CS 161 Project 3: Hidden Service Chat

Due on November 20 at 5:00 PM.

Introduction

In this project you will use a Tor hidden service to run an anonymous chat room. You will be able to chat without revealing your IP address and without knowing the IP address of the server. You will also perform the role of a TLS certificate authority, generating a root certificate and signing a leaf certificate to authenticate your chat service.

This time around, the project programs are in Python 3. As before, the parts you have to write are marked with `TODO` comments.

Step 0. Try the chat program and compile Tor

First extract the project source code.

```
cd
tar xzvf cs161proj3.tar.gz
cd cs161proj3
```

The chat system consists of two files, client and server. The programs run, but they are missing certain features that you will add in this project. You will need at least two terminal windows, one for the server and one for a client. Run the server like this:

```
./server --disable-tls 127.0.0.1 5280
```

Run the client like this:

```
./client --disable-tls 127.0.0.1 5280
```

Try typing a message into the client. You should see:

```
connecting
connected
*** user 4 entered the room.
hello
<user 4> hello
```

If you connect another client, you will be able to chat. Type Ctrl-C or Ctrl-D to exit.

By default, the programs try to use TLS—but you haven't implemented that yet. The --disable-tls option makes the programs run in an inscure plaintext mode.

The number `5280` is just the port number. You may have to change it on Hive if there are other people running the program at the same time. You will know you have to change the port number if you see the error message:

```
socket.error: [Errno 98] Address already in use
```

Next you need a copy of Tor that you can configure to run a hidden service. We will compile and install a copy of Tor on Hive. (Tor Browser comes with a copy of Tor. You could conceivably configure that copy of Tor to run a hidden service, but it would only be running while the browser was running. So we will run a separate standalone copy of Tor.)

These instructions are a short version of the instructions at

https://www.torproject.org/docs/tor-doc-unix

They are also modified slightly to install under your home directory so you don't need root privileges.

```
cd
wget https://sourceforge.net/projects/levent/files/libevent/libevent-2.0/libevent-2.0.22-stable.tar.gz
wget https://www.torproject.org/dist/tor-0.2.6.10.tar.gz
tar xzvf libevent-2.0.22-stable.tar.gz
cd libevent-2.0.22-stable
./configure --prefix=$HOME/usr --disable-shared --enable-static --with-pic
make
make install
cd
tar xzvf tor-0.2.6.10.tar.gz
cd tor-0.2.6.10
./configure --prefix=$HOME/usr --enable-static-libevent
make
make install
```

At this point you should be able to bootstrap Tor by running

```
~/usr/bin/tor
```

Press Ctrl-C to exit.

Step 1. Configure a hidden service

Begin by setting up a hidden service to run the chat server. You need to do this first, because the process of setting up a hidden service generates the service's hostname, and you need to know the hostname to create the TLS certificate in the next step. In order to connect to your hidden service as a client, you need to add proxy support to the client program, because Tor acts as a local proxy server.

How a hidden service works is you start a server listening on the localhost address 127.0.0.1:

```
./server --disable-tls 127.0.0.1 5280
```

(You might have to choose a different port number if the port is already in use.) Then you configure Tor to create a "virtual port" for the onion service that will be forwarded to the listening localhost server. The important configuration options you need are HiddenServiceDir and HiddenServicePort:

https://www.torproject.org/docs/tor-manual#HiddenServiceDir

https://www.torproject.org/docs/tor-manual#HiddenServicePort

The file server-torrc in the project source code shows how to use these options.

To run Tor using the hidden service configuration, do:

`~/usr/bin/tor -f server-torrc`

Running this command will create a directory called hs-chat-dir that contains two files:

hostname: the automatically generated onion domain name.

private_key: the hidden service's private key.

You will submit these two files. You need to know the hostname to connect to the hidden service. We will call it *yoursite.onion* in all the examples below.

Tor is a SOCKS proxy. To connect to the hidden service, you need to be running Tor locally. If you are using Tor Browser, the Tor proxy port is 9150; and if you installed Tor from apt-get or similar, the Tor proxy port is 9050. If you are running on Hive, you need to edit server-torrc and set SocksPort to a port number that doesn't conflict with other users. Once Tor is running, the command to connect the client is:

`./client --disable-tls --socks-port 9150 yoursite.onion 5280`

However, if you try that command you will see that proxy support is not implemented. Implement the `connect_with_socks` function. The easiest way to do this is to implement SOCKS4a:

https://en.wikipedia.org/wiki/SOCKS#SOCKS4a

The `connect_with_socks` function should create a socket, send the destination hostname and port number according to the SOCKS4a protocol, read the 8-byte response, and, if successful, return the socket object. If unsuccessful, raise a `socket.error` exception. The Python struct module can help with packing the request and unpacking the response. Make sure you send the port number using big-endian byte order: if you get it wrong, instead of connecting to port 5280 (0x14a0 hex), you'll try to connect to port 40980 (0xa014 hex).

If Tor says:

```
[notice] Closing stream for '[scrubbed].onion': hidden service is
unavailable (try again later).
```

wait a minute and try again.

After you've implemented SOCKS support in the client, you and anyone else who knows the hostname will be able to connect to the server and chat. The server doesn't have to be running on a traditional "server" computer; it can even just be running on an Internet-connected laptop.

Step 2. Enable TLS

In this step you will add TLS support to the client and server. You will act as your own certificate authority (CA) and generate a signing key and root certificate. On the server, you will generate a private key and get a certificate signed by the CA. On the client side, you will trust the CA certificate.

