Assignment 5

Anirudh Sivaraman

adopted with minor modifications for UT Austin's CS 356 (2024) by Venkat Arun

2024/11/18

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. For this assignment, 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 intended to give you another 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.

This assignment also uses Mypy (`http://mypy-lang.org/`). Mypy type hints give you an idea about the expected input and return types of each function signature. Mypy is entirely optional for this assignment, but can help you validate the correctness of your implementation. You can install Mypy using the following command:

```
pip3 install --upgrade mypy
```

You can run mypy on your code with this command

```
python3 -m mypy [FILE_TO_CHECK].py
```

where `FILE_TO_CHECK` represents the Python file you want to check.

Submission

"Assignment 5 Conceptual" in Gradescope contains the written questions. Some of these will depend on your solutions for the coding assignments.

Your submission to "Assignment 5 Coding" should consist of 4 files:

```
diffie_hellman_client.py
diffie_hellman_server.py
ssl_web_client.py
ssl_web_server.py
```

1 Diffie-Hellman key exchange

In this portion of the assignment you will implement a basic version of the Diffie-Hellman key exchange protocol.

1.1 Background

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 'Assignment 5

Conceptual" in Gradescope contains the written questions. ++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.

Diffie-Hellman key exchange solves this problem. It 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, and any secret data used by each party can also be discarded once the shared key is established.

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. The lecture on Oct 29, 2024 discussed it.

1.2 Starter Code

We provide some starter code in the following files.

```
diffie_hellman_server.py
diffie_hellman_client.py
```

Unlike previous assignments, this starter code is relatively bare; it only inlcudes code for processing the necessary command line arguments, and specifies the fuction signatures `dh_exchange_client` and `dh_exchange_server`. It is up to you to decide how exactly you want to implement the communication between the client and server (i.e. TCP or UDP), and how exactly that communcation should work.

However, there are a few high level requirements:

- Your client and server need to agree on a prime modulus and base to use

- The protocol should be randomized, and should not return the same shared secret key in every time (unless a random seed is set)

- No information that would allow an adversary to compute the secret key should be sent between the client and server. Obviously, this includes the secret key itself!

We do not require you to implement a pratically secure implementation (i.e. by using a large prime or modulus). For this assignment numbers between 1 to 100 are sufficient. For fun, you can try using larger modulii (2048-bit numbers are believed to be secure against today's best computers) which should be easy to do since Python integers are arbitrary precision by default. However, this will not be evaluated.

Also note, the server/client distinction is arbitrary when using the 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.

The starter code also provides some TODOs and defines a helper function that you can use as a guide for your implementation. However, you are not obligated to follow these exactly; as long as the `dh_exchange_client` and `dh_exchange_server` function signatures stay the same and return the correct information your implementation will be considered correct.

Following is the output of our solution (by running the server first, then the client). We recommend that your output be similar, but again we will actually check the return values of the `dh_exchange_client` and `dh_exchange_server`.

```
$ python3 diffie_hellman_server.py
Base int is 11
```

```
Modulus is 31
Secret is 2
Int received from peer is 29
Shared secret is 4

$ python3 diffie_hellman_client.py
Base proposal successful.
Base int is 11
Modulus is 31
Secret is 3
Int received from peer is 28
Shared secret is 4
```

1.3 Extra Credit

Submit any documents or code you write to "Assignment 5 Extra Credit" in Gradescpe.

Interoperability (10% bonus)

You can get extra credit by showing your implementation is interoperable with a partner's. To do this, you will need to run your server on one of your CloudLab VMs so it is accessible to your partner's client. Furthermore, you will have to agree on the transport protocol and format for transferring data (i.e. the integers used in the protocol). Compile a document describing the common format, and showing the clients/servers working together. Note, you may have to change the firewall rules to allow access to the port that you are using. Alternately, most (but not all) firewalls allow port numbers above 1024 by default.

We emphasize again that besides this extra credit opportunity, the assignment should be done in groups of 1 or 2. Your group should have its own implementation of both the client and the server, and should not copy any code from each other

Cracking Diffie Helman (10% bonus)

Write a simple program that takes the publicly visible information from a Diffie Helman key exchange and attempts to compute the shared secret key. Demonstrate breaking the security of your own implementation by showing the output of your client and server, and also your cracking program computing the shared secret.

Note, a "simple" program to break Diffie Hellman could consist of simply trying each possible value of the secrets. Of course, there are simple ways to improve on that as well, and we encourage you to try out any ideas you have.[1]

Create a modified version of your Diffie Hellman key exchange client/server that uses a larger modulus/base, and show that your cracking program takes longer to compute the shared keys. Can you estimate how long it would take your program to crack a realistic Diffie Hellman implementation with 2048-bit numbers?

2 Implementing SSL Web Client and "Server"

For this part of the assignment, we will use the Python ssl and socket libraries to implement a basic web server and client, both of which can be enabled to use TLS (and therfore serve/receive HTTPS rather than HTTP traffic). Starter code is provided in `ssl_web_client.py` and `ssl_web_server.py`.

