BÁO CÁO ĐỒ ÁN LẬP TRÌNH SOCKET – PROXY SERVER MẠNG MÁY TÍNH

Giáo viên hướng dẫn thực hành: Nguyễn Trung Quân

Giáo viên lý thuyết: Lê Giang Thanh

Nhóm 1712358_1712369_1712379

1. Phân công công việc

Họ tên	MSSV	Đảm nhận
Nguyễn Minh Đức	1712358	Phân tích cú pháp gói tin
Phạm Quốc Dũng	1712369	Cấu hình các kết nối
Đặng Thành Duy	1712379	Caching, blocking

2. Ngôn ngữ và môi trường lập trình

Ngôn ngữ: python 3

Editor: sublime text 3, pycharm

Interpreter: python 3.7.3

Tạo file .exe: pyinstaller

3. Ý tưởng và thực hiện

Ý tưởng:

⁂ Proxy server là server trung gian đứng ra chuyển giao dữ liệu qua lại giữa các node trên mạng. Các node ở đây, trong project này là các client muốn truy cập web và các web server đang sẵn sàng phục vụ. Dữ liệu ở đây là các gói tin http được gửi đi cùng với giao thức http. Giao thức http thực ra rất mềm dẻo, là giao thức có thể được lập trình để thay đổi, tùy theo sự đồng thuận giữa client và web server. Một client muốn kết nối với server thì đầu tiên sẽ tạo 1 kết nối TCP tới 1 port của server (ở đây là port 80). Sau đó sẽ gửi đi 1 gói tin request, server sẽ nhận và trả lời bằng 1 respond, server sẽ không gửi respond nếu client không gửi request (ít nhất là đối với http 1.0 và 1.1). Sau khi hoàn thành 1 lượt request-respond, có thể 2 bên sẽ đồng thuận ngắt kết nối, hoặc cùng giữ kết nối để gửi các lượt request-respond khác, điều kiện kết nối này có

thể tìm thấy trong header của gói tin, trong mục Connection:, nếu là keep-alive từ client tức client muốn giữ kết nối sau lượt request này, nếu là Connection: keep-alive từ server tức là server sẽ giữ kết nối sau respond, ta sẽ dựa vào yếu tố này để chạy proxy. Vậy điều proxy cần làm là, chấp nhận các kết nối từ client, nhận các request từ client, kết nối tới web server, gửi request tới server, nhận respond và gửi lại cho client. Các kết nối từ proxy tới client và server được duy trì hay phá bỏ tùy vào việc phân tích gói tin respond, liên tục kiểm tra kết nối đã bị hủy hay chưa.

⁂ Thân chương trình: Tạo socket cho proxy lắng nghe các kết nối ở port 8888. Chấp nhận các kết nối từ client, với mỗi kết nối ta tạo 1 thread xử lý riêng, từ đó sẽ phục vụ được nhiều kết nối cùng một lúc. Nhận các request của client, tìm trong request địa chỉ của web server, rồi tạo socket kết nối và gửi request tới web server, sau đó nhận lại phản hồi từ web server và gửi lại cho client. Trong các gói tin, phân tích và tìm điều kiện kết nối trong mục Connection:. Tùy thuộc vào điều kiện này mà chương trình quyết có giữ kết nối với client-server hay không, chương trình cũng phải luôn kiểm tra xem các bên client-server có tự ngắt kết nối hay chưa.

⁂ Hoạt động với blacklist: Khi khởi động chương trình, đọc các web server bị cấm từ file blacklist.conf vào biến BLACKLIST. Khi nhận request từ client nếu request gửi tới web server nằm trong BLACKLIST thì gửi 1 http message 403 về cho client và đóng kết nối.

⁂ Cơ chế caching: Dùng cơ chế max-age: trong cache của proxy có lưu các bản sao của các gói respond, mỗi gói đều có thời gian sống (max-age) riêng, mặc định là 300s, còn tùy thuộc vào thông tin của client và server. Khi nhận 1 request từ client, proxy sẽ kiểm tra trong cache để tìm bản sao của gói respond, rồi kiểm tra max-age của bản sao. Nếu bản sao còn mới → gửi lại cho client, nếu đã cũ → gửi request tới server, gửi lại cho client, sau đó lưu lại respond của server vào cache. Trong các gói respond sẽ có mục Cache-control:, ở mục này ta sẽ tìm thấy điều kiện để cache. Ta sẽ không cache nếu xuất hiện các điều kiện no-cache, no-store, private, max-age=0. Nếu xuất hiện max-age, ta sẽ chép thời gian sống vào biến max_age (nay max_age sẽ không còn là mặc định 300s nữa). Ngoài ra điều kiện cache còn xuất hiện trong gói request, thường là Cache-Control: max-age=0, ý muốn nói gói tin mà client nhận được đã cũ, client muốn 1 bản respond mới hoàn toàn, nên khi gặp điều kiện này ta phải lấy respond từ web server.

