Assignment 5 (100 points)

Anirudh Sivaraman

2020/11/20

The goal of this assignment is to familiarize yourself with some concepts in network security. This assignment mostly involves writing small Python programs for each question. Unlike previous assignments, there is no starter code because the amount of Python you need to write is very small (about 10 lines of Python for each question). Instead, you will have to refer to the Python API documentation for the secure sockets layer to help you write the programs: `https://docs.python.org/3/library/ssl.html`. This assignment is also intended to give you a taste of networking in the real world, e.g., figuring out how to use a network API and implementing a new network protocol (the Diffie-Hellman key exchange). We use the terms SSL and TLS interchangeably in this assignment.

1 SSL clients (20 points)

For this question, you need to implement an SSL client to fetch data from an already running HTTPS server (HTTP over TLS/SSL). The server's domain name is www.google.com and the port number is 443, the default TCP port for TLS. For implementing this client, you will have to open a TCP connection to www.google.com, then create a TLS connection on top of this, then issue the HTTP request below on this TLS connection.

```
GET /index.html HTTP/1.1\r\nHost: www.google.com\r\n\r\n
```

The `\r\n` is important. Then, your client must read up to 1024 bytes of response from the server. If you have written your program correctly, you should have output that looks like the output below. Our output is formatted to look neat using the Python pprint module, but if your output shows similar content, that's sufficient. If it helps, the Python API docs for SSL (`https://docs.python.org/3/library/ssl.html`) has example Python program snippets for an SSL client that you can reuse for this assignment. Repeat the same exercise for www.amazon.com, www.nytimes.com, and www.facebook.com, and paste your output as part of the assignment submission.

```
[b'HTTP/1.1 200 OK',
 b'Date: Tue, 17 Nov 2020 23:41:01 GMT',
 b'Expires: -1',
 b'Cache-Control: private, max-age=0',
 b'Content-Type: text/html; charset=ISO-8859-1',
 b'P3P: CP="This is not a P3P policy! See g.co/p3phelp for more info."',
 b'Server: gws',
 b'X-XSS-Protection: 0',
 b'X-Frame-Options: SAMEORIGIN',
 b'Set-Cookie: 1P_JAR=2020-11-17-23; expires=Thu, 17-Dec-2020 23:41:01 GMT; pat'
 b'h=/; domain=.google.com; Secure',
 b'Set-Cookie: NID=204=GAYfcyguTobM5_2ZblMQdMgF45jdYe8e-dVPD0aG661UnjcVN_0Z_bop'
 b'y9pso3Gox8SGIgKwgr3ZPXZI9imQXW3vox71wAZW2mvWJMg5tpP4Y9tFWFSvidLk5LT4DEar2LHA'
 b'IsRTGLuEtWIa8StlHPxLF7Ean4MsTOT99jLxt98; expires=Wed, 19-May-2021 23:41:01 G'
 b'MT; path=/; domain=.google.com; HttpOnly',
 b'Alt-Svc: h3-Q050=":443"; ma=2592000,h3-29=":443"; ma=2592000,h3-T051=":443";'
 b' ma=2592000,h3-Q046=":443"; ma=2592000,h3-Q043=":443"; ma=2592000,quic=":443'
```

```
 b'"; ma=2592000; v="46,43"',
 b'Accept-Ranges: none',
 b'Vary: Accept-Encoding',
 b'Transfer-Encoding: chunked',
 b'',
 b'4bed',
 b'<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang='
 b'"en"><head><meta content="Search']
```

Try increasing 1024 bytes to 4096 bytes. Does your output change? Why or why not?

2 SSL certificates (10 points)

In each of the cases above, print out the certificate presented by the Google/Amazon/Facebook/NYTimes server to your SSL client. You can use the function getpeercert() along with the pprint.pprint() function from the pprint module to print out the certificate. This is what the printed certificate should look like:

```
{'OCSP': ('http://ocsp.pki.goog/gts1o1core',),
 'caIssuers': ('http://pki.goog/gsr2/GTS1O1.crt',),
 'crlDistributionPoints': ('http://crl.pki.goog/GTS1O1core.crl',),
 'issuer': ((('countryName', 'US'),),
            (('organizationName', 'Google Trust Services'),),
            (('commonName', 'GTS CA 1O1'),)),
 'notAfter': 'Jan 20 16:23:45 2021 GMT',
 'notBefore': 'Oct 28 16:23:45 2020 GMT',
 'serialNumber': 'EC9743FA9BC4A21902000000007FD501',
 'subject': ((('countryName', 'US'),),
             (('stateOrProvinceName', 'California'),),
             (('localityName', 'Mountain View'),),
             (('organizationName', 'Google LLC'),),
             (('commonName', 'www.google.com'),)),
 'subjectAltName': (('DNS', 'www.google.com'),),
 'version': 3}
```

What do the fields crlDistributionPoints, notAfter, and notBefore mean here? It is OK to use the Web to answer this, but the answer must be written in your own words.

