모바일 컴퓨팅 HW5

2018037356 – 안동현

1. 다음은 짧은 TCP connection을 wireshark을 이용하여 capture한 trace 이다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

a. 3477번 packet의 경우 wireshark에서 Fast Retransmission이라 표현하였는데 그렇게 생각하는지 근거를 제시하여 설명하세요. 만일 Fast Retransmission이 아니라 생각할 경우, 마찬가지로 근거를 제시하여 설명하세요. (6pts)

먼저 Fast Retransimission이 아니라고 생각합니다.

위의 패킷은 포트 번호 5182, 80간의 패킷들만이 보이는 것으로 미루어 보아 먼저 와이어 샤크 내에서 tcp.port == 80 필터링을 통해 확인한 패킷 중에서 특정한 웹사이트에 대한 연결 요청과 HTTP 요청을 나타내는 패킷들의 흐름임을 추정할 수 있습니다.

A = 10.16.18.218 (클라이언트), B = 49.1.244.170 (웹 서버)

라고 할 때 순차적으로

[SYN] -> [SYN, ACK] -> [ACK] 를 통해 3-Way Handshaking이 문제 없이 진행되고 있는 것을 확인 할 수 있고 그 다음에는 A -> B로 HTTP 웹페이지 요청에 대한 패킷을 날리는 것을 확인 할 수 있습니다. 아마 해당 HTTP 패킷에 대한 TCP 헤더에는 HTTP로 바로 올리기 위해서

[PSH, ACK] 비트가 활성화되어있음을 추측할 수 있고 실제로 저의 와이어 샤크를 통해 확인해본 결과



Flags: 0x018 (PSH, ACK) 해당 옵션을 확인할 수 있었습니다. 여기에 B -> A로 해당 패킷에 대한 ACK까지 확인할 수 있었으며, 여기까지는 정상적입니다.

그러나 그 다음에 캡쳐된 패킷을 확인해보면

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

이렇게 여러 추가 옵션이 달리게 된 것을 확인 할 수 있습니다.

먼저 [TCP Previous segment lost] 입니다. 해당 옵션은 TCP패킷 전송시 SEQ/ACK 번호를 부여하는데, 이 번호와 이전 패킷에서 쓰여 있던 번호를 토대로 와야하는데 오지 못한 패킷이 있는지를 확인해줍니다. 따라서 해당 옵션이 떴다는 것은 해당 패킷 이전에 와야할 패킷이 오지 못했다는 것을 의미하고 그것은



해당 패킷과 같은 HTTP 요청에 대한 확인 패킷이라고 볼 수 있습니다. 따라서 해당 패킷은 어떠한 이유로 인해서 [TCP Previous segment lost] 옵션의 패킷보다 앞서 도착하지 못했다고 추측할 수 있고, 그것이 패킷 드랍을 의미하는지 아니면 단순히 늦게 도착하는지는 아직 모르는 상태입니다.

이제 이로 인해서 [TCP Dup ACK] 옵션이 붙은 ACK가 날아오기 시작하는 것을 확인할 수 있습니다.

텍스트, 라인, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

강의자료의 일부를 보면 손실된 패킷4에 대해서 계속 ACK를 쳐주는 것을 볼 수 있습니다.

하지만 위의 그림은 실제로 패킷이 손실되었을 때 + 중복 ACK가 연속 3번 도착해 실제로 송신자 쪽이 패킷이 손실되었음을 확정한 경우로 fast retransmission을 수행한 경우입니다.

저희의 문제의 경우를 보았을 때 바로 다음 패킷은 [TCP Out-Of-Order] 옵션을 가진 패킷이 웹서버로부터 오는 것을 확인할 수 있는데, 이는 뒤바뀐 순서로 늦게 도착한 패킷을 의미하고 이게 바로 이전에 언급한 

이 부분에 대한 패킷이라고 추측할 수 있습니다. 즉 해당 패킷은 드랍되지 않았고 단순히 모종의 이유로 늦게 도착한 것임을 추측할 수 있고,

따라서 [TCP Dup ACK] 옵션이 붙은 패킷은 3개가 아닌 2개만 캡쳐되었다고 볼 수 있습니다.