Pseudocode:

```
Khởi tạo socket cho proxy server tại localhost, port 8888
Lắng nghe các kết nối
Chấp nhận các kết nối
    Sau khi chấp nhận 1 kết nối, tạo 1 thread xử lý riêng
        khởi tạo các điều kiện nối, cache, thông tin về server, các điều kiện này sẽ thay đổi khi gọi các hàm bên dưới
        while kết-nối-còn-hiệu-lực:
            Nhận dữ liệu từ client
            Phân tích cú pháp, chích ra thông tin server, các thông tin về kết nối và caching
            Kiểm tra trong cache có gói tin yêu cầu và còn mới hay không
                Nếu có, gửi gói tin cho client, trở lại đầu vòng lặp
            Nếu là lần lặp đầu tiên -> Kết nối tới server, từ các lần tiếp theo không cần
            Gửi request cho server
            Nhận respond từ server
            Gửi respond cho client
            Phân tích gói respond, kiểm tra điều kiện kết nối, điều kiện cache
            Nếu điều kiện cache thỏa mãn, lưu lại bản sao của respond cùng với thời gian sống
```

Danh sách các hàm chính đã sử dụng:

Tên hàm	tham số	chức năng	trả về
socket()	addrerss family, kiểu kết nối	tạo socket	1 socket object
bind()	ip address, port	trói buộc kết nối	
listen()	số kết nối tối đa	bắt đầu lắng nghe các kết nối	
accept()	none	chấp nhận 1 kết nối	1 socket object và 1 tupple (ip,port)
recv()	số bytes dữ liệu tối đa trong 1 lần nhận	nhận dữ liệu	dữ liệu nhận được
send()	dữ liệu cần gửi	gửi dữ liệu	số bytes đã gửi
sendall()	dữ liệu cần gửi	gửi toàn bộ dữ liệu trong buffer	số bytes đã gửi
setblocking()	True or False	set block cho kết nối (hàm recv)	
connect()	tupple (ip,port) or (tên miền, port)	kết nối với socket ở địa chỉ (ip,port)	

close()		đóng kết nối	
start_new_thread()	hàm cần chạy riêng thread và các tham số của hàm	tạo riêng 1 thread cho hàm	
split()	chuỗi str or bytes	tách chuỗi tại các vị trí xuất hiện của tham số	chuỗi sau khi tách
encode()	kiểu encode	mã hóa thành kiểu bytes	dữ liệu sau khi mã hóa
decode()	kiểu decode	giải mã thành str	dữ liệu sau khi giải mã
find()	chuỗi str or bytes	tìm vị trí xuất hiện của tham số trong chuỗi ban đầu	vị trí xuất hiện , hoặc -1
append	phần tử cần thêm vào list	thêm 1 phần tử vào list	

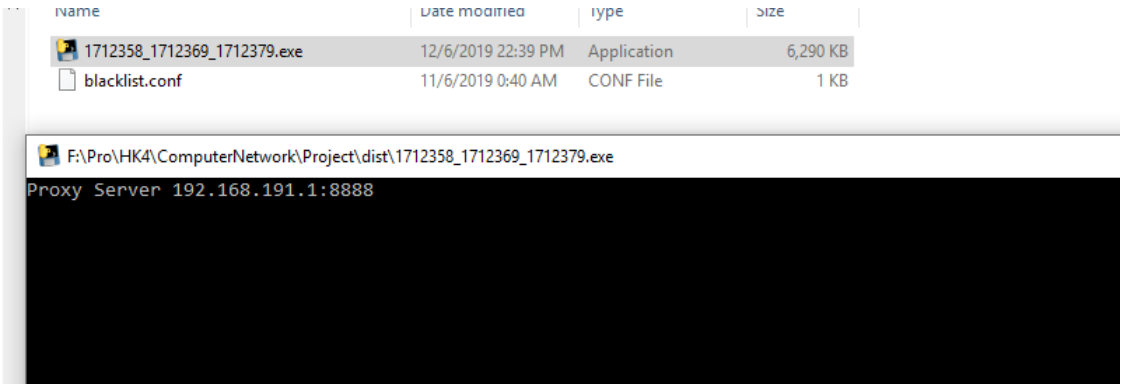
Danh sách các hàm chính tạo:

Tên hàm	tham số	chức năng	trả về
main()		nơi bắt đầu chương trình và tạo các thread	
proxy()	client_socket, client_address	chạy và giữ kết nối TCP giữa 1 client – 1 server	
fetch()	file name	tìm file trong cache	file tìm được, trạng thái cache, trạng thái kết nối
cache()	file name, respond, max-age, trạng thái kết nối	lưu respond vào trong cache	

blacklist()	client socket, client address, tên server, method của request	kiểm tra xem tên web có bị cấm không	True or False
parser_request()	tên server, port và gói request	phân tích gói request	tên server, port, method được dùng, thời gian sống của gói tin đã dùng
parser_respond()	gói respond, max-age	phân tích gói respond	trạng thái kết nối, thời gian sống của gói tin
my_recv()	socket object, timeout	nhận gói tin từ socket bằng phương thức đặc biệt, gói tin sẽ bị bỏ qua nếu thời gian chờ lớn hơn timeout	gói tin nhận được
my_send()	socket object, gói tin cần gửi	gửi gói tin	
create_proxy_socket()	host, port	tạo socket cho proxy	1 socket object
prepare()		tải các web bị cấm từ file blacklist.conf	

