

[네트워크프로그래밍]

프로젝트 결과보고서

소속: IT대학 소프트웨어학부

학번: 20192851, 20192829, 20192847

이름: 윤성준, 박민수, 양조은

프로젝트명: Cut-Thorough(컷또로)

목 차

1. 서론
 - 1.1 프로젝트 목표
 - 1.2 프로젝트 필요성
 - 1.3 프로젝트 기대효과
 - 1.4 수업 연관성
2. 이론적 배경
3. 개발환경 및 사용 라이브러리
 - 3.1 개발환경
 - 3.2 사용 라이브러리 명세
 - 3.3 기타 오픈소스
4. 프로젝트 구조도(Architecture)
 - 4.1 동영상 프로토콜 구조
 - 4.2 동영상 인코딩 및 전송 구조
 - 4.3 TCP Reset Attack
 - 4.4 Cut-Thorough flow chart
5. 프로젝트 진행 과정
 - 5.1 동영상 스트리밍 서버 구축
 - 5.2 패킷 분석
 - 5.3 DASH / HLS 패킷 필터링 프로그램 (Cut-Thorough) 개발
 - 5.4 TCP 채팅 차단 기능 추가
6. 프로젝트 소스 코드
 - 6.1 주요 모듈 및 함수
7. 결론
 - 7.1 프로젝트 성과 및 결과 요약
 - 7.2 프로젝트 정의서 대비 주요 변경사항
8. 참고문헌

1. 서론

1.1 프로젝트 목표

본 프로젝트의 목표는 MPEG-DASH / HLS 패킷 분석을 통해 MPEG-DASH / HLS 기반 동영상 콘텐츠를 차단하고 TCP Reset Attack을 이용하여 TCP 채팅서버와의 연결을 차단하는 프로그램을 개발하는 것이다. 기본적으로 사용자에게 패킷 차단 On/Off를 제공하며, On을 할 경우 HTTP 패킷을 막아 보안성을 높인다. 이를 위해 네트워크 트래픽을 모니터링하고, 프로토콜 유형을 분석하는 시스템을 설계한다. 또 TCP통신을 이용하는 서버가 있다면 TCP 리셋 공격을 통해 TCP 채팅 서버와의 연결을 종료한다. 결과적으로, 이 시스템은 프로토콜 차단을 통해 효과적으로 동영상을 차단하고 TCP 리셋 공격을 통해 기업과 같은 조직 내부에서의 상관없는 TCP 통신 기반 채팅서버 차단을 보장한다. 이러한 솔루션은 다양한 환경에서 적용될 수 있으며, 기업과 같은 조직 내부에서 차단 프로그램으로 큰 도움이 될 것이다.

1.2 프로젝트 필요성

(1) 동영상 차단

기업과 같은 조직 내부에선 보안 강화 및 문제 예방, 네트워크 리소스 관리 등 다음의 이유로 특정 콘텐츠에 대한 접근을 규제하는 시스템을 필요로 한다.

1) 콘텐츠 규제

기업은 콘텐츠 규제 정책을 시행하고 직원들이 업무 시간 동안 부적절하거나 업무와 관련이 없는 비디오 콘텐츠에 액세스하지 못하도록 막을 수 있다. 이를 통해 직원의 생산성을 높이고 방해 요소를 줄일 수 있다. 또한 교육 기관이나 의료 서비스 제공 업체와 같은 특정 산업 또는 조직에는 스트리밍 서비스 사용과 관련하여 특정 규정 준수 또는 규제 요건이 있을 수 있다.

2) 보안 문제 예방 및 강화

비디오 콘텐츠에 대한 무단 액세스를 방지하고, 스트리밍을 통해 전달되는 멀웨어 또는 악성 코드의 위험을 줄이며, 특정 스트리밍 서비스와 관련된 잠재적인 보안 취약성에 대한 노출을 줄일 수 있다. 기업은 이러한 시스템을 활용하여 네트워크 보안을 강화하고 자산을 보호할 수 있다.

3) 대역폭 관리 및 비용 절감

조직 네트워크 망에서 동영상 접근을 차단함으로써 네트워크 대역폭 소비를 제어하여 비용을 절감하고 네트워크 리소스를 더 효과적으로 관리할 수 있다. 이는 시간이 지남에 따라 상당한 비용 절감 효과를 가져올 수 있다.

본 프로젝트에서는 위와 같은 요구 사항에 대하여, 동영상 스트리밍 패킷을 차단하는 백그라운드 실행 서비스를 제공한다. 위 기능을 위하여 URL을 필터링하는 방식이 사용되기도 하는데, 기존 URL 필터링 방식과의 비교는 아래 3. 특징점에 자세히 기술하였다.

(2) HTTP 패킷 처리

일반적인 HTTP 프로토콜은 데이터를 암호화하지 않고 전송하기 때문에, 중간에서 데이터를 탈취하거나 변조될 위험이 있다. 이에 반해 HTTPS는 데이터를 SSL/TLS 암호화 프로토콜을 사용해 암호화하고, 인증서를 통해 서버의 신원을 검증한다. 본 서비스의 사용자는 필요에 의해 해당 기능을 Off함으로써 일시적으로 동영상 콘텐츠를 재생할 수 있는데, 이때 HTTP 패킷은 Drop 하여 보안성을 높인다.

(3) TCP 통신 서버 차단

이 기능은 TCP 리셋 공격을 사용하여 특정 TCP 채팅 서버와의 연결을 종료하는 것을 목표로 한다. 백그라운드 프로그램은 네트워크 트래픽을 모니터링하고, 패킷을 분석하여 프로토콜 유형을 식별할 수 있는 시스템을 설계한다. TCP 통신에 관련된 패킷을 탐지하면, 해당 패킷이 특정 TCP 채팅 서버와의 연결을 나타내는 것으로 식별한다. 이후 백그라운드 프로그램은 TCP 리셋 공격을 수행하여 해당 TCP 채팅 서버와의 연결을 강제로 종료시킨다. TCP 리셋 공격은 악의적인 공격자가 TCP 세그먼트를 조작하여 연결을 끊는 방법이다. 공격자는 TCP 세그먼트의 필드를 변경하거나 잘못된 시퀀스 번호를 전송함으로써 TCP 연결을 종료시킨다. 이를 통해 백그라운드 프로그램은 회사와 상관없는 TCP 통신 기반 채팅 서버와의 연결을 차단할 수 있다. 이는 조직 내에서 보안 및 규제 요건을 충족시키는 데 도움을 주며, 불필요한 통신을 방지하고 보안 취약성을 감소시킨다.

1.3 프로젝트 기대효과

(1) 기존 URL 필터링 방식 대비 특징점

기존의 URL 필터링 방식은 다양한 웹사이트와 콘텐츠 유형에 대응할 수 있지만, MPEG-DASH / HLS 프로토콜 차단은 특정 프로토콜에 초점을 맞추어 스트리밍 콘텐츠 제어에 효과적이다. 특정 프로토콜 차단 방식은 유지 보수가 더 적게 필요하고, 우회하기 어렵다는 장점이 있다.

1) 스트리밍 콘텐츠 제어에 더 집중할 수 있다

URL 필터링을 사용하면 사이트 전체를 차단하지 않고 사이트 내의 특정 페이지나 콘텐츠에 대한 액세스를 차단하기가 어려울 수 있다. 반면 본 프로젝트의 특정 프로토콜 차단 방식은 관련된 특정 트래픽을 타겟팅하여 해당 트래픽만 차단할 수 있으므로 사용자는 동일한 사이트의 다른 콘텐츠에 액세스할 수 있다.

2) 유지보수 감소

기존 URL 필터링의 경우 차단 목록을 지속적으로 업데이트해야 하는 반면, 특정 프로토콜을 차단 방식은 그러한 유지 보수의 번거로움을 절감할 수 있다.

3) 우회 방지

DASH 프로토콜 차단은 기본 스트리밍 기술을 대상으로 하기 때문에 사용자가 우회하기가 더 어려울 수 있다. 프록시 서버, VPN 또는 기타 방법을 사용하여 URL 필터

링을 우회할 수 있지만, DASH 프로토콜 차단을 우회하는 것은 더 어렵다.

4) On/Off 기능 제공

사용자에게 패킷 차단 기능을 제어할 수 있는 옵션을 제공한다. 이는 기존 기술에서는 일반적으로 제공되지 않는 유연성을 제공하며, 사용자가 보안성과 편의성 사이에서 선택할 수 있게 한다.

1.4 수업 연관성

프로젝트에서 네트워크 패킷을 분석 및 처리하기 위해, 강의에서 습득한 소켓 프로그래밍 기술과 소켓 API 활용에 대한 기본 개념이 필요하다. 이를 바탕으로 네트워크 패킷을 효과적으로 분석할 수 있다. 또한, 강의에서 배운 TCP/UDP 소켓과 서버 클라이언트 모델에 대한 이해를 활용하여 프로젝트에서 HLS 및 MPEG-DASH 패킷을 차단할 수 있다. 프로젝트에서 패킷 캡처 및 분석을 위해 Packet Socket을 사용하고, 이에 대한 내용은 10주차 강의에서 배워 활용할 수 있다.

프로젝트에서는 동영상 송출 서버, TCP 채팅 서버, TCP 채팅 클라이언트, 동영상 패킷 차단과 TCP 리셋 공격을 실행하는 PyQt5 프로그램 총 4개이다. 이 중에서 TCP 채팅 서버와 TCP 채팅 클라이언트는 수업의 과제 4번에 해당되는 내용이고 TCP 연결을 끊는 코드는 수업시간에서 배운 Raw Socket을 이용해 코드를 구현했다. Raw Socket를 통해 IP payload에 대한 직접 제어를 수행한다. 또한 Packet Socket을 통해 TCP 플래그를 체크하여 패킷이 SYN 플래그를 가지고 있을 때 리셋 패킷을 보내지 않도록 하고 있다.

이처럼 각각의 강의 내용이 프로젝트의 다양한 측면에서 중요한 역할을 한다. 프로젝트를 진행함으로써 소켓, TCP 통신, HTTP 등등 네트워크 프로그래밍의 전반적인 이해도가 늘 것으로 기대가 된다.

2. 이론적 배경

(1) 동영상 스트리밍 프로토콜

1) MPEG-DASH

MPEG-DASH(Dynamic Adaptive Streaming over HTTP, 이하 DASH 로 표기)는 ISO/IEC 표준의 최초의 적응 비트레이트 HTTP 기반 스트리밍 프로토콜이다. 전통적인 HTTP 웹 서버로부터 전달되는, 인터넷을 경유하는 미디어의 고품질 스트리밍을 가능케 하는 적응 비트레이트 스트리밍 기술이다. 애플의 HTTP Live Streaming(HLS) 솔루션과 비슷하게 DASH 는 내용을 일련의 작은 크기의 HTTP 기반 파일 세그먼트들로 분리시킴으로써 동작하며, 각 세그먼트는 영화나 스포츠 이벤트 생방송 등 잠재적으로 수시간에 걸친 내용물의 재생 시간의 짧은 간격(interval)을 포함하고 있다. DASH 가 사용하는 전송 프로토콜은 TCP 이다.

2) HLS

HLS(HTTP Live Streaming)는 Apple 에서 개발한 동적 비디오 스트리밍을 위한 프로토콜이다. HLS 는 콘텐츠를 작은 세그먼트로 나누어 HTTP 를 통해 전송하며, 클라이언트는 필요한 세그먼트를 요청하고 전송 속도에 따라 재생한다. HLS 는 여러 가지 비디오 코덱과 함께 사용할 수 있으며, 다양한 장치와 플랫폼에서 호환성이 좋다.

(2) 기본 프로토콜

1) TCP

TCP(Transmission Control Protocol)는 인터넷 프로토콜 스위트의 핵심 프로토콜 중 하나로, 데이터를 신뢰성 있게 전송하기 위한 프로토콜이다. TCP 는 인터넷 상에서 데이터를 분할하여 전송하고, 이를 수신 측에서 다시 합치는 역할을 한다. 이를 통해 데이터 전송 과정에서 발생할 수 있는 오류나 손실을 최소화하고, 데이터 전송의 안정성을 보장한다.

2) HTTP

HTTP 는 웹 브라우저와 웹 서버 사이의 데이터 통신을 위한 프로토콜로, 클라이언트가 웹 페이지를 요청하면 서버가 해당 페이지를 제공하는 방식으로 작동합니다. 데이터는 요청과 응답의 형태로 주고받으며, 암호화되지 않아 제 3 자에 의해 쉽게 엿볼 수 있습니다.

(3) TCP Reset Attack

TCP Reset Attack(TCP Reset 공격)은 공격자가 RST(Restart) 패킷을 보내는 방식으로 실행된다. RST 패킷은 TCP 에서 세션을 끊는 데 사용되는 패킷이다. 정상적인 경우에는

네트워크 오류나 TCP 세션 종료 시에만 보낼 수 있지만, 공격자가 송신자나 수신자의 IP 주소와 포트 번호, 시퀀스 번호 등을 알고 있다면 가짜 RST 패킷을 만들어 세션을 강제로 끊을 수 있다.

