

Proxy Server

A proxy implemented in python using websockets

Rvail Naveed - 17321983 | Advanced Telecommunications | 24-02-2020

<https://github.com/rvailnaveed/proxy-server>

# Introduction

The goal of this project was to implement a 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. The proxy should be able to handle HTTP as well as HTTPS requests, cache pages for faster access and offer dynamic blocking of URL’s as a form of access control.

The technologies I used to accomplish this were:

- **Python 3.7**

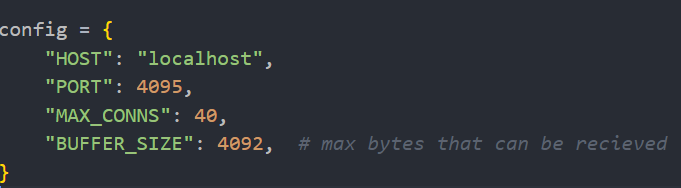
- The standard library’s **sockets** module

# Approach

I decided to use python as it is well supported and very easy to program in. You can have a working prototype in a very short amount of time.

# Configuration

I made a configuration dictionary object which can hold the designated port number and the host. This is to configure the socket for later use to specify what port it should bind to and the max bytes that can be passed to and from the client/server. It also specifies a “**MAX\_CONNS**” parameter which can be passed to the sockets *listen()* function as a backlog. This is the number of clients that the socket will accept before blocking and waiting for previous clients before accepting new ones.



# Server

Using pythons socket module from the standard library, I set up a TCP socket that would listen to incoming connections to the socket and respond appropriately to them.

A screenshot of a cell phone

Description automatically generated

# Threading

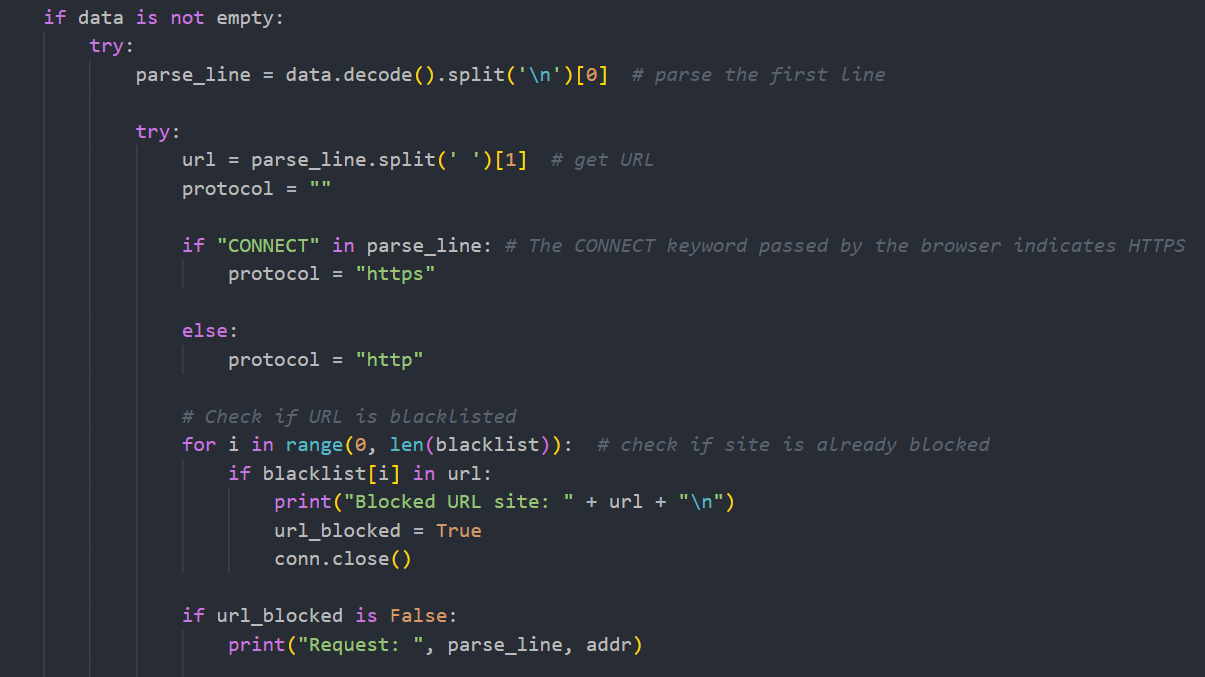
For each new request I created a new thread and passed it the proxyServer function. There can be as many threads as MAX\_CONNS.

