CSU34031 Project 1

Student name: Davy Nolan
Student number: 17330208

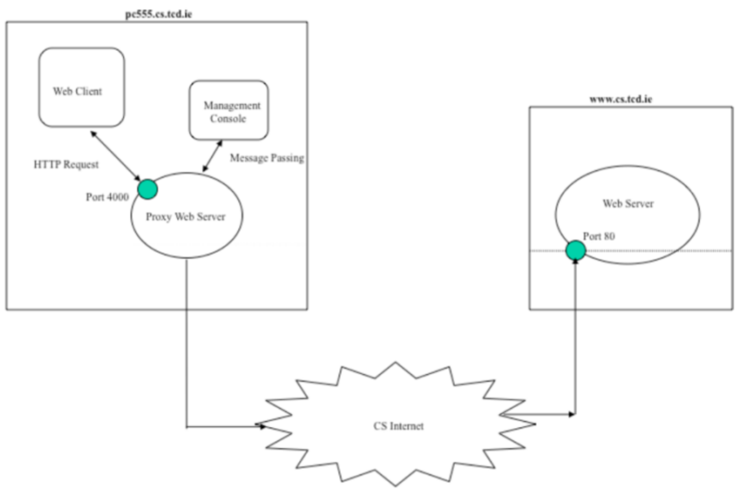
Introduction

The objective of this project is to implement a web proxy server using any language of preference. I have decided to use python for my implementation.

Requirements

The program should be able to:

1. Respond to HTTP and HTTPS requests, and should display each request on a management console. It should forward the request to the Web server and relay the response to the browser.
2. Handle Websocket connections.
3. Dynamically block selected URLs via the management console.
4. Efficiently cache requests locally and thus save bandwidth. You must gather timing and bandwidth data to prove the efficiency of your proxy.
5. Handle multiple requests simultaneously by implementing a threaded server.



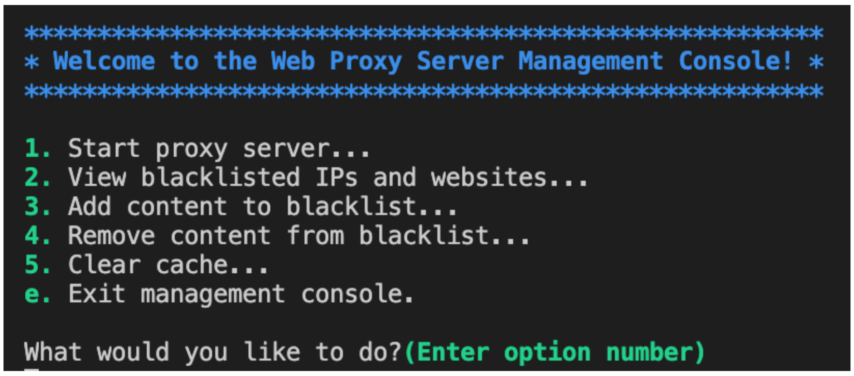
Implementation

I decided to split this project into two separate python programs that work together as one:

1. **Web_proxy_server.py** – This holds all the code involved in processing the HTTP/HTTPS requests, handling cache and also checking to see if a website or IP is blacklisted(blocked).
2. **Mng_console.py –** This program displays a UI in the console for handling processes in web_proxy_server.py in a more user-friendly way.

Code Explanation

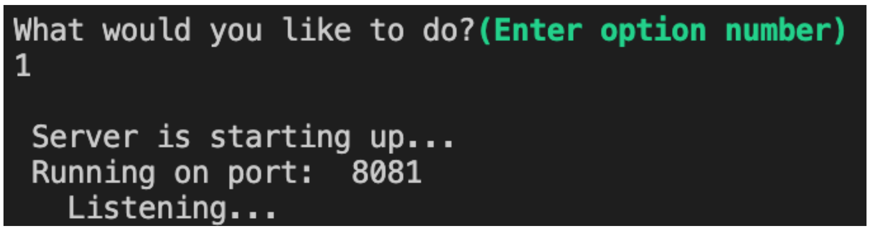
Running “python3 mng_console.py” will open up the UI for the program. You will be met by 6 options: Entering in any option from the list will perform the corresponding function explained.



```
*******************************************************
* Welcome to the Web Proxy Server Management Console! *
*******************************************************

1. Start proxy server...
2. View blacklisted IPs and websites...
3. Add content to blacklist...
4. Remove content from blacklist...
5. Clear cache...
e. Exit management console.

What would you like to do?(Enter option number)
```

