Install an IdP on Debian 8

On this page you will find instructions on how to set up a Moonshot Identity Provider (IdP) on Debian 8 (Jessie) using FreeRADIUS. It also installs and configures the Trust Router client, if you are going to use the Trust Router infrastructure.

Contents

Complexity of Installation
Many of the steps outlined below are currently necessary, but we realise the install should be simpler. As the software matures and the packaging improves, we will to make this process easier with fewer manual steps required.

1. System Preparation

1.1. Install Debian 8

The first thing that is required is a Debian 8 machine - this can be physical or virtual.

1. Install Debian 8 (Jessie) via usual mechanism (e.g., netboot CD, ISO in VMware/VirtualBox or the DVD image.
2. Choose the following server install options: "Debian desktop, SSH server, Standard system utilities".
3. Create/choose a secure root password and an initial system user account.
4. Once installed, make sure you run an `apt-get update` and `apt-get upgrade` to ensure your system is fully up to date.

Tip
We would recommend using LVM when disk partitioning to allow easier partition/disk expansion on a live system.

Warning
After install, you will want to secure/lockdown the server as best practice dictates - for both the server and any extra software installed. This is beyond the remit of this guide but there are many guides available that provide information on securing your Debian servers and applications.

1.2. Configure Debian 8

Next, there are a few Debian configuration options that need to be set in advance.

1.2.1. Networking configuration

For production deployments, it is recommended that the machine be assigned a static IP address.

For Debian networking information please refer to the Debian documentation: https://wiki.debian.org/NetworkConfiguration

1.2.2. Firewall configuration

The following ports are required to be accessible from the outside world, both in the local firewall and in any external firewalls:

- 2083/tcp (for RadSec connections to other Moonshot entities)
- 12309/tcp (for Trust Router client connections - if using the Trust Router to broker trust relationships between entities)

1.3. Add the Moonshot Repository

1. Add the Moonshot Debian Jessie repository to your system. To do this, run the following command (as root, or using sudo):

```
$ echo "deb http://repository.project-moonshot.org/debian-moonshot
jessie main" > /etc/apt/sources.list.d/moonshot.list
```

2. Install the Moonshot GPG key:

```
$ wget -O - http://repository.project-moonshot.org/key.gpg | apt-key
add -
```

> **Verifying the Moonshot GPG key**
> If you wish to verify the Moonshot GPG key's validity and integrity, please see the Packaging GPG Key for further details.

3. Update the apt cache with the new repository information:

```
$ apt-get update
```

2. Install the Moonshot IdP

We're now ready to install the Moonshot software and its required dependencies. Install the software by running the following command:

```
$ apt-get install moonshot-gss-eap moonshot-ui moonshot-trust-router
freeradius-abfab freeradius-utils dbus-x11
```

3. Configure the Moonshot IdP

Next, we need to configure the Moonshot IdP.

3.1. Configure FreeRADIUS

3.1.1. Certificates

We need to get FreeRADIUS to create some private and public keys to use for its RadSec connections. Create and install the certificates by doing the following (as root).

1. Change into the `/etc/freeradius/certs` directory

```
$ cd /etc/freeradius/certs
```

2. Edit the certificate generation properties in *client.cnf*, *server.cnf*, and *ca.cnf* as follows:
   a. In the `ca.cnf` file:
      i. In the `[ req ]` section, add `encrypt_key = no`
      ii. In the `[CA_default]` section, change the default_days from 60 to a higher number (this is how long the certificates you create will be valid for). When the certificates expire, you will have to recreate them.
      iii. in the `[ certificate_authority ]` section, change all of the parameters to match those of your organisation. e.g.

```
[certificate_authority]
countryName             = GB
stateOrProvinceName     = England
localityName            = Camford
organizationName        = Camford University
emailAddress            = support@camford.ac.uk
commonName              = "Camford University FR Certificate
Authority"
```

   b. In the `server.cnf` file:
      i. In the `[ req ]` section, add `encrypt_key = no`
      ii. In the `[CA_default]` section, change the default_days from 60 to a higher number (this is how long the certificates you create will be valid for). When the certificates expire, you will have to recreate them.
      iii. in the `[ server ]` section, change all of the parameters to match those of your organisation. e.g.

```
[server]
countryName             = GB
stateOrProvinceName     = England
localityName            = Camford
organizationName        = Camford University
emailAddress            = support@camford.ac.uk
commonName              = "Camford University FR Server
Certificate"
```

