

네트워크 HTTP 트래픽 측정 실습

Task 1: 네트워크에서의 HTTP 트래픽 측정

A)

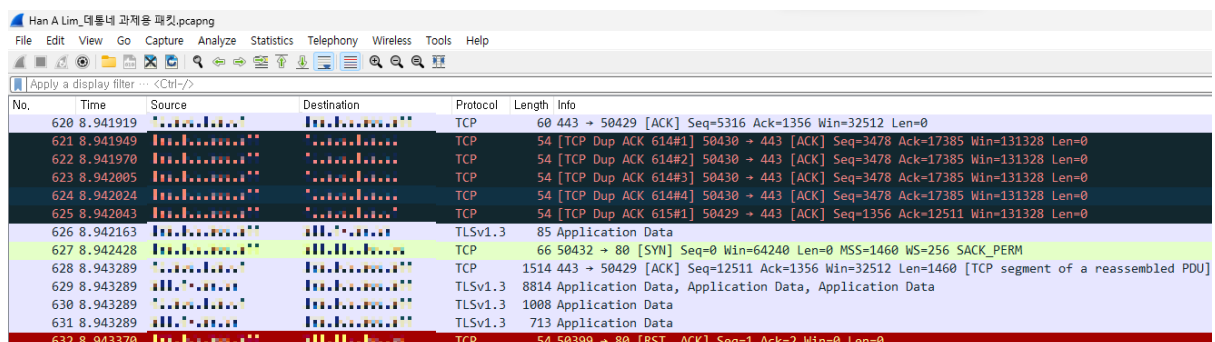
먼저 cmd 창을 연 후, ipconfig 명령어를 입력하여 나의 ip 주소를 알아내었다.

```
C:\Users\USER> ipconfig
```

이더넷 어댑터 이더넷:

```
연결별 DNS 접미사. . . . . : 
링크-로컬 IPv6 주소 . . . . . : fe80::4236:f3b7:a9f8:6f13%5
IPv4 주소 . . . . . : 
서브넷 마스크 . . . . . : 255.255.255.0
기본 게이트웨이 . . . . . :
```

이후 와이어샤크에 들어가 내 네트워크만을 대상으로 패킷 캡처를 수행하였다. 필터링 작업 전에는 TCP, HTTP, TLS, DNS 다양한 프로토콜을 사용하는 패킷들을 볼 수 있었다.



No.	Time	Source	Destination	Protocol	Length	Info
620	8.941919			TCP	60	443 → 50429 [ACK] Seq=5316 Ack=1356 Win=32512 Len=0
621	8.941949			TCP	54	[TCP Dup ACK 614#1] 50430 → 443 [ACK] Seq=3478 Ack=17385 Win=131328 Len=0
622	8.941970			TCP	54	[TCP Dup ACK 614#2] 50430 → 443 [ACK] Seq=3478 Ack=17385 Win=131328 Len=0
623	8.942005			TCP	54	[TCP Dup ACK 614#3] 50430 → 443 [ACK] Seq=3478 Ack=17385 Win=131328 Len=0
624	8.942024			TCP	54	[TCP Dup ACK 614#4] 50430 → 443 [ACK] Seq=3478 Ack=17385 Win=131328 Len=0
625	8.942043			TCP	54	[TCP Dup ACK 615#1] 50429 → 443 [ACK] Seq=1356 Ack=12511 Win=131328 Len=0
626	8.942163			TLSv1.3	85	Application Data
627	8.942428			TCP	66	50432 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
628	8.943289			TCP	1514	443 → 50429 [ACK] Seq=12511 Ack=1356 Win=32512 Len=1460 [TCP segment of a reassembled PDU]
629	8.943289			TLSv1.3	8814	Application Data, Application Data, Application Data
630	8.943289			TLSv1.3	1008	Application Data
631	8.943289			TLSv1.3	713	Application Data
632	8.943370			TCP	54	50399 → 80 [RST, ACK] Seq=1 Ack=2 Win=0 Len=0

하지만 상단에 'http' 명령어(대문자는 와이어샤크가 명령어로 인식을 못하는 것 같았다.)로 필터링 작업을 거치자, HTTP 프로토콜을 사용하는 패킷들만 골라 확인할 수 있었다.

