# Setting up a jitsi server on FreeBSD

## Introduction

This document details the steps required to set up a **jitsi** server on FreeBSD. It was written because there is quite a lot involved, and there are a lot of small things which existing instructions either don't explain at all, or which are explained in terms of Linux systems! This was done on FreeBSD 12.1; 12.0 should be fine but 11.x needs a patch to `rc.subr`. In addition, there are tips on enhancements to the basic installation.

See the end of this document for license details.

The emphasis here (mostly) is not just to be a recipe, but to explain *why* something is done. The setup is actually quite quick when it is all in one place.

This is a long document, because:

- It contains a lot of detail
- It contains explanations as well as directions
- It includes some optional enhancements
- Many pathnames are given in full, to avoid ambiguity

**Please note the changelog at the end of the document.**

## Packages required

The following packages (and their dependencies) are all required, so install them, or build from ports:

- net-im/jicofo (the **ji**tsi **co**nference **fo**cus component)
- net-im/jitsi-videobridge (the video bridge, which handles video communications)
- net-im/prosody (XMPP server)
- security/p11-kit (PKCS 11 module library)
- www/jitsi-meet (the pages and code for the jitsi server itself)
- www/nginx (HTTP server) [in principle, another server could be used]

It is strongly suggested that you install all of these packages **now**, as there are one or two cross dependencies. There are no port options to worry about, except for nginx; in that case, the defaults should be OK.

Other optional packages may also be needed if additional setup is done. These include:

- net-im/prosody-modules (additional modules for prosody)
- security/py-fail2ban (used for scanning logs and reporting/banning potential abuse). Note that the package name will include a Python version, e.g. py37-fail2ban.

These can be installed as required.