> When changing passwords in the `[ req ]` section of the server.cnf file, you must also update the `private_key_password` option in the FreeRADIUS `mods-available/eap` file with the same password.
>
> We recommend that you do **not** change these defaults.

   c. In the `client.cnf` file:
      i. In the `[ req ]` section, add `encrypt_key = no`
      ii. In the `[CA_default]` section, change the default_days from 60 to a higher number (this is how long the certificates you create will be valid for). When the certificates expire, you will have to recreate them.
      iii. in the `[ client ]` section, change all of the parameters to match those of your organisation. e.g.

```
[client]
countryName             = GB
stateOrProvinceName     = England
localityName            = Camford
organizationName        = Camford University
emailAddress            = support@camford.ac.uk
commonName              = "Camford University FR Client
Certificate"
```

> All of the organisation parameters (`countryName`, `localityName`, etc) need to match in the three .cnf files but the `commonName` must be unique in each file)

3. Clear out any old certificates in the directory:

```
$ make destroycerts
```

4. Run the bootstrap script to generate the certificates

```
$ ./bootstrap
```

5. Create a file that is the concatenation of the certificate and private key of the client.
   a. Create the file

```
$ openssl x509 -in client.crt > client.pem ; cat client.key >>
client.pem
```

   b. Verify that the client.pem file starts with "`-----BEGIN CERTIFICATE-----`".

6. Because the above command was run as root, the keys and certificates created will not be readable by the FreeRADIUS user by default, and FreeRADIUS will not be able to start. To fix this, reset the group for the files:

```
$ chgrp freerad {client,server,ca,dh}*
```

3.1.2. Moonshot UI credential store

We need to enable the freeradius user to use the Moonshot UI flatstore:

```
$ echo "freerad" >> /etc/moonshot/flatstore-users
```

3.1.3. Set up the FreeRADIUS and Trust Router users

To allow FreeRADIUS to read a key database for dynamic realm support, we need to place the FreeRADIUS user and the Trust Router users into each other's groups to allow them to read shared files of each other.

```
$ adduser freerad trustrouter
$ adduser trustrouter freerad
```

Additional freerad user configuration
Verify that the home directory for the `freerad` user exists. On this platform it should be `/etc/freeradius`.

3.1.4. RadSec

Next, we need to configure RadSec. We do this by creating a file at `/etc/radsec.conf` with the following:

```
realm gss-eap {
 type = "TLS"
 cacertfile = "/etc/freeradius/certs/ca.pem"
 certfile = "/etc/freeradius/certs/client.pem"
 certkeyfile = "/etc/freeradius/certs/client.key"
 disable_hostname_check = yes
 server {
  hostname = "127.0.0.1"
  service = "2083"
  secret = "radsec"
 }
}
```

3.1.5. Dynamic Realm support

We need to tell your FreeRADIUS server to support dynamic lookup of realms.

1. Open `/etc/freeradius/proxy.conf` for editing:
   a. Towards the top of the file is a stanza beginning "`proxy server {`". Find this.
   b. Below this, add `dynamic = yes`, like so:

```
proxy server {
        dynamic = yes
```

3.1.6. Realm

We need to configure your realm in the FreeRADIUS server so that it knows not to send any requests for your own users off to another server.

1. Configure your realm in `/etc/freeradius/proxy.conf`:
   a. Open the file for editing and find the line `realm example.com {`
   b. Above this, add the following, where YOUR_REALM should be substituted by your realm (e.g. camford.ac.uk):

```
realm YOUR_REALM {
 # Intentionally left blank
}
```

3.1.7. Channel Binding Support

We need to configure your FreeRADIUS server to support channel bindings.

1. Open `/etc/freeradius/sites-available/abfab-tls` for editing:
    a. Scroll to the `client default` stanza at the bottom of the file
    b. Edit the stanza to match the below:

```
client default {
        ipaddr = 0.0.0.0/0
        proto = tls
        gss_acceptor_realm_name = "your IDP realm here"
        trust_router_coi = ov-apc.moonshot.ja.net
}
```

> **gss_acceptor_realm_name**
> Specify the same RP realm as in the `rp_realm` option in Section 4.1 below. For simple IdP deployments, this usually matches your IDP Realm. When running a mixed IdP-RP Proxy deployment, follow the advice for an RP Proxy.
>
> Additionally, you must add a domain wildcard constraint in the Jisc Assent Portal that will match this realm value.

    c. If you have any other client definitions here, for example to distinguish between internal and external clients, also apply the change to them.

