CS3031 Advanced Telecommunications
Web Proxy Server

Anton Yamkovoy
17331565

February 24, 2020

0.1 Specification

The objective of the excercise is to implement a Web Proxy Server. A Web proxy is a local server, which fetches items from the Web on behalf of a Web client instead of the client fetching them directly. This allows for caching of pages and access control.

The program should be able to:

1. Respond to HTTP & 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.

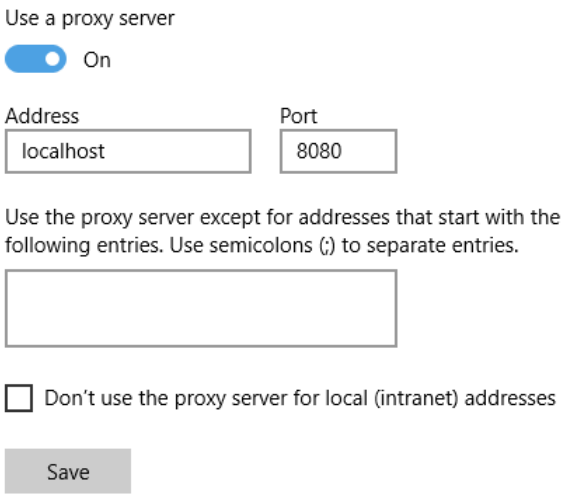
3. Dynamically block selected URLs via the management console.

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.

0.2 Implementation

To explain the implementation of the web proxy, I will go through a few typical threads of execution of the program, I have all of the features implemented that are required n the specification.
In any case when the program is run, a list of blocked sites is constructed from a file, these are stored in the list for later usage, and for interaction with the gui.
Before running the program you need to set up the proxy settings on your OS, as shown in the image below. I run the proxy on localhost, and port '8080', these settings are set at the start of the program.



When the required settings are set up, the program is run, firstly the socket is initialised on ipv4, and a tcp connection, the socket is bound to the port, specified at the start, and the socket starts to listen for connections from the browser on the given port, which is why the settings are necessary for operation, If any of these steps fail the error is handled and the program is terminated.