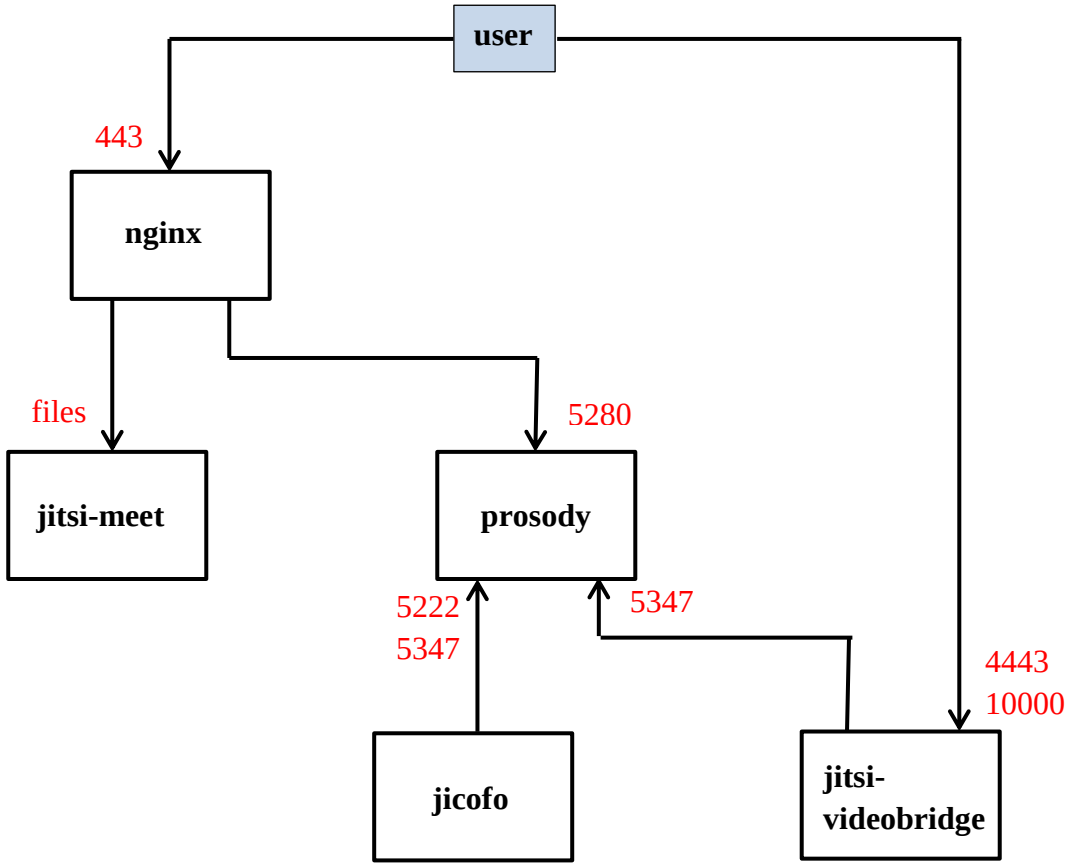
## Overview

In essence, the user visits a web page served by nginx. This uses prosody for authentication and communications. Users are coordinated by jicofo, and video communication takes place over a direct connection to the video bridge.

All this means that configuration is very distributed; hence, each component will be set up separately below. Work steadily through it, and all should be fine.

In all cases, pathnames specific to a normal FreeBSD installation have been used, in accordance with hier(7).

The number of packages and configuration settings can be a little daunting at first. The diagram below may help to explain how it all fits together. Port numbers/data are in red.




## Domain names

First, a fully qualified domain name must be chosen for the server. The base domain name will be one which can have new host names (or **CNAME**s if preferred) added easily; this is most easily done by having a cooperative ISP, or a local DNS server.

The base domain name used here is the well-worn `example.com`, and we choose to call the server `meet.example.com`. Other names can, of course, be used.

Other names are also used; and they should be added to DNS as well. Following the above naming convention, we will need the following. Other names can be substituted, but everything is easier to follow if these meaningful names are used :

- `meet.example.com`
- `auth.meet.example.com`
- `conference.meet.example.com`
- `focus.meet.example.com`
- `jitsi-videobridge.meet.example.com`

# Conventions

The following conventions are used to save space and multiple explanations below:

| Abbreviation | Meaning | Example |
|---|---|---|
| **BASEDN** | Base domain name | example.com |
| **FQDN** | Fully qualified domain name | meet.example.com |
| **SECRET1** | Password for the video bridge | See below |
| **SECRET2** | Password for the focus chooser | See below |
| **SECRET3** | Password for the authenticator | See below |
| **SECRET4** | Password for the jicofo trust store | See below |

The four passwords should be secure. Reasonable passwords can be generated in many ways; here is a suitable shell script:

```
#!/bin/sh
# generate random password
dd if=/dev/random count=1 bs=12 2>/dev/null | b64encode - | \
      sed -e 's/=*$//' -e '/^begin/d' -e '/^$/d'
```

To save confusion, there is no reason why you cannot add stuff to the passwords to indicate which is which, e.g.:

```
SECRET2-ab6$z/lsp9,y
```

# SSL/TLS

A certificate must be obtained for the **FQDN** web server; how to do this is outside the scope of this document.

One easy way is to use Lets Encrypt for free certificates; see https://letsencrypt.org. The FreeBSD package for this is `security/py-acme`.

Whatever you use, you *must* make sure that the certificate is renewed and replaced before it expires! In the case of Let's Encrypt, this is especially important as they only last three months. See the Let's Encrypt documentation for details on how to do this (of course, if you are not using Let's Encrypt you will have to make other arrangements; even a cron job sending mail, or the FreeBSD `calendar` program, will work for reminders). The instructions are not replicated here because it's better to have one up to date source.

# Configuration

Configuration is a little involved, because there are five packages to configure! These are described separately. All configuration is done as root, or using sudo.

## prosody

### Installation

First, ensure that prosody is installed, either from a package or from ports. There are no port options to worry about if it is done from ports.

