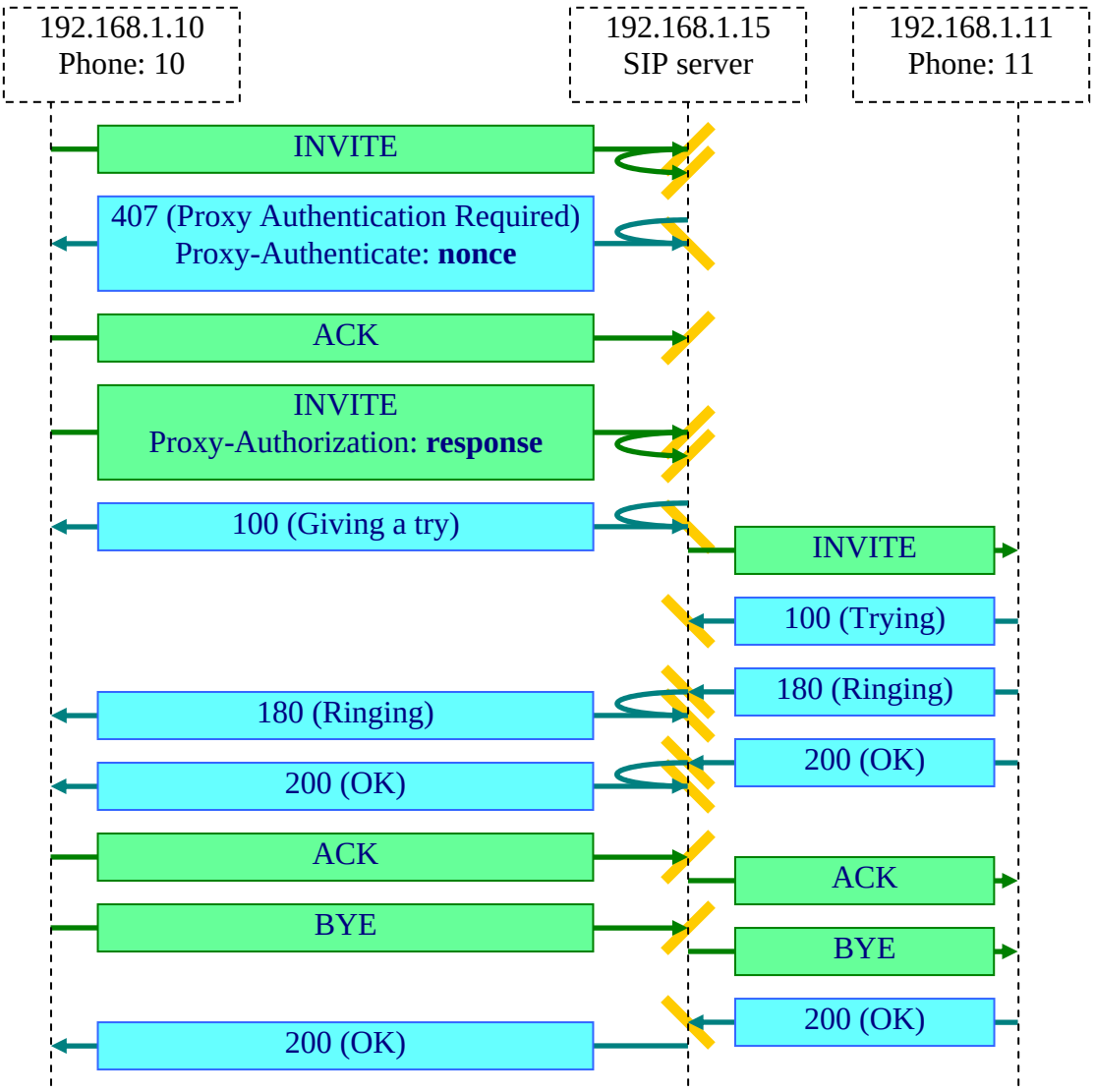
As in case of registration requests discussed in section 4, we need to see the replies of our own proxy also to INVITE requests. Our configuration file should now look for the INVITE requests, and if they are not yet looped (if it is the first arrival of invite), we must loop them once to the proxy itself without processing:

```
if(method=="INVITE")
{
  if(!search("^P-hint: [Ss]elf-[Ll]ooped"))
  {
    append_hf("P-hint: Self-Looped\r\n");
    t_relay("192.168.1.15");   # IP address of this proxy server
    exit;
  }
}
```