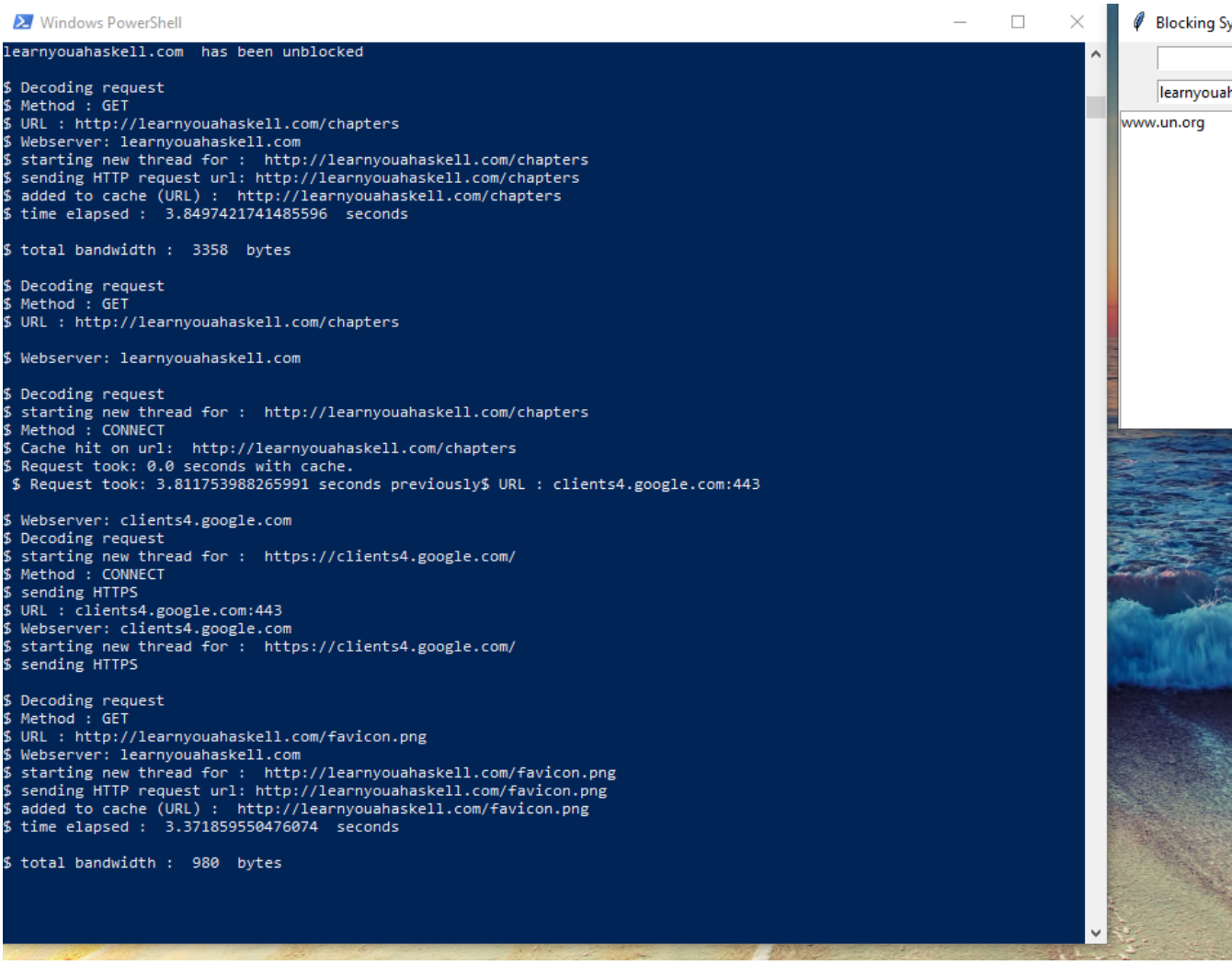
```
PS C:\Users\Anton> cd .\Documents\GitHub\advanced_telecomms\
PS C:\Users\Anton\Documents\GitHub\advanced_telecomms> python proxy.py
$ Initialising socket...
$ Bind successful
$ Server listening on port [ 8080 ]

Blocked Sites @ Init:

learnyouahaskell.com
www.un.org
```

After socket initiation, we begin to loop around trying to accept connections from the browser, when a connection is found, we extract data, calculate

