Protect the Docker daemon socket

Estimated reading time: 7 minutes

By default, Docker runs through a non-networked UNIX socket. It can also optionally communicate using an HTTP socket.

If you need Docker to be reachable through the network in a safe manner, you can enable TLS by specifying the `tlsverify` flag and pointing Docker's `tlscacert` flag to a trusted CA certificate.

In the daemon mode, it only allows connections from clients authenticated by a certificate signed by that CA. In the client mode, it only connects to servers with a certificate signed by that CA.

> **Advanced topic**
>
> Using TLS and managing a CA is an advanced topic. Please familiarize yourself with OpenSSL, x509, and TLS before using it in production.

Create a CA, server and client keys with OpenSSL

> **Note**: Replace all instances of `$HOST` in the following example with the DNS name of your Docker daemon's host.

First, on the **Docker daemon's host machine**, generate CA private and public keys:

```
$ openssl genrsa -aes256 -out ca-key.pem 4096
Generating RSA private key, 4096 bit long modulus
.....................................................................
........++
e is 65537 (0x10001)
Enter pass phrase for ca-key.pem:
Verifying - Enter pass phrase for ca-key.pem:

$ openssl req -new -x509 -days 365 -key ca-key.pem -sha256 -out ca.pem
Enter pass phrase for ca-key.pem:
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]:
State or Province Name (full name) [Some-State]:Queensland
Locality Name (eg, city) []:Brisbane
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Docker Inc
Organizational Unit Name (eg, section) []:Sales
Common Name (e.g. server FQDN or YOUR name) []:$HOST
Email Address []:Sven@home.org.au
```

Now that you have a CA, you can create a server key and certificate signing request (CSR). Make sure that “Common Name” matches the hostname you use to connect to Docker:

> **Note**: Replace all instances of `$HOST` in the following example with the DNS name of your Docker daemon’s host.

```
$ openssl genrsa -out server-key.pem 4096
Generating RSA private key, 4096 bit long modulus
.....................................................................++
.....................................................................
e is 65537 (0x10001)

$ openssl req -subj "/CN=$HOST" -sha256 -new -key server-key.pem -out server.cs
```

Next, we’re going to sign the public key with our CA:

Since TLS connections can be made through IP address as well as DNS name, the IP addresses need to be specified when creating the certificate. For example, to allow connections using `10.10.10.20` and `127.0.0.1` :

```
$ echo subjectAltName = DNS:$HOST,IP:10.10.10.20,IP:127.0.0.1 >> extfile.cnf
```

Set the Docker daemon key's extended usage attributes to be used only for server authentication:

```
$ echo extendedKeyUsage = serverAuth >> extfile.cnf
```

Now, generate the signed certificate:

```
$ openssl x509 -req -days 365 -sha256 -in server.csr -CA ca.pem -CAkey ca-key.p
  -CAcreateserial -out server-cert.pem -extfile extfile.cnf
Signature ok
subject=/CN=your.host.com
Getting CA Private Key
Enter pass phrase for ca-key.pem:
```

Authorization plugins (https://docs.docker.com/engine/extend/plugins_authorization) offer more fine-grained control to supplement authentication from mutual TLS. In addition to other information described in the above document, authorization plugins running on a Docker daemon receive the certificate information for connecting Docker clients.

For client authentication, create a client key and certificate signing request:

> **Note**: For simplicity of the next couple of steps, you may perform this step on the Docker daemon's host machine as well.

```
$ openssl genrsa -out key.pem 4096
Generating RSA private key, 4096 bit long modulus
.....................................................++
..............++
e is 65537 (0x10001)

$ openssl req -subj '/CN=client' -new -key key.pem -out client.csr
```

To make the key suitable for client authentication, create a new extensions config file:

```
$ echo extendedKeyUsage = clientAuth > extfile-client.cnf
```

Now, generate the signed certificate:

```
$ openssl x509 -req -days 365 -sha256 -in client.csr -CA ca.pem -CAkey ca-key.p
  -CAcreateserial -out cert.pem -extfile extfile-client.cnf
Signature ok
subject=/CN=client
Getting CA Private Key
Enter pass phrase for ca-key.pem:
```

After generating `cert.pem` and `server-cert.pem` you can safely remove the two certificate signing requests and extensions config files:

```
$ rm -v client.csr server.csr extfile.cnf extfile-client.cnf
```

With a default `umask` of 022, your secret keys are *world-readable* and writable for you and your group.