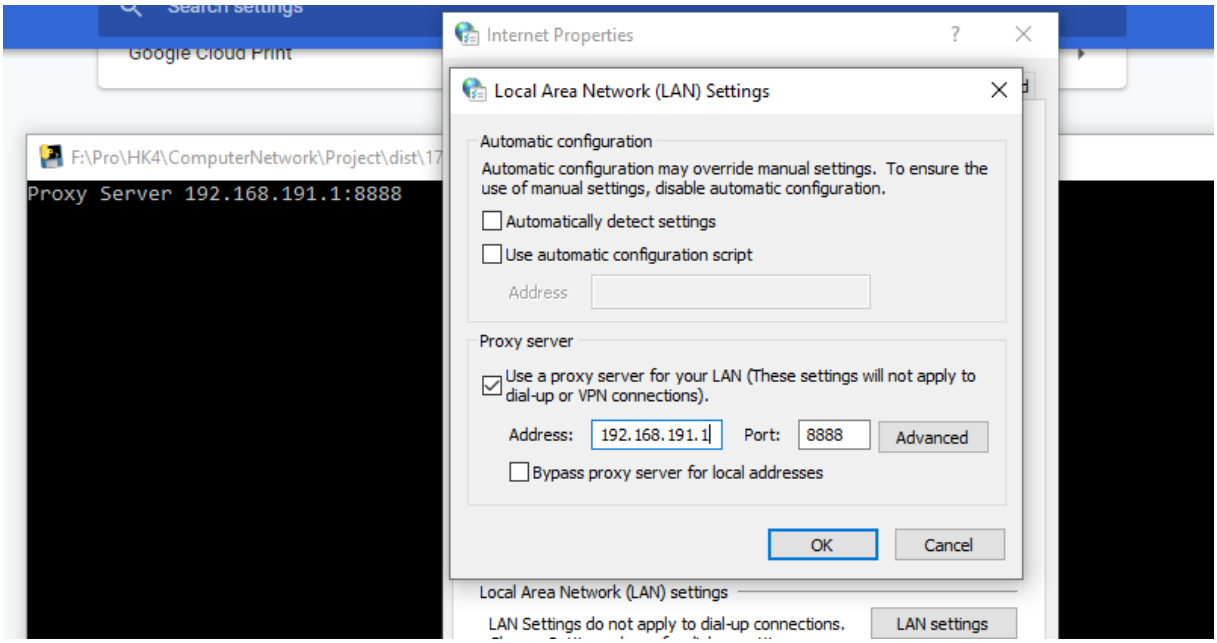
4. Cách chạy chương trình

Nhấp đúp chuột vào file .exe để thực thi, 1 cửa sổ console hiện lên như sau:



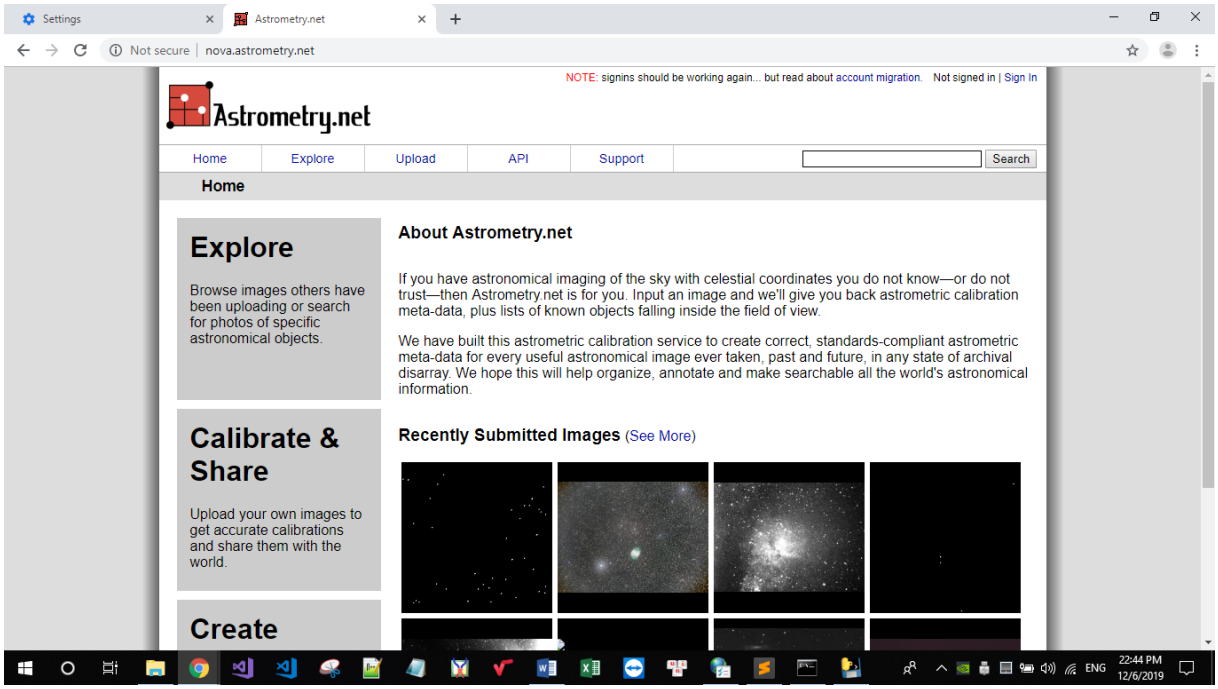
Đây là ip của proxy server, proxy này đang lắng nghe client trên port 8888.