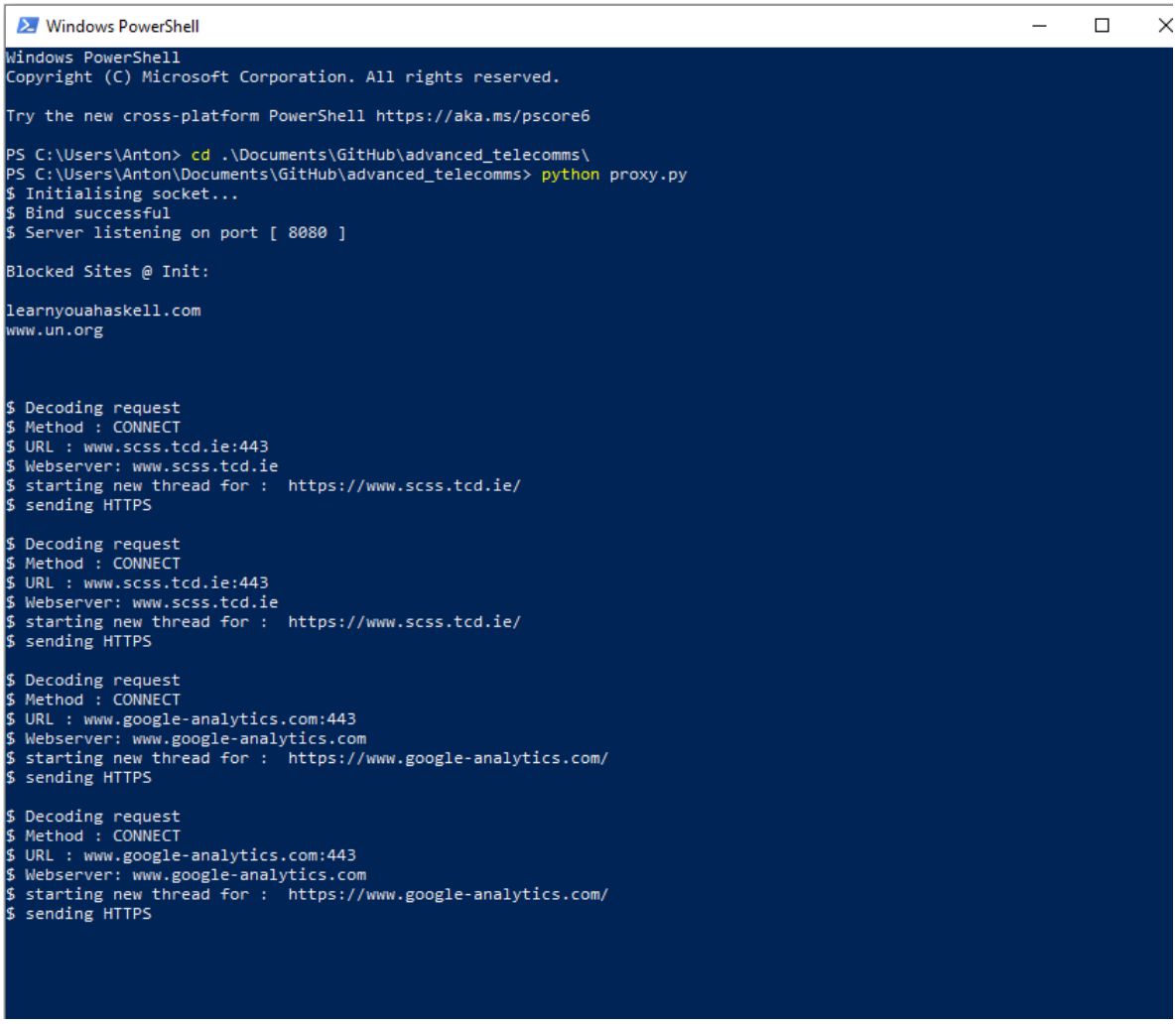
bandwidth, and start a initial timer, before entering a new thread which services this request, the thread starts a function proxy, that will decode, and enter a http or https handler based on the decoding and breakdown of the request.
When we enter, the proxy function, a few error checking cases are done, such as checking if the data is null, after this the data is passed into the parsing function, this returns a 5-tuple, a boolean to notify if it is a https request, webserver, port, url and method
After this the cache is checked, if the cache already contains a cached response based on this 'url' that was returned from the parsing function, the cached response is sent to the client, and some statistics are printed out to show the time savings and bandwidth usage previously to caching



If the request has not been previously cached, and it is a http request, it enters the http function, where a new socket is initiated, a timer is started to measure the time of the non cached request, using our client connection and the socket, we construct a http reply from the webserver, and send it out to the client, acting as a middleman receiving the data from the server and sending it on to the client.

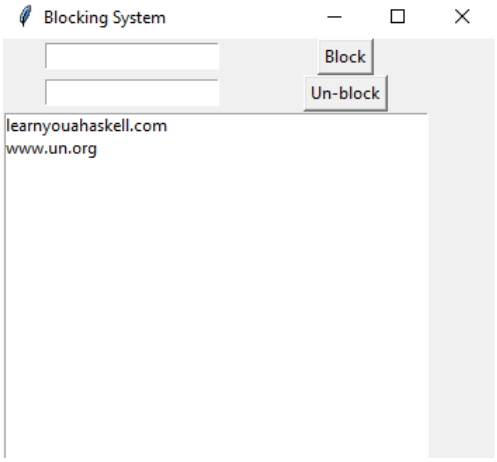
The different types of requests are identified using the methods in the headers of the request. In the case where the request is identified as a https

request after the decoding stage in the proxy function, it cannot be cached as it is https, so we jump straight to the https handler function, here we connect to the webserver, and establish a non blocking bidirectional socket connection, following this we receive the request from the client, pass it onto the server, and reply back to the client with the server response.

