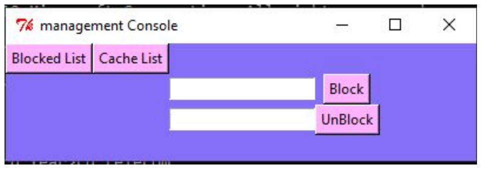
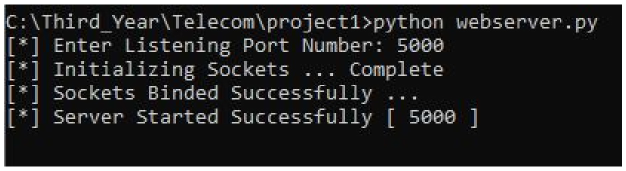
PROJECT 1

Holly Baker
17329137

PROJECT IMPLEMENTATION

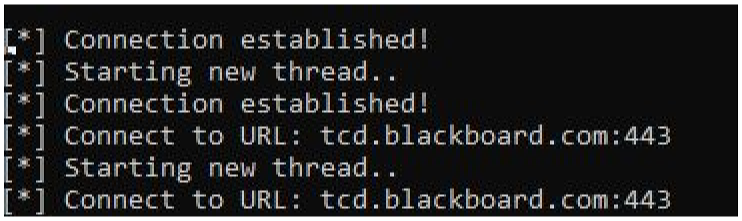
The tkinter library was imported to implement the management console. This console enables blocking and unblocking url's both http and https. It also contains buttons to display the blocked list and cached list of urls. Selected URLs can be dynamically blocked via this management console.





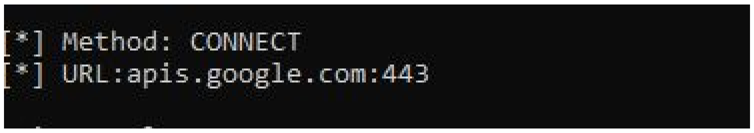
```
C:\Third_Year\Telecom\project1>python webserver.py
[*] Enter Listening Port Number: 5000
[*] Initializing Sockets ... Complete
[*] Sockets Binded Successfully ...
[*] Server Started Successfully [ 5000 ]
```

User input for the listening port number is necessary. The socket is initialized and we begin listening for connections. Multiple connections/requests are handled via threading. Each request is parsed relying on whether the connection is http(GET)or https(CONNECT). For each url running through the server, we check if this url is blocked via iterating through the dictionary, then close this connection if there is a match.



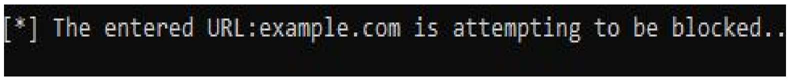
```
[*] Connection established!
[*] Starting new thread..
[*] Connection established!
[*] Connect to URL: tcd.blackboard.com:443
[*] Starting new thread..
[*] Connect to URL: tcd.blackboard.com:443
```

The proxy web server handles HTTP & HTTPS request, displaying the url on console and whether it is a CONNECT or GET method depending on the HTTP/ HTTPS.

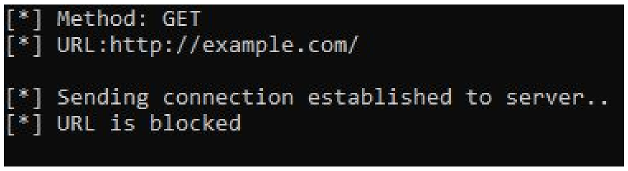


```
[*] Method: CONNECT
[*] URL:apis.google.com:443
```

The url attempts to be blocked and is successfully blocked if it is not already blocked.



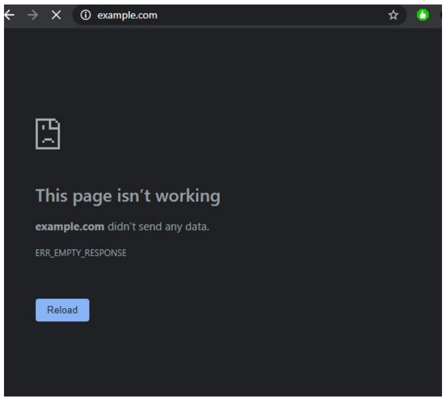
```
[*] The entered URL:example.com is attempting to be blocked..
```



```
[*] Method: GET
[*] URL:http://example.com/

[*] Sending connection established to server..
[*] URL is blocked
```

The url is inaccessible when blocked.