이 부분이 중요한 점으로 fast retransmission은 결국 송신자가 [TCP Dup ACK] 옵션을 3개를 받아서 해당 패킷이 손실되었다고 판단했을 때 보내는 패킷이므로, 위의 경우

fast retransmission이 아니었고 매우 빠른 패킷들을 캡쳐하는 와이어 샤크의 오류라고 판단할 수 있을 것 같습니다.

b. 10.16.18.218 목적지에 도착한 패킷들 중 순서대로 도착하지 않은 첫번째 패킷의 사이즈는 얼마인가? (6pts)

먼저 캡쳐한 순간의 패킷을 기준으로 하는 것이 아니라 해당 TCP에 맞는 사이즈를 기준으로 한다면, 이더넷 헤더와 IP 헤더를 제거하고 생각하는 것이 맞습니다.

따라서 [TCP Previous segment lost] 옵션이 붙은 패킷이 처음으로 순서에 맞지 않게 도착한 패킷이라고 할 수 있으므로, 해당 패킷의 사이즈를 생각해 본다면

Info 부분을 보았을 때 해당 패킷은 ACK와 함께 더 이상 보낼 데이터가 없다는 의미로 FIN 플래그를 키고 있어서 데이터 영역은 0이라고 생각됩니다. 따라서 헤더의 길이만 생각해주면 되고, 첫 패킷과는 다르게 MSS, WS, SACK 에 대한 옵션이 없는 걸 볼 수 있으므로 옵션을 제외한 TCP의 헤더 길이인 20byte라고 추측할 수 있습니다.

c. 위의 trace에는 몇개의 data segment가 있나요? (7pts)

패킷의 길이를 추측하고 data영역에 data가 존재할지 생각해본다면, 먼저 3-Way Handshaking 의 경우 SEQ이 각각 1씩 증가하는 것들로 보아 모두 데이터가 존재하지 않는다고 볼 수 있습니다.

하지만 이후



해당 HTTP 요청 이후 ACK가 468인걸 미루어 보아 해당 패킷에는 data segment가 존재함을 추측할 수 있습니다.

또한

HTTP 확인에 대한 패킷으로 추정되는

역시 이것의 ACK로 추정되는 패킷로 인해서 data segment가 있다고 추측되고

HTTP 확인에 대한 패킷의 재전송으로 추측되는

역시 똑같이 data segment를 가질것이라고 추측됩니다.

따라서 총 3개입니다.

d. 3478번 ack 패킷은 몇번 패킷까지를 ack 처리했나요? (7pts)

중요한 점은

3478번 패킷은 과연

 이것에 대한 ACK인가 아니면



이것에 대한 ACK인가 입니다. 강의 자료를 인용해 보았을 때

텍스트, 스크린샷, 라인, 도표이(가) 표시된 사진

자동 생성된 설명

이처럼 애매한 상황일 때 RTT의 샘플은 오리지널에 대해서 한다고 했습니다. 따라서 위의 경우도 그런 이유로 지연된 오리지널 패킷의 ACK라고 판단할 수 있다고 생각하고 3478 이전에 3474, 3476패킷이 중복 ACK를 내놓는 걸로 보아 사실상 3475번 뿐만 아니라, 순서상 나중에 왔어야 할 3473번 역시 ACK를 쳐준다고 볼 수 있습니다.

