Final Project: Proxy Documentation

Christopher Jensen (chjjense@ucsc.edu)

How To Use

1. Cd into folder proxy
2. There are 4 folders and 2 files. The two files are Makefile and README.txt. The four folders are doc, bin, build, and src. Doc is where the documentation is located, build is for the .o files, bin is for the binaries, and src is where the source code for this project is located. In src there is myproxy.cpp.
3. Run a quick “make” in the top directory where the Makefile is
4. Cd into bin
5. Firstly you need to set up the proxy server. You can do this by running the command:
    a. ./myproxy *listen_port forbidden_sites_file*_path *access_log_file_path*
    b. Where *listen_port* is the port the proxy server will be listening on, and is a number between 1024 and 65536
    c. *Forbidden_sites_file*_path is the path to the file you want to serve as the list of websites/IPs that will be blocked by the proxy. Below is an example of the format of this file:

    ```
    www.bookface.com
    www.youtube.com
    www.fakenews.com
    10.6.6.6
    ```

    d.
    e. *Access_log_file_path* is the path to the file which will hold a record of responses served by the proxy.
6. After the server is started, you can start sending requests to the listening port and receiving responses via curl or any other web client.

Internal Design

- **myproxy.cpp**:
    - The proxy program begins in main() by setting up two signal handlers, one for SIGINT (for the control + C signal), and a custom one that catches SIGQUIT (control + \). Both of these signals are handled by the main thread, and are masked by any other thread spawned by main to avoid any problems. SIGINT signal triggers the update of the forbidden sites file, and SIGQUIT triggers a flag that shuts down the active threads and closes down the server. I implemented this signal for ease of use, given that the normal termination command (control C), is being used for other purposes.
    - At the start of the program, the forbidden_sites_file is parsed and all of its contents are added to a hash table for easy lookup in the future.
    - The main thread gets all the command line arguments given by the user, spawns a pool of 50 threads, and starts listening for requests on accept().

    - The pool of 50 threads uses a queue to wait on tasks to become available. Once a task is added to the queue by main, a condition variable is triggered and a thread from the pool is tasked with handling this request. Once the request has been fulfilled, the thread waits again in the pool for more requests to come in.
    - Once a request has been acquired by a thread, it is parsed for any “Bad Request” issues. If there are issues, a 400 code is sent back to the client via send(). The parsing function also stores some key variables in a struct. This struct is returned to the thread function. After parsing, the “host” field is checked by looking for it in the hash table of forbidden files. If it is found in the hash table, a 403 error is sent back to the client.
    - After we’ve parsed the request and made sure the host is not a forbidden website, a DNS lookup is conducted using the host and port. If the DNS can’t resolve anything, a 502 error is sent back to the client. After this, an ssl connection to the server is set up. The original request from the client is sent to the server using SSL_write(), and the response is gathered in SSL_read().
    - SSL_read() is wrapped in a while(1) loop and is called every time through the loop until content-length bytes are sent back to the client. Or if the response is chunked-encoded, SSL_read() is called every time data is available on the socket by using select(). We know the chunked encoding has finished when select() times out after 5 seconds. If the request is a “HEAD” request, the loop is exited after we encounter “\r\n\r\n”.
    - After reading all bytes back to the client, the thread updates the log file with the relevant information, and returns to the thread pool to wait for more requests to come in.