Han A Lim_데몬네 과제용 패킷.pcapng

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

http

No.	Time	Source	Destination	Protocol	Length	Info
524	8.861375	HTTP	443	GET /jk?c=62&p=xgupBqX+qkDa197qsFSQvJRqsjRM3viZ+r5CtK_M2R4=&k=1 HTTP/1.1
526	8.871489	HTTP	415	HTTP/1.1 200 OK
642	8.949566	HTTP	467	GET /jk?c=62&p=xgupBqX+qkDa197qsFSQvJRqsjRM3viZ+r5CtK_M2R4=&k=1 HTTP/1.1
645	8.949936	HTTP	443	GET /jk?c=62&p=xgupBqX+qkDa197qsFSQvJRqsjRM3viZ+r5CtK_M2R4=&k=1 HTTP/1.1
656	8.958487	HTTP	415	HTTP/1.1 200 OK
670	8.976617	HTTP	415	HTTP/1.1 200 OK
1720	11.311112	HTTP	443	GET /jk?c=62&p=xgupBqX+qkDa197qsFSQvJRqsjRM3viZ+r5CtK_M2R4=&k=1 HTTP/1.1
1804	11.352424	HTTP	415	HTTP/1.1 200 OK
1824	11.370034	HTTP	443	GET /jk?c=62&p=xgupBqX+qkDa197qsFSQvJRqsjRM3viZ+r5CtK_M2R4=&k=1 HTTP/1.1
1829	11.381122	HTTP	415	HTTP/1.1 200 OK
2123	11.543009	HTTP	443	GET /jk?c=62&p=xgupBqX+qkDa197qsFSQvJRqsjRM3viZ+r5CtK_M2R4=&k=1 HTTP/1.1
2127	11.548513	HTTP	415	HTTP/1.1 200 OK
2191	11.603259	HTTP	443	GET /jk?c=62&p=xgupBqX+qkDa197qsFSQvJRqsjRM3viZ+r5CtK_M2R4=&k=1 HTTP/1.1
2207	11.611075	HTTP	415	HTTP/1.1 200 OK
2542	12.108156	HTTP	443	GET /jk?c=62&p=xgupBqX+qkDa197qsFSQvJRqsjRM3viZ+r5CtK_M2R4=&k=1 HTTP/1.1
2547	12.113596	HTTP	435	HTTP/1.1 200 OK
2742	12.905190	HTTP	467	GET /jk?c=62&p=xgupBqX+qkDa197qsFSQvJRqsjRM3viZ+r5CtK_M2R4=&k=1 HTTP/1.1
2743	12.909136	HTTP	415	HTTP/1.1 200 OK
2909	13.900897	HTTP	443	GET /jk?c=62&p=xgupBqX+qkDa197qsFSQvJRqsjRM3viZ+r5CtK_M2R4=&k=1 HTTP/1.1
2929	13.910797	HTTP	415	HTTP/1.1 200 OK
3169	14.143939	HTTP	443	GET /jk?c=62&p=xgupBqX+qkDa197qsFSQvJRqsjRM3viZ+r5CtK_M2R4=&k=1 HTTP/1.1
3173	14.145206	HTTP	443	GET /jk?c=62&p=xgupBqX+qkDa197qsFSQvJRqsjRM3viZ+r5CtK_M2R4=&k=1 HTTP/1.1
3192	14.160313	HTTP	415	HTTP/1.1 200 OK
3202	14.166254	HTTP	415	HTTP/1.1 200 OK
3222	14.175047	HTTP	443	GET /jk?c=62&p=xgupBqX+qkDa197qsFSQvJRqsjRM3viZ+r5CtK_M2R4=&k=1 HTTP/1.1
3245	14.196163	HTTP	415	HTTP/1.1 200 OK
4344	15.119580	HTTP	467	GET /jk?c=62&p=xgupBqX+qkDa197qsFSQvJRqsjRM3viZ+r5CtK_M2R4=&k=1 HTTP/1.1
4376	15.142756	HTTP	415	HTTP/1.1 200 OK

B)

다음으로 내가 접속해본 웹 브라우저의 HTTP 버전을 확인해 보았다. 사실 아래 사진처럼 육안으로도 확인이 가능하지만, 더욱 명확한 확인을 위해 아래와 같은 과정을 거쳤다.

HTTP 415 HTTP/1.1 200 OK

먼저 알아보고자 하는 패킷을 누르고, 마우스 우측 버튼을 클릭한다. 그럼 아래와 같은 선택창이 뜨는데, 여기서 Follow를 선택한다.

Mark/Unmark Packet(s)	Ctrl+M
Ignore/Unignore Packet(s)	Ctrl+D
Set/Unset Time Reference	Ctrl+T
Time Shift...	Ctrl+Shift+T
Packet Comments	
Edit Resolved Name	
Apply as Filter	
Prepare as Filter	
Conversation Filter	
Colorize Conversation	
SCTP	
Follow	
Copy	
Protocol Preferences	
Decode As...	
Show Packet in New Window	

Follow	TCP Stream	Ctrl+Alt+Shift+T
Copy	UDP Stream	Ctrl+Alt+Shift+U
Protocol Preferences	DCCP Stream	Ctrl+Alt+Shift+E
Decode As...	TLS Stream	Ctrl+Alt+Shift+S
Show Packet in New Window	HTTP Stream	Ctrl+Alt+Shift+H
	HTTP/2 Stream	