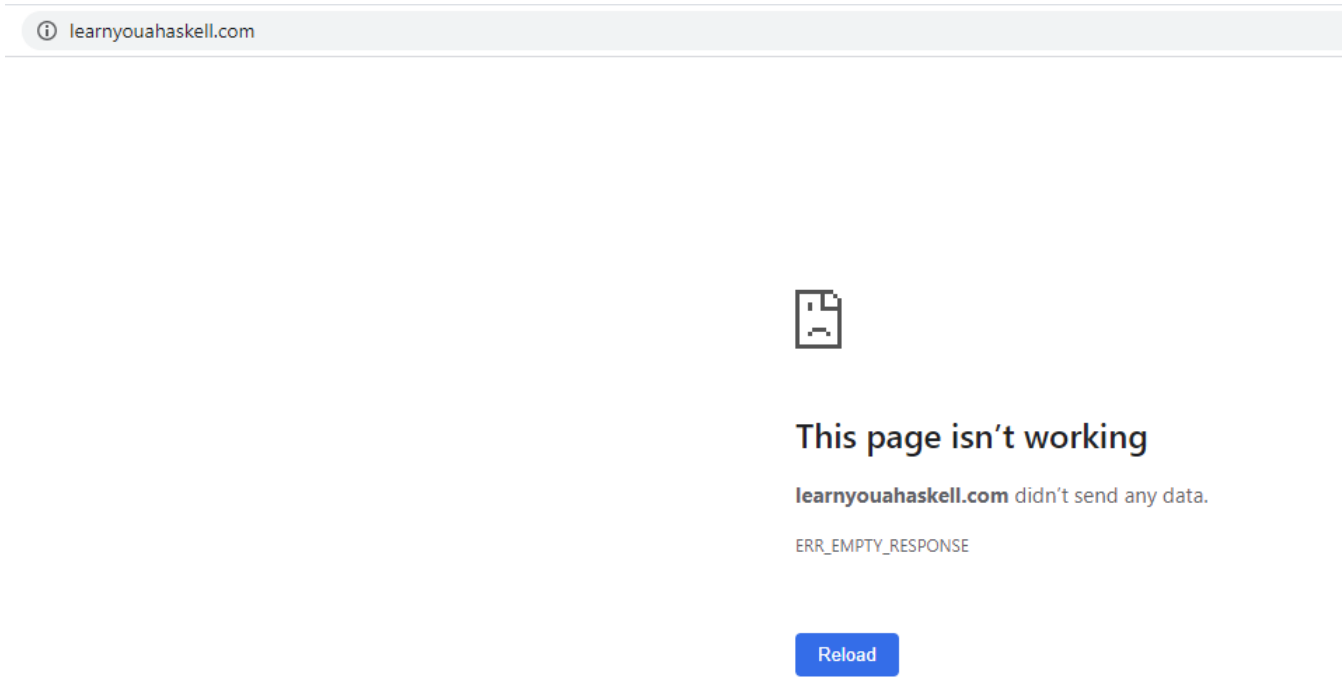


Another feature that I have implemented is a interface for blocking sites based on the webserver, a user can enter input to block or unblock certain sites,

from the being accessed, this is done dynamically using a separate thread launched at the start of the program that doesn't interfere with the rest of the program.
The initially blocked sites are found from a file at the start of the program, and printed in the console at launch, but they can be removed easily using the gui, which was implemented using a python gui library 'tkinter'



When a page is blocked the user will see this message if trying to access the blocked site.