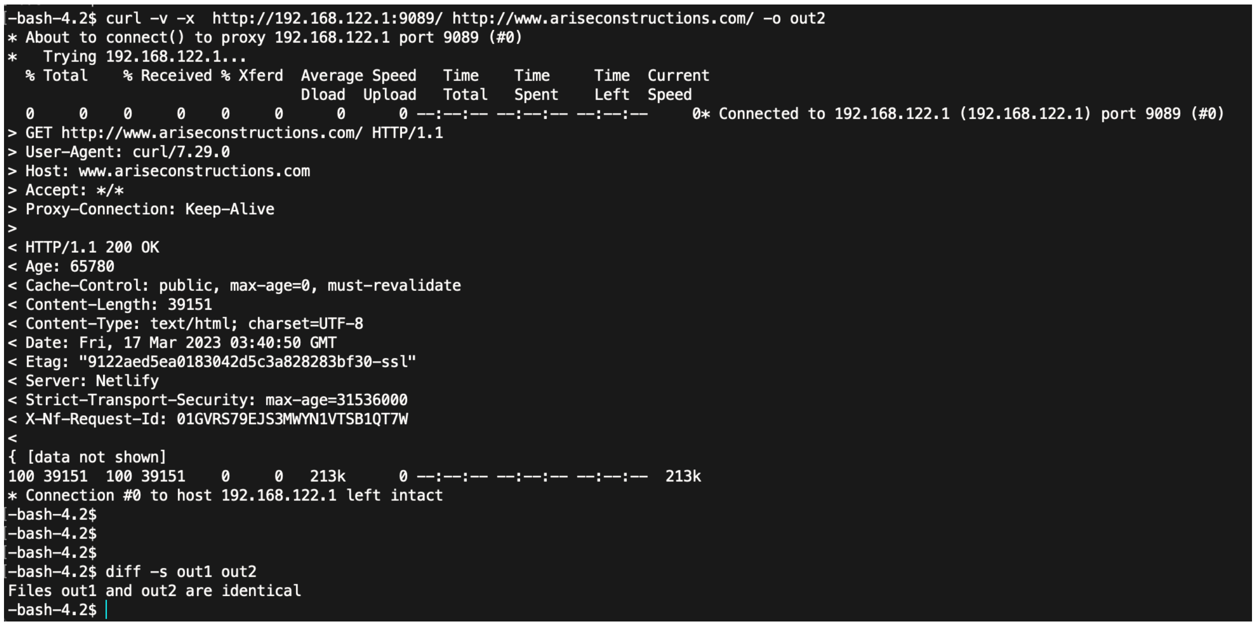
Shortcomings

- I think my proxy implementation is solid for what it is used for: turning cleartext HTTP requests into HTTPS requests using SSL. The only real shortcoming I can think of is how I handle chunked encoding. Instead of parsing for the length headers before each chunk, I instead wait for select to time out to know when the response has finished. This implementation works, but probably isn't the most practical or efficient way to handle it.

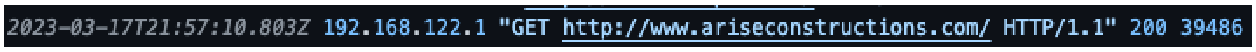
<u>Testing</u>

Request to static HTTPS website (with regular curl and proxy curl diff):

- ./myproxy 9089 ~/cse156/proxy/forbidden_sites.txt ~/cse156/proxy/access.log
- curl -v https://www.ariseconstructions.com/ -o out1
- curl -v -x http://192.168.122.1:9089/ http://www.ariseconstructions.com/ -o out2