이후 화살표에 마우스를 얹어보면 TCP Stream이라는 선택지가 뜨는데, 이 옵션을 선택하면 아래와 같이 정돈된 새로운 창이 나타난다. 여기서 빨간 부분은 클라이언트가 서버에게 요청을 전송한 내용을 담은 부분이다. 반대로 파란 부분은 서버가 클라이언트에게 응답한 메시지의 내용을 담고 있다.

```
Wireshark - Follow TCP Stream (tcp.stream eq 39) - Han A Lim_데용네 과제용 캡처.pcapng

GET /jk?c=62&p=xgupBqX+qkDa197qsF5QvJRqsjRM3viZ+r5CtK_M2R4=&k=1 HTTP/1.1
Accept: */*
User-Agent: MeDCore
Connection: keep-alive
Pragma: no-cache
Cookie:
data=2EOUWnpB6My3sZn3bFZIP4Jn9H_gt5jmtZhEouwhf8=,zaeeftY_7kbwMbIf6AJsaCibScbXyqvVsGGKecSc__RydvGa86_Bxr8p1MBC8LnX,yFhd8Ia496fZCQM19F9MbtDIespPjQxm8VAr7EkYz
YU=,AeV1k8+QScYX12SseFYAvh975MB8QcakPoY3ex4S7RI=,
Host: gms.ahnlab.com

HTTP/1.1 200 OK
Server: nginx
Date: Sun, 02 Apr 2023 08:09:41 GMT
Content-Type: application/octet-stream
Content-Length: 135
Connection: keep-alive
Cache-Control: no-cache, no-store, must-revalidate
Pragma: no-cache

U7NrG5BVdGf01bVe0NxM_UkreQ32xZh+1kV5XGV2Ygo=,drp8yerVs0GxbRprHYncopP3Xewx_vI5vbtX8F7eqVSwNp1AP3vNnzt9Ba79D15PRtNMU7kYun24ucKe0JEsdg=,.GET /jk?
c=62&p=xgupBqX+qkDa197qsF5QvJRqsjRM3viZ+r5CtK_M2R4=&k=1 HTTP/1.1
Accept: */*
User-Agent: MeDCore
Connection: keep-alive
Pragma: no-cache
Cookie:
data=vbcSrZ0n77Au4sseEnxpbCqRbyRkKw8njyubPxtsFL9w=,sDpqocTGUTN10bchC1idVXTZy9h1vGvqPA+ibb5xZVvNlqGJpcDMBsN+0XBjcZKm,yFhd8Ia496fZCQM19F9MbtDIespPjQxm8VAr7EkYz
YU=,AeV1k8+QScYX12SseFYAvh975MB8QcakPoY3ex4S7RI=,
Host: gms.ahnlab.com

HTTP/1.1 200 OK
Server: nginx
Date: Sun, 02 Apr 2023 08:09:44 GMT
Content-Type: application/octet-stream
Content-Length: 135
Connection: keep-alive
Cache-Control: no-cache, no-store, must-revalidate
Pragma: no-cache

APu0qR7akVQJ3KCfWuUi_TCvocrvwm5fjgQs0WE3bbA4=,0VHvvJT60cC6Z2SIj6rKJUvWEiMzievw7b1dohq11n22Qo0TDy2vvg7AjLC2ZH2UvEbK63mfXj5x1qYTAX41mA=,.
```

여기서 두 번째 파란 부분을 집중해서 보면 HTTP 옆 1.1이라는 소수가 보이는데, 이 수가 바로 웹 브라우저가 사용하고 있는 HTTP 버전이다. 위 창을 통해 내가 일전에 접속하였던 웹 브라우저가 1.1 버전의 HTTP를 사용하고 있음을 알 수 있었다.

C)

나는 필터링할 HTTP status code로 200 OK 상태 코드를 선정하였다. 200 OK 상태 코드를 필터링하는 명령어인 “http.response.code==200”을 사용하였다. 이 명령어는 등호 뒤에 원하는 상태 코드 숫자를 삽입하는 방식을 통해 꼭 200 status code 뿐만 아니라 다양한 상태 코드들을 필터링할 수 있다. 위 명령어를 적용한 결과는 아래와 같았다.