0.3 Code

```
1  import socket
2  import tkinter as tk
3  from tkinter import *
4  import sys,_thread
5  import time
6  import ssl
7
8  #port number & buffer size
9  #CONSTANTS
10 global list
11 MAX_DATA = 4096 # 4k default buffer
12 DEFAULT_PORT = 8080
13 BACKLOG = 1000
```

```
14 blocked_sites = [line.rstrip('\n') for line in
       open('blocked.txt')] # loading
15
16 #blocked_sites.append("scratchpads.eu") ## http blocking
17 #blocked_sites.append("www.un.org") ## https example blocking
18
19 cache = {} # dict as a cache for http responces
20 timeData = {}
21
22
23
24
25
26
27
28
29
30
31 def main():
32
33
34     try:
35         _thread.start_new_thread(gui,())
36         my_socket = socket.socket(socket.AF_INET,
               socket.SOCK_STREAM)# af_inet = ipv4, sock_stream =
               tcp
37         print("$ Initialising socket...")
38         my_socket.bind(('',DEFAULT_PORT))# all interfaces port
               : default_port
39         print("$ Bind successful")
40         my_socket.listen(BACKLOG)# will accept #|backlog|
               values until it wont accept new connections
41         print("$ Server listening on port [ %d ]\n" %
               (DEFAULT_PORT))
42         printBlocked()
43     except Exception as e0:
44         print("$ socket initiation failed")
45         sys.exit(0)
46
47     ## loop while waiting for new connections from browser
48
49     while True:
50         try:
51             my_connection, my_address = my_socket.accept() #
                   setting up connection, addr using socket
```

```
52              data = my_connection.recv(MAX_DATA)
53              bandwidth = len(data)
54              starting_time = time.time()
55              _thread.start_new_thread(proxy,(my_connection,
                    data, my_address,starting_time,bandwidth))
56              time.sleep(0.0001)
57          except ConnectionResetError:
58              pass
59          except KeyboardInterrupt:
60          #my_connection.close()
61              my_socket.close()
62              print("$ socket force closed")
63              sys.exit(1)
64
65      my_socket.close()
66      return
67
68
69
70  def printBlocked():
71      print("Blocked Sites @ Init:\n")
72      for i in blocked_sites:
73          print(i)
74      print("\n")
75
76  def gui():
77
78      def block_entry():
79          e = block.get()
80          if e not in blocked_sites:
81              blocked_sites.append(e)
82              list.insert(END,e)
83              print(e + " has been blocked")
84          else:
85              print(e," has already been blocked")
86
87      def unblock_entry():
88          e = unblock.get()
89          if e in blocked_sites:
90              blocked_sites.remove(e)
91              for i, listbox_entry in enumerate(list.get(0, END)):
92                  if listbox_entry == e:
93                      list.delete(i)
94              print(e," has been unblocked")
95          else:
```

```
96              print(e," is not blocked")
97
98
99      gui = tk.Tk()
100     gui.geometry("350x300")
101     gui.title('Blocking System')
102
103
104     block = Entry(gui)
105     block.grid(row=0, column=0)
106     unblock = Entry(gui)
107     unblock.grid(row=1, column=0)
108
109     block_button = Button(gui, text = "Block",
            command=block_entry)
110     block_button.grid(row=0, column=1)
111     block_button = Button(gui, text = "Un-block",
            command=unblock_entry)
112     block_button.grid(row=1, column=1)
113
114     list = Listbox(gui)
115     list.grid(row=2, columnspan=2)
116     list.config(width=50, height=50)
117
118     for elem in blocked_sites:
119         list.insert(END,elem)
120
121     mainloop()
122
123
124
125
126 def proxy(my_connection, data,
        my_address,starting_time,bandwidth):
127
128     if data is None:
129         return
130
131     https, web_srv, port, url,method = parse_req(data)
132
133     if web_srv in blocked_sites:
134         print("$ That host is restricted! (",web_srv,")")
135         my_connection.close()
136         return
137
```

```
138     if method != "GET" and method != "CONNECT":
139         my_connection.close()
140         print("$ Invalid method :",method, " connection closed")
141         return
142 
143 
144     print("$ starting new thread for : ",url)
145 
146     t0 = time.time()
147     cached_response = cache.get(url)
148     if cached_response != None:
149         my_connection.sendall(cached_response)
150         t1 = time.time()
151         print("$ Cache hit on url: ",url,"\n$ Request took: " +
              str(t1-t0) + " seconds with cache.\n $ Request took:
              " + str(timeData[url]) + " seconds previously")
152         #my_connection.close()
153         return
154 
155 
156     else:
157             if https == False:
158                 bw =
                        proxy_srv_http(web_srv,port,my_connection,data,my_address,url,ba
159             else:
160                 proxy_srv_https(web_srv,port,my_connection,my_address,url)
161 
162 
163             end = time.time()
164             if https == False:
165                 print("$ time elapsed :
                        ",str(end-starting_time), " seconds\n")
166                 if(bw == None):
167                     print("$ No bandwidth usage (retrieved from
                            cache) \n")
168                 else:
169                     print("$ total bandwidth : ",str(bw), "
                            bytes")
170             my_connection.close()
171             return
172 
173 
174 def proxy_srv_http(web_srv, port, my_connection, data,
        my_address, url,bandwidth):
175 
```

```
176
177         t0 = time.time()
178
179         print("$ sending HTTP request url:",url)
180         my_socket2 = socket.socket(socket.AF_INET,
                socket.SOCK_STREAM)
181         my_socket2.connect((web_srv, port))
182         my_socket2.settimeout(3)
183         my_socket2.setblocking(0)
184
185         my_socket2.send(data)
186         b = len(data)
187
188
189         if url in cache:
190             print("$ Found request in cache")
191             t1 = time.time()
192             print("$ Request took: " + str(t1-t0) + "s with cache.")
193
194         else:
195             http_reply = bytearray("",'utf-8')
196             try:
197                 while True:
198                     rcv = my_socket2.recv(MAX_DATA)
199                     if (len(rcv) > 0):
200                         my_connection.send(rcv)
201                         http_reply.extend(rcv)
202                     else:
203                         break
204             except socket.error:
205                 #print("$ HTTP socket error:",str(socket.error))
206                 pass
207             t1 = time.time()
208             cache[url] = http_reply
209
210             timeData[url] = t1-t0
211             print("$ added to cache (URL) : ",url)
212             my_socket2.close()
213             #print("$ HTTP Request completed (URL): ",url)
214             b += len(http_reply)
215             return b
216         print("$ HTTP Request completed (URL): ",url)
217         my_socket2.close()
218         #print("$ HTTP socket closed ")
219
```

```
220
221
222
223
224 def proxy_srv_https(web_srv,port,my_connection,my_address,url):
225
226
227     print("$ sending HTTPS")
228     try:
229         my_socket2 = socket.socket(socket.AF_INET,
                socket.SOCK_STREAM)
230         my_socket2.connect((web_srv, port))
231         reply = "HTTP/1.0 200 Connection
                established\r\nProxy-agent: Pyx\r\n\r\n"
232         my_connection.sendall(reply.encode())
233
234     except socket.error as err:
235                 print(err)
236                 return
237
238     my_connection.setblocking(0)
239     my_socket2.setblocking(0)
240
241     while True:
242         try:
243             request = my_connection.recv(MAX_DATA)
244             my_socket2.sendall(request)
245         except socket.error as err:
246             pass
247         try:
248             reply = my_socket2.recv(MAX_DATA)
249             my_connection.sendall(reply)
250         except socket.error as err:
251             pass
252
253     print("$ HTTPS request completed (URL) : ",url)
254
255
256 def get_host(line_data):
257     host = ""
258     for line in line_data:
259         h = line.find("Host")
260         if h != -1:
261             host = line
262             break
```

```
263     return host
264
265
266 def get_websrv(host):
267     web_srv = host.split(": ")[1]
268     port_pos = web_srv.find(":")
269     web_srv2 = ""
270     i = 0
271     if port_pos != -1:
272         while i < port_pos:
273             web_srv2 += web_srv[i]
274             i += 1
275     else:
276         web_srv2 = web_srv
277     return web_srv2
278
279
280 def parse_req(data):
281     try:
282         https = False
283         print("")
284         print("$ Decoding request")
285
286         line_data = data.decode().split("\r\n")
287
288         get = line_data[0].find("GET")
289         method = line_data[0].split(' ')[0]
290         if get == -1:
291             https = True
292
293         url_http = line_data[0].split(' ')[1]
294
295         url = "https://" + url_http.split(':')[0] + "/"
296
297         host = get_host(line_data)
298         webserver = get_websrv(host)
299
300         print("$ Method :",method)
301         print("$ URL :", url_http)
302         print("$ Webserver:",webserver)
303
304         if https is True:
305             port = 443
306         else:
307             port = 80
```

```
308             if https:
309                 return https, webserver, port, url, method
310             else:
311                 return https, webserver, port, url_http, method
312         except Exception:
313             #print("decode exception")
314             pass
315             return True, 0, 0, "", ""
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334 main()
```