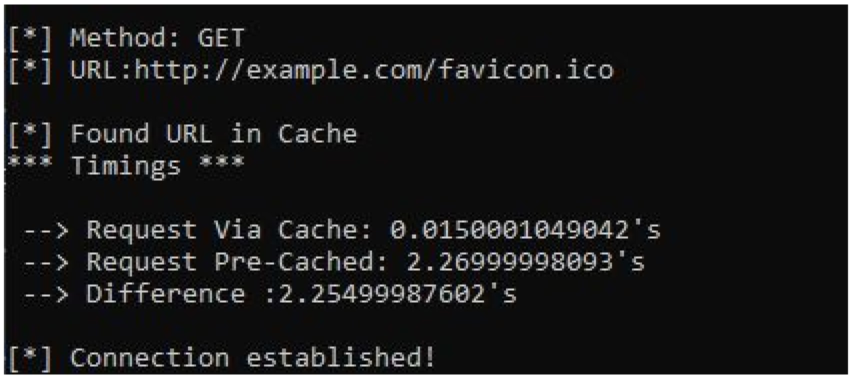

example.com

This page isn't working

example.com didn't send any data.

ERR_EMPTY_RESPONSE

The timings are recorded via a dictionary and so are the cached connections. The time it takes to access the url when it is not cached is recorded, and the time it takes to access the url when it is cached is recorded. The difference is displayed on the console.



```
[*] Method: GET
[*] URL:http://example.com/favicon.ico

[*] Found URL in Cache
*** Timings ***

 --> Request Via Cache: 0.0150001049042's
 --> Request Pre-Cached: 2.26999998093's
 --> Difference :2.25499987602's

[*] Connection established!
```

```
import os, time, socket, sys, thread
import ssl
import Tkinter as tk
from Tkinter import*

MAX_connection  = 250 # Limit no. of connections to server Max = 250
BUFFER_SIZE = 8192 # Max Socket Buffer Size
DEBUG = True

cache ={} # Dictionary - collection of cached urls
blocked = {} # Dictionary - collection of blocked urls
timings = {} # Dictionary - collection of cached urls, used for comparing
timings

# Blocks dynamically selected URls + Displays current blocked URLs + cached
URLs
def tkinter():

    console = tk.Tk(className='Management Console')
    console.geometry("400x100") #set window size
    console['background']='#856ff8'  #set window color

    # Component - Entry Boxes
    block = Entry(console)
    block.grid(row=7,column=9)
    unblock = Entry(console)
    unblock.grid(row=8,column=9)

    # handle entered url to be potentially blocked
    def block_url():
        entered_url = block.get()
        temp = blocked.get(entered_url)
        print(temp)
        if temp is None: # block url
            blocked[entered_url] = 1
            print("[*] The entered URL:"+ entered_url +" is attempting to
be blocked..\n")
        else: # url already blocked
            print("[*] The entered URL is already blocked..\n")

    # Component - block button
    block_button = Button(console, text="Block",bg='#ffb3fe',
command=block_url)
    block_button.grid(row=7,column=10)

    # handle entered url to be potentially unblocked
    def unblock_url():
        entered_url = unblock.get()
        temp = blocked.get(entered_url)
        if temp is None: # url already unblocked
            print("[*] The entered URL" + entered_url +"is already
unblocked..\n")
        else: # url  unblocked
            blocked.pop(entered_url)
            print("[*] The entered URL:" + entered_url+ " is attempting to
be unblocked..\n")

    # Component - unblock button
    unblock_button = Button(console, text="UnBlock",bg='#ffb3fe',
command=unblock_url)
    unblock_button.grid(row=8,column=10)

    # Print list of current blocked URL
    def blocked_list():
        print(blocked)

    # Component - blocked lsist button
    blocked_list = Button(console, text="Blocked List",bg='#ffb3fe',
command=blocked_list)
    blocked_list.grid(row=2,column=3)

    # Print list of currently cached pages
    def cached_pages_list():
        for key, value in cache.iteritems():
            print key

    cache_list = Button(console, text="Cache List", bg='#ffb3fe',
command=cached_pages_list)
    cache_list.grid(row=2,column=6)

    mainloop()
```

```
def main():
    thread.start_new_thread(tkinter,()) # Tkinter Thread start running

    try:
        listening_port = int(raw_input("[*] Enter Listening Port Number:
")) # Asking user for a listening port using
    except KeyboardInterrupt:
        sys.exit()

    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # initiate
socket(sock_stream = tcp,af_inet = ipv4)
        s.setsockopt(socket.SOL_SOCKET,socket.SO_REUSEADDR,1)
        s.bind(('',listening_port)) # Bind socket for listen
        s.listen(MAX_connection) # Start listening for incomming
connectionections

        print ("[*] Initializing Sockets ... Complete")
        print ("[*] Sockets Binded Successfully ...")
        print ("[*] Server Started Successfully [ %d ]\n" %
(listening_port))

    except Exception, e:
        print "[*] Unable To Initialize Socket" # Catch Errors
        sys.exit(2)

    while 1: # Listening for conections
        try:
            connection, address = s.accept() # Accept connection from
client browser
            print("[*] Connection established!")
            data = connection.recv(BUFFER_SIZE) # Recieve client data
            thread.start_new_thread( handle_browser_connection,
(connection, data, address)) # Start a new connection thread
        except KeyboardInterrupt: # Handles keyboard interrupts if the user
wants to exit the script.
            s.close()
            print ("[*] Proxy Server Shutting Down ...")
            sys.exit(1)
    s.close()
# Port in recieved client data, call function proxy_server + pass 5
```