Han A Lim_데통네 과제용 패킷.pcapng

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

http

No.	Time	Source	Destination	Protocol	Length	Info
524	8.861375			HTTP	443	GET /jk?c=62&p=xgupBqX+qkDa197qsFSQvJRqsjRM3viZ+r5CtK_M2R4=&k=1 HTTP/1.1
526	8.871489			HTTP	415	HTTP/1.1 200 OK
642	8.949566			HTTP	467	GET /jk?c=62&p=xgupBqX+qkDa197qsFSQvJRqsjRM3viZ+r5CtK_M2R4=&k=1 HTTP/1.1
645	8.949936			HTTP	443	GET /jk?c=62&p=xgupBqX+qkDa197qsFSQvJRqsjRM3viZ+r5CtK_M2R4=&k=1 HTTP/1.1
656	8.958487			HTTP	415	HTTP/1.1 200 OK
670	8.976617			HTTP	415	HTTP/1.1 200 OK
1720	11.311112			HTTP	443	GET /jk?c=62&p=xgupBqX+qkDa197qsFSQvJRqsjRM3viZ+r5CtK_M2R4=&k=1 HTTP/1.1
1804	11.352424			HTTP	415	HTTP/1.1 200 OK
1824	11.370034			HTTP	443	GET /jk?c=62&p=xgupBqX+qkDa197qsFSQvJRqsjRM3viZ+r5CtK_M2R4=&k=1 HTTP/1.1
1829	11.381122			HTTP	415	HTTP/1.1 200 OK
2123	11.543009			HTTP	443	GET /jk?c=62&p=xgupBqX+qkDa197qsFSQvJRqsjRM3viZ+r5CtK_M2R4=&k=1 HTTP/1.1
2127	11.548513			HTTP	415	HTTP/1.1 200 OK
2191	11.603259			HTTP	443	GET /jk?c=62&p=xgupBqX+qkDa197qsFSQvJRqsjRM3viZ+r5CtK_M2R4=&k=1 HTTP/1.1
2207	11.611075			HTTP	415	HTTP/1.1 200 OK
2542	12.108156			HTTP	443	GET /jk?c=62&p=xgupBqX+qkDa197qsFSQvJRqsjRM3viZ+r5CtK_M2R4=&k=1 HTTP/1.1
2547	12.113596			HTTP	435	HTTP/1.1 200 OK
2742	12.905190			HTTP	467	GET /jk?c=62&p=xgupBqX+qkDa197qsFSQvJRqsjRM3viZ+r5CtK_M2R4=&k=1 HTTP/1.1
2743	12.909136			HTTP	415	HTTP/1.1 200 OK
2909	13.900897			HTTP	443	GET /jk?c=62&p=xgupBqX+qkDa197qsFSQvJRqsjRM3viZ+r5CtK_M2R4=&k=1 HTTP/1.1
2929	13.910797			HTTP	415	HTTP/1.1 200 OK
3169	14.143939			HTTP	443	GET /jk?c=62&p=xgupBqX+qkDa197qsFSQvJRqsjRM3viZ+r5CtK_M2R4=&k=1 HTTP/1.1
3173	14.145206			HTTP	443	GET /jk?c=62&p=xgupBqX+qkDa197qsFSQvJRqsjRM3viZ+r5CtK_M2R4=&k=1 HTTP/1.1
3192	14.160313			HTTP	415	HTTP/1.1 200 OK
3202	14.166254			HTTP	415	HTTP/1.1 200 OK
3222	14.175047			HTTP	443	GET /jk?c=62&p=xgupBqX+qkDa197qsFSQvJRqsjRM3viZ+r5CtK_M2R4=&k=1 HTTP/1.1
3245	14.196163			HTTP	415	HTTP/1.1 200 OK
4344	15.119580			HTTP	467	GET /jk?c=62&p=xgupBqX+qkDa197qsFSQvJRqsjRM3viZ+r5CtK_M2R4=&k=1 HTTP/1.1
4376	15.142756			HTTP	415	HTTP/1.1 200 OK

http.response.code==200 적용 전

Han A Lim_데통네 과제용 패킷.pcapng

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

http.response.code==200

No.	Time	Source	Destination	Protocol	Length	Info
526	8.871489			HTTP	415	HTTP/1.1 200 OK
656	8.958487			HTTP	415	HTTP/1.1 200 OK
670	8.976617			HTTP	415	HTTP/1.1 200 OK
1804	11.352424			HTTP	415	HTTP/1.1 200 OK
1829	11.381122			HTTP	415	HTTP/1.1 200 OK
2127	11.548513			HTTP	415	HTTP/1.1 200 OK
2207	11.611075			HTTP	415	HTTP/1.1 200 OK
2547	12.113596			HTTP	435	HTTP/1.1 200 OK
2743	12.909136			HTTP	415	HTTP/1.1 200 OK
2929	13.910797			HTTP	415	HTTP/1.1 200 OK
3192	14.160313			HTTP	415	HTTP/1.1 200 OK
3202	14.166254			HTTP	415	HTTP/1.1 200 OK
3245	14.196163			HTTP	415	HTTP/1.1 200 OK
4376	15.142756			HTTP	415	HTTP/1.1 200 OK

http.response.code==200 적용 후

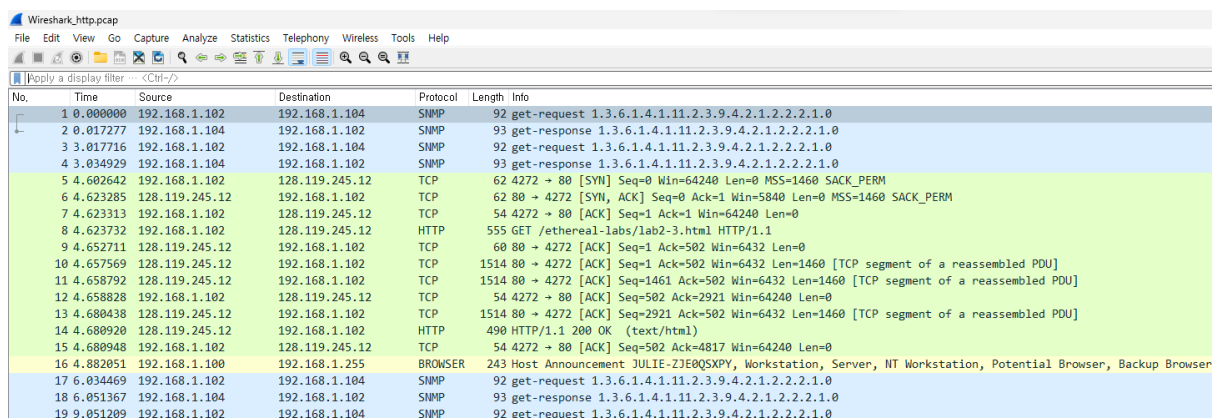
HTTP status code의 경우 필자는 200 OK 코드밖에 확보하지 못하였다. 하지만 404나 500 등 다른 유명한 status code에 대한 내용도 함께 표로 정리해 보았다. 정리한 내용은 아래와 같다.