The actual code you will write is relatively small. You will instead use these programs to answer some questions about TLS and Public Key Infrastructure. Part of these questions will also involve deploying your web server on a VM, assigning it a domain and obtaining a real, usable certificate for your server.

---

[1]If you come up with a *really* efficient method, that would be a breakthrough in Cryptography! Please let us know if you do :)

2.1 HTTP(S) Background

We have discussed only a small fraction of HTTP fields in this course. Obviously, there are many technical details that go into HTTP, but for this assignment we will implement a bare minimum HTTP client and server, such that they are interoperable with existing websites and browsers, respectively.

Therefore, all you need to know about HTTP is that it is a text based protocol run over TCP. Client requests and server responses consist of a header with a defined format, followed by some content separated from the header by new lines. A client connects to a TCP server and sends the request as text over the TCP socket, and the server replies with the response and closes the socket.[2] The starter code provides some examples of well formed HTTP requests and responses that you can use.

HTTPS is simply HTTP run over a TCP socket secured with TLS.

HTTP and HTTPS by default run on ports 80 and 443 respectively, and websites will expect HTTP connections on these. However, you can run HTTP(S) servers on arbitrary ports if desired.

2.2 Implementation

The starter code is relatively structured, with a few spots annotated with TODOs. Please make sure that your code keeps the same command line arguments. However, you are also free to add additional command line arguments, for example to deal with self signed certificates.

We recommended implementing things in the following order.

1. First get `ssl_web_client.py` to work with SSL disabled. This should be relatively easy since you should have experience implementing TCP sockets from assignment 1. You can test your implementation against any website, or on localhost by using the `netcat` tool.

2. Next, implement wrapping the client socket with an SSL socket (if the `-ssl` flag is set). Again, you will need to refer to the Python ssl libary: `https://docs.python.org/3/library/ssl.html`. You can now test your client against existing websites that support HTTPS. Remember to change the port if SSL is enabled!

   Also, as described in the starter code, you need to implement having your client print the certificate provided by the server *(hint: use `getpeercert()`)*

At this point, your client is complete, and you you should be able to complete the questions associated with Part 3 of this assignment. You can now begin implementing the server.

3. Again, first get `ssl_web_server.py` to work with SSL disabled. You should to make sure the server works with your client (with SSL also disabled). You may also want to test your server with a browser.

   Note, by default `ssl_web_server.py` runs on port 8000. Browsers support specifying a specific port like so: `http://localhost:8000`. Note you can replace `localhost` with the IP of you Cloud VM if you are using one.

4. Finally, implement wrapping connections with a SSL socket if the `-ssl` flag is set. Note your server will need a certificate; see Part 4 of the assignment for details. Make sure your implementation is compatible with your client. When testing your SSL server with a browser, also be aware that the URL needs to start with `https://` instead of `http://`.

We will run some basic automated tests with your client and server, however the majority of the assignment will consist of completing the questions in the following sections.

---

[2]Nowadays, clients and servers will keep a connection open to facilitate multiple requests without the overhead of establsihing a new connection. But again, we are doing the bare minimum for this assignment!

3 Surfing the web with SSL

Let's ue your SSL enabled web to explore some websites. Note, your client should read at least 1024 bytes of response from the server. If you have written your program correctly, you should have output that looks like the output below.

```
$ python3 ssl_web_client.py www.example.com --ssl --port 443
```

```
{'OCSP': ('http://ocsp.digicert.com',),
 'caIssuers': ('http://cacerts.digicert.com/DigiCertGlobalG2TLSRSASHA2562020CA1-1.crt',),
 'crlDistributionPoints': ('http://crl3.digicert.com/DigiCertGlobalG2TLSRSASHA2562020CA1-1.crl',
                           'http://crl4.digicert.com/DigiCertGlobalG2TLSRSASHA2562020CA1-1.crl'),
 'issuer': ((('countryName', 'US'),),
            (('organizationName', 'DigiCert Inc'),),
            (('commonName', 'DigiCert Global G2 TLS RSA SHA256 2020 CA1'),)),
 'notAfter': 'Mar  1 23:59:59 2025 GMT',
 'notBefore': 'Jan 30 00:00:00 2024 GMT',
 'serialNumber': '075BCEF30689C8ADDF13E51AF4AFE187',
 'subject': ((('countryName', 'US'),),
             (('stateOrProvinceName', 'California'),),
             (('localityName', 'Los Angeles'),),
             (('organizationName',
               'Internet\xa0Corporation\xa0for\xa0Assigned\xa0Names\xa0and\xa0'
               'Numbers'),),
             (('commonName', 'www.example.org'),)),
 'subjectAltName': (('DNS', 'www.example.org'),
                    ('DNS', 'example.net'),
                    ('DNS', 'example.edu'),
                    ('DNS', 'example.com'),
                    ('DNS', 'example.org'),
                    ('DNS', 'www.example.com'),
                    ('DNS', 'www.example.edu'),
                    ('DNS', 'www.example.net')),
 'version': 3}
========================= HTTP Response =========================
HTTP/1.1 200 OK
Accept-Ranges: bytes
Age: 465362
Cache-Control: max-age=604800
Content-Type: text/html; charset=UTF-8
Date: Fri, 12 Apr 2024 19:17:24 GMT
Etag: "3147526947"
Expires: Fri, 19 Apr 2024 19:17:24 GMT
Last-Modified: Thu, 17 Oct 2019 07:18:26 GMT
Server: ECS (nyd/D151)
Vary: Accept-Encoding
X-Cache: HIT
Content-Length: 1256
```