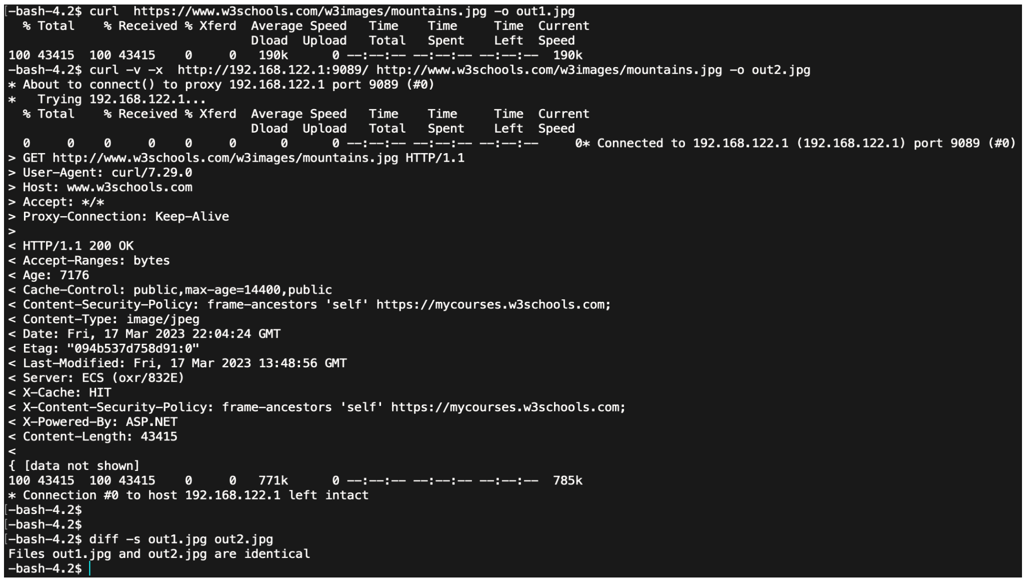
```
-bash-4.2$ curl -v -x  http://192.168.122.1:9089/ http://www.ariseconstructions.com/ -o out2
* About to connect() to proxy 192.168.122.1 port 9089 (#0)
*   Trying 192.168.122.1...
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0* Connected to 192.168.122.1 (192.168.122.1) port 9089 (#0)
> GET http://www.ariseconstructions.com/ HTTP/1.1
> User-Agent: curl/7.29.0
> Host: www.ariseconstructions.com
> Accept: */*
> Proxy-Connection: Keep-Alive
>
< HTTP/1.1 200 OK
< Age: 65780
< Cache-Control: public, max-age=0, must-revalidate
< Content-Length: 39151
< Content-Type: text/html; charset=UTF-8
< Date: Fri, 17 Mar 2023 03:40:50 GMT
< Etag: "9122aed5ea0183042d5c3a828283bf30-ssl"
< Server: Netlify
< Strict-Transport-Security: max-age=31536000
< X-Nf-Request-Id: 01GVRS79EJS3MWYN1VTSB1QT7W
<
{ [data not shown]
100 39151  100 39151    0     0   213k      0 --:--:-- --:--:-- --:--:--  213k
* Connection #0 to host 192.168.122.1 left intact
-bash-4.2$
-bash-4.2$
-bash-4.2$
-bash-4.2$ diff -s out1 out2
Files out1 and out2 are identical
-bash-4.2$
```



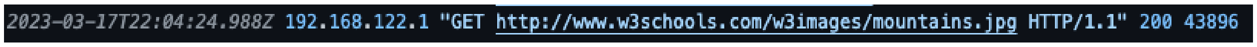
```
2023-03-17T21:57:10.803Z 192.168.122.1 "GET http://www.ariseconstructions.com/ HTTP/1.1" 200 39486
```

Request jpg from HTTPS website (with regular curl and proxy curl diff):

- ./myproxy 9089 ~/cse156/proxy/forbidden_sites.txt ~/cse156/proxy/access.log
- curl https://www.w3schools.com/w3images/mountains.jpg -o out1.jpg
- curl -v -x http://192.168.122.1:9089/ http://www.w3schools.com/w3images/mountains.jpg -o out2.jpg