상태 코드	의미	상세
200	OK	클라이언트의 요청이 성공적으로 처리되었음을 의미함
301	Moved Permanently	요청한 리소스가 다른 URL로 특정 이유 때문에 옮겨졌음을 의미함
404	Not Found	클라이언트가 요청한 리소스를 찾을 수 없음 의미함
500	Internal Server Error	서버에서 오류가 발생하여 클라이언트의 요청을 처리할 수 없음을 의미함

Task 2: 데이터 내부 패킷의 의미 분석

A)

먼저 Wireshark_http.pcapng 파일을 열어보았는데, 아래와 같이 TCP, BROWSER, SNMP 등의 다양한 프로토콜들을 확인할 수 있었다. 여기서 화면 중앙의 책갈피 표지 옆 'tcp' 명령어를 입력하여 필터링 작업을 수행하였다.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.1.102	192.168.1.104	SNMP	92	get-request 1.3.6.1.4.1.11.2.3.9.4.2.1.2.2.1.0
2	0.017277	192.168.1.104	192.168.1.102	SNMP	93	get-response 1.3.6.1.4.1.11.2.3.9.4.2.1.2.2.1.0
3	0.017716	192.168.1.102	192.168.1.104	SNMP	92	get-request 1.3.6.1.4.1.11.2.3.9.4.2.1.2.2.1.0
4	0.034929	192.168.1.104	192.168.1.102	SNMP	93	get-response 1.3.6.1.4.1.11.2.3.9.4.2.1.2.2.1.0
5	4.602642	192.168.1.102	128.119.245.12	TCP	62	4272 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM
6	4.623285	128.119.245.12	192.168.1.102	TCP	62	80 → 4272 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460 SACK_PERM
7	4.623313	192.168.1.102	128.119.245.12	TCP	54	4272 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
8	4.623732	192.168.1.102	128.119.245.12	HTTP	555	GET /etherreal-labs/lab2-3.html HTTP/1.1
9	4.652711	128.119.245.12	192.168.1.102	TCP	60	80 → 4272 [ACK] Seq=1 Ack=502 Win=6432 Len=0
10	4.657569	128.119.245.12	192.168.1.102	TCP	1514	80 → 4272 [ACK] Seq=1 Ack=502 Win=6432 Len=1460 [TCP segment of a reassembled PDU]
11	4.658792	128.119.245.12	192.168.1.102	TCP	1514	80 → 4272 [ACK] Seq=1461 Ack=502 Win=6432 Len=1460 [TCP segment of a reassembled PDU]
12	4.658828	192.168.1.102	128.119.245.12	TCP	54	4272 → 80 [ACK] Seq=502 Ack=2921 Win=64240 Len=0
13	4.680438	128.119.245.12	192.168.1.102	TCP	1514	80 → 4272 [ACK] Seq=2921 Ack=502 Win=6432 Len=1460 [TCP segment of a reassembled PDU]
14	4.680920	128.119.245.12	192.168.1.102	HTTP	490	HTTP/1.1 200 OK (text/html)
15	4.680948	192.168.1.102	128.119.245.12	TCP	54	4272 → 80 [ACK] Seq=502 Ack=4817 Win=64240 Len=0
16	4.882051	192.168.1.100	192.168.1.255	BROWSER	243	Host Announcement JULIE-ZJE0QXPY, Workstation, Server, NT Workstation, Potential Browser, Backup Browser
17	6.034469	192.168.1.102	192.168.1.104	SNMP	92	get-request 1.3.6.1.4.1.11.2.3.9.4.2.1.2.2.1.0
18	6.051367	192.168.1.104	192.168.1.102	SNMP	93	get-response 1.3.6.1.4.1.11.2.3.9.4.2.1.2.2.1.0
19	9.051209	192.168.1.102	192.168.1.104	SNMP	92	get-request 1.3.6.1.4.1.11.2.3.9.4.2.1.2.2.1.0

