Connecting to the `CotSS` IRC Service

July 3, 2020

1 Introduction

IRC (Internet Relay Chat) is a protocol that allows users to communicate over the Internet; specifically, it allows a *client* (you) to communicate with a *server* (`CotSS`, located at `frostsnow.net`). By itself, IRC sends information in *plaintext*, meaning that anyone can view and even modify said information. In order to prevent this, the IRC protocol can be encapsulated by the TLS (Transport Layer Security) protocol, which turns the plaintext into *ciphertext* which can be neither read (giving *confidentiality*) nor modified (giving *integrity*). In addition, TLS provides *authenticity*, which verifies the server to the client (letting the client know that it is indeed communicating with `CotSS` and not an imposter) and optionally verifies the client to the server (letting the server know that it is talking with a specific client). Lastly, TLS provides *authorization*, which verifies that the client is allowed to connect to the server. This guide will show you how to use TLS to connect to the `CotSS` IRC server.

TLS uses a type of cryptography often referred to as "Public-Private Key" Cryptography (formally, *Asymmetric* Cryptography), because each party has a pair of keys. A *public* key is shared between parties while a *private* key is kept secret. In TLS, the public key is called a *certificate* (more specifically, an *X.509* certificate). Therefore, in order to communicate with `CotSS`, you will need three things: 1) the server's certificate, in order for you to verify that the computer you are talking to is in fact `CotSS`; 2) your certificate, in order for the server to verify that it is talking to you; and 3) your private key, in order to verify to the server that you are the proper owner of the certificate. Note that the server will use its private key in order to verify to you that it is the proper owner of its certificate.

2 Connecting

Connecting will involve two steps: 1) verifying the server to yourself, and 2) verifying yourself to the server.

2.1 Server Verification

Verifying the server to yourself will also consist of two steps: 1) obtaining a trusted copy of the server's root certificate, and 2) configuring your client to use said certificate.

The best way to verify that you have a *valid* copy of the server's root certificate (and can therefore trust it) is to get its *fingerprint* in-person from someone whom you trust, then verify the fingerprint of the certificate which you obtain online against the trusted fingerprint. You must either do this on your own in-person, or blindly trust your luck that no one has tampered with the root certificate; if you choose to blindly trust your luck, do keep in mind that if your root certificate is an imposter's, then all subsequent connections will be vulnerable to the same imposter, thus you should make an attempt to verify the certificate if you are able to do so.

In order to get the certificate, run the commands:

```
$ mkdir ~/irc
$ wget https://www.frostsnow.net/contact/cotss.pem
$ mv cotss.pem ~/irc
```

The `$` denotes the beginning of a command. The first command creates a directory named `irc` in your home directory. The second command retrieves the certificate for CotSS from a website. The final command moves the certificate to an expected location.

Now verify the root certificate by comparing its fingerprint against one you've received from a trusted source:

```
$ openssl x509 -in ~/irc/cotss.pem -fingerprint \
    -sha256 -noout
```

The `-in ~/irc/cotss.pem` instructs the command to use the certificate which you just retrieved from the website. `-fingerprint` tells the command to generate a fingerprint of the specified certificate. `-sha256` makes the command use a more secure hash function than the default SHA-1. Last, `-noout` prevents the command from redundantly printing the certificate. If the fingerprint output by the command matches your trusted source, then you have successfully validated it!

For the curious, in order to actually view the contents of the certificate, run the command:

```
$ openssl x509 -in ~/irc/cotss.pem -text -noout
```

... which should then show various information about the certificate. For extra fun, perform the same steps except use `google.com:8080` instead of `frostsnow.net:6697`.

Now that you have a copy of the server's certificate, you need to tell your IRC client to use it as a *Certificate Authority* (CA); Certificate Authorities determine which certificates are valid (roughly speaking). This step of configuring your IRC client is client-specific, in other words, different clients will have different methods of configuration. This guide will use the `irssi` client; if you are using

another client, you will need to look at its documentation in order to figure out how to perform this step. In order to connect using the certificate you've downloaded, launch the `irssi` program and connect with the command:

```
/connect -ssl_verify -ssl_cafile ~/irc/cotss.pem \
    frostsnow.net 6697
```

This will instruct `irssi` to connect to `frostsnow.net`, port `6697` (again, where `CotSS` is located) using the file `~/irc/cotss.pem` as a Certificate Authority, after which you will propmpty be disconnected from the server as you have not yet provided a valid client certificate, which is the topic of the next section. Go ahead and type `/rmreconns` to prevent `irssi` from trying in vain to reconnect to the server, then type `/exit` to quit `irssi`.

