Project 3:

Proxy Server

Software Engineering
2016024902 윤세령

Contents

I. Project Preview

II. Code Explanation

III. Instructions

IV. Program Operating & Results

V. My Opinion

I. Project Preview

- You have to develop a small web proxy server which is also able to cache web pages.
- The proxy should be able to receive requests, forward them, read replies, and return those to the clients.
- The proxy works as follows:
  1. The proxy listens for requests from clients
  2. When there is a request, the proxy spawns a new thread for handling the request and creates an HttpRequest-object which contains the request.
  3. The new thread sends the request to the server and reads the server's reply into an HttpResponse-object.
  4. The thread sends the response back to the requesting client.

II. Code Explanation

```
class Server:
    """ The server class """

    def __init__(self, config):
        signal.signal(signal.SIGINT, self.shutdown)     # Shutdown on Ctrl+C
        self.serverSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)       # Create a TCP socket
        self.serverSocket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)    # Re-use the socket
        self.serverSocket.bind((config['HOST_NAME'], config['BIND_PORT']))  # bind the socket to a public host, and a port
        self.serverSocket.listen(10)    # become a server socket
        self.__clients = {}


    def listenForClient(self):
        """ Wait for clients to connect """
        while True:
            (clientSocket, client_address) = self.serverSocket.accept()   # Establish the connection
            d = threading.Thread(name=self._getClientName(client_address), target=self.proxy_thread, args=(clientSocket, client_address))
            d.setDaemon(True)
            d.start()
        self.shutdown(0,0)


    def _ishostAllowed(self, host):
        """ Check if host is allowed to access the content """
        for wildcard in config['HOST_ALLOWED']:
            if fnmatch.fnmatch(host, wildcard):
                return True
        return False
```

- __init__ : TCP socket setting is done in this part.
- listenForClient : exists for receive the request from client. Connection is established and thread is made in this part and starts.
- isHostAllowed : can set the allowed pages, but I set this to allow all of web pages.

```python
def proxy_thread(self, conn, client_addr):
    """
    ****************************************
    ********** PROXY_THREAD FUNC ************
      A thread to handle request from browser
    ****************************************
    """

    request = conn.recv(config['MAX_REQUEST_LEN'])        # get the request from browser
    first_line = request.split('\n')[0]                   # parse the first line
    url = first_line.split(' ')[1]                        # get url

    # Check if the host:port is blacklisted
    for i in range(0,len(config['BLACKLIST_DOMAINS'])):
        if config['BLACKLIST_DOMAINS'][i] in url:
            self.log("FAIL", client_addr, "BLACKLISTED: " + first_line)
            conn.close()
            # TODO: Create response for 403 Forbidden
            return

    # Check if client is allowed or not
    if not self._ishostAllowed(client_addr[0]):
        # TODO: Create response for 403 Forbidden
        return

    self.log("WARNING", client_addr, "REQUEST: " + first_line)

    # find the webserver and port
    http_pos = url.find("://")          # find pos of ://
    if (http_pos==-1):
        temp = url
    else:
        temp = url[(http_pos+3):]       # get the rest of url

    port_pos = temp.find(":")           # find the port pos (if any)

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

    webserver = ""
    port = -1
    if (port_pos==-1 or webserver_pos < port_pos):      # default port
        port = 80
        webserver = temp[:webserver_pos]
    else:                                               # specific port
        port = int((temp[(port_pos+1):])[:webserver_pos-port_pos-1])
        webserver = temp[:port_pos]
```

- proxy_thread : This part serves as parsing and storing http://~(URL) parts.

```python
    try:
        # create a socket to connect to the web server
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.settimeout(config['CONNECTION_TIMEOUT'])
        s.connect((webserver, port))
        s.sendall(request)                              # send request to webserver

        while 1:
            data = s.recv(config['MAX_REQUEST_LEN'])          # receive data from web server
            if (len(data) > 0):
                conn.send(data)                               # send to browser
            else:
                break
        s.close()
        conn.close()
    except socket.error as error_msg:
        self.log("ERROR", client_addr, error_msg)
        if s:
            s.close()
        if conn:
            conn.close()
        self.log("WARNING", client_addr, "Peer Reset: " + first_line)
```

This part receives the data by using socket with a parsed port and URL, and send it back the data.

```
def _getClientName(self, cli_addr):
    """ Return the clientName.
    """
    return "Client"


def shutdown(self, signum, frame):
    """ Handle the exiting server. Clean all traces """
    self.log("WARNING", -1, 'Shutting down gracefully...')
    main_thread = threading.currentThread()        # Wait for all clients to exit
    for t in threading.enumerate():
        if t is main_thread:
            continue
        self.log("FAIL", -1, 'joining ' + t.getName())
        t.join()
    self.serverSocket.close()
    sys.exit(0)
```

- Client name & shut down parts : returns name of client and close the server.