OpenSSL has command line programs that are cumbersome to use but do everything you need to simulate the certificate authority ecosystem. The main command line program is called openssl. We'll be using several subcommands:

openssl genrsa: generate a private RSA key
https://www.openssl.org/docs/manmaster/apps/genrsa.html
openssl req: create a certificate signing request (CSR)
https://www.openssl.org/docs/manmaster/apps/req.html
openssl x509: sign a CSR or display a certificate
https://www.openssl.org/docs/manmaster/apps/x509.html

You will be generating five files total:

ca.key: the CA's private TLS key
ca.crt: the CA's public TLS certificate
my.key: the hidden service's private TLS key (not the same as Tor's private_key)
my.csr: a certificate signing request containing the service hostname and public key
my.crt: a certificate for the hidden service signed by ca.key

First, set up the CA. You will generate a private key file, ca.key, and a self-signed certificate file, ca.crt. The details you fill in here (e.g. Organizational Name) don't matter; try not to make them sound too sketchy. You are a trustworthy certificate authority now.

```
openssl genrsa -out ca.key 4096
Generating RSA private key, 4096 bit long modulus
.................................++
.............................++
e is 65537 (0x10001)
openssl req -x509 -new -nodes -key ca.key -days 365 -out ca.crt
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:California
Locality Name (eg, city) []:Berkeley
Organization Name (eg, company) [Internet Widgits Pty Ltd]:TrustCo
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:
Email Address []:
```

You can examing the certificate you just generated in text form:

```
openssl x509 -noout -text -in ca.crt
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 10217653990368449391 (0x8dcc662b35c3936f)
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=US, ST=California, L=Berkeley, O=TrustCo
        Validity
            Not Before: Nov  2 05:31:48 2015 GMT
            Not After : Nov  1 05:31:48 2016 GMT
        Subject: C=US, ST=California, L=Berkeley, O=TrustCo
```

Notice that the Issuer and Subject are the same. That's what makes the certificate "self-signed". There is no higher authority certifying the validity of this certificate; it becomes trusted by being installed on client systems, just like the self-signed trusted root certificates in your browser.

Next, generate a private key for the hidden service, my.key, and a certificate signing request, my.csr. The only part that is critical is the Common Name field: this must be equal to *yoursite.onion* (i.e., your own generated onion hostname), because it is the hostname that the client will authenticate against.

```
openssl genrsa -out my.key 4096
Generating RSA private key, 4096 bit long modulus
.....++
.........................................++
e is 65537 (0x10001)
openssl req -new -key my.key -out my.csr
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:California
Locality Name (eg, city) []:Berkeley
Organization Name (eg, company) [Internet Widgits Pty Ltd]:cs161
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:yoursite.onion
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

Finally, sign the CSR using the CA's certificate.

```
openssl x509 -req -in my.csr -CA ca.crt -CAkey ca.key \
      -CAcreateserial -out my.crt -days 365
```

This time, the Issuer and Subject are not the same:

```
openssl x509 -noout -text -in my.crt
Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number: 11050466707567830388 (0x995b24ec32dc6574)
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=US, ST=California, L=Berkeley, O=TrustCo
        Validity
            Not Before: Nov  3 02:39:19 2015 GMT
            Not After : Nov  2 02:39:19 2016 GMT
        Subject: C=US, ST=California, L=Berkeley, O=cs161,
CN=yoursite.onion
```

Now you need to add TLS support to the chat programs to make use of your certificate. These are the commands to run, though you still have to implement them:

```
./server --tls-cert my.crt --tls-key my.key 127.0.0.1 5280
./client --tls-trustfile ca.crt --socks-port 9150 yoursite.onion 5280
```

Implement the `options.use_tls` code blocks in client and server. You have to *wrap* a socket object in a TLS layer so that it becomes a TLS socket. Use the Python ssl module:

https://docs.python.org/3/library/ssl.html

You can assume that the SSLContext type and the SSLContext.wrap_socket method are available. On the client side, you need to enable certificate verification (`CERT_REQUIRED`), enable checking of hostnames, load the trusted verification certificates from `options.tls_trust_filename`, and pass `server_hostname=remote_hostname` to the `wrap_socket` call. On the server side, you need to disable certificate verification (`CERT_NONE`; the server does not authenticate clients) and load the certificate chain consisting of my.crt and my.key.

Make sure that you can connect a client to a server using TLS and your certificate. When we test your code, we will also check that trying to connect to a server with a *different* hostname *does not* succeed. (You can test this yourself by running another instance of the hidden service using a separate HiddenServiceDir.)

Concluding Questions

In a separate file called answers.txt, please put down the answers to the following questions:

1. Despite being encapsulated in TLS, the chat protocol itself has flaws. How could one user impersonate another user? (Imagine that you are user 7, and user 5 is also in the room. How could you cause others to see a message on their screen that makes it look as though user 5 sent a message?)
2. Anyone who knows the onion domain of your hidden service can connect to it. Maybe you don't want that. Read the hidden service configuration options at
   https://www.torproject.org/docs/tor-manual#_hidden_service_options
   Which option allows you to restrict service to only authorized clients?
3. The chat server assigns names to connected users according to the file descriptor number of their connected socket (see the `socket_to_username` function). Why is the first connected user called “user 4”; i.e., what are file descriptors 0, 1, 2, and 3?
4. Ignoring performance issues, what were the greatest limitations of Tor you faced in this assignment? If you could make a recommendation to the Tor developers, what would it be?

Submission instructions

Submit your files using the glookup system on a Hive machine. Enter your source code directory and run the command:

```
submit proj3
```

The files you should submit are:

ca.crt
ca.key
client
hostname
my.crt
my.key
private_key
server
answers.txt

Only one member per group needs to submit.

Submissions are due on November 20 at 5:00 PM.