TCP 리셋 공격을 수행하기 위해서는 스푸핑된 RST 세그먼트에 대해 클라이언트가 신뢰할 수 있는 시퀀스 번호를 지정해야 한다. 수신자는 순차적이지 않은 시퀀스 번호를 가진 세그먼트도 수용하고 올바른 순서로 복원하는 것을 허용하지만, 이 허용 범위는 제한이 있다. 만약 수신자는 일정 범위를 벗어난 시퀀스 번호를 가진 세그먼트를 수신하면 해당 세그먼트를 폐기한다. 따라서 성공적인 TCP 리셋 공격에는 적당한 시퀀스 번호가 필요하다.

이때 수신자가 신뢰할 수 있는 시퀀스 번호는 수신자의 TCP 윈도우 크기에 의해 결정된다. 수신자의 윈도우 크기는 송신자가 한 번에 수신자에게 보낼 수 있는 확인되지 않은 바이트의 최대 수이다. 따라서 TCP 규격은 수신자가 자신의 윈도우 범위를 벗어난 시퀀스 번호를 가진 데이터를 무시하도록 규정한다. 예를 들어, 수신자가 15,000 까지의 모든 바이트에 대한 ACK 를 보낸 경우, 윈도우 크기가 30,000 이라면 15,000 부터 $(15,000 + 30,000 = 45,000)$ 까지의 시퀀스 번호를 가진 데이터를 수용한다. 이와는 달리 윈도우 범위를 벗어난 시퀀스 번호를 가진 데이터는 완전히 무시된다.

일반적인 세그먼트는 다음 예상 시퀀스 번호부터 해당 번호에 윈도우 크기를 더한 값 사이의 시퀀스 번호를 가지면 수락된다. 그러나 RST 패킷은 다음 예상 시퀀스 번호와 정확히 일치하는 시퀀스 번호를 가져야만 수락된다. 이전의 예시를 다시 생각해 보면, 수신자가 15,000 의 확인 응답 번호를 보낸 경우, 수락되려면 RST 패킷은 정확히 15,000 의 시퀀스 번호를 가져야 한다.

이러한 방식으로 세그먼트를 가로채고 세그먼트의 목적지와 소스 포트를 알아내고 시퀀스 번호를 정확하게 가로챈 패킷의 ACK 숫자로 지정하고 RST 플래그 켜서 수신자에게 보낸다면 TCP 리셋 공격을 할 수 있다.

3. 개발환경 및 사용 라이브러리

3.1 개발환경

- 운영체제: Ubuntu 20.04
- 개발 언어: Python 3.8.10
- 패킷 분석: Wireshark 3.2.3, Pyshark 0.5.3
- 개발 프레임워크: Flask 2.3.2, PyQt 5.14.1

3.2 사용 라이브러리 명세

- PyQt5: 사용자 인터페이스를 만드는 데 사용되는 라이브러리이다. QPushButton, QMessageBox, QVBoxLayout, QWidget 등의 위젯을 사용한다.
- subprocess: 시스템 커맨드를 실행하기 위해 사용되는 라이브러리이다. 여기서는 iptables 커맨드를 실행하기 위해 사용된다.
- Scapy: Scapy는 Python으로 작성된 강력한 패킷 조작 도구 및 라이브러리이다. 이를 사용하여 네트워크 패킷을 생성, 조작, 전송 및 수신할 수 있다. 여기서는 패킷 스니핑과 RST 패킷 생성 및 전송에 사용된다.
- ifaddr: 로컬 네트워크 어댑터 정보를 얻기 위해 사용되는 라이브러리이다.
- threading: 별도의 스레드에서 코드를 실행하는 데 사용되는 라이브러리이다.
- random: Python의 표준 라이브러리 중 하나로, 다양한 종류의 랜덤 연산을 지원한다. 여기서는 TCP RST 패킷의 sequence number를 무작위로 결정하는 데 사용됐다.

3.3 기타 오픈소스

<https://github.com/robert/how-does-a-tcp-reset-attack-work/blob/master/main.py>

이 Python 코드는 네트워크 패킷 조작 라이브러리 Scapy를 사용하여 패킷의 TCP 커넥션을 찾아내고, 그 커넥션에 RST 패킷을 보내서 연결을 리셋하는 코드이다. 이 코드는 로컬 호스트 인터페이스에서 TCP 패킷을 스니핑하고, 특정 서버와 클라이언트 간의 트래픽을 필터링한다.

4. 프로젝트 구조도(Architecture)

4.1 동영상 프로토콜 구조

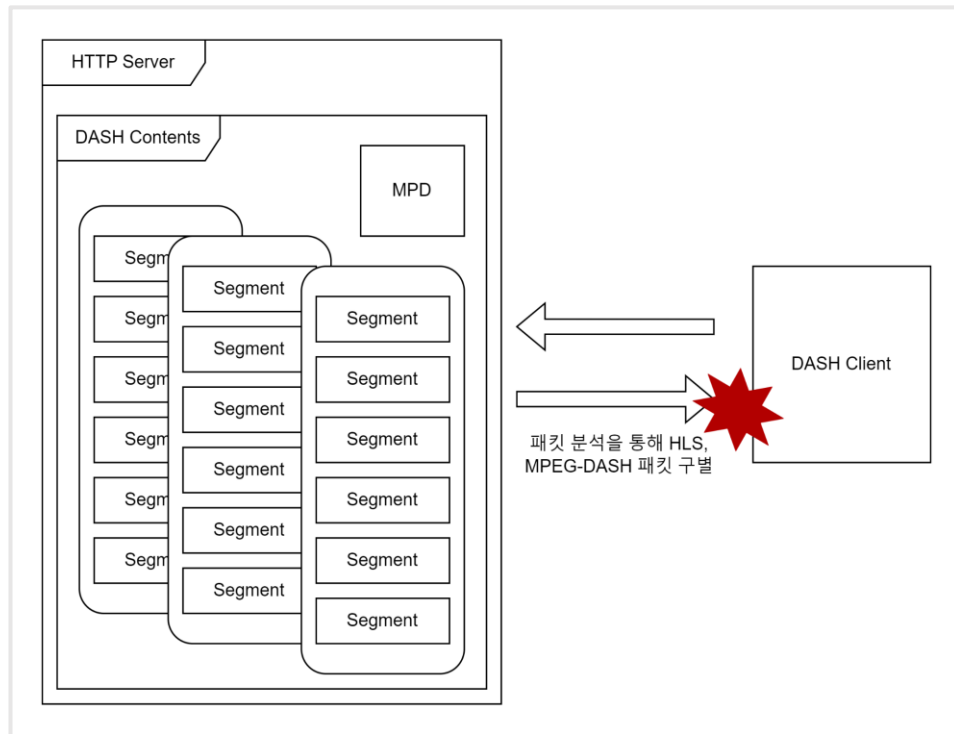


그림 1. DASH 프로토콜 구조 및 Cut-Thorough 프로그램의 blocking position

그림 1은 DASH 프로토콜의 구성과 동작 구조와 Cut-Thorough의 blocking position을 간략히 시각화한 것으로, 프로토콜 기반 동영상을 스트리밍하는 서버의 동작을 위주로 나타내었다. DASH는 동영상 파일을 여러 개의 작은 청크(chunk) 미디어 파일로 분할하고, 이러한 청크들을 클라이언트에게 동적으로 제공한다. DASH의 미디어 파일은 두 가지 주요 형식으로 나뉘는데, 서버에는 동영상의 메타데이터를 저장한 매니페스트 파일(.mpd)이 있으며, 이는 사전에 생성되어 있거나 스트리밍 중에 동적으로 생성 및 업데이트된다. 또한 잘게 쪼갠 동영상을 세그먼트(.m4s) 파일 형식으로 인코딩 된 콘텐츠를 전송한다. 동작 구조에 대한 더 자세한 설명은 아래 5.2장에 하였다.

HLS는 DASH와 구성이 유사하여 도표를 생략하였다. 동영상의 메타데이터를 저장하는 플레이리스트 파일(.m3u8)이 있고, 동영상을 분할한 세그먼트 파일(.ts)이 있다.

Cut-Thorough는 DASH / HLS 서버로부터 온 .mpd, .m4s / .m3u8, .ts 형식의 파일을 클라이언트 단에서 차단(block)하는 프로그램이다.

4.2 동영상 인코딩 및 전송 구조

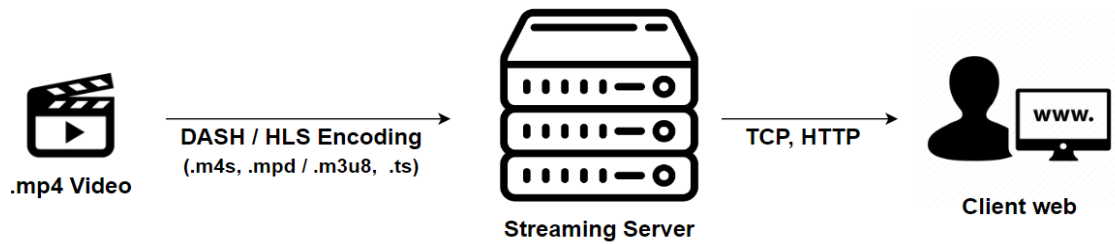


그림 2. DASH / HLS를 사용한 동영상 인코딩 및 클라이언트로 전송 과정

클라이언트는 웹 페이지에 접속하면, 첫 번째로 서버에게 '/' 경로에 대한 요청을 보낸다. 서버는 이 요청을 받고, index 함수를 실행하여 index.html 템플릿을 클라이언트에게 반환한다. 이 페이지에는 동영상을 재생하는 데 필요한 링크 또는 플레이어가 제공될 수 있다. 이후, 클라이언트가 동영상을 재생하기 위해 특정 동영상을 요청한다. 만약 클라이언트가 DASH 또는 HLS 형식으로 동영상을 재생해야 한다면, 클라이언트는 '/hls' 또는 '/dash' 경로에 대한 요청을 보낼 수 있다. 서버는 hls 또는 dash 함수를 실행하여 'hls' 또는 'dash' 디렉토리에서 해당 경로에 있는 파일을 찾아 클라이언트에게 반환한다. 클라이언트는 받은 파일을 플레이어에 전달하여 동영상을 재생한다. 자세한 설명은 5.1장에서 하겠다.

4.3 TCP Reset Attack

TCP Reset Attack: intro (I)

- Alice and Bob have a TCP connection
- Eve sends a spoofed TCP reset packet to Bob with Alice's address/port
- Bob will close connection
- (Alice won't receive any further data from Bob)

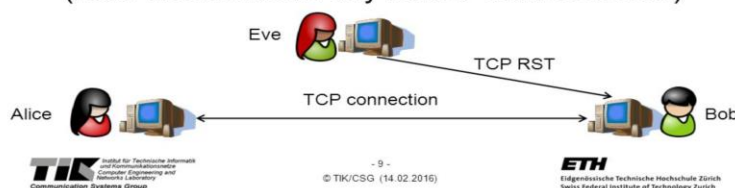


그림 3. TCP Reset Attack 순서도

(출처: https://images.slideplayer.com/31/9598472/slides/slide_9.jpg)

TCP Reset Attack은 공격자가 제3자의 TCP 연결을 강제로 종료시키는 공격 기법입니다. 공격자는 TCP RST 패킷을 생성하여 유효한 출발지 및 목적지 IP 및 포트 정보와 함께 전송하며, 이를 통해 연결을 끊을 수 있습니다. Cut-Through 프로그램은 버튼 클릭 이벤트에 의해 RST 패킷을 생성하고 전송하는 간단한 동작을 수행한다. 패킷 스니핑을 통해 네트워크 트래픽을 감시하고, 필요에 따라 연결을 리셋할 수 있는 기능을 제공한다. 자세한 설명은 5.4장에서 하겠다.

