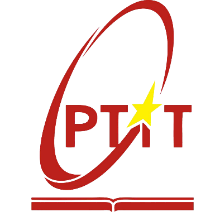
**HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG**

**KHOA AN TOÀN THÔNG TIN**

-------🙞🙜🕮🙞🙜-------



**CÁC KỸ THUẬT GIẤU TIN**

**Đề bài: Ứng dụng H264 trong video streaming**

|  |  |
| --- | --- |
| **HỌ TÊN:** | **PHẠM HẢI DƯƠNG** |
| **MÃ SINH VIÊN:** | **B21DCAT072** |

Giảng viên hướng dẫn: PGS.TS. Đỗ Xuân Chợ

**HÀ NỘI, 04/2025**

**MỤC LỤC**

[1. Giới thiệu chung về bài thực hành 5](#_Toc196590796)

[2. Nội dung và hướng dẫn bài thực hành 5](#_Toc196590797)

[1.1. Mục đích 5](#_Toc196590798)

[1.2. Yêu cầu đối với sinh viên 6](#_Toc196590799)

[1.3. Nội dung thực hành 6](#_Toc196590800)

[3. Phân tích yêu cầu của bài thực hành 8](#_Toc196590801)

[4. Cài đặt và cấu hình máy ảo 9](#_Toc196590802)

[5. Tích hợp và triển khai 14](#_Toc196590803)

[5.1. Docker Hub 14](#_Toc196590804)

[5.2. GitHub 14](#_Toc196590805)

[6. Thử nghiệm và đánh giá 15](#_Toc196590806)

**DANH MỤC HÌNH ẢNH**

[Hình 1: Giao diện Labedit 9](#_Toc196590811)

[Hình 2: Cấu hình result 10](#_Toc196590812)

[Hình 3: Cấu hình Result 10](#_Toc196590813)

[Hình 4: Cấu hình Goal 11](#_Toc196590814)

[Hình 5: Docker container bob 11](#_Toc196590815)

[Hình 6: Docker container bob 12](#_Toc196590816)

[Hình 7: Docker container alice 13](#_Toc196590817)

[Hình 8:Docker container alice 13](#_Toc196590818)

[Hình 9: Docker 14](#_Toc196590819)

[Hình 10: Đẩy lab lên github 15](#_Toc196590820)

# **Giới thiệu chung về bài thực hành**

Bài thực hành này được thiết kế nhằm giúp sinh viên hiểu rõ cách ứng dụng chuẩn mã hóa H.264-AVC trong truyền phát video (video streaming), đồng thời phân tích mức độ an toàn và khả năng bảo toàn dữ liệu khi truyền qua các giao thức khác nhau như RTMP và SRT. Thông qua quá trình thực hành, sinh viên sẽ được rèn luyện kỹ năng sử dụng công cụ FFmpeg, thiết lập kênh truyền phát trực tiếp, và đánh giá hiệu quả bảo mật của từng phương thức truyền tải.

H.264-AVC là chuẩn mã hóa video phổ biến với hiệu suất nén cao, được ứng dụng rộng rãi trong các hệ thống truyền hình, giám sát, họp trực tuyến và các nền tảng streaming. Chuẩn này cho phép truyền video chất lượng cao với băng thông tối ưu và hỗ trợ đa dạng định dạng đóng gói như MP4, MPEG-TS, hay FLV, đặc biệt phù hợp với các giao thức truyền thời gian thực.

Trong bài thực hành này, sinh viên sẽ thiết lập một luồng video H.264 và thực hiện truyền phát trực tiếp qua mạng nội bộ bằng giao thức RTMP (không mã hóa) và SRT (hỗ trợ mã hóa đầu-cuối). Sinh viên sẽ kiểm tra khả năng bắt gói, trích xuất và phân tích dữ liệu nhúng (ví dụ như thông tin giấu trong SEI NALU), từ đó đánh giá khả năng bảo vệ dữ liệu của từng giao thức khi đối mặt với các tình huống nghe trộm trong mạng cục bộ.

Thông qua bài thực hành, sinh viên sẽ nắm được quy trình thiết lập hệ thống video streaming, hiểu rõ vai trò của NAL Unit trong H.264, cách nhúng và bảo toàn dữ liệu phụ trợ như SEI, và đánh giá các rủi ro bảo mật liên quan đến kênh truyền. Bên cạnh đó, sinh viên cũng sẽ tiếp cận được cách sử dụng các công cụ phân tích mạng và xử lý video như tcpdump, ffmpeg, và strings để xác minh mức độ an toàn của quá trình truyền phát.

Sau khi hoàn thành bài thực hành, sinh viên sẽ có cái nhìn tổng quan về kiến trúc truyền phát video thời gian thực, hiểu sâu hơn về đặc điểm kỹ thuật của chuẩn H.264, và rèn luyện kỹ năng đánh giá bảo mật trong môi trường mạng thực tế – nền tảng quan trọng cho các ứng dụng trong bảo vệ nội dung số, forensics và truyền thông đa phương tiện an toàn.

# **Nội dung và hướng dẫn bài thực hành**

## **Mục đích**

* Hiểu tổng quan về quy trình mã hóa và giải mã video theo chuẩn H264-AVC.
* Nắm vững các bước thực hành sử dụng công cụ và các thuật toán liên quan đến mã hóa video trong chuẩn H264-AVC.
* Áp dụng kỹ thuật giấu tin bằng cách chèn thông tin vào trường SEI (Supplemental Enhancement Information) của luồng H264.
* Biết được cách ứng dụng của video h264 vào phát video trực tuyến

## **Yêu cầu đối với sinh viên**

* Có kiến thức cơ bản về kỹ thuật giấu tin trong đa phương tiện (steganography), đặc biệt là trên video.
* Hiểu rõ khái niệm, cấu trúc luồng dữ liệu và các ứng dụng của chuẩn video H264-AVC trong thực tế.
* Thành thạo việc sử dụng các công cụ xử lý media và các script Python hỗ trợ xử lý dữ liệu H264.