Invite request will be processed only after the second arrival. In particular, the logging of INVITE request occurs after the self-looping (so the INVITE requests will be displayed once) and the Record-Route header is also added after the self-looping (i.e. only once). Record-Route header fields (added by record_route() function) cause modifications in "The path of SIP signalling messages" during the entire SIP dialog. If we add Record-Route headers twice, all messages of the SIP dialog (i.e. during the phone call) would also pass through the proxy server twice (in particular the ACK, and BYE requests). Concerning the messages of the INVITE transaction, they will follow the path of the INVITE request in any case (irrespectively to the record_route() function which modifies the path of all successive transactions of the phone call). It means that the replies to INVITE request will follow the same path in the reverse

direction. The reply generated by the proxy itself will arrive to proxy server once (and we will be able to see it). The reply generated by a third party (i.e. by the callee UA), will arrive to proxy twice.

Similarly to the registration scenario, the first attempt of the user agent is rejected and a challenge is sent as a reply to the rejected request. A UA receives a 407 (Proxy Authentication Required) reply to its first INVITE attempt. The UA sends an ACK and retransmits its invite, this time with a Proxy-Authorization header field which contains the response parameter corresponding to the nonce parameter retrieved from proxy's 407 (Proxy Authentication Required) message. The response is the MD5 checksum of the concatenation of the password and the nonce string (with some other known parameters, such as username). The password is not transmitted over the network but the MD5 checksum proves that the UA knows the correct password. The rest of the transaction follows the usual scenario. In our case all replies of the request will pass through proxy twice until the end of the INVITE transaction (200 OK). The yellow bars of the following diagram reflect the events of arrival of messages to the proxy server:



The logs of the first failed INVITE attempt and the beginning of the following INVITE transaction are briefly shown below:

```
0(8132) INVITE (Sun Apr 22 18:46:09 2007)
INVITE sip:11@192.168.1.15 SIP/2.0
Via: SIP/2.0/UDP 192.168.1.15;branch=z9hG4bKc696.fa286516.0
Via: SIP/2.0/UDP 192.168.1.10;branch=z9hG4bKff1baf3cb8b845a2
From: <sip:10@192.168.1.15>;tag=219e32bf2a2ec6f1
To: <sip:11@192.168.1.15>
Contact: <sip:10@192.168.1.10>
Call-ID: 3357af1c8ac608cb@192.168.1.10
 0(8132) 407 (Proxy Authentication Required) concerning INVITE
SIP/2.0 407 Proxy Authentication Required
Via: SIP/2.0/UDP 192.168.1.15;branch=z9hG4bKc696.fa286516.0
Via: SIP/2.0/UDP 192.168.1.10;branch=z9hG4bKff1baf3cb8b845a2
From: <sip:10@192.168.1.15>;tag=219e32bf2a2ec6f1
To: <sip:11@192.168.1.15>;tag=4f4b3fc66ce1848604aacd9b43692540.0955
Call-ID: 3357af1c8ac608cb@192.168.1.10
Proxy-Authenticate: Digest realm="192.168.1.15",
nonce="462b927d2ab8368b1771ed4268c47ca585413f95"
 0(8132) ACK (Sun Apr 22 18:46:09 2007)
ACK sip:11@192.168.1.15 SIP/2.0
Via: SIP/2.0/UDP 192.168.1.10;branch=z9hG4bKff1baf3cb8b845a2
From: <sip:10@192.168.1.15>;tag=219e32bf2a2ec6f1
To: <sip:11@192.168.1.15>;tag=4f4b3fc66ce1848604aacd9b43692540.0955
Contact: <sip:10@192.168.1.10>
Call-ID: 3357af1c8ac608cb@192.168.1.10
 0(8132) INVITE (Sun Apr 22 18:46:09 2007)
INVITE sip:11@192.168.1.15 SIP/2.0
Via: SIP/2.0/UDP 192.168.1.15;branch=z9hG4bKd696.6412566.0
Via: SIP/2.0/UDP 192.168.1.10;branch=z9hG4bK97d3b9d4b3648ecf
From: <sip:10@192.168.1.15>;tag=219e32bf2a2ec6f1
To: <sip:11@192.168.1.15>
Contact: <sip:10@192.168.1.10>
Proxy-Authorization: Digest username="10", realm="192.168.1.15",
algorithm=MD5, uri="sip:11@192.168.1.15",
nonce="462b927d2ab8368b1771ed4268c47ca585413f95",
response="90761772224a8a43df0f31b7437181f3"
Call-ID: 3357af1c8ac608cb@192.168.1.10
 0(8132) 100 (Giving a try) concerning INVITE
SIP/2.0 100 Giving a try
Via: SIP/2.0/UDP 192.168.1.15;branch=z9hG4bKd696.6412566.0
Via: SIP/2.0/UDP 192.168.1.10;branch=z9hG4bK97d3b9d4b3648ecf
From: <sip:10@192.168.1.15>;tag=219e32bf2a2ec6f1
To: <sip:11@192.168.1.15>
Call-ID: 3357af1c8ac608cb@192.168.1.10
 0(8132) 100 (Trying) concerning INVITE
SIP/2.0 100 Trying
Via: SIP/2.0/UDP 192.168.1.15;branch=z9hG4bKd696.7412566.0
Via: SIP/2.0/UDP 192.168.1.15;branch=z9hG4bKd696.6412566.0
Via: SIP/2.0/UDP 192.168.1.10;branch=z9hG4bK97d3b9d4b3648ecf
From: <sip:10@192.168.1.15>;tag=219e32bf2a2ec6f1
To: <sip:11@192.168.1.15>
Call-ID: 3357af1c8ac608cb@192.168.1.10
```