4.4 Cut-Thorough flow chart

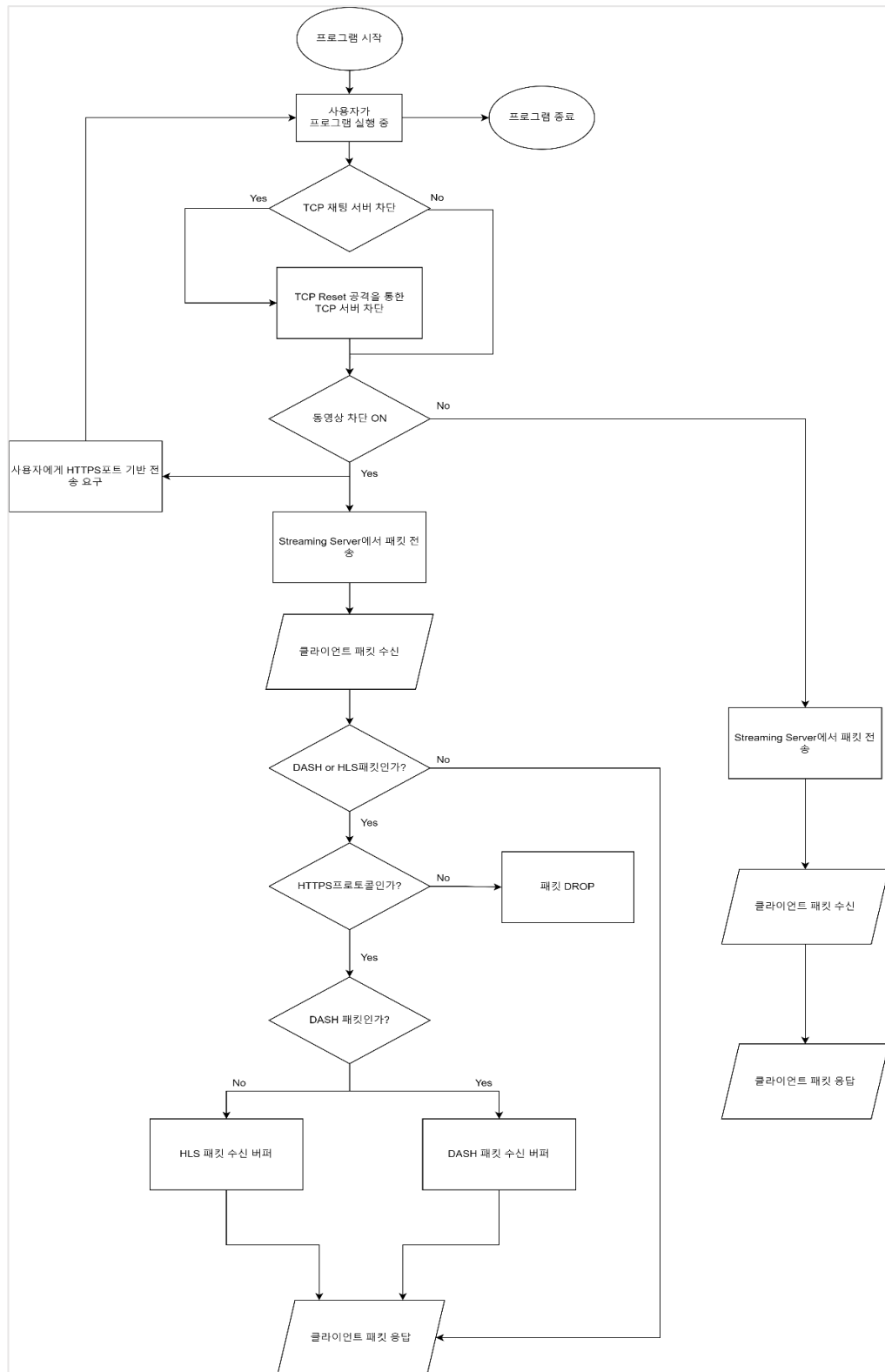


그림 4. Cut-Thorough flow chart

그림 4는 Cut-Thorough 프로그램의 flow chart를 나타낸 것이다. 사용자가 프로그램을 실행하면 '동영상 허용', '동영상 차단', 'RST 패킷 스니핑 시작' 총 3개의 버튼을 보게 된다. 사용자가 동영상 허용을 한다면 "HTTP 기반 DASH / HLS 프로토콜 동영상 허용(지연과 트래픽에 주의하세요.)" 라는 알람과 함께 HTTP 프로토콜로 전송되는 DASH / HLS 동영상 패킷을 수신하고 클라이언트는 이에 패킷 응답을 보내게 된다. 만약 클라이언트가 '동영상 차단' 버튼을 누르면 "HTTP 기반의 DASH / HLS 프로토콜 동영상 차단 (관리자에 문의하여 HTTPS로 동영상을 요청하세요.)" 라는 메시지와 함께 HTTP 기반의 DASH / HLS 패킷은 drop 되며, 만약 HTTPS 프로토콜 기반의 DASH / HLS 패킷은 클라이언트가 해당하는 패킷을 각 해당 버퍼에 수신한 후, 응답하게 된다. 세 번째 'TCP 채팅 서버 차단' 버튼을 누르면 다음과 같은 순서로 TCP 채팅서버와 클라이언트 사이의 연결을 끊는다.

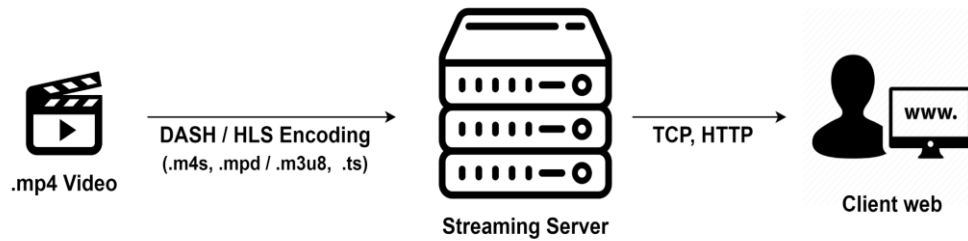
- i. 네트워크 상에서 패킷을 감시하고 대상과 통신하는 TCP 세션을 식별한다.
- ii. 공격자는 해당 TCP 세션에 대한 유효한 시퀀스 번호를 알아내거나 예측한다.
- iii. 공격자는 유효한 시퀀스 번호를 가진 TCP 리셋 세그먼트를 생성하여 대상에게 전송한다.
- iv. 대상은 이러한 잘못된 리셋 세그먼트를 수신하고, 예기치 않은 동작으로 인해 TCP 연결을 종료한다.

버튼을 누르면 50개의 패킷을 스니핑하며 각 패킷을 캡처 할 때마다 리셋 패킷을 보낸다. 이후 50개의 스니핑이 끝나면 "Finished sniffing!"라는 메시지를 로깅하여 패킷 스니핑이 끝났음을 알린다.

5. 프로젝트 진행 과정

5.1 동영상 스트리밍 서버 구축

5.1.1 DASH 프로토콜과 HLS 프로토콜을 위한 비디오 인코딩



Flask를 이용하여 DASH 프로토콜과 HLS 프로토콜을 지원하는 간단한 동영상 스트리밍 서버를 구축하였다. 이 서버는 디렉토리 내의 저장된 영상을 송출하며, 패킷 분석이 용이할 수 있도록 적당한 길이(약 3분)의 영상을 선택하였다. 영상은 DASH 프로토콜과 HLS 프로토콜에 맞게 인코딩 되어야 한다. 이를 위해 ffmpeg 프로그램을 사용하여 영상을 인코딩을 하였고, 인코딩 명령어는 다음과 같다.

1) DASH encoding

```
ffmpeg -i video02.mp4 -c:v libx264 -b:v 1M -c:a aac -b:a 128k -vf "scale=-1:720" -f dash -min_seg_duration 5000 video02.mpd
```

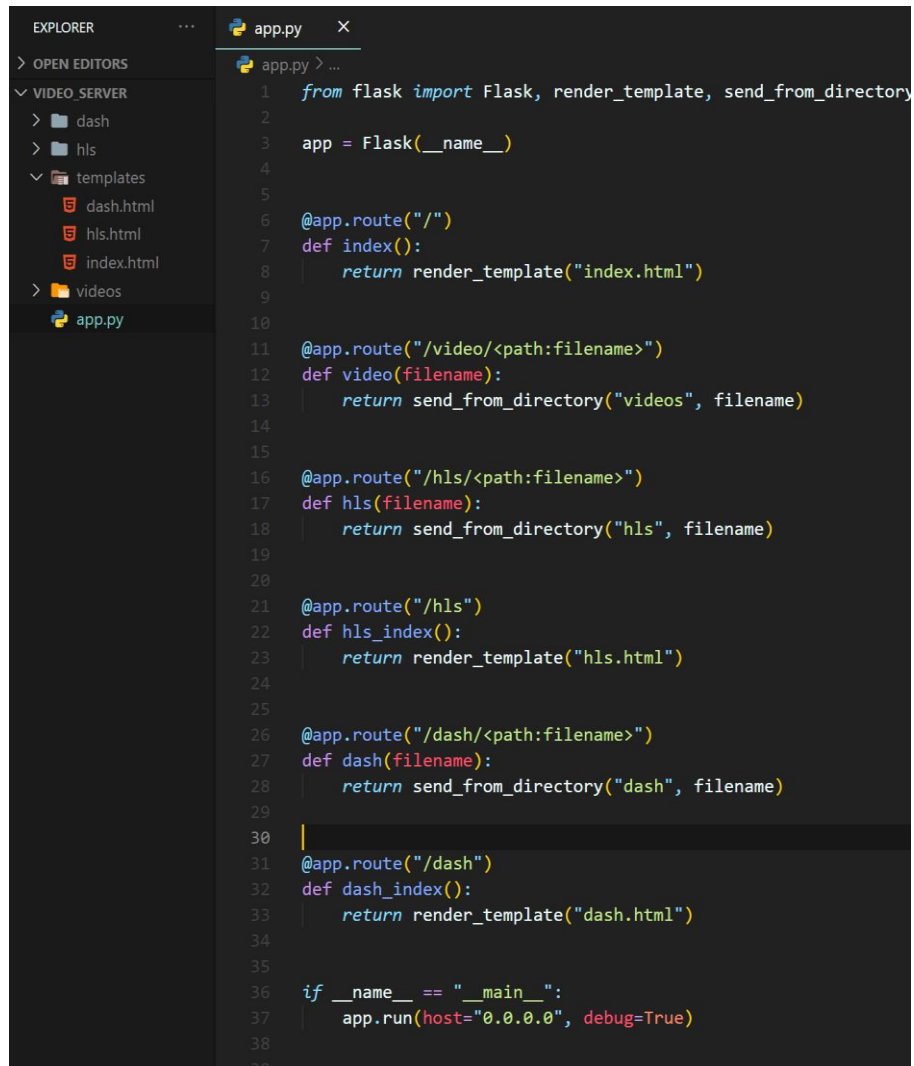
동영상은 H.264 코덱을 사용하여 인코딩하였고 비트레이트를 1Mbps로 설정했다. 오디오는 AAC코덱을 사용하여 인코딩하였고 비디오의 비트레이트를 128Kbps로 설정했다. 해상도는 가로 해상도는 유지한 채로 세로 해상도를 720 픽셀로 조정했다. 각 세그먼트의 최소 지속 시간은 5초로 설정했다.

2) HLS encoding

```
ffmpeg -i video02.mp4 -profile:v baseline -level 3.0 -s 640x480 -start_number 0 -hls_time 10 -hls_list_size 0 -f hls index.m3u8
```

동영상 해상도를 640x480로 조정하고 세그먼트 시작 번호를 0으로 지정한다. 그리고 각 세그먼트의 지속시간을 10초로 지정했다.

5.1.2 서버 프로그램 설명



The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a directory structure for a video server. The code editor shows the app.py file with the following code:

```
1 from flask import Flask, render_template, send_from_directory
2
3 app = Flask(__name__)
4
5
6 @app.route("/")
7 def index():
8     return render_template("index.html")
9
10
11 @app.route("/video/<path:filename>")
12 def video(filename):
13     return send_from_directory("videos", filename)
14
15
16 @app.route("/hls/<path:filename>")
17 def hls(filename):
18     return send_from_directory("hls", filename)
19
20
21 @app.route("/hls")
22 def hls_index():
23     return render_template("hls.html")
24
25
26 @app.route("/dash/<path:filename>")
27 def dash(filename):
28     return send_from_directory("dash", filename)
29
30
31 @app.route("/dash")
32 def dash_index():
33     return render_template("dash.html")
34
35
36 if __name__ == "__main__":
37     app.run(host="0.0.0.0", debug=True)
38
39
```

그림 5. 동영상 스트리밍 서버 구현 코드와 파일 구조

그림은 서버의 코드와 비디오가 저장된 폴더를 볼 수 있다.

핸들러 함수를 통해 경로에 대한 요청을 처리해준다. 이 서버는 총 3가지 형태로 비디오를 송출을 하고 서버는 다음과 같은 방식으로 동작한다.