3.1.8. EAP Type

1. Set the EAP type in use by moonshot (EAP-TTLS) by editing `/etc/freeradius/mods-enabled/eap`. Find the first instance of `default_eap_type = md5` and change it to TTLS.

```
default_eap_type = ttls
```

> Other EAP types should be supported (PEAP and MD5 have been tested).

3.1.9. User Authentication

FreeRADIUS offers many options on to authenticate users; common ones including using a simple local flat file (useful for initial testing), or for production deployments using a credential store in an SQL database or a connection to LDAP/AD.

> To see the full range of options available, and find out how to configure them, visit the FreeRADIUS site.

For the purposes of initial testing, we will use a simple local flat file, creating a user with username "testuser" and password "testing".

1. Open `/etc/freeradius/users` for editing and put the following at the top of the file

```
testuser Cleartext-Password := "testing"
   Reply-Message = "Hello test user. You have authenticated!"
```

> The formatting of the stanza above is very important. There should be a <tab> in between the username and Cleartext-Password, and a line break followed by a <tab> before the Reply-Message.

4. Configure the Trust Router Client

If you are going to connect your Moonshot IdP to a Trust Router network, then the next step involves configuring the Trust Router client software and configuring its connection to a Trust Router.

4.1. Configure FreeRADIUS to use Trust Router

4.1.1. Configuring FreeRADIUS realm lookup

We need to configure the community and rp_realm appropriate for your Moonshot service, and the Trust Router that it will connect to.

1. Open the `/etc/freeradius/mods-enabled/realm` for editing.
2. Find the "realm suffix {" configuration directive, and fill out the fields as appropriate.
3. For the default Jisc Assent Trust Router this will look like the following:

```
realm suffix {
  format = suffix
  delimiter = "@"
  default_community = "ov-apc.moonshot.ja.net"
  rp_realm = "Your service realm as registered in the Jisc Assent
Portal"
  trust_router = "tr.moonshot.ja.net"
}
```

Example
Camford University has a Moonshot service registered in the Jisc Assent Portal using the service realm of moonshot.camford.ac.uk, so its realm file would look like this:

```
realm suffix {
  format = suffix
  delimiter = "@"
  default_community = "ov-apc.moonshot.ja.net"
  rp_realm = "moonshot.camford.ac.uk"
  trust_router = "tr.moonshot.ja.net"
}
```

4.1.2. Register your Trust Router client with a Trust Router

At this point, the Moonshot service needs to be associated with a Trust Router. To do this, you need to contact the operator of a Trust Router you wish to join for their specific instructions on how to do this.

Once you have joined the Trust Router service, you will be issued with a Trust Router credential file in XML file format.

Keep this credential file safe. It usually will only be issued once and any subsequent requests usually invalidate any previously issued credentials. This is a security precaution.

Jisc Assent service instructions
The below instructions are specific to the world's first Trust Router service, Jisc Assent, operated by Jisc in the United Kingdom:

1. If you are not signed up to Assent, sign up to Assent first. This step may take a day or two while your organisation details are verified and you are invited to join the portal.

2. If you are signed up to Assent, log into the Assent portal.

   > For more information about the Assent Portal, see the Assent Portal Primer.

3. Download a Trust Router credential under the 'Credential' section of your organisation in the portal (in the form of an XML file). Keep this file safe!

1. You must import the issued credential file using the `moonshot-webp` command as the `freerad` user:

```
$ su - --shell /bin/bash freerad
$ unset DISPLAY
$ moonshot-webp -f [path to credential file]
```

2. Check that the credential has been correctly imported:

```
$ ls -la /etc/freeradius/.local/share/moonshot-ui/identities.txt
```

3. If the file exists, the credential file's contents should be present in the file.

4.2. Configure TIDS

The IdP also runs the Temporary ID Server (TIDS).

1. Open the `/etc/default/trust_router` file for editing. If necessary, create it.

```
ipaddr="[your server IP]"
hostname="[your server hostname]"
gssname="trustrouter@ov-apc.moonshot.ja.net"

TIDS_USER="trustrouter"
TIDS_GROUP="trustrouter"
```

5. Testing

Now that we have the Moonshot IdP installed and configured, we're now ready to test!

> **Tip**
> At this point you probably want three consoles open on the server, so that you can manually run various components separately.

5.1. Testing FreeRADIUS locally

The first test is to check whether FreeRADIUS is working in its most basic manner.