2. 다음의 TCP trace를 보고 문제에 답하세요.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

a. 데이터 교환을 위해 사용한 MSS는 얼마인가? (2pts)

더 작은 값이 협상되니 1380byte라고 할 수 있습니다.

b. 10.16.30.142로부터 64.233.189.239로 보내진 데이터는 총 몇 byte인가? (6pts)

패킷들을 보아하니, 핸드 쉐이킹과, 터미네이션 과정에서 SYN, ACK등이 1씩 증가하는 것을 제외하면 데이터의 이동은 10.16.30.142 -> 64.233.189.239의 ACK가 갑자기 증가한 부분인 5757을 보고 추측할 수 있습니다. 따라서 5757 – (핸드 쉐이킹 과정의 1) = 5756byte입니다.

b. 64.233.189.239로부터 10.16.30.142로 보내진 데이터는 총 몇 byte인가? (6pts)

위와는 반대의 이유로 64.233.189.239 -> 10.16.30.142 의 ACK가 갑자기 증가한 부분인 789을 보고 추측하면 789 – (핸드 쉐이킹 과정의 1) = 788byte입니다.

3. Generate and capture both TCP 3 way hand-shake packets (SYN, SYN-ACK, and ACK) and tear-down packets (FIN, FIN-ACK, and ACK). You can use any applications for that. Upload pcap files and explain how you obtain them. (5pts)

(in Korean: TCP 연결을 만드는 packet과 연결을 종료하는 packet을 스스로 만든 뒤 wireshark을 활용하여 capture하세요. 그 과정을 설명하고 pcap file을 업로드하세요.)

먼저 간단히 설명 드리자면, TCP 연결을 통해 클라이언트와 서버간의 연결을 맺고 서버에서

hello world 라는 문자열을 받아와서 출력한 후 연결을 해제하는 프로그램입니다.

클라이언트는 21행에서 먼저 소켓 라이브러리를 초기화 해주고

24 ~ 33행에서 소켓을 생성하고, 그것을 바탕으로 서버에 연결 요청을 합니다.

36행은 recv 함수를 호출함으로 서버로부터 전송되는 데이터를 수신해주고

마지막으로 42행에서 소켓 라이브러리를 해제해줍니다.

서버는 21행에서 소켓 라이브러리를 초기화 해주고,

24 ~ 33행에서 소켓을 생성하고 해당 소켓에 IP 주소와 PORT 번호를 할당해줍니다.

이후 36행에서 listen 함수를 호출하여 소켓을 서버 소켓으로 완성시켜줍니다.

40행에서는 클라이언트의 연결 요청을 수락하기 위해 accept 함수를 호출해주며, 44행에서는 send 함수를 통해서 클라이언트에게 데이터를 전송해줍니다. 마지막으로 47행에서 소켓 라이브러리를 해제해줍니다.

