Introduction

This guide leads you through all the necessary steps to setup your own Taskserver to sync your Taskwarrior-tasks.

Please follow the steps **carefully** and **note** all things you do differently.

Preparation – Backup Your Data

Let's reinforce a good habit and make a backup copy of your data first. Here is a very easy way to backup your data:

```
$ cd ~/.task
$ tar czf task-backup-$(date +'%Y%m%d').tar.gz *
```

Now move that file somewhere safe. All software contains bugs, so make regular backups.

Attention!

This is not only due to a good habit, we will modify your data, so this backup is highly recommended.

Preparation – Choose A Machine

A suitable machine to run your Taskserver is one that is always available. If you have such a machine, or have access to a hosted machine, that is ideal.

If your machine is not continuously available, it can still be a suitable Taskserver because the sync mechanism doesn't require continuous access. When a client cannot sync, it simply accumulates local, unpropagated changes until it can sync.

A laptop is a poor choice for a Taskserver host.

Preparation – Choose A Port

By default, Taskserver uses port 53589. You can choose any port you wish, provided it is unused. If you choose a port number that is under 1024, then Taskserver must run as root, which is not recommended.

Preparation – User/Group

Ideally you will create a new user and group solely to run the Taskserver. This helps you keep the data secure from other users on the machine, as well as controlling the privileges of Taskserver.

Preparation – Firewall

Depending on what devices you use to access your server, you may need to configure the firewall to allow incoming TCP/IP traffic on your chosen port.

Installation

Installing Taskserver from a binary package is the simplest option, but you will need to refer to your package manager's documentation and procedures for doing this.

Take a look at the Download page for examples. Generally there are too many package managers to make a complete list with instructions here.

Most importantly, for now, Taskserver is a new product, and there are very few packages available. It is expected that this situation will change soon. When it does, this page will be updated.

Meanwhile, there is installation from either git or tarball.

Installation – Introduction

Installing Taskserver from a tarball is a matter of downloading the tarball, extracting it, satisfying dependencies and building the server.

Installation – Dependencies (general)

Before building the software, you will need to satisfy the dependencies by installing the following:

- `GnuTLS` (ideally version 3.2 or newer)
- `libuuid`
- `CMake` (2.8 or newer, check with)
- `make`
- A C++ Compiler (GCC 4.7 or Clang 3.0 or newer)

Note that some OSes (Darwin, FreeBSD ...) include `libuuid` functionality in libc, check the following slides for more detailed instructions.

You don't necessarily need the latest version of all components, but it is a good idea if you can. GnuTLS is a security component, and as such, it is very important that it is current.

Using GnuTLS version 2.12.x is neither adequately secure, nor production quality. Please check the GnuTLS-Problem for details.

Installation – Dependencies (OS)

We have detailed instructions for the following operating operating systems (click on the name to continue):

- CentOS
- Debian
- Fedora
- openSUSE
- Ubuntu
- Windows with Cygwin (unsupported but working)
  Recommendation: Use the Ubuntu subsystem in Windows 10 and follow the Ubuntu instructions.
- Mac OS X

In case you can add your operating system of choice, please send an email to support@taskwarrior.org (Thank you!).

Installation – CentOS

Before building the software, you will need to satisfy the dependencies by installing the following:

```
$ sudo yum install gcc-c++
$ sudo yum install gnutls-devel
$ sudo yum install libuuid-devel
$ sudo yum install cmake
$ sudo yum install gnutls-utils
```

Continue with Download.

Installation – Debian

Before building the software, you will need to satisfy the dependencies by installing the following:

```
$ sudo apt install g++
$ sudo apt install libgnutls28-dev
$ sudo apt install uuid-dev
$ sudo apt install cmake
$ sudo apt install gnutls-utils
```

Continue with Download.

Installation – Fedora

Before building the software, you will need to satisfy the dependencies by installing the following:

```
$ sudo dnf install gcc-c++
$ sudo dnf install gnutls-devel
$ sudo dnf install libuuid-devel
$ sudo dnf install cmake
$ sudo dnf install gnutls-utils
```

Continue with Download.

Installation – openSUSE

Before building the software, you will need to satisfy the dependencies by installing the following:

```
$ sudo zypper install gcc-c++
$ sudo zypper install libgnutls-devel
$ sudo zypper install libuuid-devel
$ sudo zypper install cmake
$ sudo zypper install gnutls-utils
```

Continue with Download.