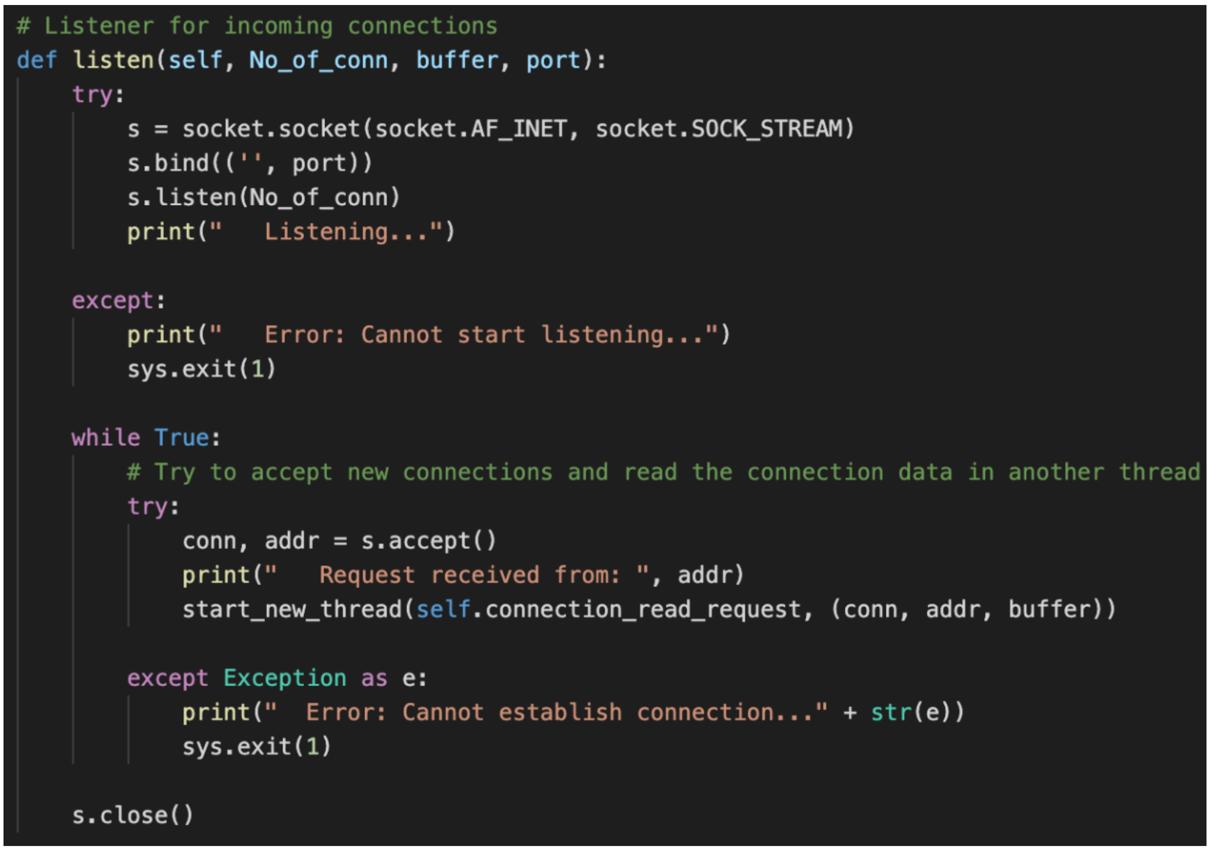
Running the Server

Selecting “option 1” from the management console will begin running the server. The port number is hardcoded into the start_server() function in web_proxy_server.py.



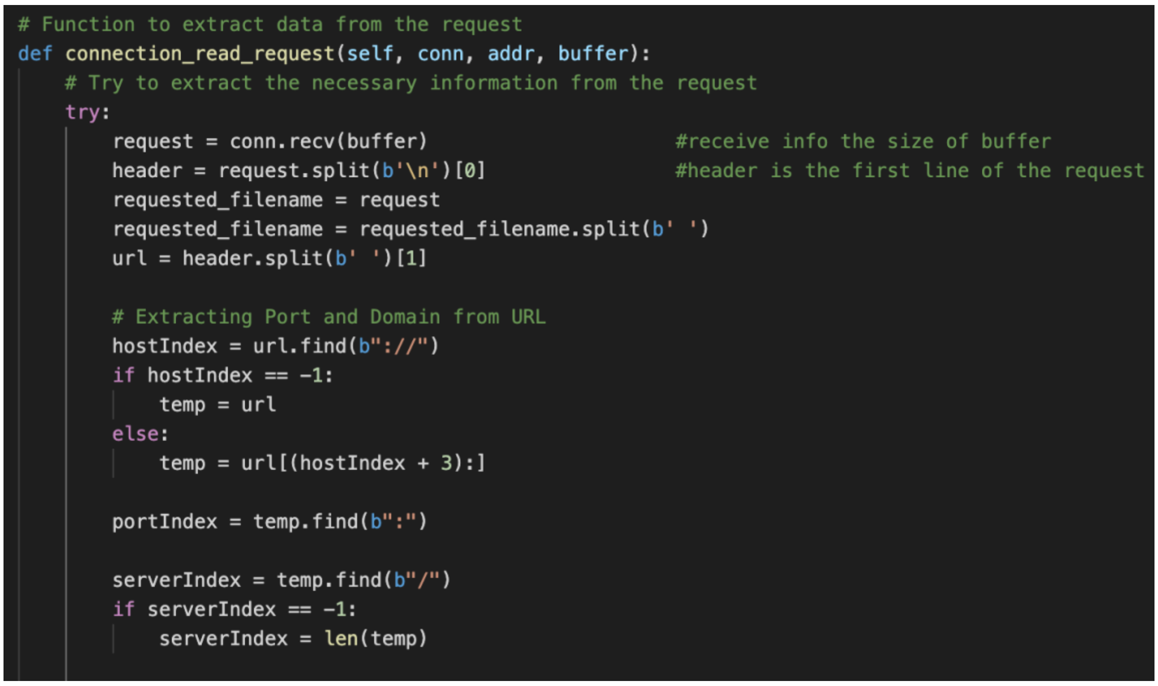
```
What would you like to do?(Enter option number)
1

 Server is starting up...
 Running on port:  8081
   Listening...
```

Once running, the server will call the listen() function and begin listening for the client (the websocket). Once a connection is received, it creates a new thread, calling the connection_read_request() function.