### Configuration

On FreeBSD, the prosody configuration directory is `/usr/local/etc/prosody`, and the work takes place there. The main configuration file is called `prosody.cfg.lua`. It makes sense to keep the jitsi configuration separate from anything else that prosody might be used for. First, remove any existing example VirtualHost lines, and then add these lines to the end of that file:

```
pidfile = "/var/run/prosody/prosody.pid";

-- disable https ports (not used)

https_ports = { }
```

```
include "conf.d/*.cfg.lua"
```

The `pidfile` line is missing from the default configuration, but is required! The `https_ports` line eliminates some annoying warnings, but should not be used if prosody is being used for something else that needs https. The `include` line allows us to keep our own configurations separate.

Create the `conf.d` directory, and in there create a file called (say) **FQDN**`.cfg.lua`, containing the following (suitably modified):

```
VirtualHost "FQDN"
    ssl = {
        key = "/var/db/prosody/FQDN.key";
        certificate = "/var/db/prosody/FQDN.crt";
    }
    authentication = "anonymous"
    modules_enabled = {
        "bosh";
        "pubsub";
    }
    c2s_require_encryption = false

VirtualHost "auth.FQDN"
    ssl = {
        key = "/var/db/prosody/auth.FQDN.key";
        certificate = "/var/db/prosody/auth.FQDN.crt";
    }
    authentication = "internal_plain"
    admins = { "focus@auth.FQDN" }

Component "conference.FQDN" "muc"

Component "jitsi-videobridge.FQDN"
    component_secret = "SECRET1"

Component "focus.FQDN"
    component_secret = "SECRET2"
```

It will be noticed that the above mentions certificates for two of the domains, but these do not yet exist. Create them with the following:

```
$ prosodyctl cert generate FQDN
$ prosodyctl cert generate auth.FQDN
```

This will create the certificates in `/var/db/prosody`. It is sufficient to answer the prompts with defaults, making sure that the 'common name' is **FQDN** or auth.**FQDN** as appropriate. These self-signed certificates expire after a year; renew them in plenty of time just by using the above commands again.

You can now check that the configuration file is OK by issuing the command:

```
$ prosodyctl check config
```

There has to be a user for jicofo to log in with, and here it will be called `focus`. Register this user with prosody:

```
$ prosodyctl register focus auth.FQDN SECRET3
```

Note that the `register` command is not mentioned in the help text. It is a legacy version of the `adduser` command, with different syntax, that doesn't prompt for the password (twice). One could equally well use:

```
$ prosodyctl adduser focus@auth.FQDN
```

and then answer the password prompts with **SECRET3**.

Now trust the two certificates:

```
$ trust anchor /var/db/prosody/FQDN.crt
$ trust anchor /var/db/prosody/auth.FQDN.crt
```

### Logging

Return to the main prosody configuration directory, and locate the `log` stanza in `prosody.cfg.lua`. Replace it with this (it can always be tweaked later):

```
log = {
        info = "/var/log/prosody/prosody.log";
        error = "/var/log/prosody/prosody.err";
}
```

Create the directory `/usr/local/etc/newsyslog.conf.d`, if it doesn't already exist. Create the file `/usr/local/etc/newsyslog.conf.d/prosody`, containing:

```
/var/log/prosody/prosody.* prosody:prosody 600    7    *     @T03 JGNC
```

Modify this as desired. Then issue the commands:

```
$ newsyslog -C /var/log/prosody/prosody.log
$ newsyslog -C /var/log/prosody/prosody.err
```

to create the logfiles with the correct ownership and permissions.

### Starting up

Before starting prosody, you can perform a quick check on the configuration. This will point out some obvious potential problems;

```
$ prosodyctl check
```

You can ignore any warnings about DNS on IPv6. Lastly, enable and start prosody:

```
$ sysrc prosody_enable="YES"
$ service prosody start
```

Check the logfiles for any errors.

More information about prosody may be found at http://prosody.im.

## jicofo

### Installation

First, ensure that jicofo is installed, either from a package or from ports. There are no port options to worry about if it is done from ports.

### Configuration

On FreeBSD, the jicofo configuration directory is `/usr/local/etc/jitsi/jicofo`, and the work takes place there. The configuration file is named `jicofo.conf`.