Note, depending on the website and how you implemented your client (specifically the specific `recv()` calls) you may only see the HTTP header and not the website content. One example of a website where the solution client would see the website body is `news.ycombinator.com/`.

Questions

(Refer to Gradescope)

4 Deploying the Web Server

This section provides details about setting up the SSL server. We also provide some instructions about how to get an usable certificate through Let's Encrypt.

4.1 Self Signed Certificate

Running an SSL server requires creating a certificate, which then needs to be signed by a certificate authority. To start we'll use a self-signed certificate, 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`.

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, you wil have to use the following Python snippet when creating the SSL context for both your client and your server.

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

You should be able at this point to run your server and client against each other without any errors.

Questions

(Answer on Gradescope)

4.2 Let's Encrypt and Public Key Infrastructure

Let's obtain an actual certificate so our web server can serve HTTPS traffic for anyone's browsers.

Background

In the past, obtaining a certificate signed by a root authority was a manual task that cost money. However, starting around 2015 or so Let's Encrypt changed the game, by creating an user-friendly, automated process for distribution of free certificates. This, along with efforts by browsers and other tech companies to encourage the use of HTTPS, has increased HTTPS adoption greatly in the last 10 years. [3] In fact, many major websites now default to HTTPS.

AT a more technical level, Let's Encrypt gives out short-term certificates that verify ownership of a domain. Let's Encrypt makes this simple, users simply have to run a script on a host machine corresponding to the domain they want a certificate for. Some details (which may be needed to answer questions) can be found here: `https://letsencrypt.org/how-it-works/`

Obtaining a VM and Subdomain

As mentioned above, we need some a host machine, with a domain that points to the machine. For the VM, we will again make use of the Google Cloud credit that was provided for the course.

Specifcally, you will need to:

1. Create a virtual machine with a public IP address.

2. Ensure that this VM can receive HTTP and HTTPS traffic (ports 80 and 443), as well as traffic on any other ports you wish to use. You will need to edit the Google cloud security groups

3. Add an "A" record to a DNS server pointing your desired subdomain to your VM's IP.

---

[3]See https://www.eff.org/deeplinks/2023/08/celebrating-ten-years-encrypting-web-lets-encrypt

Note, items (1) and (2) should be covered by the scripts that were provided with Assignment 1.

For item (3), the course staff has registered a domain `nyu-networks.com`, and we will give you access to add records to the DNS server for this domain (also through Google Cloud). We recommend creating a subdomains of the form `<netid>.nyu-networks.com`.

Once you have this VM setup, you should be able to test it by providing the domain name to "ssh" or "ping" instead of its IP address.

Obtaining a Certificate with Let's Encrypt

Once you have your VM configured with a domain name, we can obtain a certificate from Let's Encrypt. To do this we use Let's Encrypt's certbot tool (the aforementioned user-friendly, automated way to get a free certificates). Follow the instructions here `https://certbot.eff.org/instructions?ws=other&os=ubuntufocal` to install certbot. In step 6, you should run

```
sudo certbot certonly --standalone
```

and follow the provided instructions. The most likely reasons for this to fail are if your VM is unable to receive traffic on port 80 (due to firewall rules) or if your DNS is setup incorrectly.

The output of certbot should tell you where the certificate and corresponding key file are located. Run them with your web server as follows:

```
python3 ssl_web_server.py --address 0.0.0.0 --port PORT --ssl
        --cert-file <path-to-cert> --key-file <path-to-key>
```

Your SSL client should now be able to successfully request your SSL server using **the domain name**. Note you can remove the code allowing self-signed certificates to work. Your browser should also not give a warning when accessing your server by the **domain name** anymore.

Questions

(Refer to Gradescope)

5   Inspecting SSL Traffic

(Answer on Gradescope)

Once you have the SSL client and server working with each other run the following command to observe the packets on port 8000 (or other port of your choosing) over the loopback interface (i.e. localhost traffic).

```
sudo tcpdump -i lo -A port 8000
```

Then run your client and server over localhost on port 8000 **without ssl enabled**. How many packets are exchanged?

Then repeat the experiment **with ssl enabled**. How many packets are exchanged now? What is the reason for the change in number of packets, and do you see any other differences?