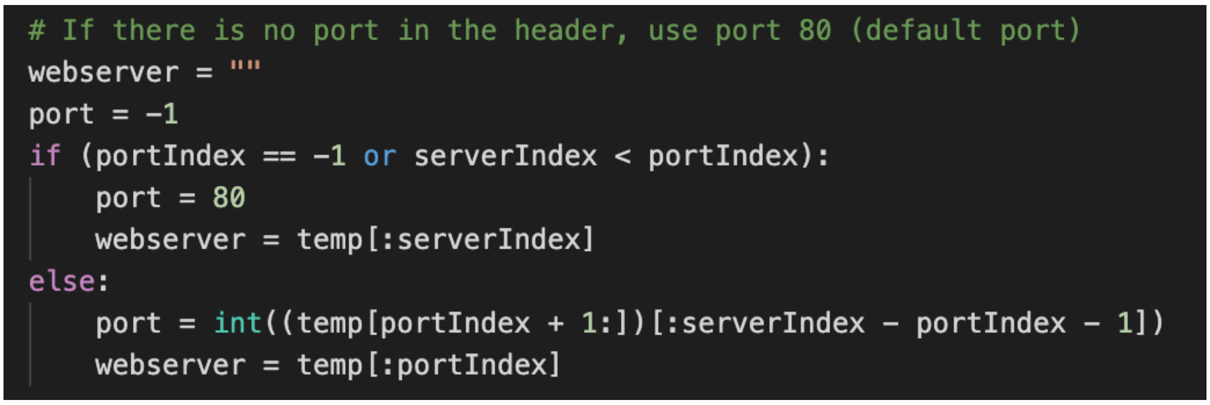
```python
# Listener for incoming connections
def listen(self, No_of_conn, buffer, port):
    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.bind(('', port))
        s.listen(No_of_conn)
        print("   Listening...")

    except:
        print("   Error: Cannot start listening...")
        sys.exit(1)

    while True:
        # Try to accept new connections and read the connection data in another thread
        try:
            conn, addr = s.accept()
            print("   Request received from: ", addr)
            start_new_thread(self.connection_read_request, (conn, addr, buffer))

        except Exception as e:
            print("  Error: Cannot establish connection..." + str(e))
            sys.exit(1)

    s.close()
```

The connection_read_request() function extracts data from the request. In the screenshot below it is extracting both the port and domain from the URL.



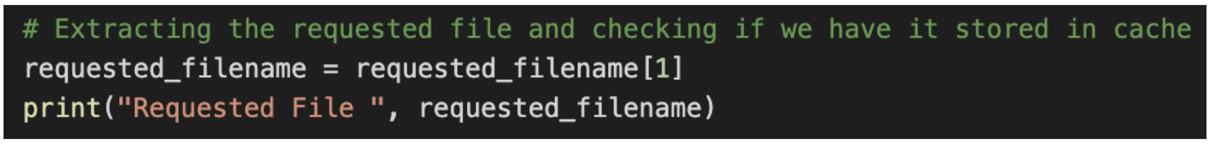
```python
# Function to extract data from the request
def connection_read_request(self, conn, addr, buffer):
    # Try to extract the necessary information from the request
    try:
        request = conn.recv(buffer)                         #receive info the size of buffer
        header = request.split(b'\n')[0]                    #header is the first line of the request
        requested_filename = request
        requested_filename = requested_filename.split(b' ')
        url = header.split(b' ')[1]

        # Extracting Port and Domain from URL
        hostIndex = url.find(b"://")
        if hostIndex == -1:
            temp = url
        else:
            temp = url[(hostIndex + 3):]

        portIndex = temp.find(b":")

        serverIndex = temp.find(b"/")
        if serverIndex == -1:
            serverIndex = len(temp)
```

If there’s no port to extract from the header, it uses the default port 80.



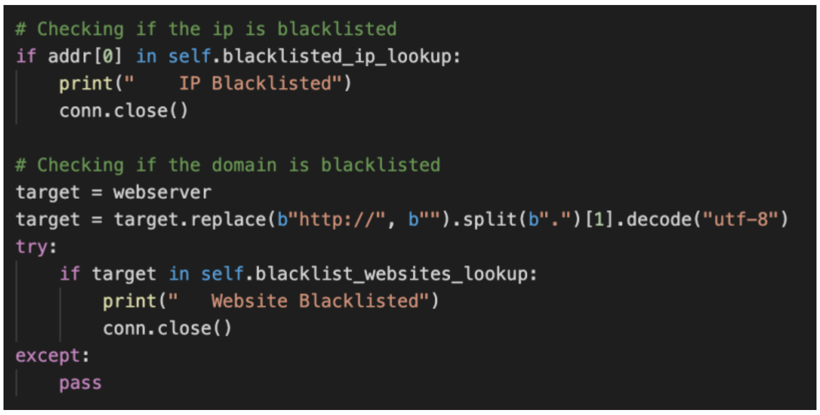
```
# If there is no port in the header, use port 80 (default port)
webserver = ""
port = -1
if (portIndex == -1 or serverIndex < portIndex):
    port = 80
    webserver = temp[:serverIndex]
else:
    port = int((temp[portIndex + 1:])[:serverIndex - portIndex - 1])
    webserver = temp[:portIndex]
```

This function also extracts the requested file name to check if it’s stored in cache already.