Make these changes to `jicofo.conf`:

```
JVB_XMPP_DOMAIN=FQDN
JVB_XMPP_SECRET=SECRET2
JVB_XMPP_USER_DOMAIN=auth.FQDN
```

```
JVB_XMPP_USER_SECRET=SECRET3
```

Leave all of the other lines alone.

### Setting up the trust store

Jicofo needs a Java trust store containing the certificate for auth.**FQDN**. The trust store is, by default, located at `/usr/local/etc/jitsi/jicofo/truststore.jks`.

To import prosody's auth.**FQDN** TLS certificate into the trust store, use:

```
$ keytool -noprompt \
-keystore /usr/local/etc/jitsi/jicofo/truststore.jks \
-importcert -alias prosody \
-file /var/db/prosody/auth.FQDN.crt
```

When prompted (twice) for a trust store password, use **SECRET4**.

### Logging

By default, jicofo logs to the file `/var/log/jicofo.log`. Some changes to logging behaviour may be made by editing the file `/usr/local/share/jicofo/lib/logging.properties`, but this is not really necessary.

Create the file `/usr/local/etc/newsyslog.conf.d/jicofo`, containing:

```
/var/log/jicofo.log 600  7 *  @T03 JNC
```

Modify this as desired. Then issue the command:

```
$ newsyslog –C /var/log/jicofo.log
```

to create the logfile with the correct ownership and permissions.

### Starting up

Enable and start jicofo:

```
$ sysrc jicofo_enable=”YES”
$ service jicofo start
```

Check the logfile for any errors.

## jitsi-meet

### Installation

First, ensure that jitsi-meet is installed, either from a package or from ports. There are no port options to worry about if it is done from ports.

### Configuration

On FreeBSD, the jitsi-meet configuration directory is `/usr/local/www/jitsi-meet`.

Configuration consists of making appropriate edits to the file `config.js`. However, much of it consists of comments, so it’s just as easy to replace it entirely with the following, which is all that is needed:

```
var domainroot = "FQDN"
var config = {
        hosts: {
                domain: domainroot,
                muc: 'conference.'+domainroot,
                bridge: 'jitsi-videobridge.'+domainroot,
                focus: 'focus.'+domainroot
        },
```

```
            useNicks: false,
            bosh: '//'+domainroot+'/http-bind',
      }
```

Be *very* careful with the contents of this file; any errors may cause silent failures!

You can also edit the file `interface_config.js` (in the same place) to select various options.

That is all that is needed. Since these are merely static pages and JavaScript files, there is nothing more to do. If you want to experiment, look at the example `config.js` file for more details.

## nginx

### Installation

It is assumed that nginx has just been installed; modify these instructions appropriately if it is already running. There are many port options, but it is safe to take the default settings for now.

### Configuration

On FreeBSD, the nginx configuration directory is `/usr/local/etc/nginx`, and the work takes place there. The configuration file is named `nginx.conf`.

