==========

Modified Steps to Run MUD Controller and FreeRadius

Megan Massey
Lalka Rieger

==========

Setting up FreeRadius Server on Ubuntu Virtual Machine

1. Ensure your system is updated:
   $ sudo apt-get update
   $ sudo apt-get upgrade

2. In your home directory, download FreeRADIUS 3.0x Series-Stable using wget

   $ wget https://github.com/FreeRADIUS/freeradius-server/archive/release_3_0_11.tar.gz

3. Certain packages may be missing. Ensure that libtalloc and libcrypto are installed. If not, run the following commands:

   $ sudo apt-get install libtalloc-dev
   $ sudo apt-get install libssl-dev

4. Unzip the resulting .tar.gz file by running the following command:

   $ tar -xvzf release_3_0_11.tar.gz

5. Enter the freeradius-server-release_3_0_11 directory and configure, make, and install by running the following commands:

   $ ./configure
   $ make
   $ sudo make install

6. All the associated FreeRADIUS folders should be in the appropriate places now. Create a new vendor dictionary file called dictionary.mudserver in /usr/local/share/freeradius/ and copy and paste the below text:

```
#########################################################################
VENDOR          CISCO-IOT       16122

BEGIN-VENDOR    CISCO-IOT

ATTRIBUTE       Cisco-MUD-URI   1       string

END-VENDOR CISCO-IOT

#########################################################################
```

7. Open usr/local/share/freeradius/dictionary file and locate the lines:

   $INCLUDE dictionary.motorola
   $INCLUDE dictinary.motrola.wimax

   and add the following on the next line:
   $INCLUDE dictionary.mudserver

8. To create a user, add a User-Name called <username> and Cleartext-Password <password> in /usr/local/etc/raddb/users file under the following lines:

   #bob Cleartext-Password := “hello”
   # Reply-Message := “Hello, %{User-Name}”

   The added line should look like the following example:
   megan Cleartext-Password := “testing123”

9. You may leave /usr/local/etc/raddb/clients.conf as it is if you are only working with localhost IP

10. Change the exec configuration file in /usr/local/etc/raddb/mods-enabled/exec from wait “no” to “yes”.

11. In /usr/local/etc/raddb/sites-enabled/default, add the following code in the “authorize” section after “filter_username”:

```
if (User-Name == "%{exec:/usr/bin/python /usr/local/etc/controller/mud_controller.py ‘null’ 'U1'}.in") {
        update control {
                Auth-Type := Accept
        }
}

if (User-Name == "%{exec:/usr/bin/python /usr/local/etc/controller/mud_controller.py 'null' 'U2'}.out") {
        update control {
                Auth-Type := Accept
        }
}
```

    Be sure that no newline character are inserted in the middle of the long commands

12. In the same file at the “post-auth” section add the below code after “exec”

```
        if (Cisco-MUD-URI) {
                if (User-Name == "<username>") {
                update reply {
                        Exec-Program = "%{exec:/usr/bin/python /usr/local/etc/controller/mud_controller.py %{Cisco- MUD-
URI} 'W'}"
                        Cisco-AVPair := "ACS:CiscoSecure-Defined-ACL=%{exec:/usr/bin/python /usr/local/etc/controller/
mud_controller.py 'null' 'U1'}.in",
                        Cisco-AVPair += "ACS:CiscoSecure-Defined-ACL=%{exec:/usr/bin/python /usr/local/etc/controller/
mud_controller.py 'null' 'U2'}.out"
                        }
                }

        if (User-Name == "%{exec:/usr/bin/python /usr/local/etc/controller/mud_controller.py 'null' 'U1'}.in") {
        update reply {
                User-Name = "%{exec:/usr/bin/python /usr/local/etc/controller/mud_controller.py 'null' 'U1'}",
                Cisco-AVPair := "ip:inacl#1=%{exec:/usr/bin/python /usr/local/etc/controller/mud_controller.py ‘null’ ‘R1’}”,
                Cisco-AVPair += "ip:inacl#2=permit udp any any eq 67",
                Cisco-AVPair += "ip:inacl#3=permit udp any any eq 68",
                Cisco-AVPair += "ip:inacl#4=%{exec:/usr/bin/python /usr/local/etc/controller/mud_controller.py ‘null’ ‘R3’}”,
                Reply-Message += "DACL Ingress Downloaded Succesfully.”,
                        }
                }

        if (User-Name == "%{exec:/usr/bin/python /usr/local/etc/controller/mud_controller.py 'null' 'U2'}.out") {
        update reply {
                User-Name = "%{exec:/usr/bin/python /usr/local/etc/controller/mud_controller.py 'null'
'U2'}. out",
                Cisco-AVPair := "ip:outacl#1=%{exec:/usr/bin/python /usr/local/etc/controller/mud_controller.py 'null' 'R2'}",
                Cisco-AVPair += "ip:outacl#2=permit udp any any eq 67",
                Cisco-AVPair += "ip:outacl#3=permit udp any any eq 68", Cisco-AVPair += "ip:outacl#4=%{exec:/usr/bin/python
/usr/local/etc/controller/mud_controller.py 'null' 'R3'}",
                Reply-Message += "DACL Egress Downloaded Succesfully."
                        }
                }
        }
```