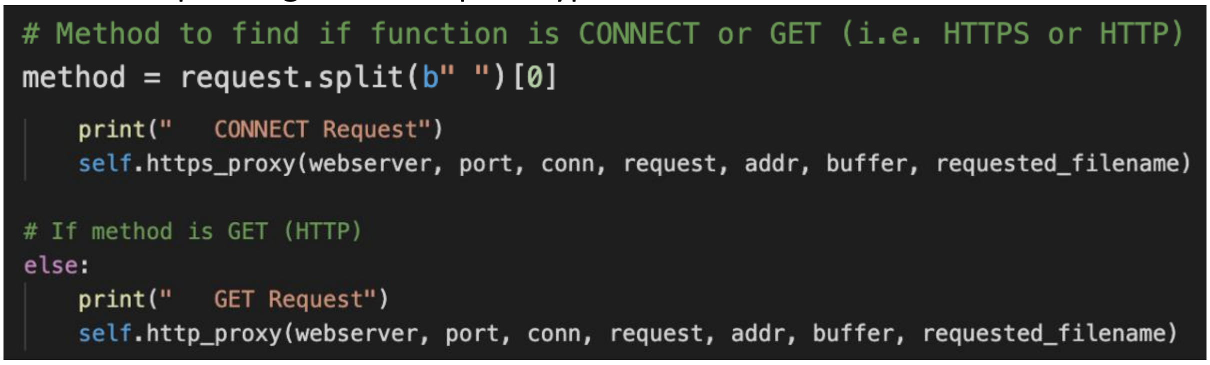
```
# Extracting the requested file and checking if we have it stored in cache
requested_filename = requested_filename[1]
print("Requested File ", requested_filename)
```

It also checks to see if the IP and/or the domain is blacklisted and closes the connection if so.



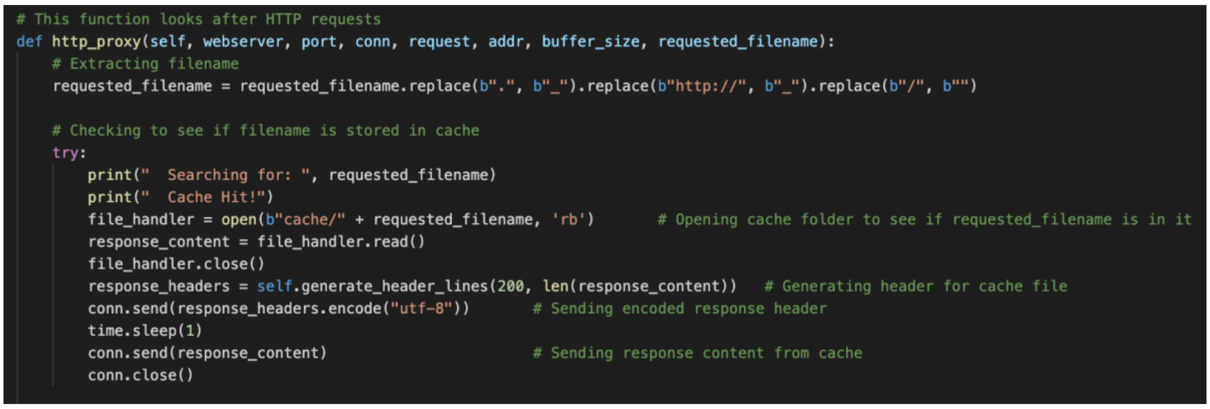
```
# Checking if the ip is blacklisted
if addr[0] in self.blacklisted_ip_lookup:
    print("     IP Blacklisted")
    conn.close()

# Checking if the domain is blacklisted
target = webserver
target = target.replace(b"http://", b"").split(b".")[1].decode("utf-8")
try:
    if target in self.blacklist_websites_lookup:
        print("    Website Blacklisted")
        conn.close()
except:
    pass
```

Lastly, the function checks to see if the request is either a CONNECT request or a GET request. CONNECT implies that it is HTTPS and GET implies that it's HTTP. Once the program checks this, it calls either the https_proxy() or http_proxy() function depending on the request type.