2.2 Client Verification

Creating a client certificate is more involved than getting the server's certificate because you must have your certificate *signed* by the appropriate certificate authority. While this can be done by hand, the `afrc` (Admin, Friend, Referrer Client) script exists in order to make the process easier and the instructions will thus make use of it. You will need to: 1) initialize your AFR client, 2) send the certificate signing request, 3) receive the certificate signing request, and 4) configure your IRC client to use the signed certificate and private key.

Begin by installing the `afrc` utility. Use the `git` command in order to clone the latest version of the utility's code repository:

```
git clone https://github.com/clinew/afr.git
```

Then change directory into the repository and install the utility:

```
cd afr
sudo make install
```

The utility should now be installed! Though you may need to configure your shell to include locally-installed utilities via:

```
export PATH=/usr/local/bin:$PATH
```

if the following commands fail with `command not found`.

With the command installed it is now time to initialize your client certificate infrastructure and generate a *certificate signing request* (CSR). Figure out what you want your *Erisian Holy Name* (think: "nickname") to be and then run the following, replacing `NAME` with your desired name:

```
afrc init NAME
```

The certificate signing request must now be signed (obviously). Ideally, this should be done in the same way as for validating the server certificate: either using GPG signatures with a previously-verified public key or by physically exchanging the files in person. In practice, you'll probably just e-mail the

`~/irc/csr.pem` file to the person who invited you like a lazy fuck. You ought to know how to do that.

The signer will then provide you with a signed certificate. Assuming that the signed cert is located in a file named `cert.pem`, install the certificate by running:

```
afrc receive-client cert.pem
```

Lastly, you must configure your client to use both the signed certificate and your secret key. As with setting up the server certificate, these instructions are specific to the `irssi` client and if you are using a different client then you will need to adapt the instructions to your client. Also, the following command will assume that you have successfully obtained and verified the server's certificate as per the preceding section. Now, in order to use your newly-obtained client certificate, start `irssi` and run the following command, replacing `USER` with your username (and without backslashes and all on one line):

```
/connect -ssl_verify -ssl_cafile ~/irc/cotss.pem \
    -ssl_cert ~/.afrc/USER/certs/USER.pem \
    -ssl_pkey ~/.afrc/USER/private/USER.pem \
    frostsnow.net 6697
```

The additional arguments, `-ssl_cert ~/.afrc/USER/certs/USER.pem` and `-ssl_pkey ~/.afrc/USER/private/USER.pem` specify your client certificate and your private key, respectively. After running the command, you should now be connected to IRC![1]

3 Using `irssi`

This section provides a quick introduction for using the `irssi` client. If you already know how to use `irssi` or some other IRC client then you may skip this section.

3.1 Autoconnect

Unless you are a fan of circumlocution, you probably do not wish to re-type the preceding `/connect` command every time you connect to `CotSS`, thus it makes sense to configure `irssi` to connect automatically on startup. In order to do this it must first be understood that IRC consists of a *network* of interconnected *servers*. Therefore, in order to set up autoconnect we must tell `irssi` both the network **and** the server that we are connecting to. For `CotSS`, which has only one server, this seems rather redundant, but remember that most IRC networks consist of multiple servers and thus `irssi` reflects this fact. Start by running the command:

```
/network add cotss
```

[1] If not then curse, take a deep breath, and carefully re-read the instructions. Failing that, feel free to contact someone knowledgeable for help.

... which adds a network with the name `cotss`. Next add the server to the `cotss` network:

```
/server add -auto -network cotss -ssl_verify \
    -ssl_cafile ~/irc/cotss.pem \
    -ssl_cert ~/.afrc/USER/certs/USER.pem \
    -ssl_pkey ~/.afrc/USER/private/USER.pem \
    frostsnow.net 6697
```

This mirrors the `/connect` command, except instead of connecting it is configuring `irssi` to automatically connect to the server with the given arguments on startup. Save the configuration with:

```
/save
```

... and then restart `irssi` to automatically connect!

3.2 Navigation

`irssi` is divided into *windows*, which can contain either *channels* or *private messages*. Windows are *numbered*, with window number `1` holding status messages and subsequent windows numbered sequentially after that (`2`, `3`, `4`, ...). You can join channels by running `/join <channelname>` where `<channelname>` is, obviously, the name of the channel that you wish to join; note that, in IRC, channels start with a `#` symbol (which is **not** a Twatter hashtag). Likewise, you may message users by typing `/msg <username>` where `<username>` is, obviously, the nickname of the user that you wish to message. Both of these actions will open up a new window (`/join` will actually **activate** that window), and you can then navigate between windows by either typing `/window <number>` or by pressing `Alt+<number>` (depending on your terminal) where `<number>` is the number of the window that you wish to activate. More information and options may be found by typing `/help window`.