In this part, make sure that in the line that says “if (User-Name == “<username>)”, change <username> to the username you have assigned in the users file.

13. Replace the tls.c file under freeradius-server-release_3_0_11/src/main/ with the tls.c file from our github

14. Under freeradius-server-release_3_0_11/share/ open dictionary.freeradius.internal:

    Locate the line:
    ATTRIBUTE TLS-PSK-Identity 1933 string

    Add below it:
    ATTRIUBTE TLS-Client-Cert-Subject-Alt-Name-URI 1934 string

15. Change directories back to the FreeRADIUS release. Reconfigure, make, and install.

    ```
    $ cd ..
    $ ./configure
    $ make
    $ sudo make install
    ```

16. Attempt to start FreeRADIUS in debugging mode by running the following command:

    ```
    $ sudo radiusd -x
    ```

If this does not work, make sure that FreeRADIUS is not already running as a process. To check, run the following commands:

```
$ ps aux
find the pid of FreeRADIUS

$ kill -9 <pid of FreeRADIUS>
```

After this, you should be able to run FreeRADIUS

At enterprise side where MUD files are stored

1. In your home directory, make a project folder entitled “elear_mud”

2. Underneath this project directory, create a new directory named “mud’ and one called “private”

   ```
   $ mkdir mud private
   ```

   Follow this link to generate MUD file
   https://www.ofcourseimright.com/mudmaker/

   Once you have specified your options and details, click on submit button to generate the MUD file

   Copy and paste the mud string and paste it on new mud file named with .json extension and save it in the mud directory

3. In the mud folder, generate key and self-signed certificate using the following commands:

   ```
   $ openssl genrsa -out key.pem 2048
   $ openssl req -new -x509 -key key.pem -out ck.pem
   ```

4. Sign and verify the MUD file using the following commands:

   ```
   $ openssl cms -sign -in <name of MUD file>.json -signer ck.pem -inkey key.pem -binary
         -outform DER -out <name of MUD file>.p7s
   $ openssl cms -verify -in  <name of MUD file>.p7s -out mud.json -CAfile ck.pem -inform
         DER -content <name of MUD file>.json
   ```

The output of the second command should be “Verification successful”

5. If you receive the following error message: “unable to write ‘random state,’” use elevated permissions

6. Move key.pem to the private folder

   ```
   $ mv key.pem ~/elear_mud/private
   ```

7. Change permissions on the private folder as well as the key

    $ chmod 700 ~/elear_mud/private
    $ chmod 700 ~/elear_mud/private/key.pem

8. Create a directory called “controller” under /usr/local/etc

9. Download and copy mud_controller.py file into /usr/local/etc/controller directory from our private github (still in process of debugging).

10. Copy ck.pem to the controller directory by running the following command from the elear_mud/mud directory

    $ cp ck.pem /usr/local/etc/controller

11. Run SimpleHTTPServer in the elear_mud directory:

    $ python -m SimpleHTTPServer

This command will by default run the HTTP server on port 8000. If you wish, you can specify a port with your IP address by running the following command:

    $ python -m SimpleHTTPServer 127.0.0.1:8080

Testing MUD Controller

1. With SimpleHTTPServer running, run the following command in the controller directory:

    $ python mud_controller.py http://localhost:8000/mud/<name of MUD file>.json W

On success, the output will be “Verification successful” and the following three files should appear in the controller directory:

    <name of MUD fille>.json
    <name of MUD file>.p7s
    mud.json

The “W” option tells the controller to get the MUD file and signature file from the MUD URI and verify the signature and store the MUD file.