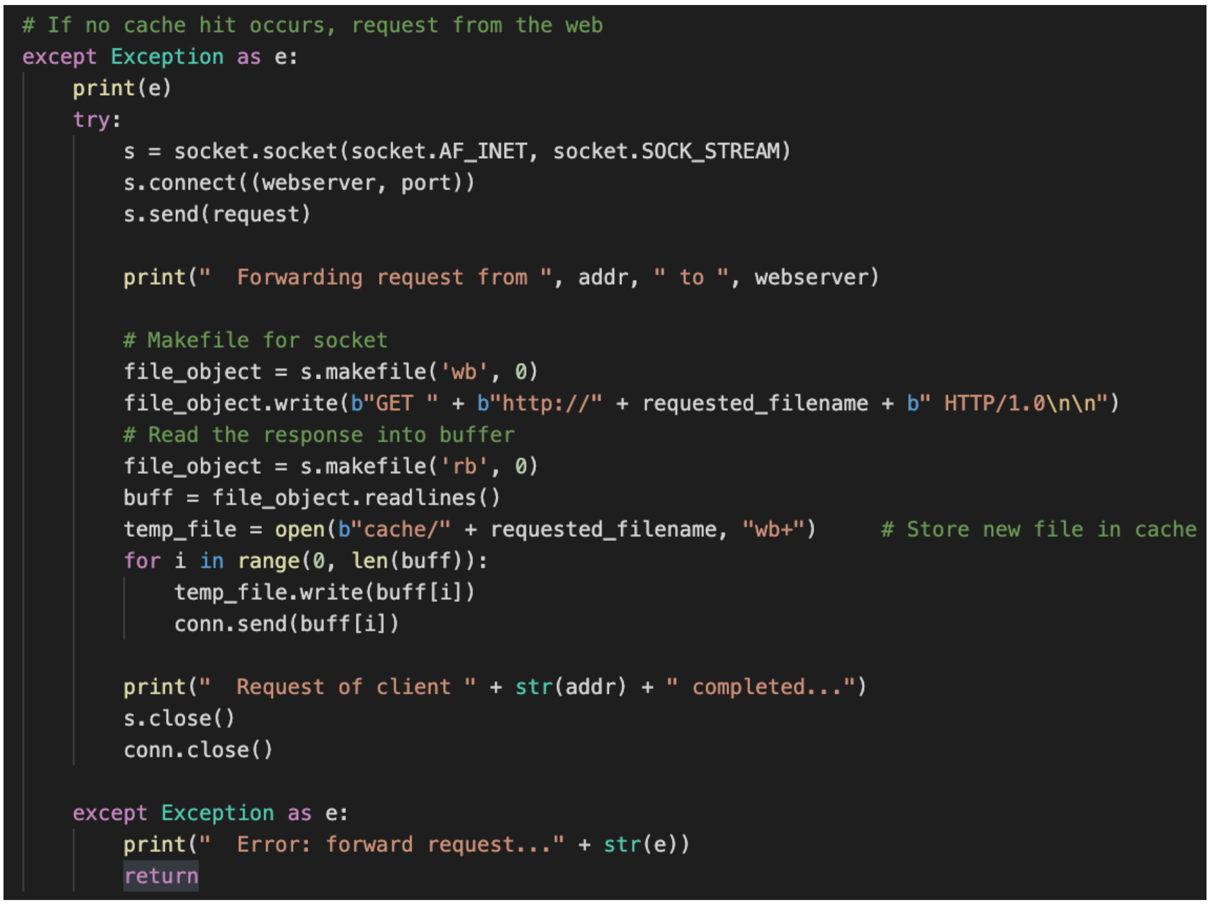
```python
# Method to find if function is CONNECT or GET (i.e. HTTPS or HTTP)
method = request.split(b" ")[0]

    print("    CONNECT Request")
    self.https_proxy(webserver, port, conn, request, addr, buffer, requested_filename)

# If method is GET (HTTP)
else:
    print("    GET Request")
    self.http_proxy(webserver, port, conn, request, addr, buffer, requested_filename)
```

The screenshot below shows how the http_proxy() function works in handling HTTP requests. It checks if the requested file is stored in cache.



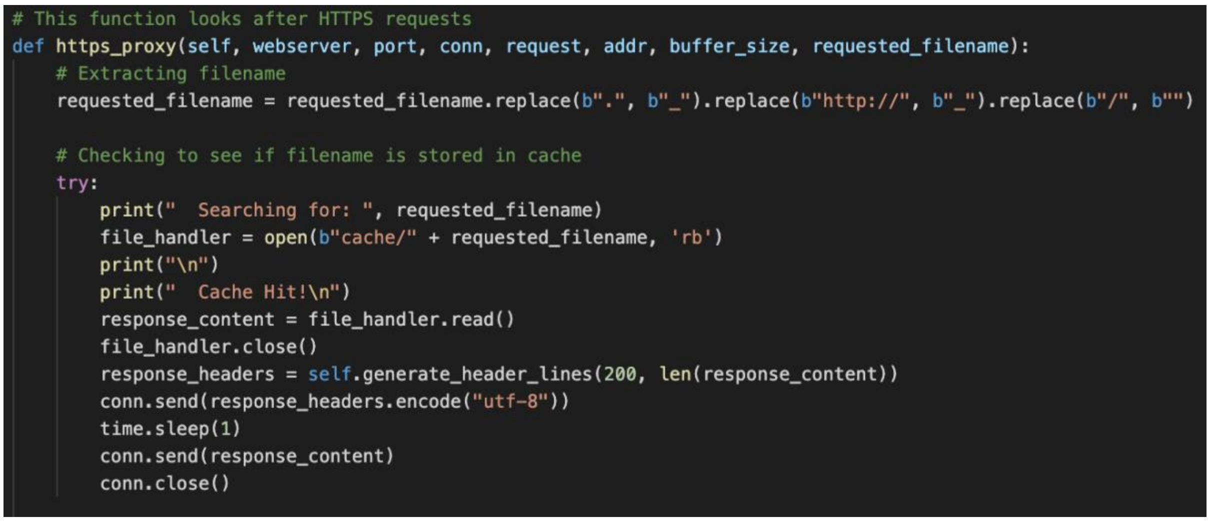
```python
# This function looks after HTTP requests
def http_proxy(self, webserver, port, conn, request, addr, buffer_size, requested_filename):
    # Extracting filename
    requested_filename = requested_filename.replace(b".", b"_").replace(b"http://", b"_").replace(b"/", b"")

    # Checking to see if filename is stored in cache
    try:
        print("  Searching for: ", requested_filename)
        print("  Cache Hit!")
        file_handler = open(b"cache/" + requested_filename, 'rb')       # Opening cache folder to see if requested_filename is in it
        response_content = file_handler.read()
        file_handler.close()
        response_headers = self.generate_header_lines(200, len(response_content))   # Generating header for cache file
        conn.send(response_headers.encode("utf-8"))       # Sending encoded response header
        time.sleep(1)
        conn.send(response_content)                       # Sending response content from cache
        conn.close()
```