1. “/” 경로 접속 → “index.html” 템플릿 → video 디렉토리에 있는 .mp4 파일 전송
2. “/dash” 경로 접속 → “dash.html” 템플릿 → dash 디렉토리에 있는 .mpd 파일 전송
3. “/hls” 경로 접속 → “hls.html” 템플릿 → hls 디렉토리에 있는 .m3u8파일 전송

5.1.3 서버 구동 사진 및 설명

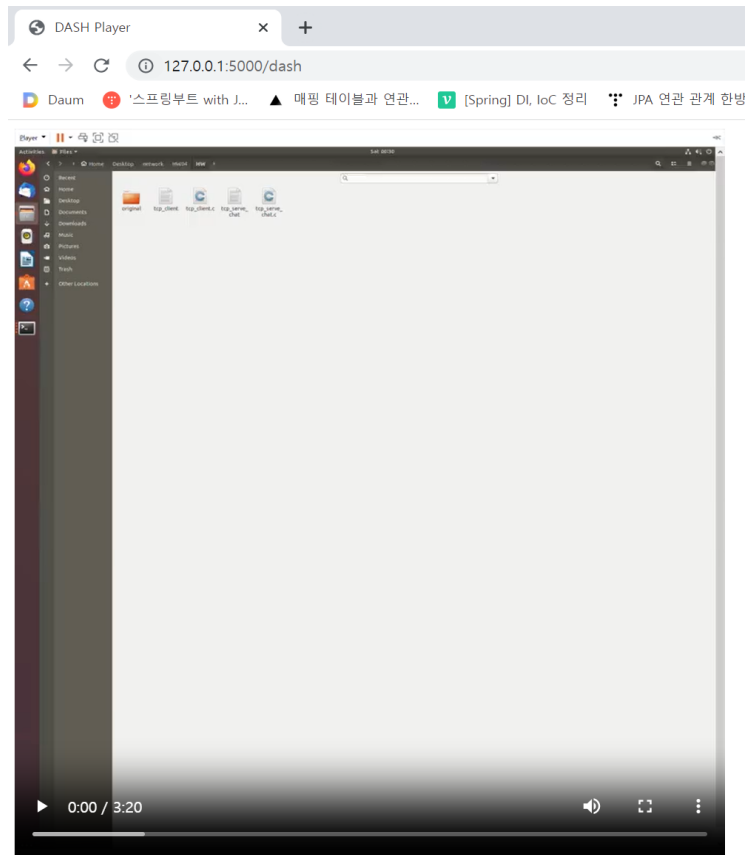


그림 6. DASH 프로토콜을 이용한 동영상 스트리밍 캡처본

```
PS C:\Users\82104\Desktop\3학년1학기\네트워크프로그래밍\프로젝트\video_server\video_server>
& C:/Python39/python.exe c:/Users/82104/Desktop/3학년1학기/네트워크프로그래밍/프로젝트/video
_server/video_server/app.py
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a produ
ction WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://192.168.45.58:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 690-110-293
127.0.0.1 - - [21/May/2023 23:52:21] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [21/May/2023 23:52:22] "GET /video/video02.mp4 HTTP/1.1" 206 -
127.0.0.1 - - [21/May/2023 23:52:22] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [21/May/2023 23:52:22] "GET /video/video02.mp4 HTTP/1.1" 206 -
127.0.0.1 - - [21/May/2023 23:52:22] "GET /video/video02.mp4 HTTP/1.1" 206 -
127.0.0.1 - - [21/May/2023 23:52:22] "GET /video/video02.mp4 HTTP/1.1" 206 -
127.0.0.1 - - [21/May/2023 23:52:48] "GET /dash HTTP/1.1" 200 -
127.0.0.1 - - [21/May/2023 23:52:48] "GET /dash/video02.mpd HTTP/1.1" 200 -
127.0.0.1 - - [21/May/2023 23:52:48] "GET /dash/init-stream0.m4s HTTP/1.1" 200 -
127.0.0.1 - - [21/May/2023 23:52:48] "GET /dash/init-stream1.m4s HTTP/1.1" 200 -
127.0.0.1 - - [21/May/2023 23:52:48] "GET /dash/chunk-stream0-00001.m4s HTTP/1.1" 200 -
127.0.0.1 - - [21/May/2023 23:52:48] "GET /dash/chunk-stream1-00001.m4s HTTP/1.1" 200 -
127.0.0.1 - - [21/May/2023 23:52:48] "GET /dash/chunk-stream0-00002.m4s HTTP/1.1" 200 -
127.0.0.1 - - [21/May/2023 23:52:48] "GET /dash/chunk-stream1-00002.m4s HTTP/1.1" 200 -
127.0.0.1 - - [21/May/2023 23:52:49] "GET /dash/chunk-stream0-00003.m4s HTTP/1.1" 200 -
127.0.0.1 - - [21/May/2023 23:52:49] "GET /dash/chunk-stream1-00003.m4s HTTP/1.1" 200 -
127.0.0.1 - - [21/May/2023 23:52:49] "GET /dash/chunk-stream0-00004.m4s HTTP/1.1" 200 -
127.0.0.1 - - [21/May/2023 23:52:49] "GET /dash/chunk-stream1-00004.m4s HTTP/1.1" 200 -
127.0.0.1 - - [21/May/2023 23:52:49] "GET /dash/chunk-stream0-00005.m4s HTTP/1.1" 200 -
127.0.0.1 - - [21/May/2023 23:52:49] "GET /dash/chunk-stream1-00005.m4s HTTP/1.1" 200 -
127.0.0.1 - - [21/May/2023 23:52:49] "GET /dash/chunk-stream0-00006.m4s HTTP/1.1" 200 -
127.0.0.1 - - [21/May/2023 23:52:49] "GET /dash/chunk-stream1-00006.m4s HTTP/1.1" 200 -
127.0.0.1 - - [21/May/2023 23:53:03] "GET /favicon.ico HTTP/1.1" 404 -
```

그림 7. DASH 스트리밍 시 http get 요청 캡처본

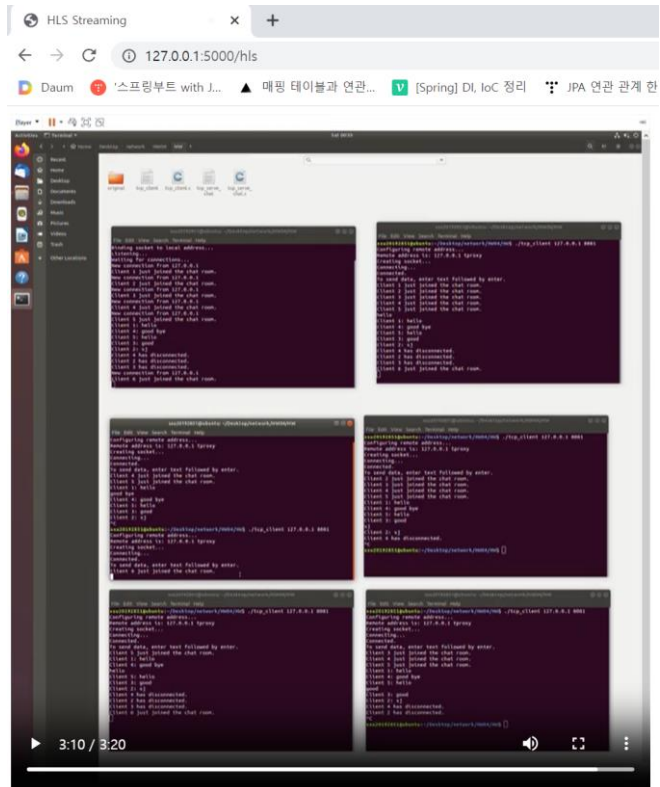


그림 8. HLS 프로토콜로 이용한 비디오 스트리밍 캡처본

```

127.0.0.1 - - [21/May/2023 23:53:03] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [21/May/2023 23:54:27] "GET /hls HTTP/1.1" 200 -
127.0.0.1 - - [21/May/2023 23:54:27] "GET /hls/index.m3u8 HTTP/1.1" 200 -
127.0.0.1 - - [21/May/2023 23:54:28] "GET /hls/index0.ts HTTP/1.1" 200 -
127.0.0.1 - - [21/May/2023 23:54:28] "GET /hls/index1.ts HTTP/1.1" 200 -
127.0.0.1 - - [21/May/2023 23:54:28] "GET /hls/index2.ts HTTP/1.1" 200 -
127.0.0.1 - - [21/May/2023 23:54:33] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [21/May/2023 23:54:34] "GET /hls/index3.ts HTTP/1.1" 200 -
127.0.0.1 - - [21/May/2023 23:54:34] "GET /hls/index4.ts HTTP/1.1" 200 -
127.0.0.1 - - [21/May/2023 23:54:34] "GET /hls/index5.ts HTTP/1.1" 200 -
127.0.0.1 - - [21/May/2023 23:54:34] "GET /hls/index6.ts HTTP/1.1" 200 -
127.0.0.1 - - [21/May/2023 23:54:35] "GET /hls/index7.ts HTTP/1.1" 200 -
127.0.0.1 - - [21/May/2023 23:54:41] "GET /hls/index8.ts HTTP/1.1" 200 -
127.0.0.1 - - [21/May/2023 23:54:43] "GET /hls/index9.ts HTTP/1.1" 200 -
127.0.0.1 - - [21/May/2023 23:54:43] "GET /hls/index10.ts HTTP/1.1" 200 -
127.0.0.1 - - [21/May/2023 23:54:43] "GET /hls/index11.ts HTTP/1.1" 200 -
127.0.0.1 - - [21/May/2023 23:54:44] "GET /hls/index12.ts HTTP/1.1" 200 -
127.0.0.1 - - [21/May/2023 23:54:46] "GET /hls/index13.ts HTTP/1.1" 200 -
127.0.0.1 - - [21/May/2023 23:54:46] "GET /hls/index14.ts HTTP/1.1" 200 -
127.0.0.1 - - [21/May/2023 23:54:47] "GET /hls/index15.ts HTTP/1.1" 200 -
127.0.0.1 - - [21/May/2023 23:54:47] "GET /hls/index16.ts HTTP/1.1" 200 -
127.0.0.1 - - [21/May/2023 23:54:47] "GET /hls/index17.ts HTTP/1.1" 200 -
127.0.0.1 - - [21/May/2023 23:54:48] "GET /hls/index18.ts HTTP/1.1" 200 -
127.0.0.1 - - [21/May/2023 23:55:40] "GET /hls/index19.ts HTTP/1.1" 200 -

```

그림 9. HLS 스트리밍 시 HTTP GET 요청 캡처본

5.2 패킷 분석

Wireshark를 통하여 이전 장 5.1 에서 구현한 동영상 서버의 동작 시 나타나는 DASH / HLS 패킷 및 관련 트래픽을 분석하였다.

DASH 패킷 분석

http.request.uri contains ".m4s" or http.request.uri contains ".mpd"

The image displays a Wireshark packet capture analysis of a DASH stream. The top pane shows a list of HTTP GET requests for various DASH segments (init-stream0.m4s, chunk-stream1-00001.m4s, etc.). The middle pane shows the details of a selected packet, including the Hypertext Transfer Protocol section. The bottom pane shows the packet bytes and the corresponding hex and ASCII data. A DASH Player window is also visible in the background, showing a video player interface.