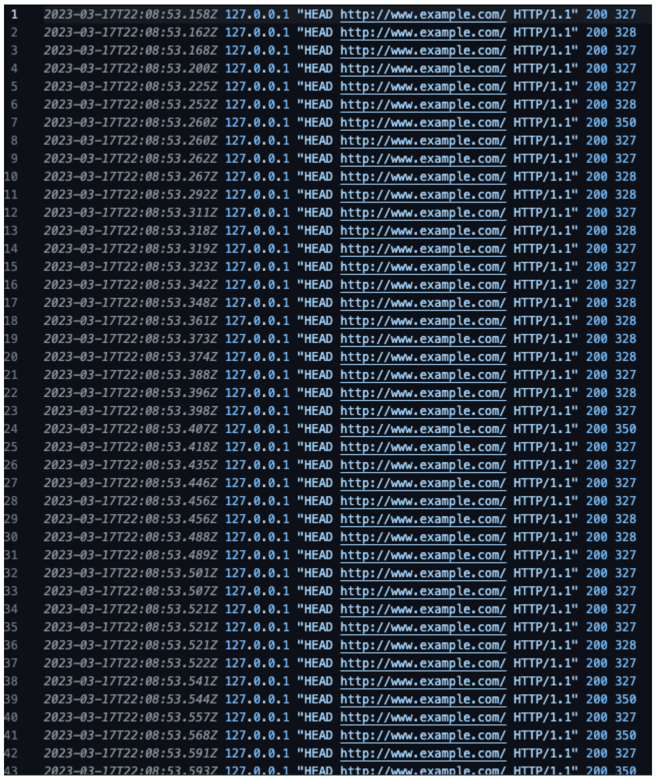


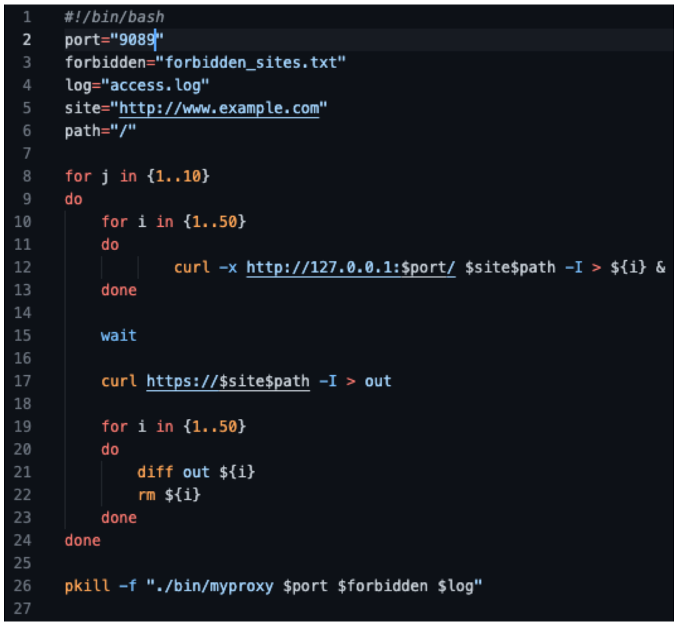
```
-bash-4.2$ curl  https://www.w3schools.com/w3images/mountains.jpg -o out1.jpg
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 43415  100 43415    0     0   190k      0 --:--:-- --:--:-- --:--:--  190k
-bash-4.2$ curl -v -x  http://192.168.122.1:9089/ http://www.w3schools.com/w3images/mountains.jpg -o out2.jpg
* About to connect() to proxy 192.168.122.1 port 9089 (#0)
*   Trying 192.168.122.1...
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0* Connected to 192.168.122.1 (192.168.122.1) port 9089 (#0)
> GET http://www.w3schools.com/w3images/mountains.jpg HTTP/1.1
> User-Agent: curl/7.29.0
> Host: www.w3schools.com
> Accept: */*
> Proxy-Connection: Keep-Alive
>
< HTTP/1.1 200 OK
< Accept-Ranges: bytes
< Age: 7176
< Cache-Control: public,max-age=14400,public
< Content-Security-Policy: frame-ancestors 'self' https://mycourses.w3schools.com;
< Content-Type: image/jpeg
< Date: Fri, 17 Mar 2023 22:04:24 GMT
< Etag: "094b537d758d91:0"
< Last-Modified: Fri, 17 Mar 2023 13:48:56 GMT
< Server: ECS (oxr/832E)
< X-Cache: HIT
< X-Content-Security-Policy: frame-ancestors 'self' https://mycourses.w3schools.com;
< X-Powered-By: ASP.NET
< Content-Length: 43415
<
{ [data not shown]
100 43415  100 43415    0     0   771k      0 --:--:-- --:--:-- --:--:--  785k
* Connection #0 to host 192.168.122.1 left intact
-bash-4.2$
-bash-4.2$
-bash-4.2$ diff -s out1.jpg out2.jpg
Files out1.jpg and out2.jpg are identical
-bash-4.2$
```


```
2023-03-17T22:04:24.988Z 192.168.122.1 "GET http://www.w3schools.com/w3images/mountains.jpg HTTP/1.1" 200 43896
```