If the file is not stored in cache (no cache hit) the server requests it from the web and stores it in cache.



```
# If no cache hit occurs, request from the web
except Exception as e:
    print(e)
    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.connect((webserver, port))
        s.send(request)

        print("  Forwarding request from ", addr, " to ", webserver)

        # Makefile for socket
        file_object = s.makefile('wb', 0)
        file_object.write(b"GET " + b"http://" + requested_filename + b" HTTP/1.0\n\n")
        # Read the response into buffer
        file_object = s.makefile('rb', 0)
        buff = file_object.readlines()
        temp_file = open(b"cache/" + requested_filename, "wb+")     # Store new file in cache
        for i in range(0, len(buff)):
            temp_file.write(buff[i])
            conn.send(buff[i])

        print("  Request of client " + str(addr) + " completed...")
        s.close()
        conn.close()

    except Exception as e:
        print("  Error: forward request..." + str(e))
        return
```

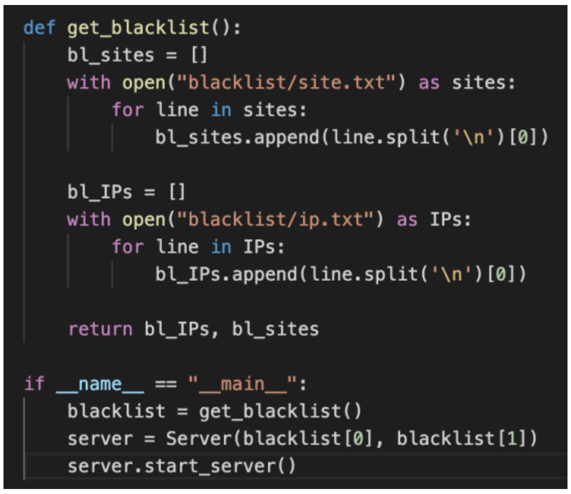
Similarly, to the above implementation, HTTPS requests are handled more or less the same way.



```
# This function looks after HTTPS requests
def https_proxy(self, webserver, port, conn, request, addr, buffer_size, requested_filename):
    # Extracting filename
    requested_filename = requested_filename.replace(b".", b"_").replace(b"http://", b"_").replace(b"/", b"")

    # Checking to see if filename is stored in cache
    try:
        print("  Searching for: ", requested_filename)
        file_handler = open(b"cache/" + requested_filename, 'rb')
        print("\n")
        print("  Cache Hit!\n")
        response_content = file_handler.read()
        file_handler.close()
        response_headers = self.generate_header_lines(200, len(response_content))
        conn.send(response_headers.encode("utf-8"))
        time.sleep(1)
        conn.send(response_content)
        conn.close()
```

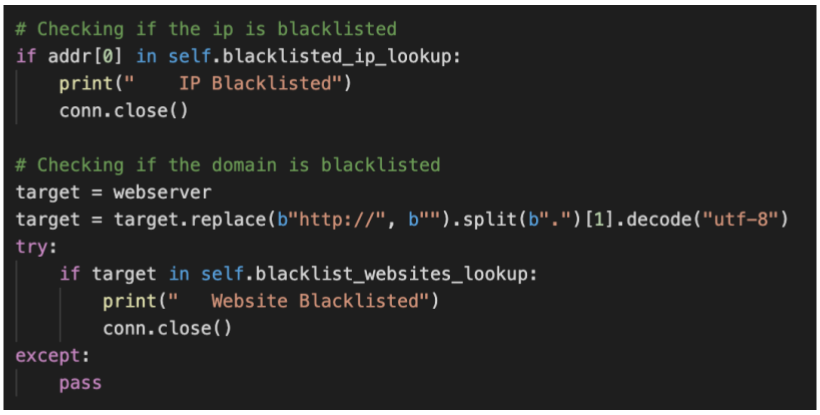
Blacklisting (blocking)

For the blacklisted IPs and websites, there are two separate text files for storing this information. Blacklisted IPs can be listed in “blacklist/ip.txt” and websites can be listed in “blacklist/site.txt”. These text files are read into the server program and are added to lists within the server.



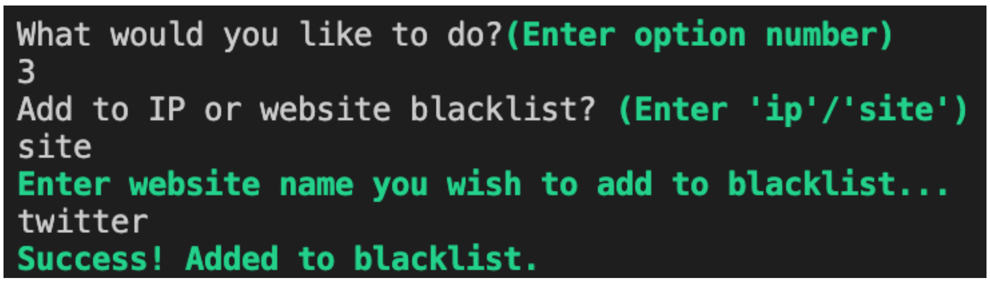
```python
def get_blacklist():
    bl_sites = []
    with open("blacklist/site.txt") as sites:
        for line in sites:
            bl_sites.append(line.split('\n')[0])

    bl_IPs = []
    with open("blacklist/ip.txt") as IPs:
        for line in IPs:
            bl_IPs.append(line.split('\n')[0])

    return bl_IPs, bl_sites

if __name__ == "__main__":
    blacklist = get_blacklist()
    server = Server(blacklist[0], blacklist[1])
    server.start_server()
```

The server will then refer to these lists to check to see if the IP or website being searched is blocked and will act accordingly. As seen in the screenshot below, the program strips the webserver URL to just be left with the name of the website ((i.e.) https://www.facebook.com becomes “facebook”). This makes it possible to blacklist HTTPS sites. It then searches for this website name in the website blacklist, and if it is in the blacklist, it closes the connection and prints to the console.