Installation – Ubuntu

Before building the software, you will need to satisfy the dependencies by installing the following:

```
$ sudo apt install g++
$ sudo apt install libgnutls28-dev
$ sudo apt install uuid-dev
$ sudo apt install cmake
$ sudo apt install gnutls-utils
```

Continue with Download.

Installation – Windows

Before building the software, you will need to satisfy the dependencies by installing the following:

Start the Cygwin GUI and install the following packages and their dependencies.

- `GnuTLS`
- `libuuid`
- `CMake`
- `make`
- `gcc-c++`
- `gnutls-utils`

Continue with Download.

Installation – Mac OS X

Before building the software, you will need to satisfy the dependencies by installing the following:

Install Xcode from Apple, via the AppStore, launch it, and select from some menu that you want the command line tools.

We expect you to have Homebrew installed on your Mac.

```
$ brew install cmake
$ brew install git
$ brew install gnutls
```

Continue with Download.

Installation – Download

The next step is to obtain the code. This means getting the Task Server 1.1.0 (or newer) source tarball. You should check for the latest stable release here:

http://taskwarrior.org/download/

You can download the tarball with `curl`, as an example of just one of many ways to download the tarball.

```
$ curl -O http://taskwarrior.org/download/taskd-latest.tar.gz
```

Installation – Build

Expand the tarball, and build the Taskserver.

```
$ tar xzf taskd-latest.tar.gz
$ cd taskd-latest
$ cmake -DCMAKE_BUILD_TYPE=release .
...
$ make
...
```

SOURCEDIR

We will refer to the directory where you extracted the data to as `SOURCEDIR` (in the example above it is `taskd-latest`).

Installation – Build Again

If you ever want to build the software again, do some cleanup.

```
$ cd taskd-latest
$ make clean
...
$ rm CMakeCache.txt
...
```

Installation – make install

Now install Taskserver. This copies files into the right place, and installs man pages.

```
$ sudo make install
...
```

Installation – Verify installation

Run the `taskd` command to verify that the server is installed, and the location is in your `$PATH`.

You should see something like this:

```
$ taskd

Usage: taskd -v|--version
       taskd -h|--help
       taskd diagnostics
       taskd validate <JSON | file>
       taskd help [<command>]

Commands run only on server:
       taskd add     [options] org   <org>
       taskd add     [options] group <org> <group>
       taskd add     [options] user  <org> <user>
       taskd config  [options] [--force] [<name> [<value>]]
       taskd init    [options]
       taskd remove  [options] org   <org>
       taskd remove  [options] group <org> <group>
       taskd remove  [options] user  <org> <user>
       taskd resume  [options] org   <org>
       taskd resume  [options] group <org> <group>
       taskd resume  [options] user  <org> <user>
       taskd server  [options] [--daemon]
       taskd status  [options]
...
```

Installation – from Git-Repository

Installing Taskserver from git is a matter of cloning the git repository and building the server.

The same dependencies as for installation from tarball apply. We have detailed instructions for the following operating operating systems (click on the name to continue), afterwards come back to this slide (23) or continue with cloning the repository:

- CentOS
- Debian
- Fedora
- openSUSE
- Ubuntu
- Windows with Cygwin
- Mac OS X

Installation – Cloning the repository

Now clone the repository like this:

```
$ git clone https://git.tasktools.org/scm/tm/taskd.git taskd.git
...
```

Use stable!

It is highly recommended that you build the stable version. This involves simply moving on to the next step, build.

Only under special circumstances should you build the unstable development version.

Installation – Special Circumstances (1)

The unstable development version is at no point guaranteed to work or even compile. The only time it does stabilize is right at the end of the development cycle, and in that case, you should wait until the release, so you are running a supported version.

The stable version is always merged to the `master` branch, which is the default branch, so ordinarily nothing needs to be done. To build an unstable branch, first determine which branch by looking at the available branches:

```
$ cd taskd.git
$ git branch -a
* master
  remotes/origin/1.1.0
  remotes/origin/1.1.1
  remotes/origin/1.2.0
  remotes/origin/HEAD -> origin/master
  remotes/origin/master
```

Installation – Special Circumstances (2)

The convention we use is that `master` represents the stable release. The numbered branches represent the latest development (1.2.0, the 'highest' branch number, ending in '.0') and a patch branch (1.1.1, ending in a non-zero number).