What follows is a *complete* configuration file; modify as necessary, or merge it with any existing configuration. It is assumed that a certificate has been obtained from Lets Encrypt for the server.

```
user                    nobody;
worker_processes        auto;
pid                     /var/run/nginx.pid;

events {
      worker_connections  1024;
      use               kqueue;
}

http {
      include           mime.types;
      default_type      application/octet-stream;
      log_format        main '$remote_addr - $remote_user [$time_local] "$request" '
                        '$status $body_bytes_sent "$http_referer" '
                        '"$http_user_agent" "$http_x_forwarded_for"';
      error_log         /var/log/nginx/error.log error;
      access_log        /var/log/access.log  main;
      keepalive_timeout 65;

      # jitsi meet HTTPS server
      server {
            listen                  80;
            listen                  443 ssl;
            server_name             FQDN;
            root                    /usr/local/www/jitsi-meet;
            index                   index.html;
            ssl_certificate         /usr/local/etc/letsencrypt/live/FQDN/fullchain.pem;
            ssl_certificate_key     /usr/local/etc/letsencrypt/live/FQDN/privkey.pem;
            ssl_session_cache shared:SSL:1m
            ssl_session_timeout     1400m;
            ssl_session_tickets     off;
            ssl_protocols           TLSv1 TLSv1.2 TLSv1.3;
            # Redirect non-https traffic to https
            if ($scheme != "https") {
                  return 301 https://$host$request_uri;
            }
            location ~ ^/([a-zA-Z0-9=?]+)$ {
```

```
            rewrite ^/(.*)$ / break;
        }
        location / {
            ssi on;
        }
        # BOSH
        location /http-bind {
            proxy_pass        http://localhost:5280/http-bind;
            proxy_set_header  X-Forwarded-For $remote_addr;
            proxy_set_header  Host $http_host;
            tcp_nodelay       on;
            proxy_buffering   off;
        }
        # XMPP websockets
        location /xmpp-websocket {
            proxy_pass        http://localhost:5280/xmpp-websocket;
            proxy_http_version      1.1;
            proxy_set_header  Upgrade $http_upgrade;
            proxy_set_header  Connection "Upgrade";
            proxy_set_header  Host $host;
            proxy_set_header  X-Forwarded-For $remote_addr;
            tcp_nodelay       on;
        }
    } # server end
    } # http end
```

Check the configuration with the command:

```
$ nginx –t
```

### Kernel module

FreeBSD includes a kernel module that optimises HTTP requests; it is advisable to use this. Add the following line to `/boot/loader.conf`:

```
accf_http_load=”YES”
```

This will load the module when the system is booted. Meanwhile, to save a reboot right now, load the module manually with the command:

```
$ kldload accf_http
```

### Logging

Logging is defined in `nginx.conf` using the `access_log` and `error_log` directives. It is possible to change the verbosity level of the access log; see the nginx documentation for details.

Create the file `/usr/local/etc/newsyslog.conf.d/nginx`, containing:

```
/var/log/nginx/*  600  7     *    @T03  JGC   /var/run/nginx.pid SIGUSR1
```

Modify this as desired. Then issue the commands:

```
$ newsyslog –C /var/log/nginx/access.log
$ newsyslog –C /var/log/nginx/error.log
```

to create the logfiles with the correct ownership and permissions.

### Starting up

Enable and start nginx:

```
$ sysrc nginx_enable=”YES”
$ service nginx start
```

Check the logfile for any errors.

More information about nginx may be found at https://www.nginx.com.

## jitsi-videobridge

### Installation

First ensure that jitsi-videobridge is installed, either from a package or from ports. There are no port options to worry about if it is done from ports.

### Configuration

On FreeBSD, the jitsi-videobridge configuration directory is `/usr/local/etc/jitsi/videobridge`, and the work takes place there. The configuration file is named `jitsi-videobridge.conf`.

Make these changes to `jitsi-videobridge.conf`:

```
JVB_XMPP_DOMAIN=FQDN
JVB_XMPP_SECRET=SECRET1
```

Leave all of the other lines alone.

### Logging

By default, jitsi-videobridge logs to the file `/var/log/jitsi-videobridge.log`. Some changes to logging behaviour may be made by editing the file `/usr/local/share/java/jitsi-videobridge-2.1.183/lib/logging.properties`, but this is not really necessary (note that the version number in this path will almost certainly be different).

Create the file `/usr/local/etc/newsyslog.conf.d/jtsi-videobridge`, containing:

```
/var/log/jitsi-videobridge.log 600  7 *  @T03 JNC
```

Modify this as desired. Then issue the command:

```
$ newsyslog –C /var/log/jitsi-videobridge.log
```

to create the logfile with the correct ownership and permissions.

### Starting up

Enable and start jitsivideobridge:

```
$ sysrc jitsi_videobridge_enable=”YES”
$ service jitsi-videobridge start
```

Note the first underscore (not hyphen!) in the first line. Check the logfile for any errors.

## Testing

If all of prosody, jicofo, nginx and jitsi-videobridge are now running, check the logfiles.