The full messages are in the complete printout [htm], [txt], [doc].

The authentication script of the INVITE messages is similar to that of REGISTER messages:

```
route
{
  ...
  if (method=="INVITE") {
    route(3);
    exit;
  }
  ...
```

```
}

route[3]
{
  if(!proxy_authorize("","subscriber")) {
    proxy_challenge("","0");
    exit;
  }
  if (!check_from()) {
    sl_send_reply("403","Forbidden, Use From=ID");
    exit;
  }
  ...
  lookup("aliases");
  if(uri!=myself) {
    route(1);
    exit;
  }
  if(!lookup("location")) {
    sl_send_reply("404","User Not Found");
    exit;
  }
  route(1);
}
```

[configuration file]

The proxy_authorize("","subscriber") function verifies credentials according to RFC2617. If the credentials are verified successfully then the function will return “true”. If the function was unable to verify the credentials then it will fail and the script calls proxy_challenge("","0") which will challenge the user. The first parameter of proxy_authorize("","subscriber") function is the realm (usually the domain or hostname of the server). Realm is a string that the user agent should present to the user so he can decide what username and password to use [rfc2617, p.4], [rfc2617, p.8]. If an empty string "" is used then the server will generate it from the request. “From” header field domain will be used as realm. The second parameter of proxy_authorize("","subscriber") function is the table to be used to lookup usernames and passwords.