필터링 전

No.	Time	Source	Destination	Protocol	Length	Info
5	4.602642	192.168.1.102	128.119.245.12	TCP	62	4272 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM
6	4.623285	128.119.245.12	192.168.1.102	TCP	62	80 → 4272 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460 SACK_PERM
7	4.623313	192.168.1.102	128.119.245.12	TCP	54	4272 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
8	4.623732	192.168.1.102	128.119.245.12	HTTP	555	GET /ethereal-labs/lab2-3.html HTTP/1.1
9	4.652711	128.119.245.12	192.168.1.102	TCP	60	80 → 4272 [ACK] Seq=1 Ack=502 Win=6432 Len=0
10	4.657569	128.119.245.12	192.168.1.102	TCP	1514	80 → 4272 [ACK] Seq=1 Ack=502 Win=6432 Len=1460 [TCP segment of a reassembled PDU]
11	4.658792	128.119.245.12	192.168.1.102	TCP	1514	80 → 4272 [ACK] Seq=1461 Ack=502 Win=6432 Len=1460 [TCP segment of a reassembled PDU]
12	4.658828	192.168.1.102	128.119.245.12	TCP	54	4272 → 80 [ACK] Seq=502 Ack=2921 Win=64240 Len=0
13	4.680438	128.119.245.12	192.168.1.102	TCP	1514	80 → 4272 [ACK] Seq=2921 Ack=502 Win=6432 Len=1460 [TCP segment of a reassembled PDU]
14	4.680920	128.119.245.12	192.168.1.102	HTTP	490	HTTP/1.1 200 OK (text/html)
15	4.680948	192.168.1.102	128.119.245.12	TCP	54	4272 → 80 [ACK] Seq=502 Ack=4817 Win=64240 Len=0

필터링 후

여기서 tcp로 필터링했음에도 HTTP가 남아있는 것을 확인할 수 있었는데, 이는 HTTP 프로토콜은 전송 계층에서 사용되는 TCP 프로토콜을 기반으로 하기 때문이다. HTTP 프로토콜을 사용하는 패킷은 모두 TCP 패킷으로 포장되어 전송되기에, TCP 프로토콜을 필터링하면 HTTP를 포함한 모든 TCP 패킷을 볼 수 있게 되는 것이다.

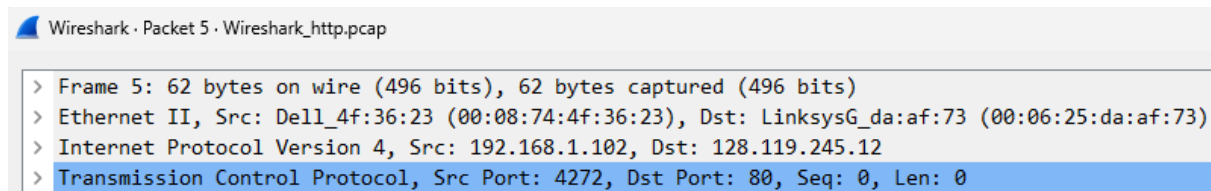
B)

다음으로는 HTTP 프로토콜과 GET 매소드를 가지는 패킷을 찾아보았다. 효율적인 탐색을 위해 `http.request.method == GET` 명령어를 사용하였다. 이 명령어는 앞에서 서술했듯 GET 메소드를 가지면서 HTTP 프로토콜을 사용하는 패킷만 필터링해서 보여주는 명령어이다. 탐색 결과, 8번 패킷이 조건에 부합하였다.

No.	Time	Source	Destination	Protocol	Length	Info
8	4.623732	192.168.1.102	128.119.245.12	HTTP	555	GET /ethereal-labs/lab2-3.html HTTP/1.1

Wireshark_http.pcapng 파일에 담긴 여러 패킷들을 통해 Server & Client의 포트 번호와 ip를 알아낼 수 있다. 여기서 패킷의 개수가 많아 혼동이 올 수 있는데, SYN 패킷의 상세만 보면 쉽게 클라이언트와 서버의 ip, 포트번호를 알아낼 수 있다. SYN은 클라이언트가 서버에게 TCP 연결을 요청할 때 전송하는 패킷으로, Destination(서버)과 소스(클라이언트)값을 통해 원하는 값들을 확인할 수 있었다.

5	4.602642	192.168.1.102	128.119.245.12	TCP	62	4272 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM
---	----------	---------------	----------------	-----	----	--

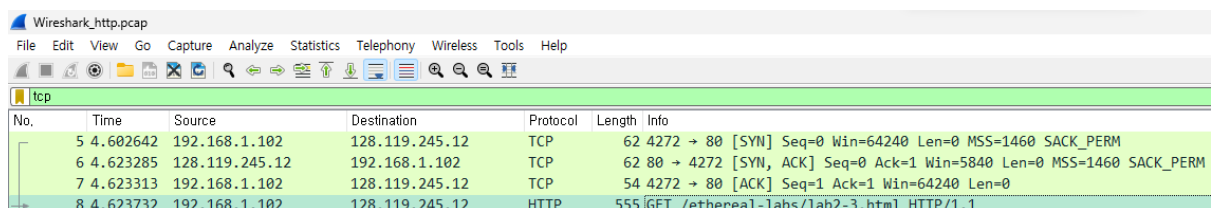
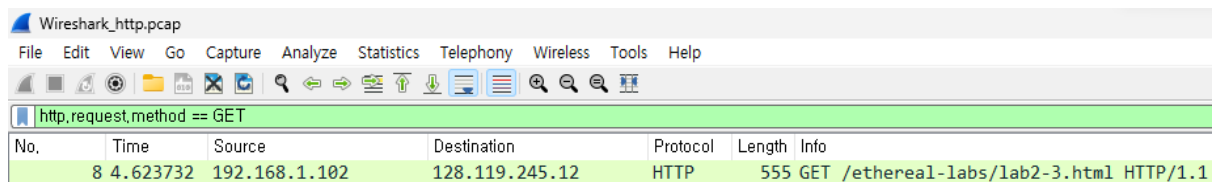


5번 패킷을 통해 알아낸 클라이언트, 서버의 포트번호와 ip 주소는 아래와 같다.