First check the prosody logfiles, in particular `prosody.log`. If all is well, you should see that the focus and jitsi-videobridge components have authenticated; if not, re-check that the correct secrets are in the correct places (it is easy to get wrong, with so many of them).

jicofo spills a lot of initial information into `jicofo.log`, but at the end there should be one or many lines indicating a successful health check. If not, check for errors caused by incorrect or missing secrets and/or certificates.

jitsi-videobridge is subject to the same advice as jicofo.

Lastly, check the nginx logfiles. There is unlikely to be much wrong here, as most errors prevent nginx from starting. If there is anything, it is almost certainly in `/var/log/nginx/error.log`.

To test, open a browser and go to **http://FQDN**. You should see a jitsi Meet screen!

If jitsi is running behind a firewall, some ports may need to be opened or forwarded. These are:

| Protocol | Port | Component |
|---|---|---|
| TCP | 443 | Web server |
| TCP | 4443 | Video bridge |
| UDP | 10000 | Video bridge |

## Log rotation

nginx honours a signal to close its logs, so that newsyslog can easily rotate them; this has already been sorted out Things are a little harder with jicofo, jitsi-videobridge and prosody, as rotated logs will stay connected. One simple solution is to rotate the logs at an unsocial hour (e.g. 0300), at the same time restarting the relevant program.

To do this, edit the appropriate file in `/usr/local/etc/newsyslog.conf.d` (prosody, jicofo, jitsi-videobridge). Add the letter R to the flags column, remove any N, and at the end add the name of a suitable shell script (give the full path). This script simply needs a single line:

```
service program restart
```

substituting the appropriate program name for *program*.

# Further work

## Adding prosody users

Jitsi normally does not require registration of users; however, in certain circumstances this may be necessary. For example, if running a public server, it may be wise to limit room creation to specified users.

Users are added to **FQDN**, *not* to auth.**FQDN**. This is done by issuing a command such as:

```
$ prosodyctl register user FQDN password
```

Note that the `register` command is not mentioned in the help text. It is a legacy version of the `adduser` command, with different syntax, that doesn't prompt for the password (twice). One could equally well use:

```
$ prosodyctl adduser user@FQDN
```

and then answer the password prompts with the chosen password. This does not provide any mechanism for users to change their own passwords; if this is required, see below.

To delete users, use one of:

```
$ prosodyctl unregister user FQDN
```

or

```
$ prosodyctl deluser user@FQDN
```

## Listing prosody users

One of the shortcomings of `prosodyctl` is that, although it provides facilities for adding and deleting users (`register`/`adduser`, `unregister`/`deluser`), it does not provide a way of listing the registered users.

This functionality can be added by using a plugin. First, install or build `net-im/prosody-modules`. The additional modules are installed into `/usr/local/lib/prosody-modules`. This directory needs to be added to the module search path; to do this, add the following line to `/usr/local/etc/prosody/prosody.cfg.lua`, immediately before the `modules` section:

```
plugin_paths = { "/usr/local/lib/prosody-modules" }
```

Then add the following as the last part of the `modules` section:

```
-- Additional modules
      "listusers";
```

Restart prosody to activate the plugin.

The command:

```
$ prosodyctl mod_listusers
```

will now provide a list of registered users.

## Restrictions on room creation

The server so far is *open*; anyone can connect, create a room or join a room. This is obviously open to abuse. A reasonable solution is to restrict *room creation* to registered users, while allowing anyone else to *join* a room. Naturally, room passwords can be set to control the latter.

Start by editing `/usr/local/etc/prosody/conf.d/`**FQDN**`.cfg.lua` to add a *new* virtual host; this is a copy of the **FQDN** one, but this time it is called "`guest.`**FQDN**". There is no need to add this to DNS. The `guest` part is internal to jitsi, and is simply a way of introducing a virtual host for unauthenticated users.

Now enable authentication on **FQDN**; simply change the authentication line:

```
VirtualHost "FQDN"
    .
     authentication = "internal_plain"
     modules_enabled = {
   .
```

Restart prosody to pick this up. It is possible to use other authentication methods; see the prosody documentation for details.