클라이언트

텍스트, 전자제품, 스크린샷, 디스플레이이(가) 표시된 사진

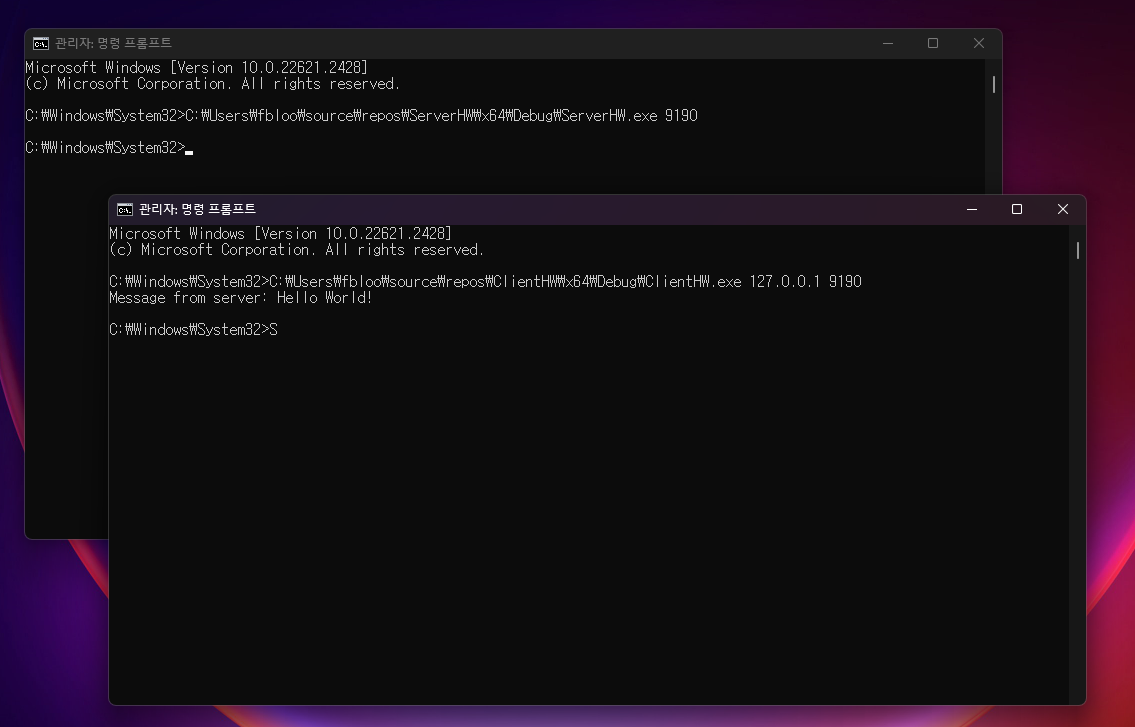
자동 생성된 설명

서버

텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

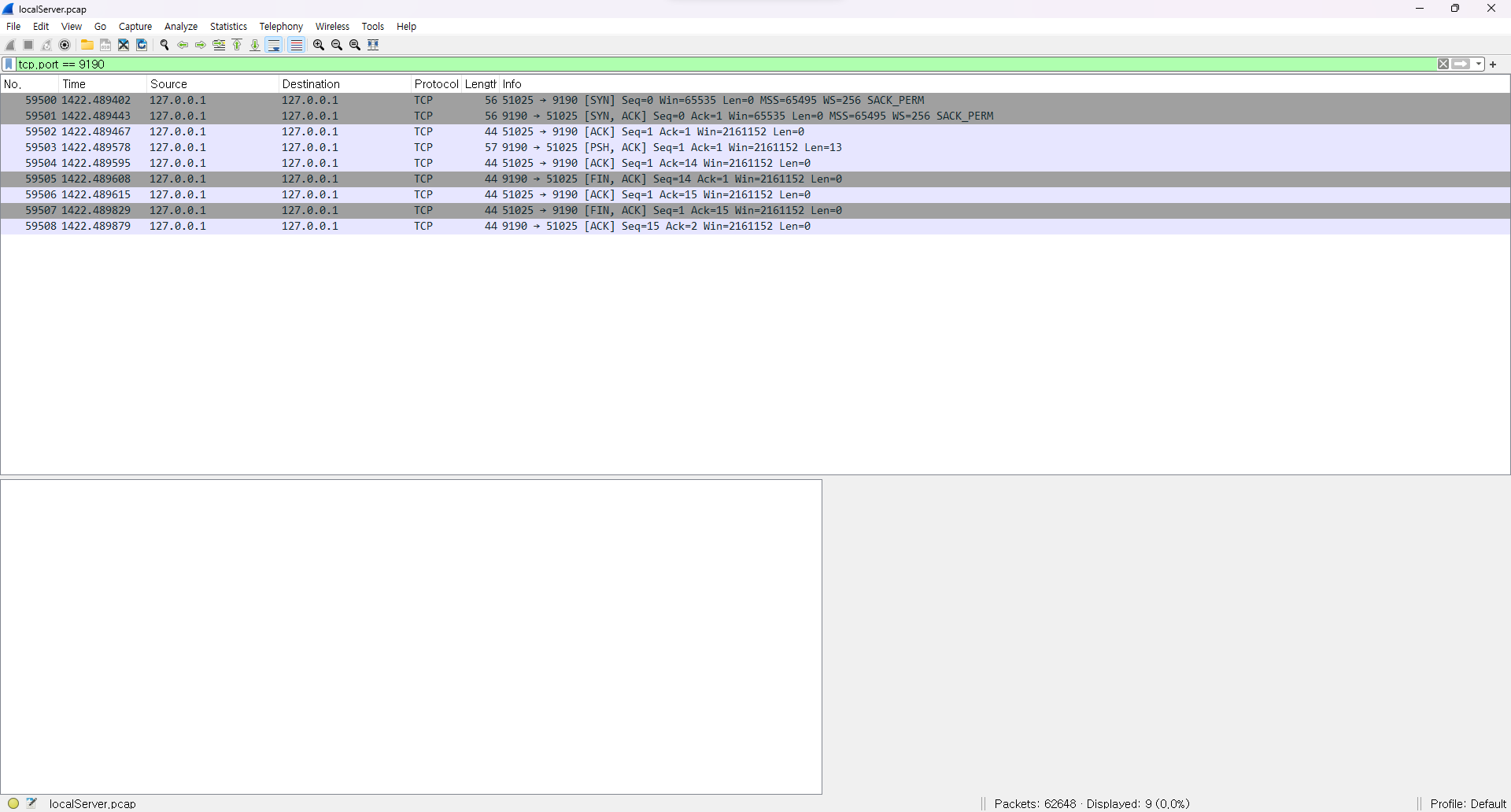
자동 생성된 설명

서버 작동



이런식으로 서버를 먼저 작동시켜서 임의의 PORT 번호를 통해 서버를 열어주고, 클라이언트를 작동시켜서 루프백 IP와 할당한 PORT 번호를 통해 저 자신이 다시 받을 수 있도록 합니다.

패킷 캡쳐



그렇게 진행해서 와이어 샤크의 loopback 인터페이스를 살펴보면 이렇게 정상적으로 해당 포트에 대한 3 way hand-shake 와 데이터 전송, 그리고 tear-down 을 확인하는 것이 가능합니다.

해당 데이터를 자세히 살펴보면

텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

첫 3 way hand-shake에서 SEQ와 ACK를 정상적으로 확인할 수 있고, 바로 종료하지 않고 데이터를 주고받은 것 만큼



이부분을 통해 ACK가 갑자기 14로 뛴 만큼 13바이트의 데이터를 주고받았음을 확인할 수 있습니다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