```
agruments to it.
def handle_browser_connection(connection, data, address):
    #Client Browser Request Appears Here
    print("[*] Starting new thread..")

    try:
        data = data.decode("utf-8")

        first_line = data.split('\n')[0]
        url = first_line.split(' ')[1]
        webserver = ""
        port = ""
        method = first_line.split(" ")[0]

        print("[*] Connect to URL: " + url +"\n")
        print ("[*] Method: " + method)
        if(DEBUG):
            print("[*] URL:" + url  +"\n")

        if method == "CONNECT": # set webserver for HTTPS
            webserver = url.split(':')[0]
            print (webserver)
            port = url.split(':')[1]
        else: # set webserverfor HTTP
            http_position = url.find("://") # Find ://
            if(http_position == -1):
                temp = url
            else:
                temp = url[(http_position+3):] # Rest of URL

            port_position = temp.find(":") # Find position of the port, if
specified, if not will take default number

            webserver_position = temp.find("/") # Find the end of the web
server
            if webserver_position == -1:
                webserver_position = len(temp)

            port = -1

            if(port_position == -1 or webserver_position < port_position):
                # Port was not specified, therefore default port is 80
                port = 80
                webserver = temp[:webserver_position]

            else:
                # Port was specified
                port =
int((temp[(port_position+1):])[:webserver_position-port_position-1])
                webserver = temp[:port_position]


        for key, value in blocked.iteritems():
            if key in webserver and value is 1:
                print("[*] URL is blocked\n")
                connection.close()
                return

        # Cache Section
        start_time = time.time()
        get_url = cache.get(webserver)
        if get_url is not None:
            print("[*] Found URL in Cache")
            connection.sendall(get_url)
            end_time =time.time()
            print("*** Timings *** \r\n")
            print(" --> Request Via Cache: " +
str(end_time-start_time)+"'s")
            print(" --> Request Pre-Cached: " +
str(timings[webserver])+"'s")
            print(" --> Difference :" +
str(timings[webserver]-(end_time-start_time))+"'s\n")
        else:
            proxy_server(webserver, port, connection, address, data,
method)
    except socket.error as err:
        print(err)
        return

#Creates new socket, connectionects to webserver and sends the clients
request, recieves the reply, forwards it back to the client without
modifying it.
```

```
def proxy_server (webserver, port, connection, client_address, data,
method) :

    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)


    if method == "CONNECT":
        try:
            s.connect((webserver, int(port)))
            reply = "HTTP/1.0 200 connection established \r\n"
            reply += "Proxy-agent: Pyx\r\n"
            reply += "\r\n"
            print("[*] Sending connection established to server..\n")
            connection.sendall(reply.encode())
        except socket.error as err:
            print(err)
            return
        connection.setblocking(0)
        s.setblocking(0)
        print ("[*] Websocket connection set up...\n")
        #.send(data)
        while True: # loop to keep connection alive instead of dropping
            # Read reply
            try:
                request = connection.recv(BUFFER_SIZE)
                s.sendall(request)
            except socket.error as err:
                pass
            try:
                reply = s.recv(BUFFER_SIZE)
                connection.sendall(reply)
            except socket.error as err:
                pass
        print("[*] Sending resposnes to client..")
    else:
        start_time =time.time()
        string_builder = bytearray("",'utf-8')
        s.connect((webserver,int(port)))
        print("[*] Sending request to server..")
        s.send(data)
        s.settimeout(2)
        try:
            while True:
                reply= s.recv(BUFFER_SIZE)
                if(len(reply) >0):
                    connection.send(reply)
                    string_builder.extend(reply)
                else:
                    break
        except socket.error:
            pass
        print("[*] Sending response to client..")
        end_time = time.time()
        print("[*] Request took:" + str(end_time-start_time)+"s")
        timings[webserver] = end_time-start_time
        cache[webserver] = string_builder

        s.close()
        connection.close()

if __name__ == '__main__':
      main()
```