	서버	클라이언트
포트번호	80	4272
ip	128.119.245.12	192.168.1.102

C)

앞선 탐색으로 A 패킷은 8번 패킷임을 확인할 수 있었다. tcp 필터링 상태를 유지한 채로 패킷 목록을 보면 7, 6, 5번 패킷들이 8번 앞에 위치해있음을 확인할 수 있다.



위 패킷들을 자세히 살펴보니, 일전에 언급되었던 3 Way Handshake 와 관련이 있었다. 3 Way Handshake는 TCP/IP 프로토콜에서 사용되는 핵심 네트워크 연결 설정 방식 중 하나로, 클라이언트와 서버 간에 연결 설정 과정에서 총 3번의 메시지 교환을 통해 연결을 설정한다는 특징이 있다. 위 방식에서 핵심이 되는 3가지 패킷이 있는데, 바로 SYN, SYN + ACK, ACK가 바로 그것들이다. 이 3가지 패킷들에 대한 요약은 아래와 같다.

	SYN	SYN + ACK	ACK
전송	클라이언트	서버	클라이언트

수신	서버	클라이언트	서버
설명	클라이언트가 자신의 연결 준비 상태와 함께 서버에 TCP 연결 요청	서버가 클라이언트의 요청에 응답하며 자신의 준비상태에 대한 정보 전송	클라이언트가 서버가 보냈던 SYN 패킷(= 연결준비 상태 정보 패킷)의 수신 여부 전송

위 사진에서 각 패킷들의 info를 보면 5번 패킷이 SYN, 6번은 SYN + ACK, 7번이 ACK 패킷임을 확인할 수 있다. 먼저 클라이언트가 서버로 SYN 메시지(클라이언트 포트 번호 & 초기 시퀀스 번호 포함)를 5번 패킷을 통해 보냈음을 확인할 수 있다.

```
5 4.602642 192.168.1.102 128.119.245.12 TCP 62 4272 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM
```

이후 서버는 클라이언트의 SYN 메시지를 받고, 클라이언트가 사용할 포트 번호를 복제한 후 자신이 사용할 포트 번호와 초기 시퀀스 번호를 SYN+ACK 메시지에 담아 다시 클라이언트에게 6번 패킷을 이용해 보냈음을 알 수 있다.

```
6 4.623285 128.119.245.12 192.168.1.102 TCP 62 80 → 4272 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460 SACK_PERM
```

클라이언트는 서버가 전송한 SYN+ACK 메시지를 받고, 서버가 사용할 포트 번호와 초기 시퀀스 번호를 복제하였을 것이다. 그리고 자신이 사용할 포트 번호와 서버의 초기 시퀀스 번호를 ACK 메시지에 담아 다시 서버에게 전송하였음을 확인할 수 있다(7번 패킷).

```
7 4.623313 192.168.1.102 128.119.245.12 TCP 54 4272 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
```

이를 통해 패킷들을 주고받으며 클라이언트와 서버는 서로 상대방이 정상 및 생존 여부를 확인하고, 데이터를 안정적으로 전송하기 위한 초기화 작업을 수행하였음을 확인할 수 있다.

D)

A 패킷(= 8번 패킷) 뒤에는 9번부터 14번까지 총 6개의 패킷이 존재하고, 각 패킷은 상이한 의미를 가지고 있다.

```
9 4.652711 128.119.245.12 192.168.1.102 TCP 60 80 → 4272 [ACK] Seq=1 Ack=502 Win=6432 Len=0
10 4.657569 128.119.245.12 192.168.1.102 TCP 1514 80 → 4272 [ACK] Seq=1 Ack=502 Win=6432 Len=1460 [TCP segment of a reassembled PDU]
11 4.658792 128.119.245.12 192.168.1.102 TCP 1514 80 → 4272 [ACK] Seq=1461 Ack=502 Win=6432 Len=1460 [TCP segment of a reassembled PDU]
12 4.658828 192.168.1.102 128.119.245.12 TCP 54 4272 → 80 [ACK] Seq=502 Ack=2921 Win=64240 Len=0
13 4.680438 128.119.245.12 192.168.1.102 TCP 1514 80 → 4272 [ACK] Seq=2921 Ack=502 Win=6432 Len=1460 [TCP segment of a reassembled PDU]
14 4.680920 128.119.245.12 192.168.1.102 HTTP 490 HTTP/1.1 200 OK (text/html)
```


먼저 9번 패킷은 TCP ACK 패킷이자 이전에 전송된 데이터를 성공적으로 수신하였음을 알리기 위한 응답 패킷으로, 80번 포트에서 4272번 포트에 전송되었음을 알 수 있다.