Ta lấy địa chỉ này để cài đặt proxy cho chrome tại setting →advanced → proxy setting → LAN setting → OK

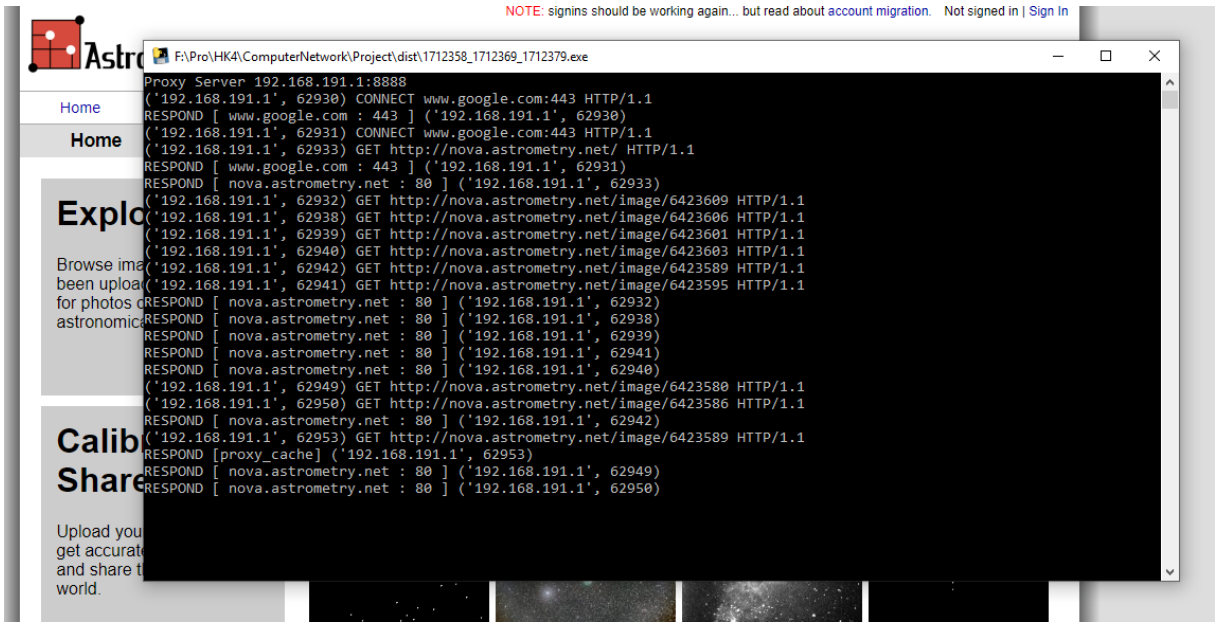


5. Mô tả các gói tin qua lại client – server

Thử vào 1 trang http:

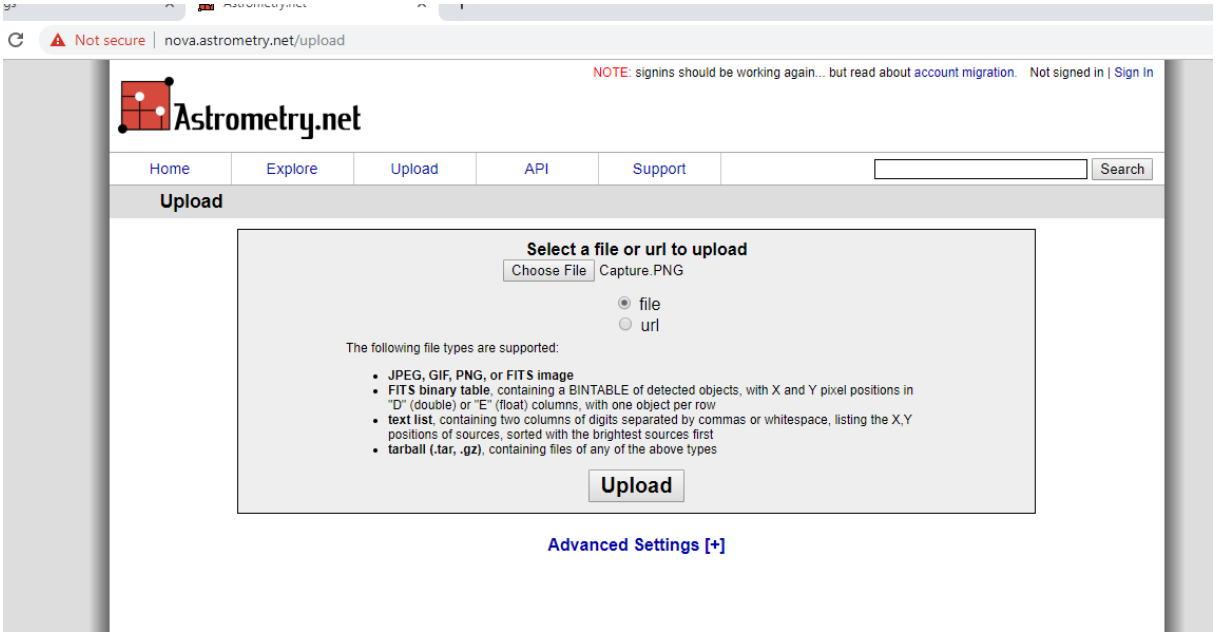


Web hiện lên nguyên vẹn, ta kiểm tra tại console của proxy thấy có các gói tin qua lại:

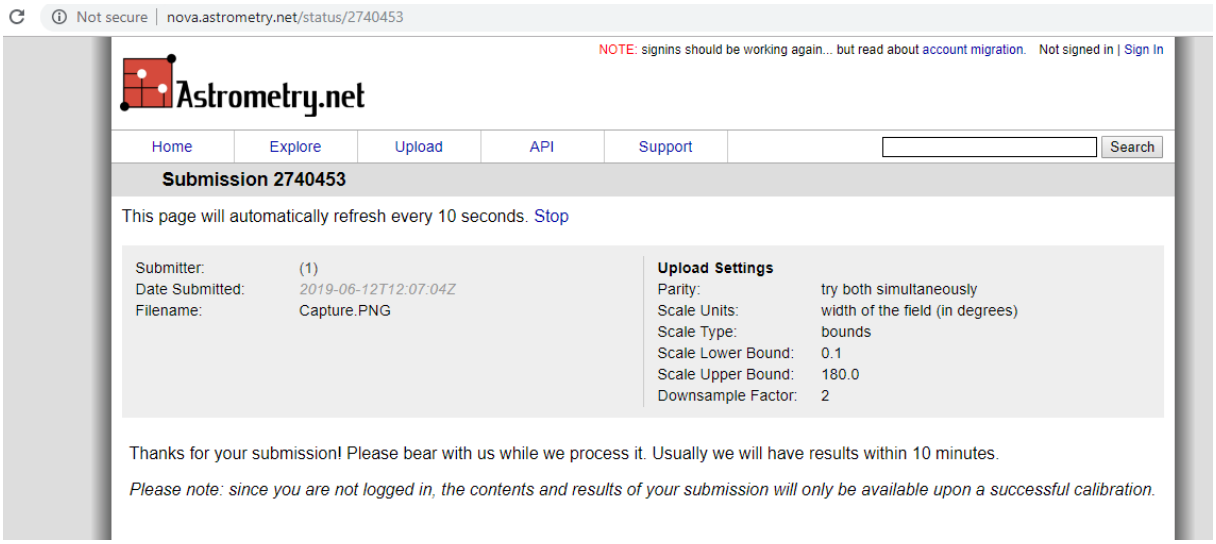


Ở đây ta thấy có các gói request với method GET từ clienn t, các gói RESPOND từ tên miền trang web gửi về.

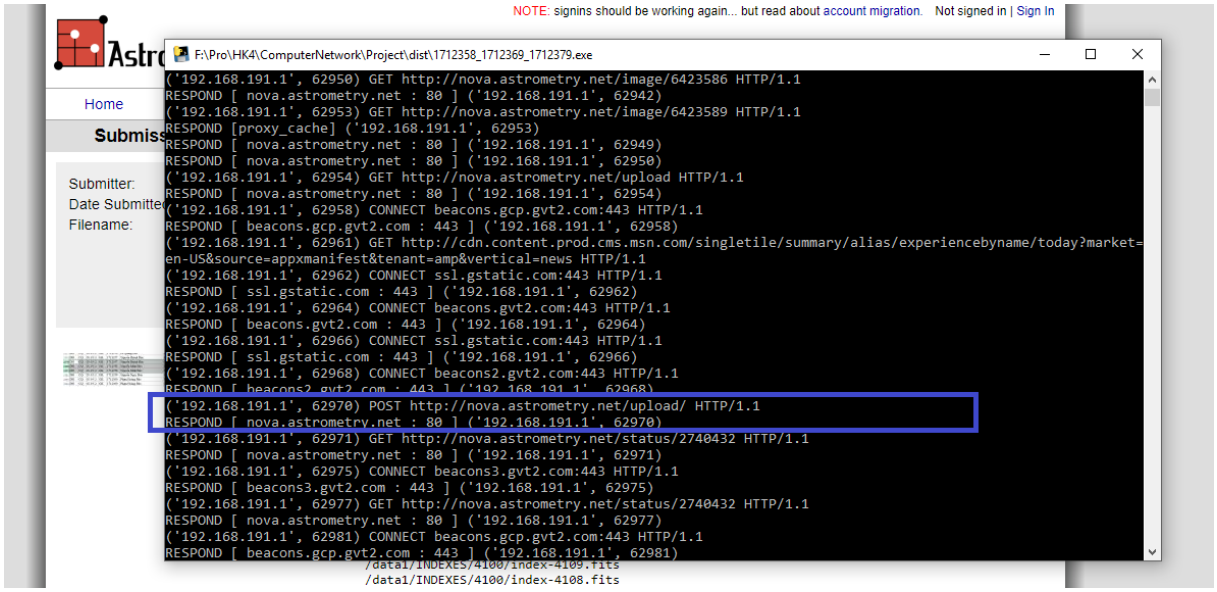
Ta thử test 1 chức khác, giao thức POST với phần upload 1 bức hình