Patch branches are reserved for emergency releases, so in this example you would choose to build 1.2.0 as the latest development branch like this:

```
$ git checkout 1.2.0
Branch 1.2.0 set up to track remote branch 1.2.0 from origin.
Switched to a new branch '1.2.0'
```

Installation – Build from Git

Now build the Taskserver.

```
$ cd taskd.git
$ cmake -DCMAKE_BUILD_TYPE=release .
...
$ make
...
```

SOURCEDIR

In this case the `SOURCEDIR` is `taskd.git`.

Installation – Test your build

Having built the server, now build and run the unit tests. Although this is an optional step, it is a good idea to know whether the build works on your platform.

```
$ cd test # from SOURCEDIR
$ make
...
$ ./run_all

Pass:      2920
Fail:         0
Skipped:      0
Runtime:      1 seconds

$ cd ..
```

This example shows that all 2,920 tests pass. If you see test failures, stop and report them.

Note that there are some unit tests that fail if you have not built the latest commit. Seeing 4 test failures may mean all is well. Seeing 30 failures does not.

Installation

Now install Taskserver. This copies files into the right place, and installs man pages.

```
$ sudo make install
...
```

Verify your Installation

Run the `taskd` command to verify that the server is installed, and the location is in your `$PATH`. You should see something like this:

```
$ taskd

Usage: taskd -v|--version
taskd -h|--help
taskd diagnostics
taskd validate <JSON | file>
taskd help [<command>]

Commands run only on server:
taskd add     [options] org   <org>
taskd add     [options] group <org> <group>
taskd add     [options] user  <org> <user>
taskd config  [options] [--force] [<name> [<value>]]
taskd init    [options]
taskd remove  [options] org   <org>
taskd remove  [options] group <org> <group>
taskd remove  [options] user  <org> <user>
taskd resume  [options] org   <org>
taskd resume  [options] group <org> <group>
taskd resume  [options] user  <org> <user>
taskd server  [options] [--daemon]
taskd status  [options]
taskd suspend [options] org   <org>
taskd suspend [options] group <org> <group>
taskd suspend [options] user  <org> <user>
...
```

taskd-User

All configuration is done with the taskd-User

We assume that you will do all configuration with the taskd user you chose to run the server with.

Server Configuration – Data Location

Configuring the server is straightforward, but needs a little planning.

A location for the data must be chosen and created. The `TASKDDATA` environment variable will be used to indicate that location to all the `taskd` commands.

```
$ export TASKDDATA=/var/taskd
$ mkdir -p $TASKDDATA
```

If the `TASKDDATA` variable is not set, then most `taskd` commands require the `--data ...` argument, otherwise the commands rely on the `TASKDDATA` value to indicate the location.

Everything the server does will be confined to that directory.

There are two ’D’s in `TASKDDATA`, and omitting one is a common mistake.

The user that will run the server must have write permissions in that directory.

Server Configuration – Initialization

Now we let the server initialize that directory:

```
$ taskd init
You must specify the 'server' variable, for example:
taskd config server localhost:53589

Created /var/taskd
```

pki directory

It is a good idea to copy the `pki` subdirectory from your `SOURCEDIR` to your `TASKDDATA` directory.

Server Configuration – Keys & Certificates (1)

Now we create certificates and keys. The command below will generate all the certs and keys for the server, but this uses self-signed certificates, and this is not recommended for production use. This is for personal use, and this may be acceptable for you, but if not, you will need to purchase a proper certificate and key, backed by a certificate authority.

Server Configuration – Keys & Certificates (2)

The certificate and key generation scripts make assumptions ***that are guaranteed to be wrong for you***. Specifically the `generate.server` script has a hard-coded `CN` entry that is not going to work. You ***need*** to edit the `vars` file, which you find in the `pki` subdirectory in your `SOURCEDIR`.

```
CN=localhost
...
```

You will need to modify this value to match your server.

Most probably the result of `hostname -f` is exactly what you need ("yourserver.example.com").

Server Configuration – Keys & Certificates (3)

The value of CN (Common Name) is important.

It is this value against which Taskwarrior validates the servername, so use a value similar to `ack.tasktools.org`, which is what we use, but of course don't expect that to work for you. If you do not change this value, the only option for the client is to skip some or all certificate validation, ***which is a bad idea***.