Sending 500 concurrent HEAD requests:

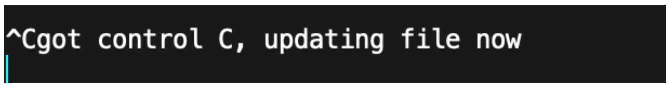
- ./myproxy 9089 ~/cse156/proxy/forbidden_sites.txt ~/cse156/proxy/access.log
- ./script.sh

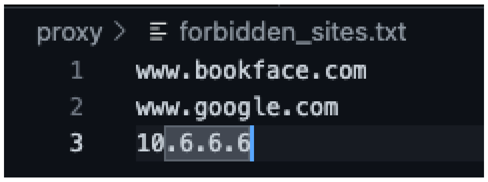

```
2023-03-17T22:08:53.158Z 127.0.0.1 "HEAD http://www.example.com/ HTTP/1.1" 200 327
2023-03-17T22:08:53.162Z 127.0.0.1 "HEAD http://www.example.com/ HTTP/1.1" 200 328
2023-03-17T22:08:53.168Z 127.0.0.1 "HEAD http://www.example.com/ HTTP/1.1" 200 327
2023-03-17T22:08:53.200Z 127.0.0.1 "HEAD http://www.example.com/ HTTP/1.1" 200 327
2023-03-17T22:08:53.225Z 127.0.0.1 "HEAD http://www.example.com/ HTTP/1.1" 200 327
2023-03-17T22:08:53.252Z 127.0.0.1 "HEAD http://www.example.com/ HTTP/1.1" 200 328
2023-03-17T22:08:53.260Z 127.0.0.1 "HEAD http://www.example.com/ HTTP/1.1" 200 350
2023-03-17T22:08:53.260Z 127.0.0.1 "HEAD http://www.example.com/ HTTP/1.1" 200 327
2023-03-17T22:08:53.262Z 127.0.0.1 "HEAD http://www.example.com/ HTTP/1.1" 200 327
2023-03-17T22:08:53.267Z 127.0.0.1 "HEAD http://www.example.com/ HTTP/1.1" 200 328
2023-03-17T22:08:53.292Z 127.0.0.1 "HEAD http://www.example.com/ HTTP/1.1" 200 328
2023-03-17T22:08:53.311Z 127.0.0.1 "HEAD http://www.example.com/ HTTP/1.1" 200 327
2023-03-17T22:08:53.318Z 127.0.0.1 "HEAD http://www.example.com/ HTTP/1.1" 200 328
2023-03-17T22:08:53.319Z 127.0.0.1 "HEAD http://www.example.com/ HTTP/1.1" 200 327
2023-03-17T22:08:53.323Z 127.0.0.1 "HEAD http://www.example.com/ HTTP/1.1" 200 327
2023-03-17T22:08:53.342Z 127.0.0.1 "HEAD http://www.example.com/ HTTP/1.1" 200 327
2023-03-17T22:08:53.348Z 127.0.0.1 "HEAD http://www.example.com/ HTTP/1.1" 200 328
2023-03-17T22:08:53.361Z 127.0.0.1 "HEAD http://www.example.com/ HTTP/1.1" 200 328
2023-03-17T22:08:53.373Z 127.0.0.1 "HEAD http://www.example.com/ HTTP/1.1" 200 328
2023-03-17T22:08:53.374Z 127.0.0.1 "HEAD http://www.example.com/ HTTP/1.1" 200 328
2023-03-17T22:08:53.388Z 127.0.0.1 "HEAD http://www.example.com/ HTTP/1.1" 200 327
2023-03-17T22:08:53.396Z 127.0.0.1 "HEAD http://www.example.com/ HTTP/1.1" 200 328
2023-03-17T22:08:53.398Z 127.0.0.1 "HEAD http://www.example.com/ HTTP/1.1" 200 327
2023-03-17T22:08:53.407Z 127.0.0.1 "HEAD http://www.example.com/ HTTP/1.1" 200 350
2023-03-17T22:08:53.418Z 127.0.0.1 "HEAD http://www.example.com/ HTTP/1.1" 200 327
2023-03-17T22:08:53.435Z 127.0.0.1 "HEAD http://www.example.com/ HTTP/1.1" 200 327
2023-03-17T22:08:53.446Z 127.0.0.1 "HEAD http://www.example.com/ HTTP/1.1" 200 327
2023-03-17T22:08:53.456Z 127.0.0.1 "HEAD http://www.example.com/ HTTP/1.1" 200 327
2023-03-17T22:08:53.456Z 127.0.0.1 "HEAD http://www.example.com/ HTTP/1.1" 200 328
2023-03-17T22:08:53.488Z 127.0.0.1 "HEAD http://www.example.com/ HTTP/1.1" 200 328
2023-03-17T22:08:53.489Z 127.0.0.1 "HEAD http://www.example.com/ HTTP/1.1" 200 327
2023-03-17T22:08:53.501Z 127.0.0.1 "HEAD http://www.example.com/ HTTP/1.1" 200 327
2023-03-17T22:08:53.507Z 127.0.0.1 "HEAD http://www.example.com/ HTTP/1.1" 200 327
2023-03-17T22:08:53.521Z 127.0.0.1 "HEAD http://www.example.com/ HTTP/1.1" 200 327
2023-03-17T22:08:53.521Z 127.0.0.1 "HEAD http://www.example.com/ HTTP/1.1" 200 327
2023-03-17T22:08:53.521Z 127.0.0.1 "HEAD http://www.example.com/ HTTP/1.1" 200 328
2023-03-17T22:08:53.522Z 127.0.0.1 "HEAD http://www.example.com/ HTTP/1.1" 200 327
2023-03-17T22:08:53.541Z 127.0.0.1 "HEAD http://www.example.com/ HTTP/1.1" 200 327
2023-03-17T22:08:53.544Z 127.0.0.1 "HEAD http://www.example.com/ HTTP/1.1" 200 350
2023-03-17T22:08:53.557Z 127.0.0.1 "HEAD http://www.example.com/ HTTP/1.1" 200 327
2023-03-17T22:08:53.568Z 127.0.0.1 "HEAD http://www.example.com/ HTTP/1.1" 200 350
2023-03-17T22:08:53.591Z 127.0.0.1 "HEAD http://www.example.com/ HTTP/1.1" 200 327
2023-03-17T22:08:53.593Z 127.0.0.1 "HEAD http://www.example.com/ HTTP/1.1" 200 350
```