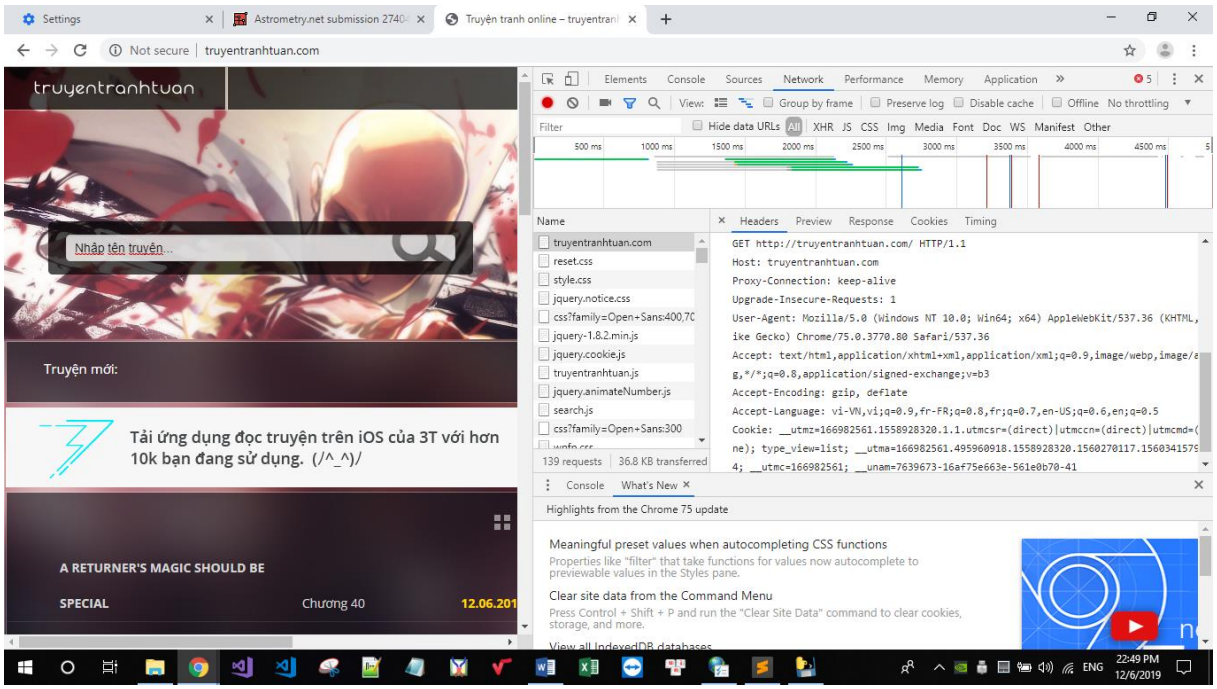
sau khi nhấn Upload, kết quả hiện lên như sau:



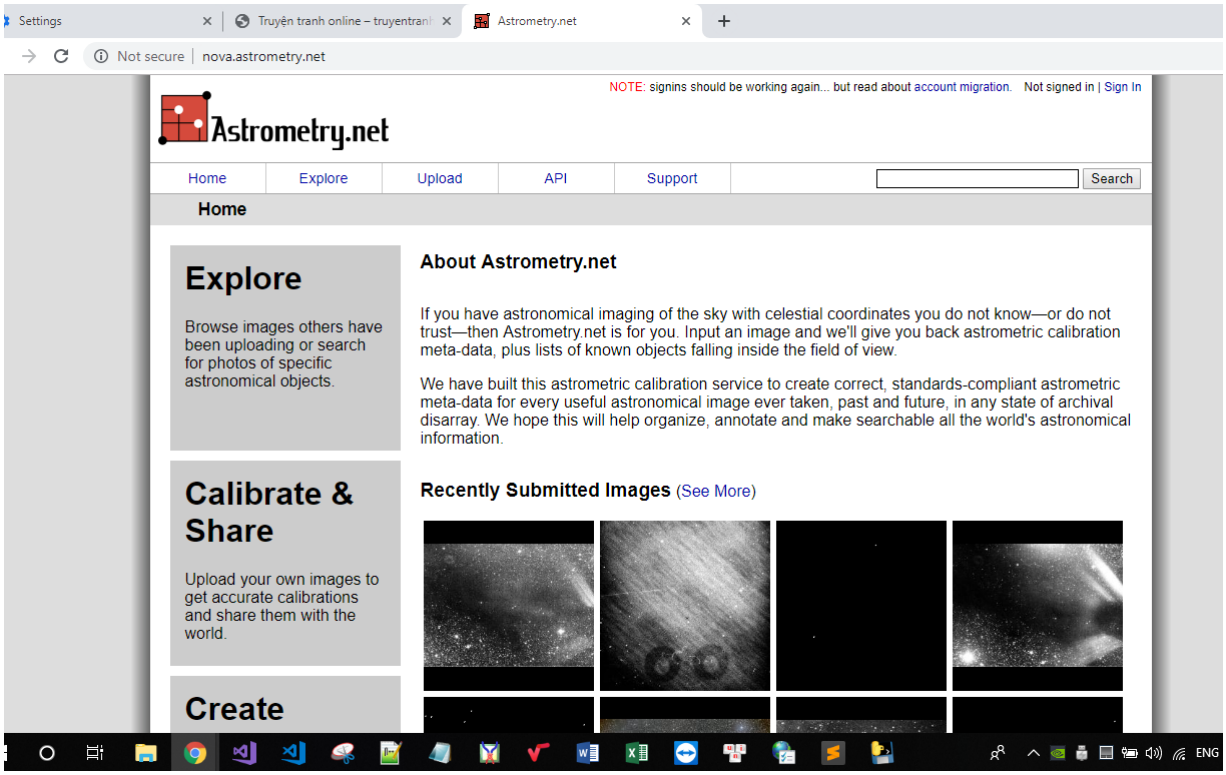
Như vậy ta đã post thành công, phần trang web này sẽ xử lý bức ảnh và gửi lại kết quả cho client sau mỗi 10s (tất nhiên web server không tự động gửi respond, mà do cứ sau 10s client sẽ tự gửi 1 request tới webserver). Khi nhìn vào console proxy, ta sẽ thấy phần gói tin POST này hiện ra cùng với RESPOND ngay bên dưới:



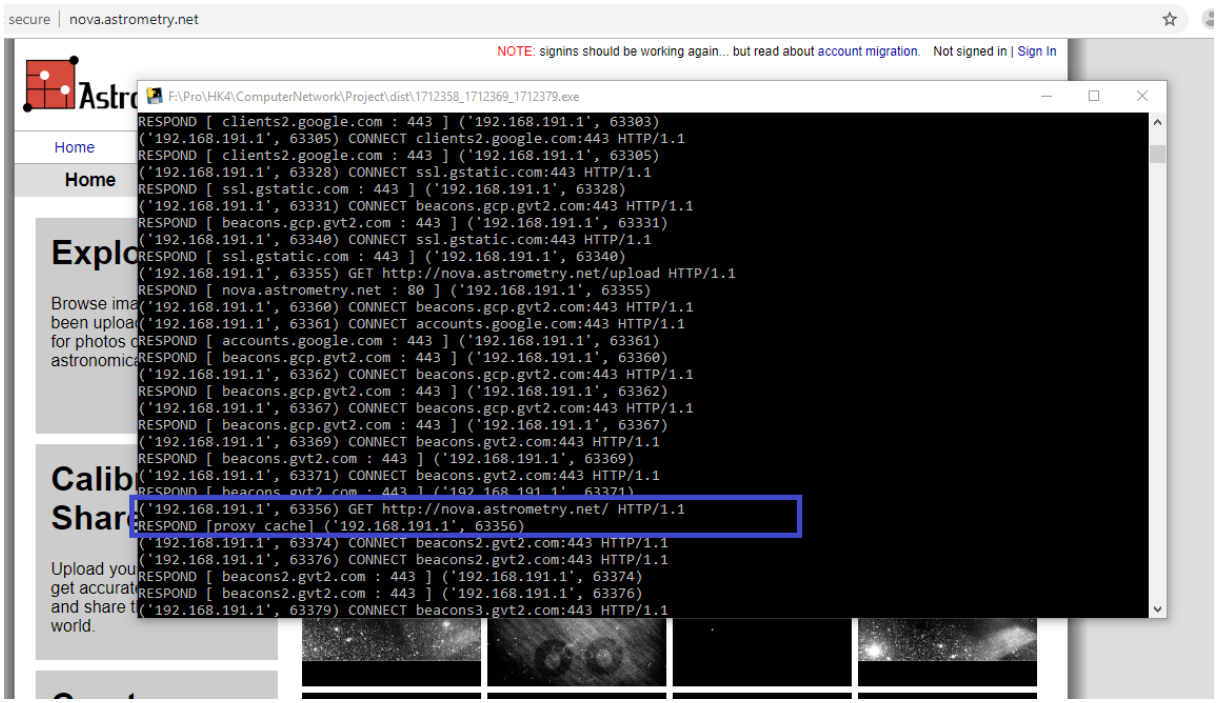
Thử vào 1 trang web khác:



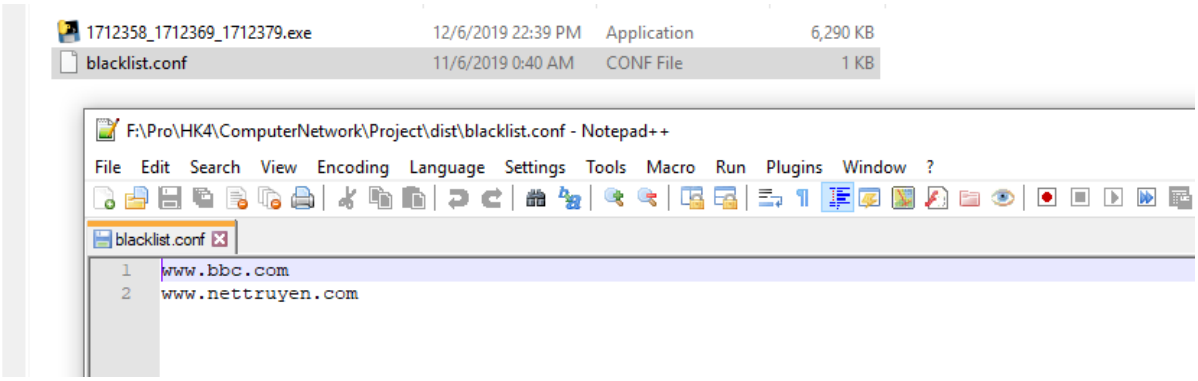
Bây giờ ta thử vào lại trang web đầu tiên xem thời gian vào web có nhanh hơn không (test caching):