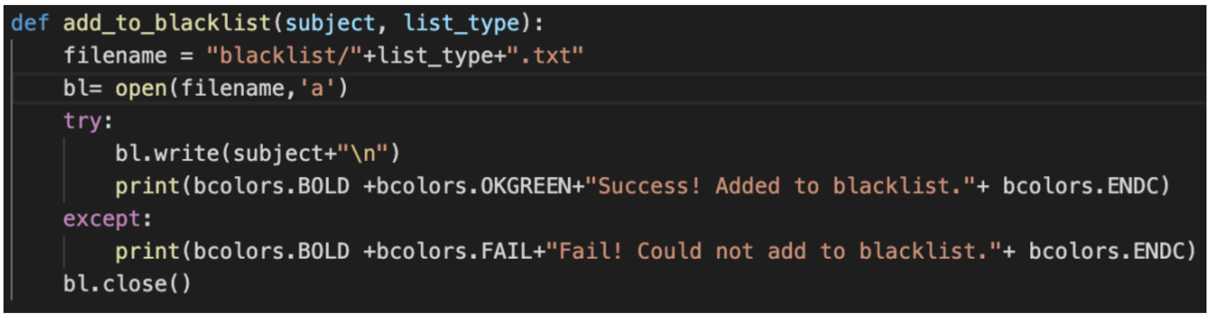


```python
# Checking if the ip is blacklisted
if addr[0] in self.blacklisted_ip_lookup:
    print("    IP Blacklisted")
    conn.close()

# Checking if the domain is blacklisted
target = webserver
target = target.replace(b"http://", b"").split(b".")[1].decode("utf-8")
try:
    if target in self.blacklist_websites_lookup:
        print("   Website Blacklisted")
        conn.close()
except:
    pass
```

To add new content to the blacklists, you can easily do this via the management console. Selecting “option 3” will prompt you to enter what list you want to add to (either IP or website name) and then ask you to enter either the IP address or website you wish to block. (The example below demonstrates adding a website only but it is the same idea for IP, you just enter ‘ip’ instead of ‘site’)

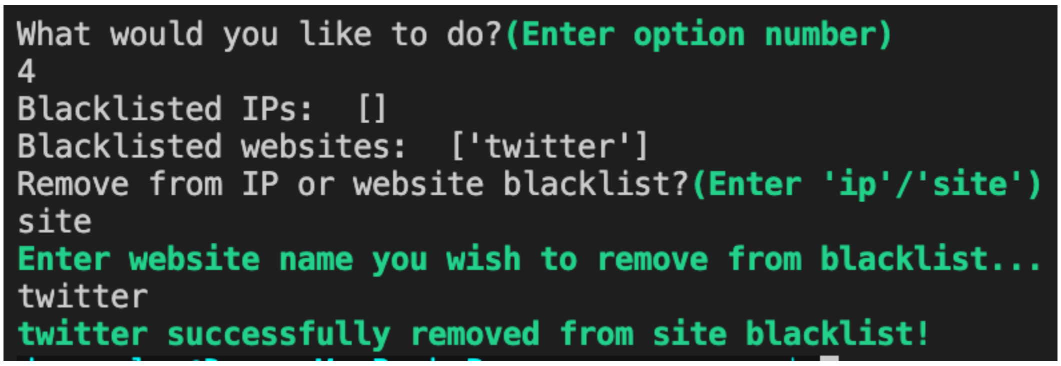


```
What would you like to do?(Enter option number)
3
Add to IP or website blacklist? (Enter 'ip'/'site')
site
Enter website name you wish to add to blacklist...
twitter
Success! Added to blacklist.
```

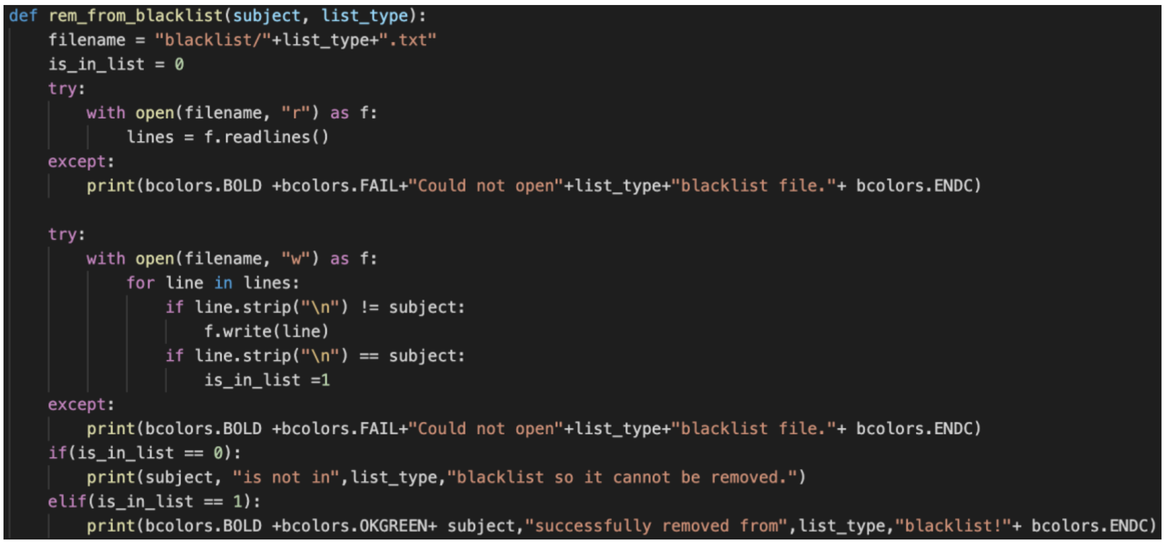


```python
def add_to_blacklist(subject, list_type):
    filename = "blacklist/"+list_type+".txt"
    bl= open(filename,'a')
    try:
        bl.write(subject+"\n")
        print(bcolors.BOLD +bcolors.OKGREEN+"Success! Added to blacklist."+ bcolors.ENDC)
    except:
        print(bcolors.BOLD +bcolors.FAIL+"Fail! Could not add to blacklist."+ bcolors.ENDC)
    bl.close()
```