A screenshot of a cell phone

Description automatically generated

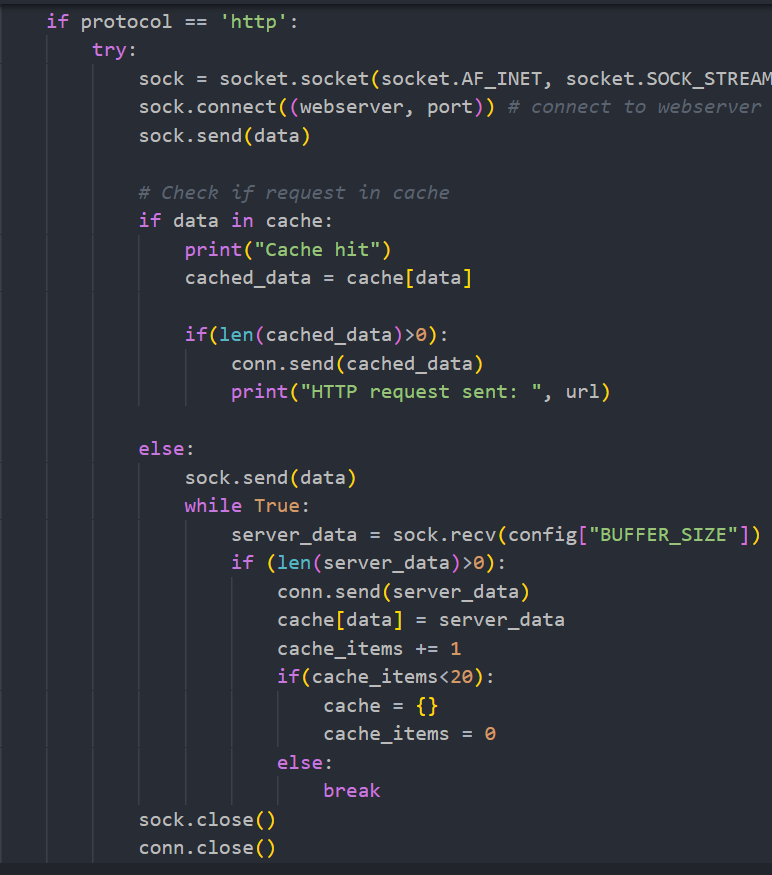
# Handling requests

The data retrieved from the request is parsed to find if the protocol used by the website is either http or https. HTTPS is determined by it containing the “CONNECT” keyword. The URL is then checked to see if it is in the list of blacklisted URL’s, if so the connection between the client and webserver is closed.