Go to your `SOURCEDIR`, which depends on which installation method you chose. Here is is assumed that you installed from the source tarball.

```
$ cd ~/taskd-1.1.0/pki
$ ./generate
...

$ cp client.cert.pem $TASKDDATA
$ cp client.key.pem  $TASKDDATA
$ cp server.cert.pem $TASKDDATA
$ cp server.key.pem  $TASKDDATA
$ cp server.crl.pem  $TASKDDATA
$ cp ca.cert.pem     $TASKDDATA
```

Server Configuration – Keys & Certificates (4)

```
$ taskd config --force client.cert $TASKDDATA/client.cert.pem
$ taskd config --force client.key  $TASKDDATA/client.key.pem
$ taskd config --force server.cert $TASKDDATA/server.cert.pem
$ taskd config --force server.key  $TASKDDATA/server.key.pem
$ taskd config --force server.crl  $TASKDDATA/server.crl.pem
$ taskd config --force ca.cert     $TASKDDATA/ca.cert.pem
```

There are three classes of key/cert here. There is the CA (Certificate Authority) cert, which has cert signing capabilities and is used to sign and verify the other certs.

There are the server key/certs, which are used to authenticate the server and encrypt.

Finally there are client key/certs, which are not what you might expect. These are for API access, and not for your Taskwarrior client. Those are created later.

Server Configuration – Other Configuration

Now we configure some basic details for the server. The chosen port is 53589. Note that we allow Taskwarrior clients specifically.

```
$ cd $TASKDDATA/..
$ taskd config --force log $PWD/taskd.log
$ taskd config --force pid.file $PWD/taskd.pid
$ taskd config --force server localhost:53589
```

Note that we have chosen `localhost:53589`, but this choice has consequences. The name `localhost` is not network visible, which limits the server to only serving clients on the same machine. Use your full machine name for proper network addressability.

You can look at all the configuration settings:

```
$ taskd config
```

You can view all the supported settings with:

```
$ man taskdrc
```

Server – Control

You can now to launch the server:

```
$ taskdctl start
```

This command launched the server as a daemon process. This command requires the `TASKDDATA` variable. Your server is now running, and ready for syncing.

Note that to stop the server, you use:

```
$ taskdctl stop
```

Check that your server is running by looking in the `taskd.log` file, or running this:

```
$ ps -leaf | grep taskd
```

Server – Interactive or Non-Daemon Server

A daemon server is typically how you would want to run Taskserver, but there may be times when you need to run the server attached to a terminal. These two commands are identical:

```
$ taskdctl start
$ taskd server --data $TASKDDATA --daemon
```

By omitting the `--daemon` option, the server remains attached to the terminal. Then to stop the server you can enter `Ctrl-C`.

The interactive mode is really only useful for debugging, in conjunction with TLS debug mode, like this:

```
$ taskd config debug.tls 3
$ taskd server --data $TASKDDATA
...
```

With a `debug.tls` setting that is non-zero, you see lots of security-related diagnostic output.

Server – systemd unit file

You can start Taskserver using a systemd-unitfile (called `taskd.service`) like the following (please add the contents of `$TASKDDATA` not the variable itself). Running the Taskserver as root is not recommended, please add an appropriate user and group to run the daemon with (`$TASKDUSER` and `$TASKDGROUP`).

```
[Unit]
Description=Secure server providing multi-user, multi-client access to Taskwarrior data
Requires=network.target
After=network.target
Documentation=http://taskwarrior.org/docs/#taskd

[Service]
ExecStart=/usr/local/bin/taskd server --data $TASKDDATA
Type=simple
User=$TASKDUSER
Group=$TASKDGROUP
WorkingDirectory=$TASKDDATA
PrivateTmp=true
InaccessibleDirectories=/home /root /boot /opt /mnt /media
ReadOnlyDirectories=/etc /usr

[Install]
WantedBy=multi-user.target
```

Server – Control with systemd

Afterwards prepare systemd to recognise the file. The following commands need to be run as root-user.

```
$ cp taskd.service /etc/systemd/system
$ systemctl daemon-reload
$ systemctl start taskd.service
$ systemctl status taskd.service
```

In case everything is running fine, enable the script to start Taskserver on every boot.

```
$ systemctl enable taskd.service
```

Add User/Organization to Server

A user account must be created, along with a key, cert and ID, before syncing may occur.