Now edit `/usr/local/www/jitsi-meet/config.js`, and add a line:

```
domain: domainroot,
anonymousdomain: 'guest.'+domainroot,
muc: 'conference.'+domainroot,
```

Edit (or create) the file `/usr/local/etc/jitsi/jicofo/sip-communicator.properties`, and add this line:

```
org.jitsi.jicofo.auth.URL=XMPP:FQDN
```

to enable the jicofo authentication service. Restart jicofo.

Lastly, create some users using prosodyctl as before. These will be in the domain **FQDN** (*not* in auth.**FQDN**). When logging in, users will have to give the full username, including domain.

## Security enhancements

If the server is not run on a private network, there will inevitably be attempts to abuse it. The restrictions on room creation will probably result in many failed login attempts by those attempting to break in.

One easy way of mitigating this is to use the fail2ban package. This detects repeated logfile entries indicating authorisation failures, and blocks the offending IP address for a specified time (by instructing the firewall).

Fail2ban may be found at `security/py-fail2ban`. It can of course be installed either from a package or from ports. If it is done from ports, take the default configuration options.

There needs to be an enhancement to prosody's logging behaviour; that is, to add logging of authentication successes and failures. This functionality can be added by using a plugin. First, install or build `net-im/prosody-modules`. The additional modules are installed into `usr/local/lib/prosody-modules`. This directory needs to be added to the module search path; to do this, add the following line to `/usr/local/etc/prosody/prosody.cfg.lua`, immediately before the `modules` section:

```
plugin_paths = { "/usr/local/lib/prosody-modules" }
```

Then add the following as the last part of the `modules` section:

```
-- Additional modules
      "log_auth";
```

Restart prosody to activate the plugin. You can now try some failed logins to your server, and you should see messages about it in `/var/log/prosody/prosody.log`. You are also likely to see a problem; the offending IP address is that of the jitsi server! This is because, although nginx sends an `X-Forwarded-For` header with the correct IP address, prosody does not trust the machine on which nginx is running – even if it's the same one. To fix this, add the following line near the bottom of `/usr/local/etc/prosody/prosody.cfg.lua`:

```
trusted_proxies = { "127.0.0.1", "::1", "192.168.1.1" }
```

substituting the IP address of the jitsi server for `192.168.1.1`, of course. Restart prosody. Check that the correct (offending) IP address now appears in the log file.

It is now time to configure fail2ban. First, we need to add a filter for prosody's logfile entries, so create a file named `/usr/local/etc/fail2ban/filter.d/prosody-auth.local`, containing these lines:

```
# Fail2Ban filter for prosody authentication failures
[INCLUDES]
# Read common prefixes. If any customizations available -- read them from common.local
before = common.conf
[Definition]
failregex = Failed authentication attempt \(not-authorized\) for user .* from IP: <HOST>
ignoreregex =
```

The filter *detects* the authentication failure in the logfile; now we need to define an *action*. In fact, there are many predefined ones – look in `/usr/local/etc/fail2ban/action.d`. Or write your own – if the firewall is on a different machine, use ssh and a script at the far end.

Once a suitable action has been located (or constructed), create the file `/usr/local/etc/fail2ban/jail.local`. This can include directives concerning the number of failed attempts before action, the length of a ban, and so on. There can be multiple actions. A typical file might look like this:

```
[DEFAULT]
ignoreip  = 127.0.0.1
bantime   = 43200
```

```
# Host is banned if it's generated "maxretry" in the last "findtime" secs.
findtime  = 600
# "maxretry" is the number of failures before a host gets banned.
maxretry  = 3
[prosody]
enabled   = true
port      = 443,5222,5269
filter    = prosody-auth
destemail = root@BASEDN
sender    = fail2ban@FQDN
action    = ipfw[name=PROSODY]
            sendmail[name="PROSODY",dest=BASEDN,sender=fail2ban@FQDN]
logpath   = /var/log/prosody/prosody.log
```

Enable and start fail2ban:

```
$ sysrc fail2ban_enable="YES"
$ service start fail2ban
```

You can now test this out by attempting some (failed) logins and checking the effect on the firewall. See the manual page for `fail2ban-client` to see how to remove bans.