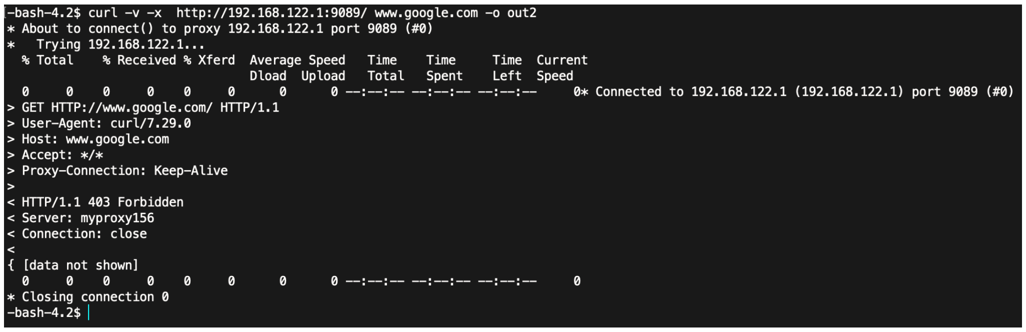

```bash
#!/bin/bash
port="9089"
forbidden="forbidden_sites.txt"
log="access.log"
site="http://www.example.com"
path="/"

for j in {1..10}
do
    for i in {1..50}
    do
            curl -x http://127.0.0.1:$port/ $site$path -I > ${i} &
    done

    wait

    curl https://$site$path -I > out

    for i in {1..50}
    do
        diff out ${i}
        rm ${i}
    done
done

pkill -f "./bin/myproxy $port $forbidden $log"
```