Content can also be removed from the blacklists using the management console by selecting “option 4”. This will display both the IP and website blacklists with their current content. You just have to enter what you want to remove.



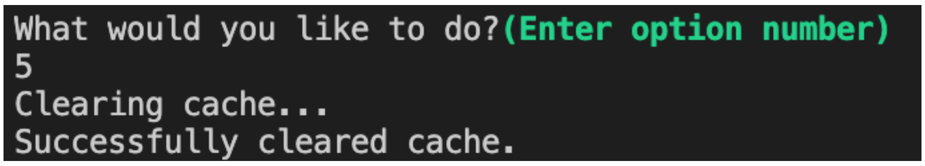
```
What would you like to do?(Enter option number)
4
Blacklisted IPs:  []
Blacklisted websites:  ['twitter']
Remove from IP or website blacklist?(Enter 'ip'/'site')
site
Enter website name you wish to remove from blacklist...
twitter
twitter successfully removed from site blacklist!
```



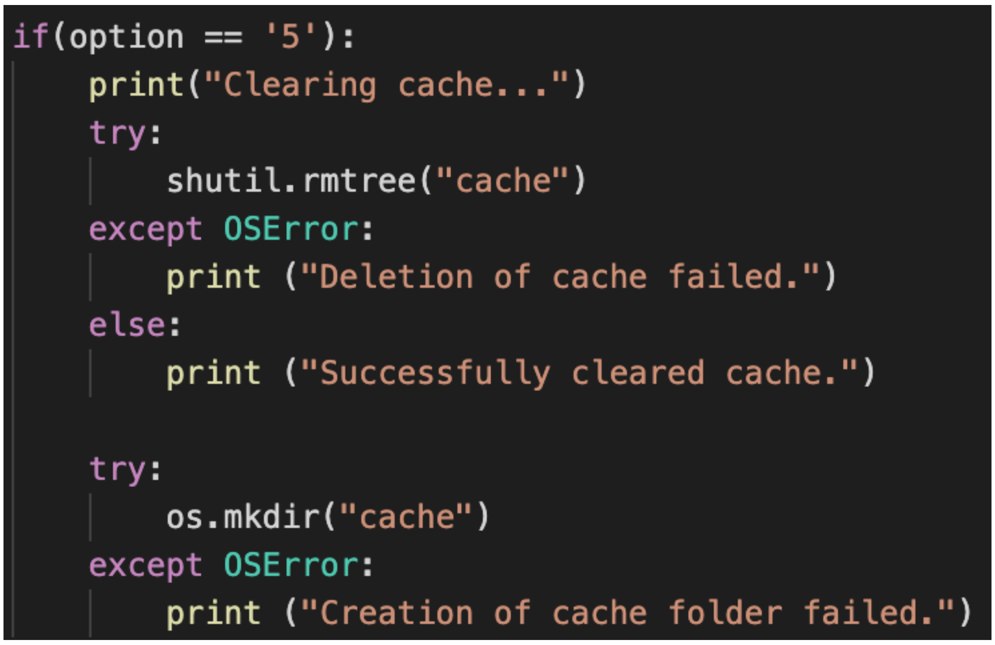
```python
def rem_from_blacklist(subject, list_type):
    filename = "blacklist/"+list_type+".txt"
    is_in_list = 0
    try:
        with open(filename, "r") as f:
            lines = f.readlines()
    except:
        print(bcolors.BOLD +bcolors.FAIL+"Could not open"+list_type+"blacklist file."+ bcolors.ENDC)

    try:
        with open(filename, "w") as f:
            for line in lines:
                if line.strip("\n") != subject:
                    f.write(line)
                if line.strip("\n") == subject:
                    is_in_list =1
    except:
        print(bcolors.BOLD +bcolors.FAIL+"Could not open"+list_type+"blacklist file."+ bcolors.ENDC)
    if(is_in_list == 0):
        print(subject, "is not in",list_type,"blacklist so it cannot be removed.")
    elif(is_in_list == 1):
        print(bcolors.BOLD +bcolors.OKGREEN+ subject,"successfully removed from",list_type,"blacklist!"+ bcolors.ENDC)
```

Clear Cache

Cache can be freed up by selecting “option 5” in the management console. This will practically delete the cache directory and create a new one.



```
What would you like to do?(Enter option number)
5
Clearing cache...
Successfully cleared cache.
```



```python
if(option == '5'):
    print("Clearing cache...")
    try:
        shutil.rmtree("cache")
    except OSError:
        print ("Deletion of cache failed.")
    else:
        print ("Successfully cleared cache.")

    try:
        os.mkdir("cache")
    except OSError:
        print ("Creation of cache folder failed.")
```