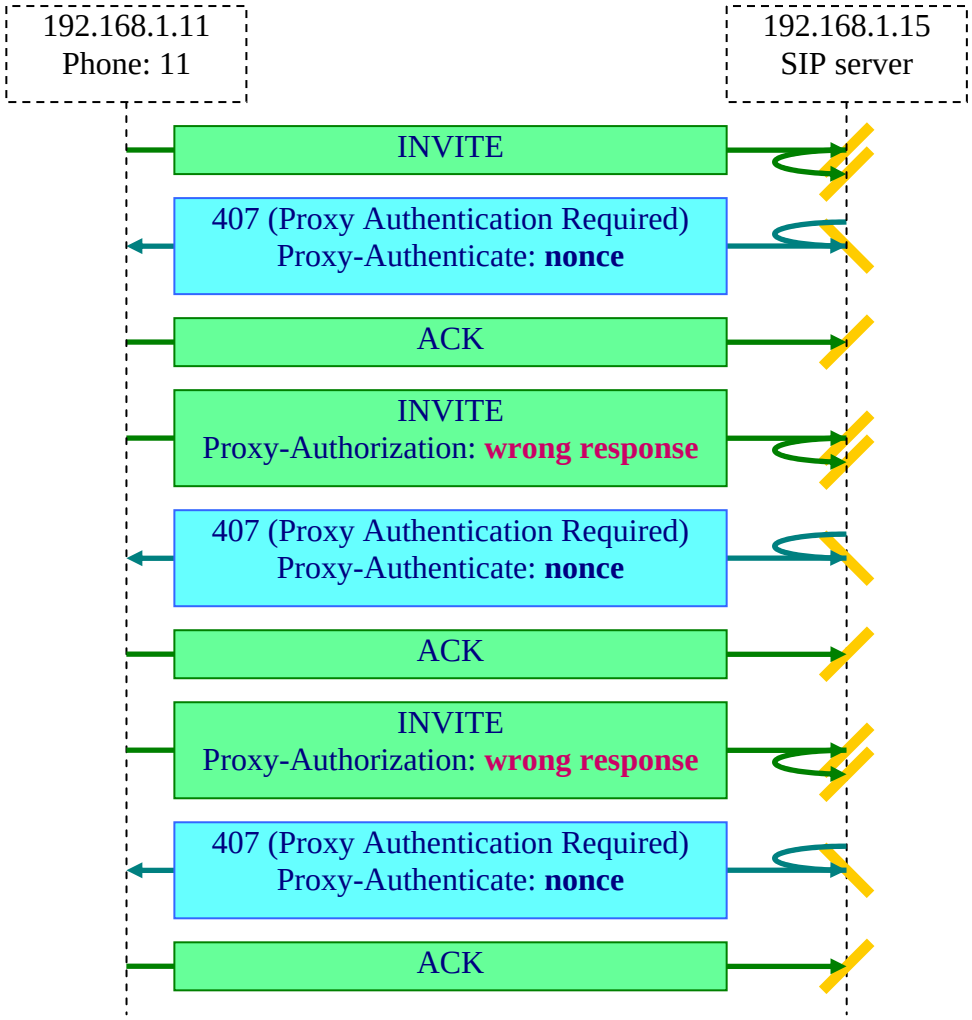
The proxy_challenge("","0") function challenges the user agent. It will generate Proxy-Authenticate header field containing a digest challenge. It will put this header field into the 407 (Proxy Authentication Required) reply generated from the request by the server. Upon reception of such a reply the user agent should compute credentials and retry the request. The first parameter is realm. When an empty "" string is passed, the server uses as realm the domain appearing in the “From” header field of the request.

The check_from() function checks “From” username against the digest credentials to make sure that the INVITE request is not using hijacked credentials of another valid request.

5.1. Unsuccessful proxy authentication

When the UA does not know the correct password and therefore sends a wrong response, the SIP proxy server replies to such re-transmitted request of UA with the

same 407 (Proxy Authentication Required) message. For examining such a scenario we use the same configuration file. The INVITE messages at their first arrival to the proxy server are re-transmitted to the proxy itself. As a consequence, the replies to INVITE also inherit the same looped path (and we can see the replies of our own proxy):



The successive challenges of the SIP server contain the same Proxy-Authenticate parameters. The UA with a wrong password abandons its attempts after two unsuccessful responses (i.e. after three requests):

```
INVITE sip:10@192.168.1.15 SIP/2.0
From: <sip:11@192.168.1.15>;tag=e97fb7b7290040c2
Call-ID: f51f014d589e1ca7@192.168.1.11
SIP/2.0 407 Proxy Authentication Required
Call-ID: f51f014d589e1ca7@192.168.1.11
Proxy-Authenticate: Digest realm="192.168.1.15",
nonce="462c7effba07312005e1116c8236295cad5c0c84"
ACK sip:10@192.168.1.15 SIP/2.0
Call-ID: f51f014d589e1ca7@192.168.1.11
INVITE sip:10@192.168.1.15 SIP/2.0
From: <sip:11@192.168.1.15>;tag=e97fb7b7290040c2
Proxy-Authorization: Digest username="11", realm="192.168.1.15",
algorithm=MD5, uri="sip:10@192.168.1.15",
nonce="462c7effba07312005e1116c8236295cad5c0c84",
response="ee74d173c7c03094af9aeb4ab5acaf6f"
Call-ID: f51f014d589e1ca7@192.168.1.11
```

```
SIP/2.0 407 Proxy Authentication Required
Call-ID: f51f014d589e1ca7@192.168.1.11
Proxy-Authenticate: Digest realm="192.168.1.15",
nonce="462c7effba07312005e1116c8236295cad5c0c84"
ACK sip:10@192.168.1.15 SIP/2.0
Call-ID: f51f014d589e1ca7@192.168.1.11
INVITE sip:10@192.168.1.15 SIP/2.0
From: <sip:11@192.168.1.15>;tag=e97fb7b7290040c2
Proxy-Authorization: Digest username="11", realm="192.168.1.15",
algorithm=MD5, uri="sip:10@192.168.1.15",
nonce="462c7effba07312005e1116c8236295cad5c0c84",
response="ee74d173c7c03094af9aeb4ab5acaf6f"
Call-ID: f51f014d589e1ca7@192.168.1.11
SIP/2.0 407 Proxy Authentication Required
Call-ID: f51f014d589e1ca7@192.168.1.11
Proxy-Authenticate: Digest realm="192.168.1.15",
nonce="462c7effba07312005e1116c8236295cad5c0c84"
ACK sip:10@192.168.1.15 SIP/2.0
Call-ID: f51f014d589e1ca7@192.168.1.11
```