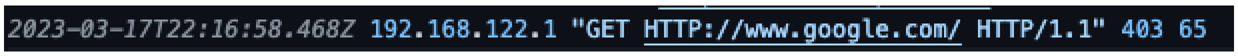
Sending a Request to a Server on the Forbidden Website List (using control C signal):

- ./myproxy 9089 ~/cse156/proxy/forbidden_sites.txt ~/cse156/proxy/access.log
- curl -I -v -x http://192.168.122.1:9089/ http://www.example.com -o out2


```
^Cgot control C, updating file now
```


```
proxy > forbidden_sites.txt
1  www.bookface.com
2  www.google.com
3  10.6.6.6
```

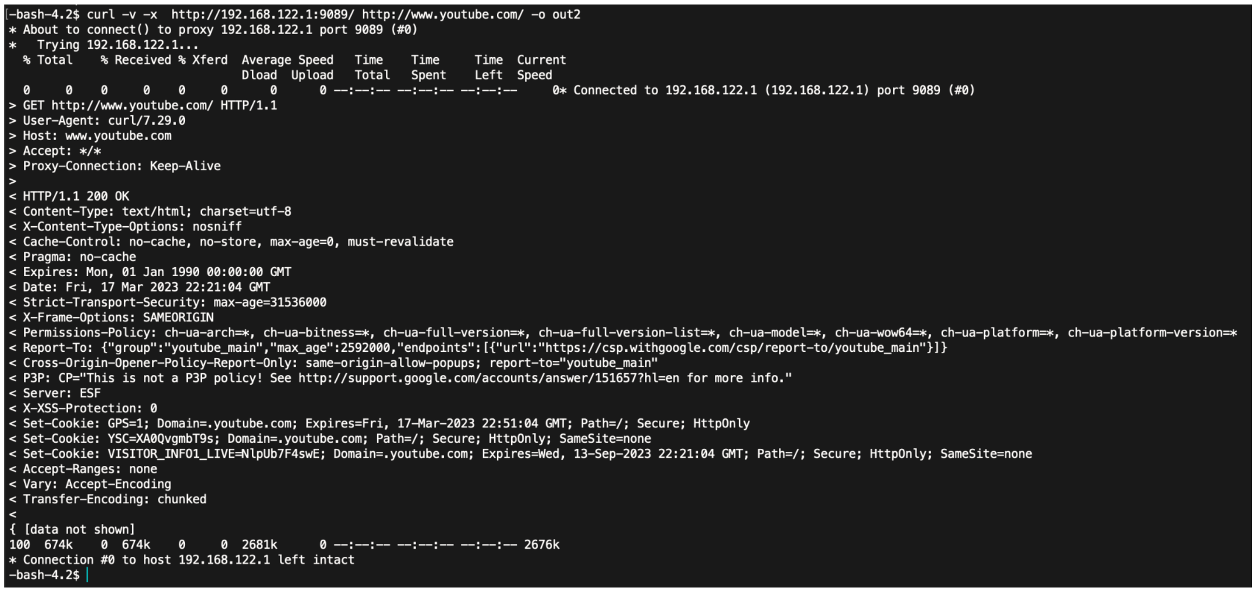

```
-bash-4.2$ curl -v -x  http://192.168.122.1:9089/ www.google.com -o out2
* About to connect() to proxy 192.168.122.1 port 9089 (#0)
*   Trying 192.168.122.1...
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0* Connected to 192.168.122.1 (192.168.122.1) port 9089 (#0)
> GET HTTP://www.google.com/ HTTP/1.1
> User-Agent: curl/7.29.0
> Host: www.google.com
> Accept: */*
> Proxy-Connection: Keep-Alive
>
< HTTP/1.1 403 Forbidden
< Server: myproxy156
< Connection: close
<
{ [data not shown]
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
* Closing connection 0
-bash-4.2$
```