Testing the Radius Client

1. Start the radius server in debugging mode.

2. With SimpleHTTPServer still running, run the following command:

    $ echo “User-Name=<username>, User-Password=<password>, Cisco-MUD-URI=http://127.0.0.1:8000/mud/<name of MUD file>.json” | /usr/local/bin/radclient -x localhost:1812 auth testing123

Change <username> and <password> to username and password specified in the users file

3. On success, the output will indicate that an Access-Request Id was sent and an Access-Accept Id was received.

Appendix
Supplicant Authentication with Certificates (in progress)

1. Generate client key

   $ openssl genrsa -out light.key

2. Make certificate request config file mudREQ.conf:

```
# The main section is named req because the command we are using is req
# (openssl req ...)
[ req ]
# This specifies the default key size in bits. If not specified then 512 is
# used. It is used if the -new option is used. It can be overridden by using
# the -newkey option.
default_bits = 2048

# This is the default filename to write a private key to. If not specified the
# key is written to standard output. This can be overridden by the -keyout
# option.
default_keyfile = light.key

# If this is set to no then if a private key is generated it is not encrypted.
# This is equivalent to the -nodes command line option. For compatibility
# encrypt_rsa_key is an equivalent option.
encrypt_key = no

# This option specifies the digest algorithm to use. Possible values include
# md5 sha1 mdc2. If not present then MD5 is used. This option can be overridden
# on the command line.
default_md = sha1

# if set to the value no this disables prompting of certificate fields and just
# takes values from the config file directly. It also changes the expected
# format of the distinguished_name and attributes sections.
prompt = no

# if set to the value yes then field values to be interpreted as UTF8 strings,
# by default they are interpreted as ASCII. This means that the field values,
# whether prompted from a terminal or obtained from a configuration file, must
# be valid UTF8 strings.
utf8 = yes

# This specifies the section containing the distinguished name fields to
# prompt for when generating a certificate or certificate request.
distinguished_name = my_req_distinguished_name

# this specifies the configuration file section containing a list of extensions
# to add to the certificate request. It can be overridden by the -reqexts
# command line switch. See the x509v3_config(5) manual page for details of the
# extension section format.
req_extensions = my_extensions

[ my_req_distinguished_name ]
C = PT
ST = Lisboa
L = Lisboa
O  = Oats In The Water
CN = *.oats.org

[ my_extensions ]
subjectAltName=@my_subject_alt_names

[ my_subject_alt_names ]
URI = http://localhost/mud/lighting-example.json
```

3. Generate request

```
$ openssl req -new light.csr -config mudREQ.conf

Generating a 2048 bit RSA private key
. . . . . . . . . . . . +++
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . +++
writing new private key to 'oats.key'
— — — —
```

4. Optionally read request contents:

```
$ openssl req -in light.csr -noout -text
```

Create CA

5. Generate ca's key:

```
$ openssl genrsa -out mudCA.key 2048
Generating RSA private key, 2048 bit long modulus
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . +++
. . . . . . . . . +++
e is 65537 (0x10001)
```

6. Generate self-signed certificate for CA:

```
$ openssl req -new -x509 -key mudCA.key -out mudCA.crt

You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:PT
State or Province Name (full name) [Some-State]:Lisboa
Locality Name (eg, city) []:Lisboa
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Sz CA
Organizational Unit Name (eg, section) []:SZ CA
Common Name (e.g. server FQDN or YOUR name) []:
Email Address []:An optional company name []:
```

7. Create CA configuration file ca.conf:

```
# we use 'ca' as the default section because we're usign the ca command
# we use 'ca' as the default section because we're usign the ca command
[ ca ]
default_ca = my_ca

[ my_ca ]
#  a text file containing the next serial number to use in hex. Mandatory.
#  This file must be present and contain a valid serial number.
serial = ./serial

# the text database file to use. Mandatory. This file must be present though
# initially it will be empty.
```

```
database = ./index.txt
# specifies the directory where new certificates will be placed. Mandatory.
new_certs_dir = ./newcerts

# the file containing the CA certificate. Mandatory
certificate = ./mudCA.crt

# the file contaning the CA private key. Mandatory
private_key = ./mudCA.key

# the message digest algorithm. Remember to not use MD5
default_md = sha1

# for how many days will the signed certificate be valid
default_days = 365

# a section with a set of variables corresponding to DN fields
policy = my_policy

[ my_policy ]
# if the value is "match" then the field value must match the same field in the
# CA certificate. If the value is "supplied" then it must be present.
# Optional means it may be present. Any fields not mentioned are silently
# deleted.
countryName = match
stateOrProvinceName = supplied
organizationName = supplied
```