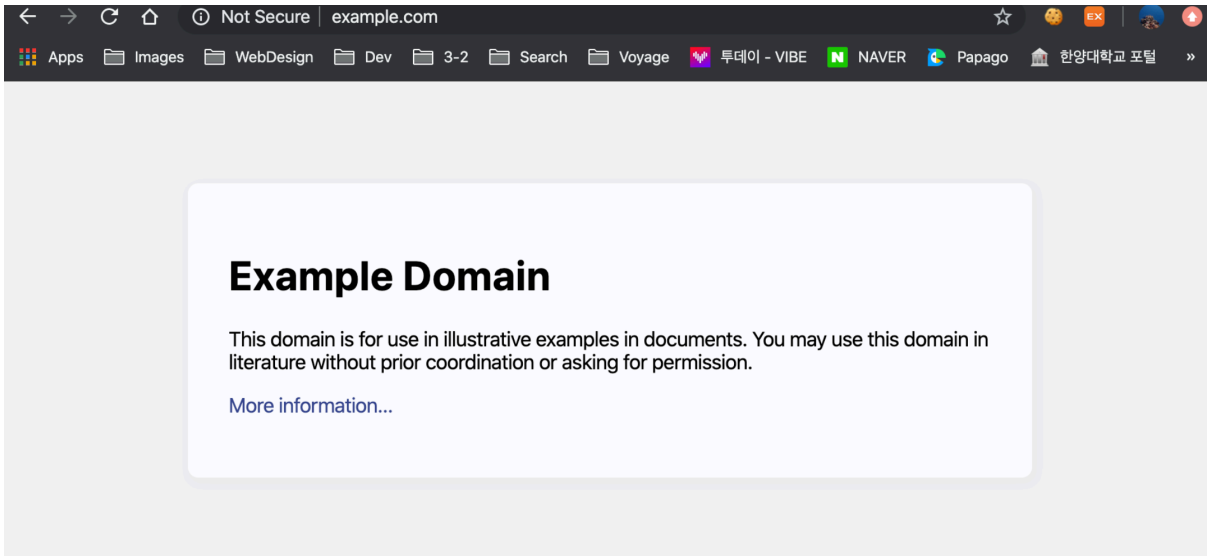
III. Instructions

Configure my Chrome browser to use the proxy, so I give the address of the proxy to 127.0.0.1(localhost), and port: 12345. Then go to url http://www.example.com while the proxy server is connected.

IV. Program Operating & Results

```
→  Project_3 python proxy_server.py
[Tue, 03 Dec 2019 16:36:02] (Client    ) 127.0.0.1:57827 REQUEST: GET http://example.com/ HTTP/1.1
[Tue, 03 Dec 2019 16:36:07] (Client    ) 127.0.0.1:57827 timed out
[Tue, 03 Dec 2019 16:36:07] (Client    ) 127.0.0.1:57827 Peer Reset: GET http://example.com/ HTTP/1.1
[Tue, 03 Dec 2019 16:36:07] (Client    ) 127.0.0.1:57828 REQUEST: GET http://example.com/favicon.ico HTTP/1.1
[Tue, 03 Dec 2019 16:36:12] (Client    ) 127.0.0.1:57828 timed out
[Tue, 03 Dec 2019 16:36:12] (Client    ) 127.0.0.1:57828 Peer Reset: GET http://example.com/favicon.ico HTTP/1.1
```

The logs above by sending a proxy server to url(http://www.example.com) and sending it from the host.



V. My Opinion

The concept of receiving a request when creating a proxy server, parsing it and sending it back open was simple. However, I thought the proxy server would not work because it only works in https, and I had lots of time tried to fix it. Finally I tested it using a simple website frame that is only http(http://www.example.com), It worked well. The understanding of Python language also increased, and it was fun to make and test the Proxy.