Before creating a user account, you may need to create an organization. An organization consists of a group of zero or more users. You can get away with just one organization, and in this example, we will create just one, named ’Public’.

You can create as many organizations as you wish (even one per user), and the purpose is simply to group users together. Future features will utilize this.

```
$ taskd add org Public
Created organization ’Public’
```

Now the organization ’Public’ exists, we can add users to that organization.

Create User

Now we add a new user, named ’First Last’ as an example. You can use any name you wish, and if it contains spaces, quote the name as shown.

```
$ taskd add user ’Public’ ’First Last’
New user key: cf31f287-ee9e-43a8-843e-e8bbd5de4294
Created user ’First Last’ for organization ’Public’
```

Note that you will get a different ’New user key’ than is shown here, and you will need to retain it, to be used later for client configuration. Note that the key is just a unique id, because your name alone is not necessarily unique.

Create Certificate and Key

Go to your `SOURCEDIR`, which depends on which installation method you chose. Here it is assumed that you installed from the source tarball.

```
$ cd ~/taskd-1.1.0/pki
$ ./generate.client first_last
...
```

This will generate a new key and cert, named `first_last.cert.pem` and `first_last.key.pem`. It is not important that 'first_last' was used here, just that it is something unique, and valid for use in a file name. It has no bearing on security.

Let's encrypt

Certificates coming from Let's encrypt have **not** been successfully used by anyone. A working scenario would be highly appreciated.

Client Configuration

You have now created a new user account on the server, created a new client cert and key, and have details that need to be transferred to the user, to set up a sync client. The details needed are:

- `ca.cert.pem` is the certificate authority, and the only way to validate self-signed certs such as the ones we have created here.
- `first_last.cert.pem` is the client certificate.
- `first_last.key.pem` is the client key.
- The new user key (yours will be different): `cf31f287-ee9e-43a8-843e-e8bbd5de4294`
- The organization, `Public`.
- The full and proper user name, `First Last`.
- The server address and port, `host.domain:53589`. In the server configuration we used `localhost` as an example. Whatever you actually used there, should be used here.

Configure Taskwarrior – Certificates

If you have configured Taskserver and created a user account (or better yet, someone created an account for you) then you now have details needed in the configuration of your Taskwarrior client. You should have the files and information mentioned on the last slide.

Now we feed this information to Taskwarrior.

Copy the Cert, Key and CA to your /.task directory. The reason we are copying the CA cert is because this is a self-signed cert, and we need the CA to validate against. Alternately we could force Taskwarrior to trust all certs, but that is not recommended.

```
$ cp first_last.cert.pem ~/.task
$ cp first_last.key.pem  ~/.task
$ cp ca.cert.pem        ~/.task
```

Now we need to make Taskwarrior aware of these certs:

```
$ task config taskd.certificate  -- ~/.task/first_last.cert.pem
$ task config taskd.key         -- ~/.task/first_last.key.pem
$ task config taskd.ca          -- ~/.task/ca.cert.pem
```

Configure Taskwarrior – Certificates

Now set the server info:

```
$ task config taskd.server      -- host.domain:53589
```

Finally we provide the credentials, which combine the organization, account name and user key:

```
$ task config taskd.credentials -- Public/First Last/cf31f287-ee9e-43a8-843e-e8bbd5de4294
```

Taskwarrior – Trust Level

Trust

It is possible to configure Taskwarrior's trust level, which determines how the server certificate is treated.

For Taskwarrior 2.3.0 you can specify `taskd.trust=yes` in order to skip certificate validation. ***This is a bad idea.*** The default is `taskd.trust=no`, which does not trust the server certificate, which is more secure.

For Taskwarrior 2.4.0 you must specify `taskd.trust=ignore hostname` in order to skip certificate hostname validation. ***This is a bad idea***. You can also specify `taskd.trust=allow all` to perform no validation. ***This is a worse idea***. The default value is `taskd.trust=strict` which performs the most stringent verification, and is more secure.

Your Taskwarrior is now ready to sync.

First Time Sync

You are now ready to sync your Taskwarrior client. You will do this differently depending on whether this is the first sync per device, or one of the many subsequent syncs.