You will probably need to manage the logfiles. This is done with `newsyslog.conf.d`, as usual.

## User passwords

There is no facility for users to change their own passwords in the system as set up; the system administrator must do it using prosodyctl. While this may not be an issue, it may violate local laws; for example, in the EU it may be covered by relevant provisions in the GDPR - see article 5(1)(f).

Since prosody is a real XMPP server, an XMPP client can be used to log in to it and change a password. On some systems, Thunderbird might be used for this.

It is suggested that the 'psi' XMPP client is simple, and adequate for this task. For some platforms, it can be downloaded from https://psi-im.org/download/. On FreeBSD, it is a port and can be found in `net-im/psi`.

A minor problem is that prosody is using a self-signed certificate for **FQDN**. psi will actually allow a connection after warning the user, but it may be better (using the principle of least astonishment to your users) to have a proper certificate. One already exists, as it was created for nginx. This can be used by prosody simply by changing the `ssl` stanza in `/usr/local/etc/prosody/prosody.cfg.lua`. Assuming that the certificate is from Let's Encrypt:

```
ssl = {
        certificate = "/usr/local/etc/letsencrypt/live/FQDN/fullchain.pem";
              key = "/usr/local/etc/letsencrypt/live/FQDN/privkey.pem";
}
```

There is no need to do this for auth.**FQDN**, as that is only used internally.

Note that, by default, the private key `privkey.pem` will probably *not* be readable by prosody, but only by root (since the nginx master process runs as root, it works OK there). There are various solutions to this; one is to make the key file 'group readable', and change its group to `prosody`; it will also be necessary to make the directory `/usr/local/etc/letsencrypt/archive` 'group searchable' in the same way (the key file is actually stored there, and there is a symbolic link to it):

```
$ chgrp prosody /usr/local/etc/letsencrypt/live/FQDN/privkey.pem
$ chmod 640 /usr/local/etc/letsencrypt/live/FQDN/privkey.pem
$ chgrp prosody /usr/local/etc/letsencrypt/archive
$ chmod 750 /usr/local/etc/letsencrypt/archive
```

Restart prosody and the certificate should work; check the logfile to be sure, as any errors may not be apparent immediately.

It is also possible to import the key into prosody's own certificate store. Both this method, and the method above, require regular management, as keys do expire and have to be replaced (particularly when using Let's Encrypt, as certificates expire after 90 days). For this reason it is often easiest just to leave the certificate where it is, and not bother about the import.

The 'regular management' will probably take the form of a deployment script run as a hook. As before, consult the Let's Encrypt (or other) documentation for details.

Certificates will need to be renewed; this is straightforward for Let's Encrypt, and can be automated. The online documentation at http://prosody.im will help here.

## Acknowledgements

Thanks are due to those who worked hard getting the FreeBSD port of jitsi ready in record time, particularly Alonso Cárdenas Márquez, and to the numerous people from the FreeBSD and Linux worlds from whose web pages I have extracted (and sometimes corrected) various snippets of information. I have tried to fill in the many gaps, and resolve the inconsistencies between the various sources.

## License



## Changelog

| | |
|---|---|
| 2020-10-02 | Clarified that it is best to install all packages first |
| 2020-09-07 | Added warning about errors in `config.js`, and hints about `interface_config.js` |
| 2020-09-07 | Clarified removal of VirtualHost lines from example configuration file |
| 2020-09-01 | Corrected typo in videobridge startup instructions |
| 2020-06-08 | Added hint about adding `SECRETn` to passwords (thanks to Olaf Kolkman) |
| 2020-06-08 | Added material about certificate renewal |
| 2020-05-31 | Added material about permissions on certificates |
| 2020-05-30 | Credited Alonso in the Acknowledgement (once I had his name!) |

Bob Eager
bob@eager.cx
October 2020