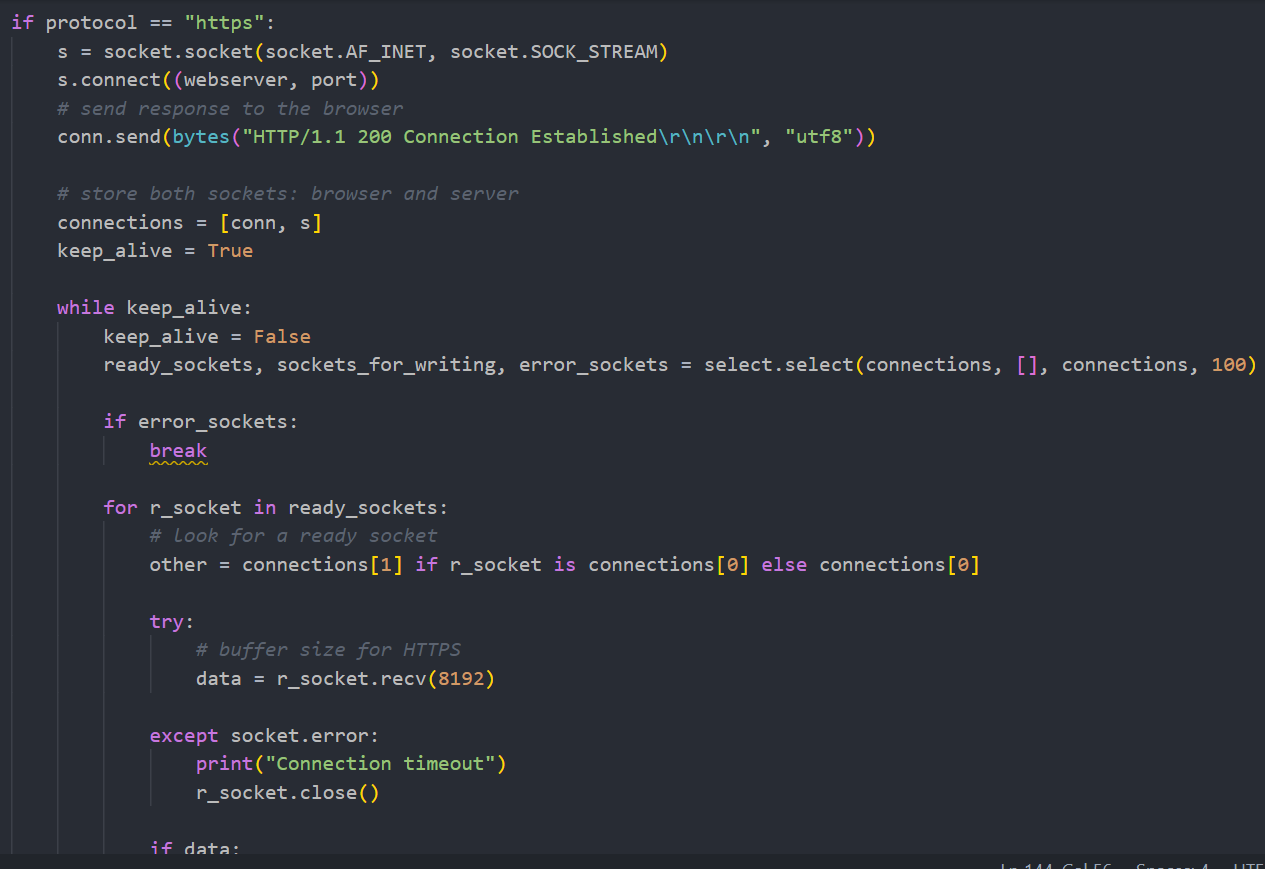
# HTTP

If the protocol is http, then a new socket object is created that connects to the webserver. Then the data being requested is checked against the cache to see if it is present. If so, then it is served from cache, otherwise the request is forwarded to the server by the proxy.



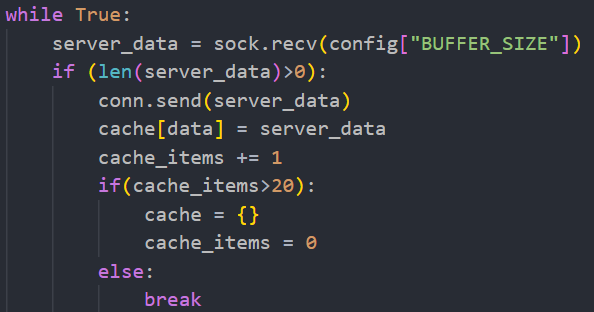
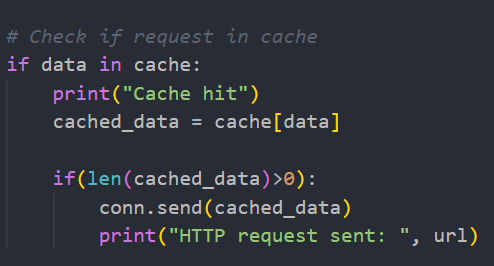
# HTTPS

HTTPS requests are handled a bit differently. The connection must be kept alive between the browser and the server. The server will then perform a TLS handshake with the site being requested and the request will be processed. The buffer size is also increased for HTTPS sites,



# Caching

The cache was implemented using a dictionary. If a http request is sent, it is checked if that request exists in the dictionary, if not is added and the request is served from the webserver the first time. Any subsequent requests to the same URL will be served by the cache. If more than 20 entries exist in the cache, the cache is then flushed.



# Dynamic Blocking

Dynamic blocking is implemented with the file block.py. While the server is running, in a terminal you can run the command: *python block.py <url>* and the URL will then be added to the blacklist by the proxy. This works by writing the url to a file and the proxy checking the file for new entries every set interval.

**Proxy In-Action Screenshots:**

A screenshot of a computer screen

Description automatically generated

*Dynamic blocking*

A picture containing text

Description automatically generated



*Cache hit*

A screenshot of a computer screen

Description automatically generated

*Normal Execution*