1. In window 1, run (as the freerad user)

```
$ su --shell /bin/bash freerad
$ unset DISPLAY
$ freeradius -fxx -l stdout
```

2. Check that no errors are output.
3. In window 2, run (as root user)

```
$ radtest testuser@YOURREALM testing localhost 2222 testing123
```

This uses the "radtest" utility which is used in the following way - *radtest username password servername port shared-secret*

4. If this is working correctly you should see something like the following:

In window 1 - FreeRADIUS server output

```
Sending Access-Accept of id 57 from 127.0.0.1 port 1812 to 127.0.0.1
port 33363
     Reply-Message = 'Hello test user. You have authenticated!'
(1) Finished request 1.
Waking up in 0.3 seconds.
Waking up in 4.6 seconds.
(1) Cleaning up request packet ID 57 with timestamp +94
Ready to process requests.
```

In window 2 - radtest client output

```
Sending Access-Request of id 57 from 0.0.0.0 port 33363 to 127.0.0.1
port 1812
     User-Name = 'testuser'
     User-Password = 'testing'
     NAS-IP-Address = 127.0.1.1
     NAS-Port = 2222
     Message-Authenticator = 0x00
rad_recv: Access-Accept packet from host 127.0.0.1 port 1812, id=57,
length=61
     Reply-Message = 'Hello test user. You have authenticated!'
```

5.2. Testing the Trust Router connection

To test the connection to Trust Router, we need to make sure the Temporary Identity Server (TIDS) software is running, then use the Temporary Identity Client (TIDC) software to simulate a connection to the Trust Router.

5.2.1. Testing using the Temporary Identity Client (TIDC)

1. In window 2, (as the freerad user) run the tidc command:

```
$ su --shell /bin/bash freerad
$ unset DISPLAY
$ tidc tr.moonshot.ja.net [your rp-realm] ov-apc.moonshot.ja.net
ov-apc.moonshot.ja.net
```

This uses the "tidc" binary which is used in the following way - *tidc [hostname-of-trust-router] [rp-realm] [hostname-of-apc-server] [apc-name]*

2. If the Trust Router connection was successful, you should see something like the following:

In window 2 - TIDC output

```
TIDC Client:
Server = tr.moonshot.ja.net, rp_realm = moonshot-idp.camford.ac.uk,
target_realm = ov-apc.moonshot.ja.net, community =
ov-apc.moonshot.ja.net
connecting to host 'tr.moonshot.ja.net' on port 12309
CTRL-EVENT-EAP-STARTED EAP authentication started
CTRL-EVENT-EAP-PROPOSED-METHOD vendor=0 method=21
CTRL-EVENT-EAP-METHOD EAP vendor 0 method 21 (TTLS) selected
CTRL-EVENT-EAP-PEER-CERT [...]
CTRL-EVENT-EAP-SUCCESS EAP authentication completed successfully
tidc_fwd_request: Sending TID request:

[...]

tr_msg_decode_tidresp(): Success! result = success.
tr_msg_decode_servers(): Number of servers = 1.
Response received! Realm = ov-apc.moonshot.ja.net, Community =
ov-apc.moonshot.ja.net.
Client Key Generated (len = 256):

[...]
```

5.2.2. Testing the Temporary Identity Server (TIDS)

1. In window 3 (as the trustrouter user, window 1 should still be still running the FreeRADIUS server and window 2 the TIDC command), run the TIDS software:

```
$ su --shell /bin/bash trustrouter
$ unset DISPLAY
$ tids [your server IP] trustrouter@ov-apc.moonshot.ja.net [your
server hostname] /var/lib/trust_router/keys
```

trustrouter@ov-apc.moonshot.ja.net is the identity that the trust router will use when provisioning keys - this makes it easy to spot in your own log files.
Specifying your server's IP *and* hostname may seem redundant (and for single server deployments, it is!). You'll need to set the hostname and IP arguments a little differently if you want to enable some more advanced configurations (such as load balancing and key sharing).

This uses the "tids" binary which is used in the following way - *tids [your-ip-address] trustrouter-gss-name] [your-hostname] [path-to-key-database]*

When using Network Address Translation (NAT) or a firewall, you must specify your external IP address.