The first time you sync is special – the client sends all your tasks to the server. This is something you should only do once. Run this:

```
$ task sync init
Please confirm that you wish to upload all your pending tasks to the Task Server (yes/no) yes
Syncing with host.domain:53589

Sync successful.  2 changes uploaded.
```

You should get an indication that tasks were uploaded, in this case 2 of them.

Taskwarrior before Version 2.5.1

Please note that older Taskwarrior versions only sync the **pending** tasks and not all tasks.

General Sync

After the first time sync, you switch and just use this command:

```
$ task sync
Syncing with host.domain:53589

Sync successful.  No changes.
```

This will give you feedback about what happened. Please note that it is perfectly safe to run this command as often as you wish. Syncing is safe and does not consume great system resources.

Note that if your client is a mobile device, a sync command may consume some of your data usage. Act accordingly.

But it does require network connectivity, and if there is no connectivity you will be notified. It is not a problem if a sync fails because of this, because the next sync that works will catch up with all the changes, and do the right merging. *Taskwarrior and Taskserver were designed to work together, and tolerate intermittent connectivity*.

Sync Reminder

After you modify data locally, Taskwarrior will start notifying you that you need to sync, after commands, like this:

```
$ task project:foo list
No matches.
There are local changes.  Sync required.
```

This is just a reminder to sync. Respond with a sync, and the reminder goes away:

```
$ task sync
Syncing with <server>:<port>

Sync successful.  1 changes uploaded.
```

If you do not respond with a sync, then local changes accumulate unseen by other clients. When you do eventually sync, the data will be properly propagated, so it is a question of whether you *need* current data on the server. It is perfectly fine to allow *weeks*< to go by without a sync.

Troubleshooting Sync

Here is a list of problems you may encounter, with the most common ones listed first.

The single most common problem has been that the Taskserver Setup Instructions (this guide) were not properly followed. Please review the steps you took.

It is always a good idea to make sure that you are using the latest release of Taskwarrior and Taskserver, not just because bugs are fixed that may help you, but also because the solutions below are geared toward the current releases.

If you upgrade from an older release of Taskserver, you will need to follow the upgrade instructions.

Problems (1)

You tried `task sync` but Taskwarrior showed you a task list instead

You have a version of Taskwarrior older than `2.3.0`, which means there was no `sync` command, and you are seeing a list filtered by the search term 'sync'. Upgrading is the only solution.

Problems (2)

You tried `task sync` and saw 'Taskwarrior was built without GnuTLS support. Sync is not available.'

You are using version `2.3.0` or later, but the Taskwarrior binary was compiled without GnuTLS support.

If you installed Taskwarrior using your OS's package manager, you may be suffering from an out of date package. Prod your OS's package maintainer for an update.

Recent releases make GnuTLS support opt-out instead of opt-in, so upgrading to the latest version may help. Otherwise, you will need to build Taskwarrior from the latest source tarball, following the instructions in the `INSTALL` file. If you are a developer, do that. If you are not, then installing a development environment is probably not something you want to do, in which case contact your OS's package maintainer.

Continued on next page.

Problems (2) – continued

Verify that your Taskwarrior was built with GnuTLS support by running `task diagnostics`:

```
$ task diagnostics | grep libgnutls
libgnutls: 3.3.18
```

Problems (3)

nodename nor servname provided, or not known

Despite the terrible wording, this means the Taskwarrior setting `taskd.server=<host>:<port>` refers to a host name that cannot be seen by Taskwarrior.

- Is it spelled correctly?
- Is the domain correct?
- Is there a valid DNS resolution for the name?
- Is there a firewall between Taskwarrior and Taskserver that is not letting through `<port>` traffic?

Problems (4)

Could not connect to <host> <port>

Taskserver may not be running on `<host>`.

Check with `ps -leaf | grep taskd`.

Problems (5)

Unable to use port 53589?

By default, port 53589 is used, but whichever you chose must be open on the server.

If you are unable to open firewall ports, you can use an SSH Tunnel to route port 53589 traffic over port 22:

```
$ ssh -L localport:dsthost:dstport user@example.com
```

Problems (6)

Certificate fails validation, Handshake failed

There are many reasons that the TLS handshake can fail.

When you generated certificates, you modified a `vars` file, in particular the `CN=<name>` setting. That name must match the output of `$ hostname -f` on the server for the certificate to validate.