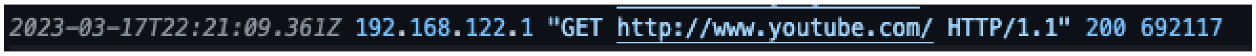
```
2023-03-17T22:16:58.468Z 192.168.122.1 "GET HTTP://www.google.com/ HTTP/1.1" 403 65
```

Sending Request To Server With Chunked-Encoding:

- ./myproxy 9089 ~/cse156/proxy/forbidden_sites.txt ~/cse156/proxy/access.log
- curl -v -x http://192.168.122.1:9089/ http://www.youtube.com/ -o out2
- curl -v https://www.youtube.com/ -o out1



```
-bash-4.2$ curl -v -x  http://192.168.122.1:9089/ http://www.youtube.com/ -o out2
* About to connect() to proxy 192.168.122.1 port 9089 (#0)
*   Trying 192.168.122.1...
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0* Connected to 192.168.122.1 (192.168.122.1) port 9089 (#0)
> GET http://www.youtube.com/ HTTP/1.1
> User-Agent: curl/7.29.0
> Host: www.youtube.com
> Accept: */*
> Proxy-Connection: Keep-Alive
>
< HTTP/1.1 200 OK
< Content-Type: text/html; charset=utf-8
< X-Content-Type-Options: nosniff
< Cache-Control: no-cache, no-store, max-age=0, must-revalidate
< Pragma: no-cache
< Expires: Mon, 01 Jan 1990 00:00:00 GMT
< Date: Fri, 17 Mar 2023 22:21:04 GMT
< Strict-Transport-Security: max-age=31536000
< X-Frame-Options: SAMEORIGIN
< Permissions-Policy: ch-ua-arch=*, ch-ua-bitness=*, ch-ua-full-version=*, ch-ua-full-version-list=*, ch-ua-model=*, ch-ua-wow64=*, ch-ua-platform=*, ch-ua-platform-version=*
< Report-To: {"group":"youtube_main","max_age":2592000,"endpoints":[{"url":"https://csp.withgoogle.com/csp/report-to/youtube_main"}]}
< Cross-Origin-Opener-Policy-Report-Only: same-origin-allow-popups; report-to="youtube_main"
< P3P: CP="This is not a P3P policy! See http://support.google.com/accounts/answer/151657?hl=en for more info."
< Server: ESF
< X-XSS-Protection: 0
< Set-Cookie: GPS=1; Domain=.youtube.com; Expires=Fri, 17-Mar-2023 22:51:04 GMT; Path=/; Secure; HttpOnly
< Set-Cookie: YSC=XA0QvgmbT9s; Domain=.youtube.com; Path=/; Secure; HttpOnly; SameSite=none
< Set-Cookie: VISITOR_INFO1_LIVE=NlpUb7F4swE; Domain=.youtube.com; Expires=Wed, 13-Sep-2023 22:21:04 GMT; Path=/; Secure; HttpOnly; SameSite=none
< Accept-Ranges: none
< Vary: Accept-Encoding
< Transfer-Encoding: chunked
<
{ [data not shown]
100  674k    0  674k    0     0  2681k      0 --:--:-- --:--:-- --:--:-- 2676k
* Connection #0 to host 192.168.122.1 left intact
-bash-4.2$
```



```
2023-03-17T22:21:09.361Z 192.168.122.1 "GET http://www.youtube.com/ HTTP/1.1" 200 692117
```