실제로 전송한 13바이트의 문자열로 인해 정상적으로 작동함을 확인할 수 있고

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

와이어 샤크에서 데이터 영역에 13바이트를 확인할 수 있었습니다.

또한

텍스트, 폰트, 스크린샷, 번호이(가) 표시된 사진

자동 생성된 설명

해당 부분을 통해서도 종료 역시 잘 됨을 확인할 수 있었습니다.

느낀점

개인적으로는 이번 과제가 지금까지 중 가장 어렵다고 느꼈습니다. ARP, IP, ICMP 등은 그 관계가 꽤나 명확함을 볼 수 있지만, TCP의 경우 양방향으로 계속해서 패킷 주고받는데, 그 과정에서 각각의 패킷들이 언제 도착할지 보장하지 못한다는 점이 해석하는데 애를 먹었습니다. 최대한 논리적으로 생각하려고 노력한 결과, 어느정도 이야기를 만들어낼 수 있었지만 아직도 많이 헷갈리는 것 같아서 강의 자료에 있던 그림들처럼 하나하나 순서를 그려가며 패킷을 추적해보는 것이 많은 도움이 될 것이라고 생각했고, 개인적으로 궁금했던 점은 그냥 tcp.port == 80 을 통해서 패킷들을 관찰해보고 있었는데, 서버 -> 클라 쪽으로 가는 데이터에는 항상 이더넷 프레임에 6바이트의 패딩이 붙는다는 점이였습니다. arp가 아니라서 패딩이 붙지 않을줄 알았는데, 패딩이 붙어서 이 부분에 대해 고민을 해본 결과 아직도 이유를 모르겠어서 이 부분에 대해서 면밀한 조사가 필요하겠다고 생각했습니다.