“Seq=1, Ack=502, Win=6432, Len=0”이라는 정보를 통해 이전에 전송한 데이터의 첫 번째 바이트 번호가 1이고, 수신 측에서는 이전에 전송된 데이터를 잘 받았으며, 다음에 받을 데이터의 첫 번째 바이트 번호가 502임을 알 수 있다. 또 수신 측에서 전송 가능한 데이터의 양은 6432이며, 전송되는 데이터의 길이는 0임을 파악할 수 있다. 따라서 9번 패킷은 4272번 포트로부터 수신된 데이터에 대한 응답을 나타내고 있고, 이 데이터는 이 패킷 이전에 전송된 것임을 알 수 있다.

```
9 4.652711 128.119.245.12 192.168.1.102 TCP 60 80 → 4272 [ACK] Seq=1 Ack=502 Win=6432 Len=0
```

10번 패킷은 TCP ACK 패킷으로, 80번 포트에서 4272번 포트에 전송되었다. Seq=1, Ack=502, Win=6432, Len=1460이라는 정보를 통해 1460바이트의 데이터를 전송하고, 이전에 전송된 9번 패킷에 대한 ACK(승인)을 포함하고 있음을 알 수 있다. 10번 패킷은 TCP 프로토콜의 특징 중 하나인 데이터 분할 전송을 보여주는 예시라고 볼 수 있으며, 데이터를 안정적으로 전송하기 위한 중간 과정에 해당한다.

```
10 4.657569 128.119.245.12 192.168.1.102 TCP 1514 80 → 4272 [ACK] Seq=1 Ack=502 Win=6432 Len=1460 [TCP segment of a reassembled PDU]
```

11번 패킷은 이전에 전송된 10번 패킷의 뒷부분인 TCP segment of a reassembled PDU를 전송하는 TCP ACK 패킷이다. 11번 패킷은 128.119.245.12에서 192.168.1.102로 전송되며, Seq는 1461, Ack는 502, Win은 6432, Len은 1460으로 설정되어 있다. 10번 패킷이 전송한 데이터가 커 데이터는 분할되어 전송되는데, 이때 11번 패킷이 이전에 전송된 10번 패킷의 뒷부분에 해당하는 데이터를 전송하여 10번 패킷이 전송한 데이터를 완전히 재조립하는 역할을 수행한다.

```
11 4.658792 128.119.245.12 192.168.1.102 TCP 1514 80 → 4272 [ACK] Seq=1461 Ack=502 Win=6432 Len=1460 [TCP segment of a reassembled PDU]
```

12번 패킷은 TCP ACK 패킷으로, 이 패킷은 4272번 포트에서 80번 포트에 전송되었다. 이 패킷은 Seq=502, Ack=2921, Win=64240, Len=0이라는 정보가 포함되어 있으며, 이 패킷은 80번 포트로부터 수신된 데이터에 대한 응답을 나타내는 역할과 4272 포트에서 전송된 데이터를 성공적으로 수신하였음을 알리는 역할을 수행한다. 추가적으로 이전 패킷(11번 패킷)에서 80번 포트에서 전송된 데이터 중 일부를 재조립하였기 때문에, 4272 포트에 ACK를 보내는 작업을 진행한다.

```
12 4.658828 192.168.1.102 128.119.245.12 TCP 54 4272 → 80 [ACK] Seq=502 Ack=2921 Win=64240 Len=0
```

13번 패킷은 128.119.245.12에서 192.168.1.102로 전송된 TCP 데이터 패킷이다. 이 패킷은 이전에 4272 포트에서 재조립된 PDU의 일부를 나타낸다. Seq 값은 2921이고, Ack 값은 502이며, 이전에 전송된 데이터를 수신하였음을 나타낸다. 13번 패킷은 데이터를 전송하기 위해 여러 개의 패킷으로 분할된 상태일 경우 마지막 패킷에 대한 응답이며, 이 패킷 이전에 전송된 데이터의 마지막 부분에 해당한다는 특징이 있다.

```
13 4.680438 128.119.245.12 192.168.1.102 TCP 1514 80 → 4272 [ACK] Seq=2921 Ack=502 Win=6432 Len=1460 [TCP segment of a reassembled PDU]
```

14번 패킷은 128.119.245.12에서 192.168.1.102로 전송된 HTTP 응답 패킷이다. 이 패킷은 이전에 80 포트에서 전송된 데이터를 포함하고 있으며, HTTP/1.1 200 OK (text/html) 상태 코드를 반환하는 역할을 수행한다. 14번 패킷은 HTTP 요청에 대한 응답으로, 80 포트에서 수신된 요청에 대한 내용을 포함하고 있으며, 4272 포트로부터 받은 이전 패킷들과 함께 HTTP 요청과 응답의 전체 흐름을 이해하는 데 중요한 비중을 차지한다.

```
14 4.680920 128.119.245.12 192.168.1.102 HTTP 490 HTTP/1.1 200 OK (text/html)
```