To protect your keys from accidental damage, remove their write permissions. To make them only readable by you, change file modes as follows:

```
$ chmod -v 0400 ca-key.pem key.pem server-key.pem
```

Certificates can be world-readable, but you might want to remove write access to prevent accidental damage:

```
$ chmod -v 0444 ca.pem server-cert.pem cert.pem
```

Now you can make the Docker daemon only accept connections from clients providing a certificate trusted by your CA:

```
$ dockerd --tlsverify --tlscacert=ca.pem --tlscert=server-cert.pem --tlskey=ser
  -H=0.0.0.0:2376
```

To connect to Docker and validate its certificate, provide your client keys, certificates and trusted CA:

> **Run it on the client machine**
>
> This step should be run on your Docker client machine. As such, you need to copy your CA certificate, your server certificate, and your client certificate to that machine.

> **Note**: Replace all instances of `$HOST` in the following example with the DNS name of your Docker daemon's host.

```
$ docker --tlsverify --tlscacert=ca.pem --tlscert=cert.pem --tlskey=key.pem \
  -H=$HOST:2376 version
```

> **Note**: Docker over TLS should run on TCP port 2376.

> **Warning**: As shown in the example above, you don't need to run the `docker` client with `sudo` or the `docker` group when you use certificate authentication. That means anyone with the keys can give any instructions to your Docker daemon, giving them root access to the machine hosting the daemon. Guard these keys as you would a root password!

Secure by default

If you want to secure your Docker client connections by default, you can move the files to the `.docker` directory in your home directory --- and set the `DOCKER_HOST` and `DOCKER_TLS_VERIFY` variables as well (instead of passing `-H=tcp://$HOST:2376` and `--tlsverify` on every call).

```
$ mkdir -pv ~/.docker
$ cp -v {ca,cert,key}.pem ~/.docker

$ export DOCKER_HOST=tcp://$HOST:2376 DOCKER_TLS_VERIFY=1
```

Docker now connects securely by default:

```
$ docker ps
```

Other modes

If you don't want to have complete two-way authentication, you can run Docker in various other modes by mixing the flags.

Daemon modes

- `tlsverify`, `tlscacert`, `tlscert`, `tlskey` set: Authenticate clients
- `tls`, `tlscert`, `tlskey`: Do not authenticate clients

Client modes

- `tls`: Authenticate server based on public/default CA pool
- `tlsverify`, `tlscacert`: Authenticate server based on given CA
- `tls`, `tlscert`, `tlskey`: Authenticate with client certificate, do not authenticate server based on given CA
- `tlsverify`, `tlscacert`, `tlscert`, `tlskey`: Authenticate with client certificate and authenticate server based on given CA

If found, the client sends its client certificate, so you just need to drop your keys into `~/.docker/{ca,cert,key}.pem`. Alternatively, if you want to store your keys in another location, you can specify that location using the environment variable `DOCKER_CERT_PATH`.

```
$ export DOCKER_CERT_PATH=~/.docker/zone1/
$ docker --tlsverify ps
```

Connecting to the secure Docker port using `curl`

To use `curl` to make test API requests, you need to use three extra command line flags:

```
$ curl https://$HOST:2376/images/json \
  --cert ~/.docker/cert.pem \
  --key ~/.docker/key.pem \
  --cacert ~/.docker/ca.pem
```

Related information

- Using certificates for repository client verification (https://docs.docker.com/engine/security/certificates/)
- Use trusted images (https://docs.docker.com/engine/security/trust/)

docker (https://docs.docker.com/glossary/?term=docker), docs (https://docs.docker.com/glossary/?term=docs), article (https://docs.docker.com/glossary/?term=article), example (https://docs.docker.com/glossary/?term=example), https (https://docs.docker.com/glossary/?term=https), daemon (https://docs.docker.com/glossary/?term=daemon), tls (https://docs.docker.com/glossary/?term=tls), ca (https://docs.docker.com/glossary/?term=ca), certificate (https://docs.docker.com/glossary/?term=certificate)