The full list of messages of this scenario is available in the OpenSER printout [htm], [txt], [doc].

6. Consuming the credentials

In the configuration script block handling the invite request we call the consume_credentials() function before processing the INVITE request:

```
route[3]
{
  if(!proxy_authorize("","subscriber")) {
    proxy_challenge("","0");
    exit;
  } else if (!check_from()) {
    sl_send_reply("403","Use From=ID");
    exit;
  }
  consume_credentials();
  lookup("aliases");
  if(uri!=myself) {
    route(1);
    exit;
  }
  if(!lookup("location")) {
    sl_send_reply("404","User Not Found");
    exit;
  }
  route(1);
}
```

[configuration file]

The consume_credentials() function removes the credentials from the message being processed by the server. That means that when re-transmitting the message to the next hop, the message will not contain anymore the credentials used by the server. The proxy should not reveal the information about the credentials of the user and the message will be also a little bit shorter.

To demonstrate the effect of the consume_credentials() function we use a configuration file, which, after authentication of an INVITE request, sends the

message to the proxy server itself in order to display the message upon its arrival and examine whether the credentials were removed or not.

The configuration file displays only the contents of INVITE request. Therefore during a call setup we should see three INVITE requests: the first one sent by UA without credentials, the second one sent by UA with credentials (after the challenge of the proxy), and the third one sent by proxy to itself after successful authentication. The third INVITE request corresponds to the one, which is sent to the destination UA or to the next-hop proxy. When the consume_credentials() function is commented the third INVITE request (sent out by proxy after successful authentication) still contains the Proxy-Authorization header field of the originating UA:

```
0(4649) INVITE from 192.168.1.10
INVITE sip:11@192.168.1.15 SIP/2.0
From: <sip:10@192.168.1.15>;tag=c9eb088d452d8629
To: <sip:11@192.168.1.15>
Call-ID: bcf44bf2657e2d7c@192.168.1.10
 0(4649) Processing by INVITE handler ...
 0(4649) INVITE from 192.168.1.10
INVITE sip:11@192.168.1.15 SIP/2.0
From: <sip:10@192.168.1.15>;tag=c9eb088d452d8629
To: <sip:11@192.168.1.15>
Proxy-Authorization: Digest username="10", realm="192.168.1.15",
algorithm=MD5, uri="sip:11@192.168.1.15",
nonce="462c9b46ef5a432a0e580d90bbe1485d35803d4f",
response="f3e283276058a016cd96cf33e8c9e8bb"
Call-ID: bcf44bf2657e2d7c@192.168.1.10
 0(4649) Processing by INVITE handler ...
 0(4649) Sending out first to myself for examining the content
 0(4649) INVITE from 192.168.1.15
INVITE sip:11@192.168.1.11 SIP/2.0
From: <sip:10@192.168.1.15>;tag=c9eb088d452d8629
To: <sip:11@192.168.1.15>
Proxy-Authorization: Digest username="10", realm="192.168.1.15",
algorithm=MD5, uri="sip:11@192.168.1.15",
nonce="462c9b46ef5a432a0e580d90bbe1485d35803d4f",
response="f3e283276058a016cd96cf33e8c9e8bb"
Call-ID: bcf44bf2657e2d7c@192.168.1.10
 0(4649) Self-Looped INVITE is received
 0(4649) The credentials are being forwarded to the next hop:
 0(4649) Digest username="10", realm="192.168.1.15", algorithm=MD5,
uri="sip:11@192.168.1.15", nonce="462c9b46ef5a432a0e580d90bbe1485d35803d4f",
response="f3e283276058a016cd96cf33e8c9e8bb"
```

All messages of the dialog are in the full printout [htm], [txt], [doc].