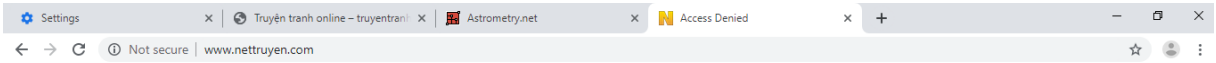


Quan sát trong console proxy, ta sẽ thấy proxy dùng bộ nhớ cache để trả lời request của client:



Giờ ta thử vào 1 trang web nằm trong blacklist:

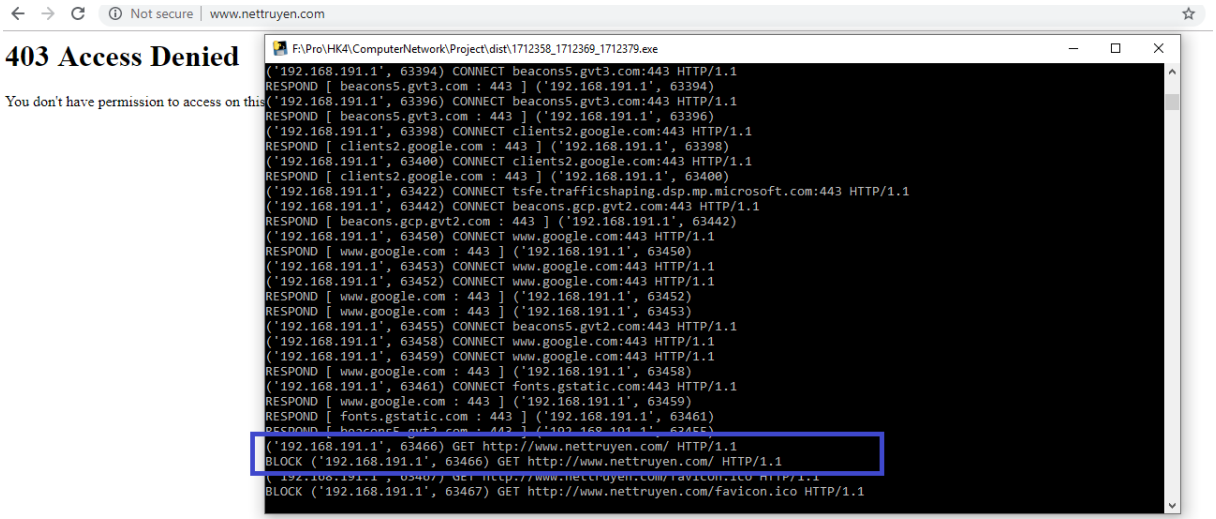




403 Access Denied

You don't have permission to access on this server.

Kết quả là nhận được 1 http 403, mở console proxy ta sẽ thấy gói tin được phản hồi bằng 1 BLOCK respond:



6. Proxy trong thực tế

Với các yêu cầu và bảo mật an ninh (cho công ty, trường học), yêu cầu về một bọ lọc thông tin giữa mạng nội bộ với bên ngoài, yêu cầu về cache để giảm giao thông trên mạng,... proxy trong thực tế rất được ưa dùng với nhiều chức năng và ý nghĩa khác nhau:

- Là một bộ lọc thông tin, cấm các web nguy hại, cấm xâm nhập khả nghi từ bên ngoài, dùng trong các trường học và công ty.

- Là 1 cache bộ nhớ hiệu quả, giúp giảm thời gian tải gói tin, hạn chế lưu thông trên mạng, dùng cho cả server và công sở. Caching trong thực tế là 1 quá trình rất phức tạp, các proxy đóng 1 vai trò không thể thiếu trong quá trình này.

- Công cụ để truy cập tới nhiều trang web bị cấm tại quốc gia, vùng lãnh thổ (các proxy public trên mạng).

- Ngăn chặn các hacker muốn xâm nhập tới 1 máy tính nào đó nằm bên kia 1 proxy.

7. Tham khảo

https://vi.wikipedia.org/wiki/M%C3%A1y_ch%E1%BB%A7_proxy

https://docs.python.org/2/library/socket.html

https://www.tutorialspoint.com/python/python_networking.htm

https://www.w3schools.com/whatis/whatis_http.asp

https://vi.wikipedia.org/wiki/Hypertext_Transfer_Protocol

https://en.wikipedia.org/wiki/List_of_HTTP_header_fields