.m4s, .mpd 확장자 확인 가능

http.request.uri contains ".m4s" or http.request.uri contains ".mpd" or tcp					
Time	Source	Destination	Protocol	Length	Info
1.0.000000000	127.0.0.1	127.0.0.1	TCP	74	44546 → 5000 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=
2.0.000052426	127.0.0.1	127.0.0.1	TCP	74	5000 → 44546 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495
3.0.000093435	127.0.0.1	127.0.0.1	TCP	66	44546 → 5000 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=61392849
4.0.000365590	127.0.0.1	127.0.0.1	HTTP	716	GET /dash HTTP/1.1
5.0.000371747	127.0.0.1	127.0.0.1	TCP	66	5000 → 44546 [ACK] Seq=1 Ack=651 Win=64896 Len=0 TSval=613928
6.0.005464143	127.0.0.1	127.0.0.1	TCP	240	5000 → 44546 [PSH, ACK] Seq=1 Ack=651 Win=65536 Len=174 TSval=
7.0.005479204	127.0.0.1	127.0.0.1	TCP	66	44546 → 5000 [ACK] Seq=651 Ack=175 Win=65408 Len=0 TSval=6139
8.0.005507911	127.0.0.1	127.0.0.1	HTTP	485	HTTP/1.1 200 OK (text/html)
9.0.005512087	127.0.0.1	127.0.0.1	TCP	66	44546 → 5000 [ACK] Seq=651 Ack=594 Win=65024 Len=0 TSval=6139
10.0.016356712	127.0.0.1	127.0.0.1	TCP	66	5000 → 44546 [FIN, ACK] Seq=594 Ack=651 Win=65536 Len=0 TSval=
11.0.059395605	127.0.0.1	127.0.0.1	TCP	66	44546 → 5000 [FIN, ACK] Seq=651 Ack=595 Win=65536 Len=0 TSval=
12.0.059418992	127.0.0.1	127.0.0.1	TCP	66	5000 → 44546 [ACK] Seq=595 Ack=652 Win=65536 Len=0 TSval=6139
17.0.538498572	127.0.0.1	127.0.0.1	TCP	74	44562 → 5000 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=
18.0.538550808	127.0.0.1	127.0.0.1	TCP	74	5000 → 44562 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495
19.0.538560135	127.0.0.1	127.0.0.1	TCP	66	44562 → 5000 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=61393387
20.0.538653310	127.0.0.1	127.0.0.1	HTTP	583	GET /dash/video02.mpd HTTP/1.1
21.0.538656261	127.0.0.1	127.0.0.1	TCP	66	5000 → 44562 [ACK] Seq=1 Ack=518 Win=65024 Len=0 TSval=613933
22.0.544733870	127.0.0.1	127.0.0.1	TCP	438	5000 → 44562 [PSH, ACK] Seq=1 Ack=518 Win=65536 Len=372 TSval=
23.0.544754715	127.0.0.1	127.0.0.1	TCP	66	44562 → 5000 [ACK] Seq=518 Ack=373 Win=65280 Len=0 TSval=6139
24.0.544783682	127.0.0.1	127.0.0.1	HTTP	2138	HTTP/1.1 200 OK
25.0.544789756	127.0.0.1	127.0.0.1	TCP	66	44562 → 5000 [ACK] Seq=518 Ack=2445 Win=63744 Len=0 TSval=613
26.0.547491223	127.0.0.1	127.0.0.1	TCP	66	44562 → 5000 [FIN, ACK] Seq=518 Ack=2445 Win=65536 Len=0 TSval=
27.0.547662370	127.0.0.1	127.0.0.1	TCP	66	5000 → 44562 [FIN, ACK] Seq=5000 Ack=519 Win=65536 Len=0 TSval=
28.0.547672035	127.0.0.1	127.0.0.1	TCP	66	44562 → 5000 [ACK] Seq=519 Ack=2446 Win=65536 Len=0 TSval=613
29.0.654209343	127.0.0.1	127.0.0.1	TCP	74	44574 → 5000 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=
30.0.654265843	127.0.0.1	127.0.0.1	TCP	74	5000 → 44574 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495
31.0.654282710	127.0.0.1	127.0.0.1	TCP	66	44574 → 5000 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=61393503
32.0.654446304	127.0.0.1	127.0.0.1	HTTP	588	GET /dash/init-stream0.m4s HTTP/1.1
33.0.654452430	127.0.0.1	127.0.0.1	TCP	66	5000 → 44574 [ACK] Seq=1 Ack=523 Win=65024 Len=0 TSval=613935
34.0.660248624	127.0.0.1	127.0.0.1	TCP	440	5000 → 44574 [PSH, ACK] Seq=1 Ack=523 Win=65536 Len=374 TSval=
35.0.660276453	127.0.0.1	127.0.0.1	TCP	66	44574 → 5000 [ACK] Seq=523 Ack=375 Win=65280 Len=0 TSval=6139
36.0.660304499	127.0.0.1	127.0.0.1	HTTP	899	HTTP/1.1 200 OK
37.0.660308616	127.0.0.1	127.0.0.1	TCP	66	44574 → 5000 [ACK] Seq=523 Ack=1208 Win=64512 Len=0 TSval=613
38.0.660353491	127.0.0.1	127.0.0.1	TCP	66	44574 → 5000 [FIN, ACK] Seq=523 Ack=1208 Win=65536 Len=0 TSval=
39.0.660362006	127.0.0.1	127.0.0.1	TCP	66	5000 → 44574 [FIN, ACK] Seq=1208 Ack=524 Win=65536 Len=0 TSval=

1) TCP 연결

클라이언트는 DASH 동영상을 요청하기 위해 우선 서버와 TCP 연결 (3-way handshake)을 설정한다. (위 사진에서 1, 2, 3번 라인) TCP는 신뢰성 있는 연결 지향형 프로토콜로, 클라이언트와 서버 간에 신뢰성 있는 데이터 전송을 제공한다.

1	0.000000000	127.0.0.1	127.0.0.1	TCP	74 44546 → 5000 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=
2	0.000052426	127.0.0.1	127.0.0.1	TCP	74 5000 → 44546 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495
3	0.000093435	127.0.0.1	127.0.0.1	TCP	66 44546 → 5000 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=61392849

2) HTTP 요청

TCP 연결이 설정되면 클라이언트는 서버에게 DASH 동영상에 대한 정보를 요청하는 HTTP GET 요청 메시지를 보낸다. (20번 라인)

20	0.538653310	127.0.0.1	127.0.0.1	HTTP	583 GET /dash/video02.mpd HTTP/1.1
21	0.538656261	127.0.0.1	127.0.0.1	TCP	66 5000 → 44546 [ACK] Seq=1 Ack=51

3) HTTP 응답

서버는 클라이언트 요청에 대한 HTTP 응답 메시지를 전송한다. 응답에는 DASH 동영상에 대한 메타데이터와 함께 DASH 매니페스트 파일(.mpd)이 포함된다. DASH 동영상 서버에 위치하며 클라이언트가 동영상을 요청 시 클라이언트에게 제공되는 매니페스트는, DASH 동영상의 구성 요소와 세그먼트에 대한 정보를 담은 파일이다. 각 세그먼트는 동영상의 작은 조각을 나타낸다.

24	0.544783682	127.0.0.1	127.0.0.1	HTTP	2138 HTTP/1.1 200 OK
25	0.544790755	127.0.0.1	127.0.0.1	TCP	66 44546 → 5000 [ACK]

그런데 아래와 같이 200이 아닌 304가 뜨는 경우가 있다.

0.081248796	127.0.0.1	127.0.0.1	HTTP	340 HTTP/1.1 304 NOT MODIFIED
0.081254072	127.0.0.1	127.0.0.1	TCP	66 44546 → 5000 [ACK] Seq=615 Ack=

```

▶ Frame 20: 340 bytes on wire (2720 bits), 340 bytes captured (2720 bits) on interface lo, id 0
▶ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▶ Transmission Control Protocol, Src Port: 5000, Dst Port: 48834, Seq: 1, Ack: 615, Len: 274
▼ Hypertext Transfer Protocol
  HTTP/1.1 304 NOT MODIFIED\r\n
    [Expert Info (Chat/Sequence): HTTP/1.1 304 NOT MODIFIED\r\n]
    [HTTP/1.1 304 NOT MODIFIED\r\n]
    [Severity level: Chat]
    [Group: Sequence]
    Response Version: HTTP/1.1
    Status Code: 304
    [Status Code Description: Not Modified]
    Response Phrase: NOT MODIFIED
    Server: Werkzeug/2.3.4 Python/3.8.10\r\n
    Date: Sun, 21 May 2023 15:42:11 GMT\r\n
    Content-Disposition: inline; filename=video02.mpd\r\n
    Cache-Control: no-cache\r\n
    ETag: "1684148077.0-2072-3458666916"\r\n
    Date: Sun, 21 May 2023 15:42:11 GMT\r\n
    Connection: close\r\n
    \r\n
    [HTTP response 1/1]
    [Time since request: 0.001803060 seconds]
    [Request in frame: 18]
    [Request URI: http://127.0.0.1:5000/dash/video02.mpd]
  
```

HTTP 상태 코드 304 (Not Modified)는 클라이언트가 이전에 요청한 리소스가 변경되지 않았음을 나타낸다. 처음 .mpd 파일을 요청&응답할 때 브라우저 캐시에 해당 .mpd 파일이 남아있어서, 클라이언트가 동일한 .mpd 파일을 다시 요청할 경우 서버는 304 코드를 반환하며 실제 파일 데이터를 다시 전송하지 않는다고 한다.

4) DASH 세그먼트 요청

클라이언트는 매니페스트에 나열된 각 세그먼트에 대해 비디오 세그먼트 파일(.m4s)을 요청하는 HTTP GET 요청을 보낸다. 클라이언트는 필요한 세그먼트를 지속적으로 요청하여 전체 동영상을 받게 된다.

5) DASH 세그먼트 응답

서버는 클라이언트의 세그먼트 요청에 대해서 비디오 세그먼트 파일(.m4s)을 포함한 HTTP 응답을 전송한다.

클라이언트는 수신한 세그먼트 파일을 재생해 동영상을 렌더링한다. DASH는 동적 스트리밍을 제공하므로, 클라이언트는 플레이어가 적절한 세그먼트를 요청하여 품질을 조절할 수 있도록 매니페스트 파일을 주기적으로 업데이트한다.

http.request.uri contains ".m4s" or http.request.uri contains ".mpd" or http					
Time	Source	Destination	Protocol	Length	Info
4 0.000365590	127.0.0.1	127.0.0.1	HTTP	716	GET /dash HTTP/1.1
8 0.005507911	127.0.0.1	127.0.0.1	HTTP	485	HTTP/1.1 200 OK (text/html)
20 0.538653310	127.0.0.1	127.0.0.1	HTTP	583	GET /dash/video02.mpd HTTP/1.1
24 0.544783682	127.0.0.1	127.0.0.1	HTTP	2138	HTTP/1.1 200 OK
32 0.654446304	127.0.0.1	127.0.0.1	HTTP	588	GET /dash/init-stream0.m4s HTTP/1.1
36 0.660304499	127.0.0.1	127.0.0.1	HTTP	899	HTTP/1.1 200 OK
44 0.664996659	127.0.0.1	127.0.0.1	HTTP	588	GET /dash/init-stream1.m4s HTTP/1.1
48 0.668498831	127.0.0.1	127.0.0.1	HTTP	830	HTTP/1.1 200 OK
56 0.705457704	127.0.0.1	127.0.0.1	HTTP	595	GET /dash/chunk-stream0-00001.m4s HTTP/1.1
61 0.710249394	127.0.0.1	127.0.0.1	HTTP	595	GET /dash/chunk-stream1-00001.m4s HTTP/1.1
98 0.723881153	127.0.0.1	127.0.0.1	HTTP	3114	HTTP/1.1 200 OK
100 0.732715677	127.0.0.1	127.0.0.1	HTTP	5536	HTTP/1.1 200 OK
111 0.825225602	127.0.0.1	127.0.0.1	HTTP	595	GET /dash/chunk-stream0-00002.m4s HTTP/1.1

6) TCP 연결 해제

클라이언트와 서버의 단일 세그먼트의 요청과 응답이 완료되면 TCP는 종료 (4-way handshake)된다.

5.2.2 HLS 패킷 분석

http.request.uri contains ".ts" or http.request.uri contains ".m3u8"

HLS도 DASH와 마찬가지로 ‘TCP 연결 → HTTP 기반 세그먼트 요청/응답 → TCP 연결 해제’ 사이클을 동일하게 거친다.

http.request.uri contains ".ts" or http.request.uri contains ".m3u8" or tcp					
No.	Time	Source	Destination	Protocol	Length Info
73	13.399849002	127.0.0.1	127.0.0.1	TCP	74 34440 → 5000 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=...
74	13.399865322	127.0.0.1	127.0.0.1	TCP	74 5000 → 34440 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495...
75	13.399880229	127.0.0.1	127.0.0.1	TCP	66 34440 → 5000 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=10886745...
76	13.400386819	:::1	:::1	TCP	94 49640 → 5000 [SYN] Seq=0 Win=65476 Len=0 MSS=65476 SACK_PERM=...
77	13.400394274	:::1	:::1	TCP	74 5000 → 49640 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
78	13.400459841	127.0.0.1	127.0.0.1	TCP	74 34456 → 5000 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=...
79	13.400467536	127.0.0.1	127.0.0.1	TCP	74 5000 → 34456 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495...
80	13.400474517	127.0.0.1	127.0.0.1	TCP	66 34456 → 5000 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=10886745...
81	13.402301677	127.0.0.1	127.0.0.1	HTTP	741 GET /hls HTTP/1.1
82	13.402320116	127.0.0.1	127.0.0.1	TCP	66 5000 → 34440 [ACK] Seq=1 Ack=676 Win=64896 Len=0 TSval=108867...
83	13.404014204	127.0.0.1	127.0.0.1	TCP	240 5000 → 34440 [PSH, ACK] Seq=1 Ack=676 Win=65536 Len=174 TSval=...
84	13.404165179	127.0.0.1	127.0.0.1	TCP	66 34440 → 5000 [ACK] Seq=676 Ack=175 Win=65408 Len=0 TSval=1088...
85	13.404174099	127.0.0.1	127.0.0.1	HTTP	870 HTTP/1.1 200 OK (text/html)
86	13.404179971	127.0.0.1	127.0.0.1	TCP	66 34440 → 5000 [ACK] Seq=676 Ack=979 Win=64640 Len=0 TSval=1088...
87	13.409372944	127.0.0.1	127.0.0.1	TCP	66 34440 → 5000 [FIN, ACK] Seq=676 Ack=979 Win=65536 Len=0 TSval=...
88	13.409596301	127.0.0.1	127.0.0.1	TCP	66 5000 → 34440 [FIN, ACK] Seq=979 Ack=677 Win=65536 Len=0 TSval=...
89	13.409614226	127.0.0.1	127.0.0.1	TCP	66 34440 → 5000 [ACK] Seq=677 Ack=980 Win=65536 Len=0 TSval=1088...
90	13.489096899	127.0.0.1	127.0.0.1	HTTP	676 GET /hls/index.m3u8 HTTP/1.1
91	13.489121279	127.0.0.1	127.0.0.1	TCP	66 5000 → 34456 [ACK] Seq=1 Ack=611 Win=64896 Len=0 TSval=108867...

1) TCP 연결

클라이언트는 HLS 동영상을 요청하기 위해 우선 서버와 TCP 연결 (3-way handshake)을 설정한다. (위 사진에서 73, 74, 75번 라인)

73	13.399849002	127.0.0.1	127.0.0.1	TCP	74 34440 → 5000 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=...
74	13.399865322	127.0.0.1	127.0.0.1	TCP	74 5000 → 34440 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495...
75	13.399880229	127.0.0.1	127.0.0.1	TCP	66 34440 → 5000 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=10886745...