When the consume_credentials() function is called after the successful authentication, the third INVITE request of the printout (which is the one sent out by the proxy after the successful authentication), does not contain anymore the Proxy-Authorization header field of the originating UA:

```
0(4827) INVITE from 192.168.1.10
INVITE sip:11@192.168.1.15 SIP/2.0
From: <sip:10@192.168.1.15>;tag=1c43b02579e58415
To: <sip:11@192.168.1.15>
Call-ID: 40dc3876fa6602d7@192.168.1.10
 0(4827) Processing by INVITE handler ...
 0(4827) INVITE from 192.168.1.10
INVITE sip:11@192.168.1.15 SIP/2.0
From: <sip:10@192.168.1.15>;tag=1c43b02579e58415
To: <sip:11@192.168.1.15>
Proxy-Authorization: Digest username="10", realm="192.168.1.15",
algorithm=MD5, uri="sip:11@192.168.1.15",
nonce="462c9cc7efab60dc0884aa5506b1ab7598e67b48",
response="61ddee4ae8bca6ff75a056e083e6b7dd"
```

```
Call-ID: 40dc3876fa6602d7@192.168.1.10
 0(4827) Processing by INVITE handler ...
 0(4827) Sending out first to myself for examining the content
 0(4827) INVITE from 192.168.1.15
INVITE sip:11@192.168.1.11 SIP/2.0
From: <sip:10@192.168.1.15>;tag=1c43b02579e58415
To: <sip:11@192.168.1.15>
Call-ID: 40dc3876fa6602d7@192.168.1.10
 0(4827) Self-Looped INVITE is received
 0(4827) No credentials of the originating user are found
```

All messages of the dialog using the consume_credentials() function are available in a full printout of the server [htm], [txt], [doc].

7. Final configuration file without debug re-transmissions

The final configuration file authenticating both, the registration and call setup attempts, may look as follows:

```
route
{
  ...
  xlog("L_NOTICE","$rm from $si at $Tf\n");
  if(method!="REGISTER") record_route();
  if(loose_route()) route(1);
  if(uri!=myself) route(1);
  if(method=="ACK") route(1);
  else if (method=="INVITE") route(3);
  else if (method=="REGISTER") route(2);
  ...
}

route[1]
{
  if(!t_relay()) sl_reply_error();
  exit;
}

route[2]
{
  sl_send_reply("100","Trying to Register");
  if(!www_authorize("","subscriber")) {
    www_challenge("","0");
    exit;
  }
  if(!check_to()) {
    sl_send_reply("401","Unauthorized");
    exit;
  }
  consume_credentials();
  if(!save("location")) {
    sl_reply_error();
  }
  exit;
}

route[3]
{
  if(!proxy_authorize("","subscriber")) {
    proxy_challenge("","0");
    exit;
```

```
  }
  if (!check_from()) {
    sl_send_reply("403","Forbidden");
    exit;
  }
  consume_credentials();
  lookup("aliases");
  if(uri!=myself) {
    route(1);
  }
  if(!lookup("location")) {
    sl_send_reply("404","User Not Found");
    exit;
  }
  route(1);
}
```

In our example of the configuration file we still use xlog("L_NOTICE","...") function calls to monitor the SIP messages. The printout of a call setup scenario with this configuration is logged [htm], [txt], [doc]. You can remove all xlog("L_NOTICE","...") function calls from the script for running OpenSER in the background mode.

8. Glossary

BNF	Backus Normal Form or Backus-Naur Form
Base64	Positional notation using a base of 64
HTTP	Hyper Text Transfer Protocol
MD5	Message Digest version 5
OpenSER	a spin-off of SER project
RFC	Request for Comment
SER	SIP Express Router
SIP	Session Initiation Protocol
SQL	Structured Query Language
UA	User Agent
UDP	User Datagram Protocol
URI	Universal Resource Identifier

9. Relevant links

Examining the STUN settings of a SIP phone

Creating and sending INVITE and CANCEL SIP text messages

Direct calls between two SIP phones without passing through a SIP proxy

SIP messages, transactions, and dialogs (understanding SIP exchanges by experimentation)

The path of SIP signaling messages (understanding Via, Record-Route, and Route headers)

SIP transaction flags in OpenSER

Looping SIP messages in an OpenSER proxy (viewing messages transmitted by server)

SIP proxy with a user location database and authentication of registration and call setup requests

9.1. Local files

This document is available in web and printable formats [htm], [doc], [pdf]. This web page is available for a download [zip].