Additionally, that name must also be used in the `taskd.server=<host>:<port>` setting for Taskwarrior. Otherwise you'll need `taskd.trust=ignore hostname`.

If you are using a self-signed certificate, did you specify it using the `taskd.ca` setting?

Setting `taskd.ciphers` can force the use of different ciphers. Use `gnutls-cli --list` to see a list of installed ciphers, and confirm that there is overlap between client and server. There needs to be a cipher that is available to both, otherwise they cannot communicate.

Problems (7)

Is your certificate still valid?

Certificates have expiration dates, and if you followed our instructions, you created a certificate that is valid for one year. Check your certificate with this command:

```
$ certtool -i --infile ~/.task/<YOUR NAME>.cert.pem
```

If your certificate has expired, you need a new one. You may also need to regenerate expired server certificates.

Note that creating certificates that never expire is a bad idea. Certificates may be compromised. A certificate that is considered secure today, may not be considered secure in a year. Is the key length you chose something that will remain suitable in the future? Will the algorithms you chose remain secure? For these reasons, choose an expiration date that lets you reevaluate your choices in the relatively near future.

Problems (8)

Is your GnuTLS library up to date?

As a security product, it is imperative that you keep your GnuTLS up to date.

As with many security products, GnuTLS is maintained by a responsible and quick-responding team that takes security very seriously. Benefit from their diligence by keeping your GnuTLS up to date.

We have received reports of issues with older GnuTLS releases. Specifically, version 2.12.20 has problems validating certificates under certain conditions. Newer releases have addressed memory leaks that were able to take down Taskserver.

Please keep in mind that you have to recompile Taskserver completely to benefit from the new version.

Continued on next page.

Problems (8) – continued

Upgrading GnuTLS does nothing to upgrade taskd – it has to be rebuilt from scratch, which means:

```
$ cd taskd.git
$ rm CMakeCache.txt
$ cmake -DCMAKE_BUILD_TYPE=release .
$ make
$ sudo make install
```

This can then be verified using `taskd diagnostics`.

Problems (9)

ERROR: Could not find common ancestor for ...
Client sync key not found.

You skipped the important step of running:

```
$ task sync init
```

This performs an initial upload of your tasks, and sets up a local sync key, which identifies the last sync transaction.

Taskwarrior before Version 2.5.1

Please note that older Taskwarrior versions only sync the **pending** tasks and not all tasks.

Debugging – Diagnostics

You may wish to try and debug the problem yourself. You will probably not. But if you do, here is how.

Both Taskwarrior and Taskserver have a `diagnostics` command, the purpose of which is to show you relevant troubleshooting details. Additionally it will indicate problems directly, for example, it will tell you if your cert/key files are not readable. The output from `diagnostics` is intended to be included in bug reports, and doing so saves you a lot of time, because it's the first thing we'll ask for.

```
$ task diagnostics
...
$ taskd diagnostics
...
```

Read the output of the `diagnostics` commands carefully, there may be several types of problems mentioned, which need to be addressed before going further.

Debug Mode

The next step would be to run the server in debug mode. First shutdown your Taskserver, then launch it interactively:

```
$ taskdctl stop
...
$ taskd server
...
```

You can hit `Ctrl-C` to stop this server. For highly verbose output, try this:

```
$ taskd server --debug --debug.tls=2
...
```

Similarly, Taskwarrior has a verbose debug mode, and debug TLS mode:

```
$ task rc.debug=1 rc.debug.tls=2 sync
...
```

Getting Help

As a last resort, ask for help. But please make sure you have carefully reviewed your setup, and gone through the checks above before asking. No one wants to lead you through the steps above to discover that you didn't.

We'll ask you to provide the `diagnostics` output for both Taskwarrior and Taskserver, then we're going to go through the steps above, because this is our checklist also.

Getting Help

There are several ways of getting help:

- Submit your details to our Q & A site, then wait patiently for the community to respond.
- Email us at support@taskwarrior.org, then wait patiently for a volunteer to respond.
- Join us IRC in the #taskwarrior channel on Freenode.net, and get a quick response from the community, where, as you have anticipated, we will walk you through the checklist above.
- Even though Twitter is no means of support, you can get in touch with @taskwarrior.
- We have a User Mailinglist which you can join anytime to discuss about Taskwarrior and techniques.
- The Developer Mailinglist is focussing on a more technical oriented audience.