2) HTTP 요청

클라이언트는 TCP 연결을 통해 서버에게 HLS 동영상에 대한 정보를 요청하는 HTTP GET 요청 메시지를 보낸다. (90번 라인)

3) HTTP 응답

서버는 클라이언트 요청에 대한 HTTP 응답 메시지를 전송한다. 응답에는 HLS 동영상에 대한 메타데이터와 함께 플레이리스트 파일(.m3u8)이 포함된다. 플레이리스트 파일은 HLS 동영상의 시간별로 분할된 세그먼트에 대한 정보를 담고 있다.

클라이언트는 수신한 플레이리스트 파일(.m3u8)을 분석하여 사용 가능한 비디오 세그먼트와 해당 세그먼트의 URL을 확인한다.

99	13.632317322	127.0.0.1	127.0.0.1	TCP	74 34472 → 5000 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=...
100	13.632329029	127.0.0.1	127.0.0.1	TCP	74 5000 → 34472 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495...
101	13.632338904	127.0.0.1	127.0.0.1	TCP	66 34472 → 5000 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=10886747...
102	13.632451714	127.0.0.1	127.0.0.1	HTTP	678 GET /hls/index0.ts HTTP/1.1
103	13.632456414	127.0.0.1	127.0.0.1	TCP	66 5000 → 34472 [ACK] Seq=1 Ack=613 Win=64896 Len=0 TSval=108867...
104	13.635788538	127.0.0.1	127.0.0.1	HTTP	340 HTTP/1.1 304 NOT MODIFIED
105	13.635811481	127.0.0.1	127.0.0.1	TCP	66 34472 → 5000 [ACK] Seq=613 Ack=275 Win=65280 Len=0 TSval=1088...
106	13.636210748	127.0.0.1	127.0.0.1	TCP	66 34472 → 5000 [FIN, ACK] Seq=613 Ack=275 Win=65536 Len=0 TSval=...
107	13.636397250	127.0.0.1	127.0.0.1	TCP	66 5000 → 34472 [FIN, ACK] Seq=275 Ack=614 Win=65536 Len=0 TSval=...
108	13.636404230	127.0.0.1	127.0.0.1	TCP	66 34472 → 5000 [ACK] Seq=614 Ack=276 Win=65536 Len=0 TSval=1088...

4) 세그먼트 요청 (HTTP)

클라이언트는 플레이리스트에 나열된 각 세그먼트에 대해 비디오 세그먼트 파일(.ts)을 요

청하는 HTTP GET 요청을 보낸다.

http.request.uri contains ".ts" or http.request.uri contains ".m3u8"					
No.	Time	Source	Destination	Protocol	Length Info
20	0.091772408	127.0.0.1	127.0.0.1	HTTP	676 GET /hls/index.m3u8 HTTP/1.1
32	0.139520252	127.0.0.1	127.0.0.1	HTTP	678 GET /hls/index0.ts HTTP/1.1
44	0.286706635	127.0.0.1	127.0.0.1	HTTP	678 GET /hls/index1.ts HTTP/1.1
56	0.490190130	127.0.0.1	127.0.0.1	HTTP	678 GET /hls/index2.ts HTTP/1.1
82	12.836770059	127.0.0.1	127.0.0.1	HTTP	678 GET /hls/index3.ts HTTP/1.1
94	21.139980716	127.0.0.1	127.0.0.1	HTTP	678 GET /hls/index4.ts HTTP/1.1
106	29.539929229	127.0.0.1	127.0.0.1	HTTP	678 GET /hls/index5.ts HTTP/1.1
118	37.836321265	127.0.0.1	127.0.0.1	HTTP	678 GET /hls/index6.ts HTTP/1.1
130	46.144735670	127.0.0.1	127.0.0.1	HTTP	678 GET /hls/index7.ts HTTP/1.1
146	62.837710591	127.0.0.1	127.0.0.1	HTTP	678 GET /hls/index8.ts HTTP/1.1

5) 세그먼트 응답 (HTTP)

서버는 클라이언트의 세그먼트 요청에 대해서 비디오 세그먼트 파일(.ts)을 포함한 HTTP 응답을 전송한다.

클라이언트는 수신한 세그먼트 파일을 재생해 동영상을 렌더링한다. 재생 중에 클라이언트는 플레이어가 적절한 세그먼트를 요청하여 품질을 조절할 수 있도록 플레이리스트를 주기적으로 업데이트한다.

6) TCP 연결 해제

클라이언트와 서버의 단일 세그먼트의 요청과 응답이 완료되면 TCP는 종료 (4-way handshake)된다.

아래 사진을 통해 TCP 연결 (3-way handshake)부터 클라이언트의 HTTP 요청, 서버로부터 받는 플레이리스트 및 세그먼트 응답, TCP 연결 해제 (4-way handshake)까지의 사이클을 확인할 수 있고, 실제로 이 사이클이 주기적으로 매 세그먼트 마다 반복된다.

99	13.632317322	127.0.0.1	127.0.0.1	TCP	74 34472 → 5000 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1
100	13.632329029	127.0.0.1	127.0.0.1	TCP	74 5000 → 34472 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495
101	13.632338904	127.0.0.1	127.0.0.1	TCP	66 34472 → 5000 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=10886747...
102	13.632451714	127.0.0.1	127.0.0.1	HTTP	678 GET /hls/index0.ts HTTP/1.1
103	13.632456414	127.0.0.1	127.0.0.1	TCP	66 5000 → 34472 [ACK] Seq=1 Ack=613 Win=64896 Len=0 TSval=108867...
104	13.635788538	127.0.0.1	127.0.0.1	HTTP	340 HTTP/1.1 304 NOT MODIFIED
105	13.635811481	127.0.0.1	127.0.0.1	TCP	66 34472 → 5000 [ACK] Seq=613 Ack=275 Win=65280 Len=0 TSval=1088...
106	13.636210748	127.0.0.1	127.0.0.1	TCP	66 34472 → 5000 [FIN, ACK] Seq=613 Ack=275 Win=65536 Len=0 TSval...
107	13.636397250	127.0.0.1	127.0.0.1	TCP	66 5000 → 34472 [FIN, ACK] Seq=275 Ack=614 Win=65536 Len=0 TSval...
108	13.636404230	127.0.0.1	127.0.0.1	TCP	66 34472 → 5000 [ACK] Seq=614 Ack=276 Win=65536 Len=0 TSval=1088...

이렇게 매 세그먼트마다 TCP 연결이 됐다 해제되므로, 실제로 세그먼트 응답 패킷을 보면 매 src port가 랜덤으로 바뀌는 것도 확인된다. (cf. 클라이언트 src port는 처음 TCP 연결을 요청할 때 클라이언트 측에서 랜덤하게 선택하여 할당)

▶ Transmission Control Protocol, Src Port: 52658, Dst Port: 5000, Seq: 1, Ack: 1, Len: 612
▼ Hypertext Transfer Protocol
▶ GET /hls/index0.ts HTTP/1.1\r\n
▶ Transmission Control Protocol, Src Port: 52660, Dst Port: 5000, Seq: 1, Ack: 1, Len: 612
▼ Hypertext Transfer Protocol
▶ GET /hls/index1.ts HTTP/1.1\r\n
▶ Transmission Control Protocol, Src Port: 52662, Dst Port: 5000, Seq: 1, Ack: 1, Len: 612
▼ Hypertext Transfer Protocol
▶ GET /hls/index2.ts HTTP/1.1\r\n

추가로 조사해본 결과에 따르면, DASH / HLS는 각각 매니페스트 / 플레이리스트를 주기적으로 업데이트하여 새로운 세그먼트를 요청할 수 있다고 한다. 클라이언트가 플레이리스트를 주기적으로 다시 요청하여 업데이트된 세그먼트 목록을 확인하는 경우, 이전 TCP 연결이 유지되며, 새로운 세그먼트 요청을 위해 계속 사용된다.

5.2.3 패킷 분석 비하인드 스토리

사실 프로젝트 초반에는 DASH / HLS 패킷을 식별하기 위해 패킷 헤더의 특정 시그니처를 찾을 수 있을 것이라 생각했다. 그러나 결론적으로는 패킷 헤더 자체에는 HLS 동영상 요청과 관련된 특정 시그니처가 없다는 것을 알게 되었다.

DASH / HLS 동영상 요청은 HTTP 프로토콜을 기반으로 이루어지며, 보통 패킷의 "페이로드"에 있는 HTTP 요청 메시지에 해당 프로토콜과 관련된 특정 정보가 포함되어 있다. **(TODO: 논문있으면 출처 넣으면 좋을 듯)** 따라서 HLS 동영상 요청에 대한 내용을 파악하고자 한다면, Wireshark를 사용하여 HTTP 요청 메시지의 페이로드를 확인하는 것이 일반적인 방법이다.

위에서 언급한 바와 같이, DASH / HLS 프로토콜에 대한 시그니처 정보는 페이로드(Hypertext Transfer Protocol 부분)에서만 확인할 수 있다. 프로젝트 시작 시, DASH와 HLS의 매직 헤더를 찾아내기 위해 패킷 헤더를 상세하게 분석했다. 그러나 우리는 결론적으로 DASH와 HLS의 매직 헤더에 해당하는 특정 시그니처를 발견하지 못했다.

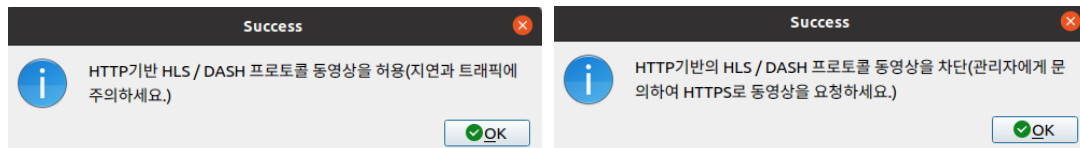
그러나 HTTPS는 트래픽을 암호화하기 때문에 페이로드 데이터를 확인할 수가 없다. 즉, HTTP가 아닌 HTTPS인 경우, Wireshark에서 암호화된 HTTPS 패킷의 내용은 모니터링이 불가하다. 그런데 동영상을 제공하는 거의 대부분의 웹 서비스가 HTTPS를 사용하기 때문에 패킷 필터링 실습에 어려움을 겪었다. 이로 인해 DASH / HLS 프로토콜 기반 동영상을 서비스하는 간단한 스트리밍 서비스를 직접 구축하게 되었다.

5.3 DASH / HLS 패킷 필터링 프로그램 (Cut-Thorough) 개발

Cut-Thorough 프로그램의 메인 화면은 아래 그림과 같다.



‘동영상 허용’, ‘동영상 차단’ 버튼을 클릭하면 다음과 같이 알림을 이벤트가 각각 발생한다.



처음 iptables 리스트를 조회하면 아래 사진과 같이 안에 내용이 비어 있지만

```
ssu20192829@ubuntu:~/python_file$ sudo iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

‘동영상 차단’ 버튼 클릭 이벤트가 발생하면 아래 1~4 번 내용이 추가된 것을 볼 수 있다.

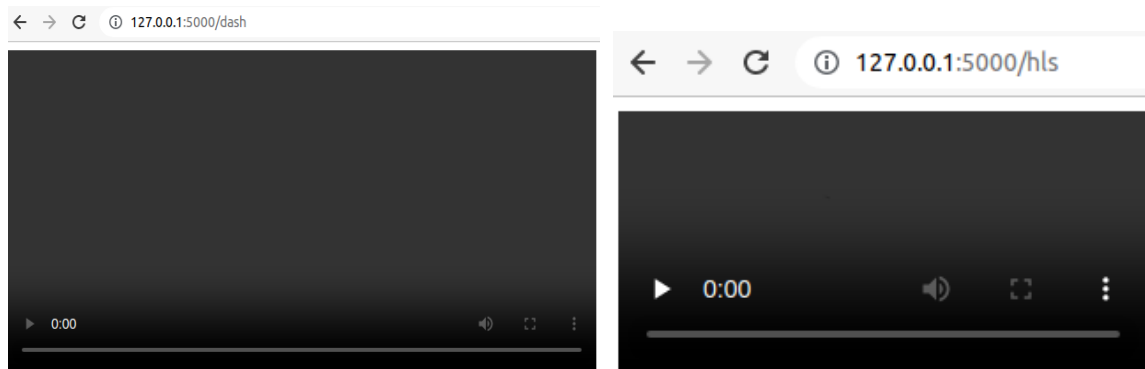
1. `sudo iptables -A INPUT -p tcp --dport 0:65535 -m string --string "GET /" --algo kmp -m string --string ".m3u8" --algo kmp -j DROP`
2. `sudo iptables -A INPUT -p tcp --dport 0:65535 -m string --string "GET /" --algo kmp -m string --string ".ts" --algo kmp -j DROP`
3. `sudo iptables -A INPUT -p tcp --dport 0:65535 -m string --string "GET /" --algo kmp -m string --string ".m4s" --algo kmp -j DROP`
4. `sudo iptables -A INPUT -p tcp --dport 0:65535 -m string --string "GET /" --algo kmp -m string --string ".mpd" --algo kmp -j DROP`