3 SSL server (30 points)

We'll now create an SSL server. Creating a server requires creating a certificate, which then needs to be signed by a certificate authority. While considerable progress has been made in automating this process and reducing manual intervention (e.g., `https://letsencrypt.org`), this process still requires you to demonstrate ownership of a domain before getting a certificate. Hence, to simplify things for this assignment, we'll use self-signed certificates, where you yourself will certify that you are who you say you are. This is obviously not secure and not recommended for practical use. To create a self-signed certificate, you can use the instructions provided here: `https://docs.python.org/3/library/ssl.html#certificates`.

Once you have created a self-signed certificate, which consists of both a certificate as well as a private key, use the SSL API reference to create a server SSL socket using that self-signed certificate. To do so, you should bind the server's socket to localhost on a port of your choice. Then connect to this server's socket using a client that creates an SSL client socket.

Because SSL enforces good security practices by default, this won't work out of the box because the client will reject the server's self-signed certificate as invalid. To override this setting, use the following Python snippet:

```
context.check_hostname=False
context.verify_mode=ssl.CERT_NONE
```

At the end of this process, to earn points for this assignment, the client should be able to send Hello to the server on an encrypted TLS connection and the server should be able to receive and print it out.

4 The mechanics of SSL (10 points)

Once you have the SSL client and server in the previous connection working on localhost, use Wireshark on your loopback interface (usually named lo or something similar) to observe the packets that are exchanged between the SSL client and server. Attach a screenshot of all the packets exchanged between your SSL client and server during the process of the client sending Hello to the server. Can you identify the TCP handshake? Besides the TCP handshake packets and the encrypted data packets, you should see packets related to TLS. What function do these packets serve? It's OK to search the Web for an answer to this question, but again, you should write the answer in your own words.

5 Diffie-Hellman key exchange (30 points)

Recall that asymmetric encryption and decryption (i.e., with private and public keys) is slow, while symmetric encryption and decryption is much faster. However, for symmetric encryption and decryption, we need a key that is shared between the 2 communicating parties. One way to establish this key is to create the shared key at either of the 2 parties and use asymmetric encryption and decryption to transfer it securely to the other end. However, if the asymmetric private keys are stolen in the future, then the shared key created for past communication sessions between the two parties becomes vulnerable. This is because if the adversary who stole the private keys happened to passively record previous sessions, the adversary can then use these private keys to decrypt the beginning of these sessions to determine the shared key. With the shared key in hand, the adversary can then decrypt the rest of the session, which is encrypted using this shared key.

A solution to this is the Diffie-Hellman key exchange. This is a simple, but very effective algorithm to create a shared key for 2 parties without ever transmitting this shared key between the 2 parties (either as clear text or encrypted text). Diffie-Hellman key exchange relies on 2 parties exchanging some amount of data, but never the shared key. This data exchange is sufficient for each of the 2 parties to then independently compute the shared key.

Your goal for this question is to implement Diffie-Hellman key exchange. The Wikipedia page: `https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange#Cryptographic_explanation` is a good reference for implementing it. For this assignment, you can use a UDP socket for exchanging data between 2 parties (which we'll call the client and server, even though there is no real distinction between the two for Diffie-Hellman key exchange) both running on localhost. On localhost, packet drops are very rare, making UDP sufficient, even though it is unreliable. You can use code snippets from the sockets lecture to transfer data between the UDP sockets. Your output should look like this:

First, execute the server:

```
python3 diffie_hellman_server.py
```

Then, execute the client:

```
python3 diffie_hellman_client.py
```

After both commands are executed, the shared secret computed by the server and client should be printed out on their respective terminals. Please use the values of g and p suggested in `https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange#Cryptographic_explanation`. In practice, you should randomize the choice of a and b by picking both uniformly from a large range of natural numbers. However, for this assignment alone, you can pick from the range 1 to 10 to simplify computation of exponents like g^a. In practice, an approach called modular exponentiation is used when the exponents are very large numbers.

The server vs. client distinction is completely arbitrary in Diffie-Hellman key exchange. We are using the term server in our implementation to refer to the entity that listens for data from the other entity (the client)

before sending data to the other entity. This is so that both entities don't wait around indefinitely for the other to send data first. Feel free to use TCP instead of UDP for this question if you prefer that more.

6 Extra credit

With a full score on this question, you can earn 2 points of extra credit towards your final grade by answering at most 1 out of the 2 questions below. Recall that extra credit is capped at 10 points.

6.1 Option 1

Download, install, and use the Tor browser to access `https://cs.nyu.edu/~anirudh/CSCI-UA.0480-062/`. Attach a screencast of your browsing process using Tor. Use a moderate resolution (e.g., 720 p) to ensure your browser details show up in the screencast while reducing the storage required for the screencast. Please ensure your screencast is under 50 MBytes, ideally around 25 MBytes or so due to GitHub limitations: `https://docs.github.com/en/free-pro-team@latest/github/managing-large-files/conditions-for-large-files`

6.2 Option 2

The Heartbleed bug was a serious bug that affected the widely used OpenSSL TLS/SSL library. Describe this bug in your own words and explain how this bug could be exploited to steal sensitive information such as private keys. You might find the following links helpful: `https://jhalderm.com/pub/papers/heartbleed-imc14.pdf` and `https://blog.cloudflare.com/searching-for-the-prime-suspect-how-heartbleed-leaked-private-keys/`. You will have to synthesize information from these links to answer the question, but the final answer should only be a couple of paragraphs (think at most 500 words).