4 NickServ

By default, anyone may use any nickname that is not currently in use after connecting to IRC. In order to claim ownership of a nickname, one must *register* it with *services* by messaging the bot named `NickServ` and then *identify* to that nickname on subsequent connections. The easiest way to do this on `cotss` is to 1) register with `NickServ` and then 2) identify using your client certificate. In order to register, run the following command while on the nick that you wish to register:

```
/msg NickServ register <your_password> \
    <your_bogus_email>
```

... where `<your_password>` should be a **unique** password for your account (don't send me a password that you've previously used elsewhere, that's just

stupid) and <your_bogus_email> is some gibberish (all e-mail functionality has been disabled for cotss). After that, you may manually identify to NickServ on subsequent connections by running the command:

```
/msg NickServ identify <your_password>
```

. . . but that's a pain, so the easier option is to add your client certificate's *fingerprint*, which is a unique identifier for that certificate, to NickServ's list of valid fingerprints for your account. You can add your current fingerprint to NickServ's list of valid fingerprints for your account by running:

```
/msg NickServ cert add
```

. . . and you should now be identified automatically when you connect!

More information about NickServ's services may be found by running /msg NickServ help.

5 Inviting Friends

Depending on your relationship with me, the server admin, you may be able to *sign* additional client certificates, thereby inviting your friends to cotss. This certificate architecture is known as "Admin, Friend, Referred" based on the idea that there is a server admin, their friends, and those whom said friends have referred. In order to do this you will need to request and receive a *referrer* certificate from me and then use that certificate in order to supply a certificate revocation list (CRL) to me. This process is extremely cumbersome on its own, but the afrc utility should make it much easier.

5.1 Obtaining a referrer certificate

Begin by requesting a referrer certificate:

```
afrc request-referrer
```

Read the command output in order to find out where the CSR for the referrer certificate is located, then send that request to me for signing. Once (and "if") I send you the signed referrer certificate, then, assuming the signed certificate is located at ~/referrer.pem, run the following command in order to install the referrer certificate:

```
afrc receive-referrer ~/referrer.pem
```

When you install the certificate, the relevant CRL will automatically be generated for you. Read the command output and then be sure to send the CRL file to me, otherwise none of the users you invite will be able to connect!

5.2 Signing Friend's Certificates

In order to sign your friend's certificate, your friend must first create a *certificate signing request* (CSR), as explained earlier in this guide, and then send the

signing request to you. When you receive the signing request, then, assuming the request is in the file `~/csr.pem`, you should first verify that it is of the appropriate key *length* and has the correct *Common Name* before signing it by running:

```
$ openssl req -in ~/csr.pem -text -noout
```

. . . and making sure that the "Public-Key" reads `4096 bit` (and not, say, `2048 bit` or lower) and that the "Subject" line reads something akin to `Subject: CN=yourfriend`[2].

Now that you've verified the certificate signing request, you may sign the certificate by running the following command with `NAME` replaced by a name with which to refer to your friend with:

```
$ afrc sign-referred ~/csr.pem NAME
```

Then all you need to do is read the command output in order to find the signed certificate and then send your friend their signed client certificate!

5.3 Revoking a Friend's Certificate

In the unfortunate event that you need to revoke a friend's certificate, it may be done by running the following command, where `NAME` is replaced with the name of the friend whose certificate you are revoking:

```
$ afrc revoke-referred NAME
```

Although you have revoked your friend's certificate, you must now inform the server by sending the updated CRL file to me (the admin); the command output will tell you where to find the CRL file. You will have then completed revoking your (possibly former) friend's certificate. Beware: this action cannot be undone.

6 Migrating from friend-of-friend

This guide prior to mid-2020 used a Public Key Infrastructure (PKI) which I called "Friend-of-Friend"; that was essentialy version 0.1 of what I now call "Admin, Friend, Referrer" version 0.2. This was done in order to address several potential security issues with the old vesion. As a result, the old server certificates and client certificates will need to be migrated away from.

Thankfully, due to the wonders of certificate validation, it is possible to configure the server such that clients which trust either the old or new root server certificate will trust the server and also both old and new client certificates will be trusted by the server! However, it is not desireable to maintain this state of affairs indefinitely due to the potential security issues, thus you will need to migrate to the new root server and client certificate.

[2]Note that key lengths below `4096` *may* be accepted by the server as-is, but are not supported and may break at any time, as weak crypto will be deprecated today and not tomorrow.

While it is possible to use your old private key in order to generate a new client certificate, it'll be easiest to follow the above instructions from the beginning and get rid of your old client certificate and the old root server certificate.