```

lsu20192829@ubuntu:~/python_file$ sudo iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source                destination            tcp STRING match "GET /" ALGO name knp TO 65535 STRING match ".m3u8" ALGO name knp
DROP      tcp  -- anywhere             anywhere               tcp STRING match "GET /" ALGO name knp TO 65535 STRING match ".ts" ALGO name knp T
DROP      tcp  -- anywhere             anywhere               tcp STRING match "GET /" ALGO name knp TO 65535 STRING match ".m4s" ALGO name knp
DROP      tcp  -- anywhere             anywhere               tcp STRING match "GET /" ALGO name knp TO 65535 STRING match ".mpd" ALGO name knp
Chain FORWARD (policy ACCEPT)
target     prot opt source                destination
Chain OUTPUT (policy ACCEPT)
target     prot opt source                destination

```

실행 중인 웹 서버에서 DASH / HLS 기반 동영상을 확인해보면, 아래 그림과 같이 동영상 재생이 불가능한 것을 확인 가능하다.



또한, TCP 는 신뢰성 있는 데이터의 신뢰성과 무결성을 보장하기 위해 사용하는 메커니즘 중 하나는 ‘재전송’ 이다. 재전송은 송신자가 패킷을 전송한 후 일정 시간 동안 확인 응답을 받지 못하거나, 확인 응답이 손실된 경우에 해당 패킷을 다시 전송하는 것을 말한다. 아래 Wireshark 캡처본을 통해 패킷이 차단됐을 때의 패킷 흐름을 확인할 수 있다. 첫 번째 그림 21~26 라인, 28~33 번 라인에서의 video02.mpd 즉, DASH 동영상 패킷에 대해 [TCP Retransmission] 패킷들을 살펴보면, 45240 에서 5000 으로 가는 패킷이 여러 개 보이는 것을 확인할 수 있다. 두 번째 그림 또한, 18~25 라인, 27~28 라인, 30 라인, 32~33 라인에서의 index.m3u8 / video02.mpd 즉, HLS 동영상 패킷에 대해 [TCP Retransmission] 패킷들을 살펴보면, 51726 에서 5000 으로 가는 패킷이 여러 개 보이는 것을 확인할 수 있다. 이처럼 프로토콜은 데이터를 전송하기 전에 SYN 을 보내고, SYN+ACK 가 도착하면 데이터를 전송한다. 그러나 일정 시간 내에 SYN+ACK 를 받지 못하면 해당 SYN 을 다시 보내게 되는데, 이것이 바로 TCP Retransmission (TCP 재전송) 의 기본 원리로 해당 패킷이 지속적으로 발생하는 것을 관찰할 수 있다.

20	1.094141152	127.0.0.1	127.0.0.1	HTTP	583 GET /dash/video02.mpd HTTP/1.1
21	1.298493600	127.0.0.1	127.0.0.1	TCP	583 [TCP Retransmission] 45240 → 5000 [PSH, ACK] Seq=1 Ack=1 Win=...
22	1.298498040	127.0.0.1	127.0.0.1	TCP	583 [TCP Retransmission] 45264 → 5000 [PSH, ACK] Seq=1 Ack=1 Win=...
23	1.500361928	127.0.0.1	127.0.0.1	TCP	583 [TCP Retransmission] 45264 → 5000 [PSH, ACK] Seq=1 Ack=1 Win=...
24	1.724951833	127.0.0.1	127.0.0.1	TCP	583 [TCP Retransmission] 45240 → 5000 [PSH, ACK] Seq=1 Ack=1 Win=...
25	1.922293096	127.0.0.1	127.0.0.1	TCP	583 [TCP Retransmission] 45264 → 5000 [PSH, ACK] Seq=1 Ack=1 Win=...
26	2.557128092	127.0.0.1	127.0.0.1	TCP	583 [TCP Retransmission] 45240 → 5000 [PSH, ACK] Seq=1 Ack=1 Win=...
27	2.619204854	127.0.0.1	127.0.0.1	HTTP	583 GET /dash/video02.mpd HTTP/1.1
28	2.759475694	127.0.0.1	127.0.0.1	TCP	583 [TCP Retransmission] 45264 → 5000 [PSH, ACK] Seq=1 Ack=1 Win=...
29	4.218211541	127.0.0.1	127.0.0.1	TCP	583 [TCP Retransmission] 45240 → 5000 [PSH, ACK] Seq=1 Ack=1 Win=...
30	4.390452575	127.0.0.1	127.0.0.1	TCP	583 [TCP Retransmission] 45264 → 5000 [PSH, ACK] Seq=1 Ack=1 Win=...
31	7.741724334	127.0.0.1	127.0.0.1	TCP	583 [TCP Retransmission] 45264 → 5000 [PSH, ACK] Seq=1 Ack=1 Win=...
32	7.741734045	127.0.0.1	127.0.0.1	TCP	583 [TCP Retransmission] 45240 → 5000 [PSH, ACK] Seq=1 Ack=1 Win=...
33	9.277649858	127.0.0.1	127.0.0.1	TCP	583 [TCP Retransmission] 51942 → 5000 [PSH, ACK] Seq=1 Ack=1 Win=...

17	0.233429745	127.0.0.1	127.0.0.1	HTTP	727 GET /hls/index.m3u8 HTTP/1.1
18	0.448110101	127.0.0.1	127.0.0.1	TCP	727 [TCP Retransmission] 51726 → 5000 [PSH, ACK] Seq=1 Ack=1 Win=...
22	0.653682113	127.0.0.1	127.0.0.1	TCP	727 [TCP Retransmission] 51726 → 5000 [PSH, ACK] Seq=1 Ack=1 Win=...
25	1.100294331	127.0.0.1	127.0.0.1	TCP	727 [TCP Retransmission] 51726 → 5000 [PSH, ACK] Seq=1 Ack=1 Win=...
26	1.935386623	127.0.0.1	127.0.0.1	HTTP	583 GET /dash/video02.mpd HTTP/1.1
27	1.935397344	127.0.0.1	127.0.0.1	TCP	727 [TCP Retransmission] 51726 → 5000 [PSH, ACK] Seq=1 Ack=1 Win=...
28	3.541990891	127.0.0.1	127.0.0.1	TCP	727 [TCP Retransmission] 51726 → 5000 [PSH, ACK] Seq=1 Ack=1 Win=...
29	4.725895133	127.0.0.1	127.0.0.1	HTTP	583 GET /dash/video02.mpd HTTP/1.1
39	6.785745288	127.0.0.1	127.0.0.1	TCP	727 [TCP Retransmission] 51726 → 5000 [PSH, ACK] Seq=1 Ack=1 Win=...
31	10.370653345	127.0.0.1	127.0.0.1	HTTP	583 GET /dash/video02.mpd HTTP/1.1
32	13.430761406	127.0.0.1	127.0.0.1	TCP	727 [TCP Retransmission] 51726 → 5000 [PSH, ACK] Seq=1 Ack=1 Win=...
33	15.226952162	127.0.0.1	127.0.0.1	TCP	583 [TCP Retransmission] 46886 → 5000 [PSH, ACK] Seq=1 Ack=1 Win=...

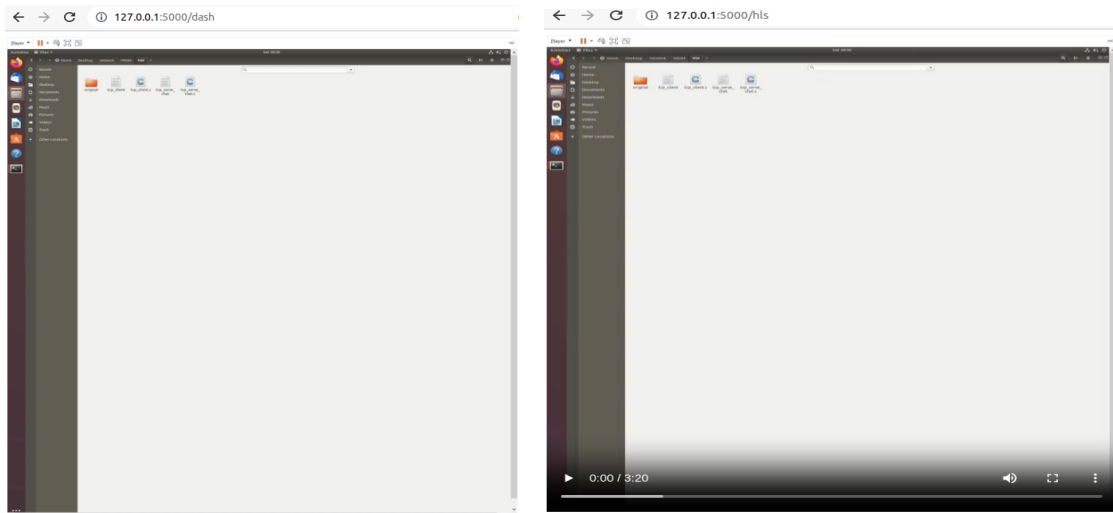
‘동영상 허용’ 버튼 이벤트 발생시 아래 그림과 같이 기존 iptables 의 리스트 목록이 삭제가 된다.

```
ssu20192829@ubuntu:~/python_file$ sudo iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

그리고, 기존에 재생되지 않았던 DASH / HLS 동영상이 재생이 가능한 것을 확인할 수 있다.



5.4 TCP 채팅 차단 기능 추가

5.4.1 TCP Reset Attack 이론 적용하기

프로그램에서 TCP Reset Attack 의 가장 핵심적인 함수인 'send_reset' 을 설명하겠다. send_reset 함수는 TCP RST 패킷을 만들어서 보내는 함수이다. 이 함수는 먼저 패킷의 여러 필드, 즉 소스 IP, 소스 포트, 목적지 IP, 목적지 포트, 시퀀스 번호, 인증 번호 및 TCP 플래그를 추출한다. 그 다음 이 정보를 로깅하고, 패킷이 SYN 플래그를 가지고 있으며 SYN 플래그 무시 옵션이 활성화된 경우 RST 패킷을 보내지 않고 함수를 종료한다. 그리고 시퀀스 번호에 임의의 jitter(변동량)을 더하여 RST 패킷의 시퀀스 번호를 계산한다. 그리고 이를 기반으로 RST 패킷을 생성한다. 패킷 생성시 원본 패킷의 출발지와 목적지를

반전시키고, RST 플래그를 설정한다. 이후 생성된 RST 패킷 정보를 로깅하고 해당 패킷을 네트워크 인터페이스를 통해 전송한다.

5.4.2 TCP Reset Attack Wireshark 패킷 분석

The image displays a terminal window and a Wireshark packet capture. The terminal window shows a chat server script running on a machine with IP 127.0.0.1. The script handles client connections and messages. The Wireshark window shows a packet capture of a TCP reset attack. The packet list shows a sequence of packets, including a TCP reset packet (No. 10) with a source IP of 127.0.0.1 and a destination IP of 127.0.0.1. The packet details pane shows the packet structure, including Ethernet II, Internet Protocol Version 4, and Transmission Control Protocol. The packet bytes pane shows the raw data of the packet.

No.	Time	Source	Destination	Protocol
1	0.000000000	127.0.0.1	127.0.0.1	TCP
2	0.000123551	127.0.0.1	127.0.0.1	TCP
3	0.000139419	127.0.0.1	127.0.0.1	TCP
4	0.000879667	127.0.0.1	127.0.0.1	TCP
5	0.000918168	127.0.0.1	127.0.0.1	TCP
6	0.003678788	127.0.0.1	127.0.0.1	TCP
7	0.003693781	127.0.0.1	127.0.0.1	TCP
8	0.003703750	127.0.0.1	127.0.0.1	TCP
9	0.003762233	127.0.0.1	127.0.0.1	TCP
10	0.003771724	127.0.0.1	127.0.0.1	TCP
11	0.003779374	127.0.0.1	127.0.0.1	TCP
12	0.003851983	127.0.0.1	127.0.0.1	TCP
13	41.868091335	127.0.0.1	127.0.0.53	DNS
14	41.868106204	127.0.0.1	127.0.0.53	DNS
15	41.874626329	127.0.0.53	127.0.0.1	DNS
16	41.874689729	127.0.0.53	127.0.0.1	DNS

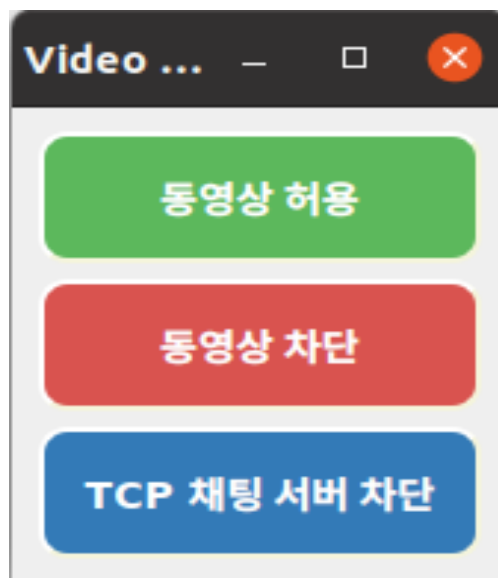
Frame 1: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on
Ethernet II, Src: 00:00:00:00:00:00 (00:00:00:00:00:00), Dst: 00:00:00:00:00:00
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 36662, Dst Port: 8888, Seq: 123456789

TCP 채팅 서버와 두 채팅 클라이언트가 통신하고 있다. Wireshark 에서 연보라색으로 칠해진 부분이다. TCP 채팅 서버와 채팅 클라이언트 코드는 과제 4 의 요구사항에 맞게 프로그래밍 되어 있다.