2. In window 2, (as the freerad user) run the tidc command again, but this time modify it slightly and specifying the realm you defined in Section 3.1.5 above:

```
$ su --shell /bin/bash freerad
$ unset DISPLAY
$ tidc tr.moonshot.ja.net [your rp-realm] [YOUR_REALM]
ov-apc.moonshot.ja.net
```

This uses the "tidc" binary which is used in the following way - *tidc [hostname-of-trust-router] [rp-realm] [identity realm specified in Section 3.1.5] [apc-name]*

3. If the Trust Router connection was successful, you should see something like the following:

In window 2 - TIDC output

```
TIDC Client:
Server = tr.moonshot.ja.net, rp_realm = moonshot-idp.camford.ac.uk,
target_realm = camford.ac.uk, community = ov-apc.moonshot.ja.net
connecting to host 'tr.moonshot.ja.net' on port 12309
CTRL-EVENT-EAP-STARTED EAP authentication started
CTRL-EVENT-EAP-PROPOSED-METHOD vendor=0 method=21
CTRL-EVENT-EAP-METHOD EAP vendor 0 method 21 (TTLS) selected
CTRL-EVENT-EAP-PEER-CERT [...]
CTRL-EVENT-EAP-SUCCESS EAP authentication completed successfully
tidc_fwd_request: Sending TID request:

[...]

tr_msg_decode_tidresp(): Success! result = success.
tr_msg_decode_servers(): Number of servers = 1.
Response received! Realm = camford.ac.uk, Community =
ov-apc.moonshot.ja.net.
Client Key Generated (len = 256):

[...]
```

Additionally, in window 1, where FreeRADIUS is running, you should see something similar to this:

In window 1 - FreeRADIUS output

```
(9) Found Auth-Type = Accept
(9) Auth-Type = Accept, accepting the user
(9) # Executing section post-auth from file
/etc/raddb/sites-enabled/abfab-tr-idp
(9)   post-auth {
(9)     [exec] = noop
(9)     policy remove_reply_message_if_eap {
(9)       if (&reply:EAP-Message && &reply:Reply-Message) {
(9)       if (&reply:EAP-Message && &reply:Reply-Message)  -> FALSE
(9)       else {
(9)         [noop] = noop
(9)       } # else = noop
(9)     } # policy remove_reply_message_if_eap = noop
(9)   } # post-auth = noop
(9) Sent Access-Accept Id 0 from 0.0.0.0:2083 to 127.0.0.1:56352
length 196
(9)   Message-Authenticator = 0x856c08f200d29d641e486a1ccc48799a
(9)   User-Name = 'trustrouter@ov-apc.moonshot.ja.net'
(9)   MS-MPPE-Recv-Key =
0xac76ebd3df5fd9f11b57ad0fbc57b92175a8c8f7b4f3ae18bc4ccab5184e6158
(9)   MS-MPPE-Send-Key =
0xb891112bc014a06dbbeb8170d64136d2e0359a0c255d846fb5772f2733205782
(9)   EAP-Message = 0x03090004
(9) Finished request
Closing TLS socket from client port 56352
(0) >>> TLS 1.0 Alert [length 0002], warning close_notify
Client has closed connection
Waking up in 3.7 seconds.
(0) <done>: Cleaning up request packet ID 0 with timestamp +25
```

4. With the tests successful, you can now terminate the FreeRADIUS (window 1) and TIDS (window 3) processes.

6. Next Steps

At this point, you now have a Moonshot IdP that is working and registered with a Trust Router. Now for the next steps:

6.1. Automatically start the software

6.1.1. FreeRADIUS

To automatically start FreeRADIUS, issue the following command (as root):

```
$ sudo update-rc.d freeradius defaults
$ sudo service freeradius start
```

If this is working correctly, you should see FreeRADIUS running as a daemon process.

6.1.2. TIDS

To automatically start TIDS, issue the following command (as root):

```
$ sudo update-rc.d tids defaults
$ sudo service tids start
```

If this is working correctly, you should see TIDS running as a daemon process.

6.2. Configure a real source of Authentication

Your FreeRADIUS server can currently only authenticate a single user - "testuser". At this point, you will want to connect to Active Directory, LDAP, an SQL database, or some other source of credentials.

See Configuring FreeRADIUS to Use a Local Identity Store for more information and instructions for how to do this.

6.3. Integrate SAML

As currently configured, this Moonshot IdP can only use RADIUS attributes. If you wish to also include SAML assertions, visit the Issue SAML Assertions page to see the options available to you.

6.4. Configure clients

If you are going to also use your Moonshot IdP as a Moonshot RP (i.e., connect services to it that you wish to allow people to authenticate to using Moonshot), then see the Configure the Moonshot RP Proxy to Talk to Applications/Services page.