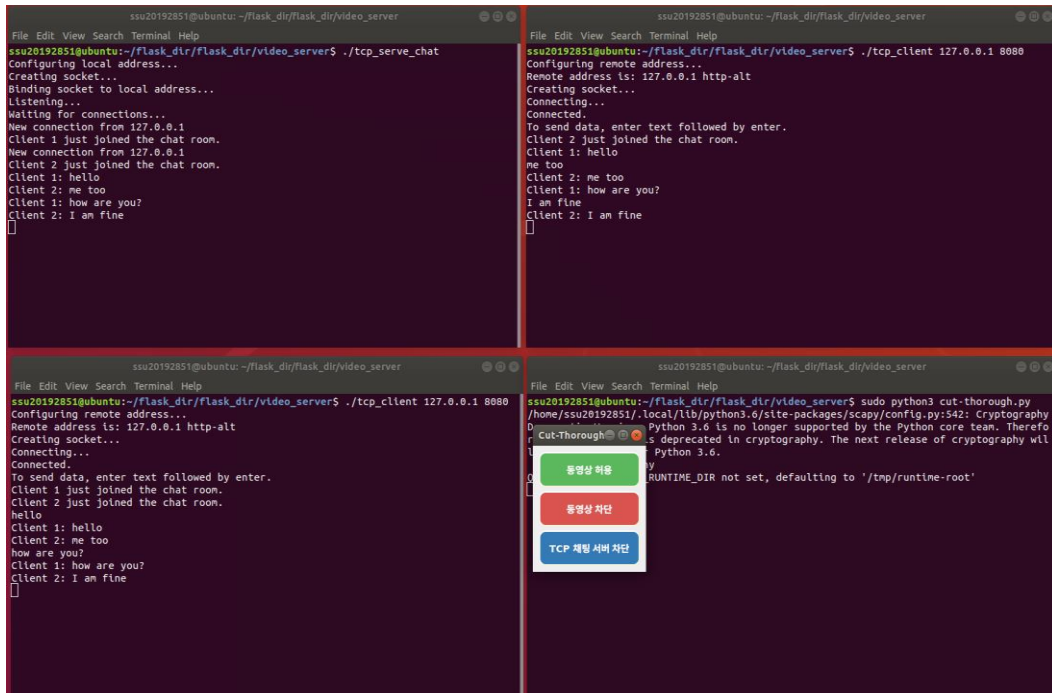
The image shows a Wireshark packet capture of a TCP connection. The top pane displays a list of packets, with packet 129 highlighted in red. This packet is a RST (Reset) packet from source 127.0.0.1 to destination 127.0.0.1, port 8080. The bottom pane shows the details of this packet, including the RST flag and sequence number 3250559807. The packet is labeled as 'ssu20192851@ubuntu:~/flask_dir/flask_dir/video_server'.

클라이언트가 TCP 연결을 종료한 것을 볼 수 있다. 따라서 RST 패킷을 보내어 TCP Reset 공격이 성공한 것을 볼 수 있다.

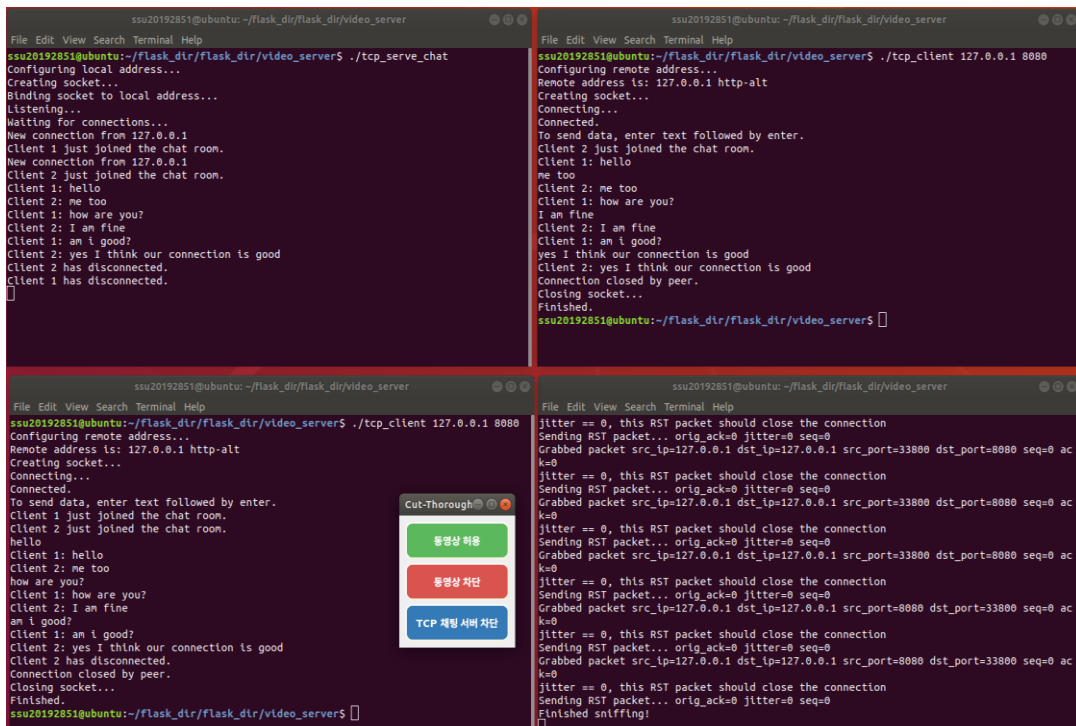
5.4.3 TCP 채팅 서버 차단 버튼 구동 과정



기존 두개의 버튼과 다른 색을 이용하여 TCP 채팅 서버 차단 버튼을 추가했다. 이 버튼을 누르면 50개의 패킷을 스니핑하여 RST 패킷을 송신하여 TCP 채팅 서버를 차단한다. 버튼 구동과정은 다음과 같다.



다음과 같이 하나의 채팅 서버와 두개의 클라이언트 프로그램을 구동시켰다. 서버 프로그램과 클라이언트 프로그램은 네트워크 프로그래밍 과제 #4 의 요구사항에 맞는 TCP 채팅 서버와 클라이언트 프로그램이다. 그리고 “sudo python3 cut-thorough.py” 명령어를 이용해 프로그램을 작동시킨다. 그리고 파란 TCP 채팅 서버 차단 버튼을 누르면 “Starting sniff...” 메시지가 나오면서 앞으로 로컬 환경에서 생기는 다음 50 개의 TCP 패킷을 스니핑한다.



스니핑이 시작되면 클라이언트가 메시지를 보내자마자 프로그램이 패킷을 캡처하여 해당 ip 주소와 포트 번호로 RST 패킷을 보내 TCP 연결을 끊어버리게 한다. 위 사진을 보면 서버에는 클라이언트가 disconnected 됐다는 메시지를 보내고 클라이언트는 TCP 연결이 종료되어 채팅 프로그램이 종료된 것을 볼 수 있다.

6. 프로젝트 소스 코드

6.1 주요 모듈 및 함수 설명

- ◆ 주요 모듈

- PyQt5.QtWidgets: PyQt5에서 GUI 애플리케이션을 개발하기 위한 위젯 클래스들을 제공하는 모듈
- subprocess: 외부 프로세스를 생성하고 제어하는 모듈
- scapy.all: 네트워크 패킷 분석과 조작을 위한 Scapy 라이브러리를 사용
- ifaddr: 네트워크 인터페이스의 IP 주소를 가져오기 위한 모듈
- threading: 멀티스레딩을 지원하는 모듈

- ◆ 주요 함수

- is_adapter_localhost(adapter, localhost_ip): 주어진 어댑터가 로컬 호스트 어댑터인지 확인하는 함수
- is_packet_on_tcp_conn(server_ip, server_port, client_ip): 주어진 서버 IP, 포트 및 클라이언트 IP에 해당하는 TCP 연결에 패킷이 있는지 확인하는 함수
- is_packet_tcp_server_to_client(server_ip, server_port, client_ip): 주어진 서버 IP, 포트 및 클라이언트 IP에 해당하는 TCP를 통해 서버에서 클라이언트로 이동하는지 확인하는 함수
- is_packet_tcp_client_to_server(server_ip, server_port, client_ip): 주어진 서버 IP, 포트 및 클라이언트 IP에 해당하는 TCP를 통해 클라이언트에서 서버로 이동하는지 확인하는 함수
- send_reset(iface, seq_jitter=0, ignore_syn=True): TCP 연결을 리셋하는 패킷을 생성하고 보내는 함수
- run_command(self): "동영상 허용" 버튼이 클릭되었을 때 실행되는 함수입니다. subprocess.run()을 사용하여 iptables 명령어를 실행하고, 실행 결과를 처리하여 성공 메시지를 QMessageBox를 통해 표시하는 함수
- stop_command(self): "동영상 차단" 버튼이 클릭되었을 때 실행되는 함수입니다. subprocess.run()을 사용하여 iptables 명령어를 실행하고, 실행 결과를 처리하여 성공 메시지를 QMessageBox를 통해 표시하는 함수

7. 결론

7.1 프로젝트 성과 및 결과 요약

- HLS/DASH 스트리밍을 허용하거나 차단하는 기능을 제공하는 것을 통해 사용자는 원하는 동영상 스트리밍을 제어할 수 있다. 예를 들어, 특정 동영상 스트리밍을 차단하거나 사용자가 접근할 수 있는 동영상 스트리밍을 제한이 가능하다.
- TCP 연결 유무를 통해 사내, 조직, 단체 내 시스템을 안전한 환경에서의 TCP 연결을 통해 동영상 스트리밍을 전송할 수 있다.
- 프로그램으로 서버에서 클라이언트로 패킷 흐름을 확인하는 것을 통해 시스템은 동영상 스트리밍 중 데이터의 무결성을 보장할 수 있다.
- 다양한 동영상 전송 프로토콜에 대한 종류와 기능을 조사하고 분석할 수 있다.

또한, 이 프로젝트에서는 다양한 동영상 전송 프로토콜에 대한 종류와 기능을 조사하고 분석하였습니다. 이를 통해 프로젝트는 다양한 스트리밍 서비스와의 호환성을 갖추며, 사용자들에게 다양한 동영상 콘텐츠에 대한 접근 제어 및 보안성을 제공할 수 있게 되었다.

이러한 프로젝트의 성과와 결과는 기업이나 조직 내에서 보안 강화, 생산성 향상, 비용 절감 등을 달성할 수 있도록 도움을 줄 것으로 기대된다.

7.2 프로젝트 정의서 대비 주요 변경사항

- 기존 Flow Chart는 기본적으로 프로그램을 동작 시 동영상 차단을 기본 전제로 하였으며, 프로그램 내에서 HTTP차단 기능을 ON할 경우 HTTPS가 아닌 HTTP 프로토콜 기반 DASH / HLS를 사용한 동영상을 클라이언트가 시청한 경우에 사용자에게 HTTPS로 동영상 시청을 권하는 알람을 전송하는 것이었다. 새로운 Flow Chart에선 프로그램을 시작하여 동영상 차단을 ON할 시에 HTTP프로토콜 기반의 DASH / HLS를 사용한 동영상을 차단한다. 클라이언트에게 프로그램을 스위치 ON 중에 DASH / HLS 기반의 동영상을 시청하고 싶다면, 해당 페이지 관리자에게 HTTPS 프로토콜을 사용한 DASH / HLS 기반 동영상을 전송 요청 알람을 보내는 것으로 진행되는 것으로 바뀌었다.
- TCP Reset Attack에 대한 기능은 프로젝트 정의서에 미리 정의하진 않았으나 해당 TCP 채팅서버와의 연결을 차단하는 프로그램을 개발하는 것이 기업과 같은 조직 내부에서의 기밀 유출을 방지에 효과적일 것 같아 기능을 추가하였다.

8. 참고 문헌

- (1) 정상호, 2011, HTTP상에서 동적 적응적 스트리밍 소프트웨어 개발
- (2) 한국전자통신연구원, 2011, HTTP상에서 동적 적응적 스트리밍 소프트웨어 개발