## **Nội dung thực hành**

Sinh viên khởi động bài lab

Chạy lệnh:

*labtainer -r stego\_h264\_video\_sei\_code*

*(Chú ý: sinh viên sử dụng <TÊN\_TÀI\_KHOẢN> của mình để nhập thông tin người thực hiện bài lab khi có yêu cầu, để sử dụng khi chấm điểm.)*

Sau khi khởi động bài lab,có 3 container hiện lên là bob và alice. Sinh viên thực hiện làm lab theo yêu cầu trên 3 container này

***TASK 1: Giấu tin vào trong video h264***

* Trong terminal streamer đã có sẵn 1 file video\_h264.mp4.
* Ta tiến hành chuyển đổi nó thành dạng raw h264,phục vụ cho việc giấu tin sau này

***ffmpeg -i video\_h264.mp4 -c:v copy -bsf:v h264\_mp4toannexb raw.h264***

* Script python patch\_sei sẽ giúp ta giấu 1 thông điệp vào phần SEI trong raw file ta vừa tạo ra (xem chi tiết cách giấu tin trong phần SEI trong bài lab stego\_h264\_video\_sei\_code): **python3 patch\_sei.py raw.h264 steg.h264 "rtmp not secure, srt is better"** (thông điệp được giấu là rtmp not secure, srt is better)
* Sau khi hona ftất giấu tin, ta đóng gói raw file thành dạng video mp4 để sử dụng trong phát video trực tuyến sau này: **ffmpeg -f h264 -i steg.h264 -c:v copy stream.mp4**

***TASK 2: Cài đặt môi trường phát video trực tuyến RTMP***

* RTMP (Real-Time Messaging Protocol) là một giao thức truyền tải dữ liệu do Adobe phát triển, được thiết kế chủ yếu cho việc truyền phát âm thanh, video và dữ liệu điều khiển trong thời gian thực giữa máy chủ và client. RTMP hoạt động dựa trên giao thức TCP và thường sử dụng cổng 1935.
* Trong lĩnh vực video streaming, RTMP được sử dụng phổ biến để đẩy luồng video từ client (streamer) lên máy chủ (ví dụ: NGINX RTMP Module), từ đó máy chủ có thể phân phối video đến các người xem (viewer) hoặc các dịch vụ CDN.
* Một số đặc điểm chính của RTMP:
  + Kết nối liên tục (persistent TCP connection), giúp giảm độ trễ khi truyền tải luồng.
  + Chia luồng thành các "message" nhỏ, hỗ trợ truyền đa phương tiện hiệu quả.
  + Hỗ trợ metadata và control command, phù hợp cho việc truyền thông tin bổ sung ngoài dữ liệu hình ảnh.
* Streamer tiến hành config nginx rtmp để có thể tiến hành phát video trực tuyến: ***sudo nano /etc/nginx/nginx.conf***

Thêm phân này vào cuối file

rtmp {

server {

listen 1935;

chunk\_size 4096;

application live {

live on;

record off;

access\_log /home/ubuntu/rtmp\_access.log;

}

}

}

* Luu file và khởi động lại nginx: **sudo systemctl restart nginx**

***TASK 3: Phát video trực tuyến***

* Trên terminal viewer, tiến hành chạy trình phát vlc để có thể xem video phát trực tuyến:

***cvlc --no-audio --avcodec-hw=none -q rtmp://<IP streamer>/live/stream 2>vlc.log***

* Tiến hành stream trên terminal streamer: ***ffmpeg -re -i stream.mp4 -c copy -b:v 800k -r 24 -f flv rtmp://localhost/live/stream***
* Sẽ có 1 vài đỗ trễ, nhưng ta sẽ thấy cửa sổ của trình phát VLC hiển thị trên terminal viewer phát video stream.mp4 lưu trên máy streamer

***TASK 4: Tấn công vào RTMP***

* RTMP tồn tại một số hạn chế về bảo mật:
  + Không mã hóa dữ liệu theo mặc định (trừ khi dùng RTMPS – phiên bản sử dụng SSL/TLS).
  + Dễ bị nghe lén nếu attacker có thể truy cập vào tuyến mạng giữa streamer và viewer.
  + Không có cơ chế xác thực và bảo vệ nội dung nâng cao như các giao thức hiện đại (SRT, HLS...).
* Do đó, trong các môi trường yêu cầu an toàn và bảo mật cao, RTMP dần được thay thế bởi các giao thức bảo mật hơn như SRT (Secure Reliable Transport) hoặc HLS (HTTP Live Streaming).
* Trên terminal attacker, kẻ tấn công sẽ cố gắng ghi lại video mà streamer phát trực tuyến: ***ffmpeg -i rtmp://<IP streamer>/live/stream -c copy -bsf:v h264\_mp4toannexb -f h264 captured.h264***
* Sau streamer phát video trực tuyến: ***ffmpeg -re -i stream.mp4 -c copy -b:v 800k -r 24 -f flv rtmp://localhost/live/stream*** , trên máy attacker đax xuất hiện video captured.h264.
* Attacker thực hiện kiểm tra file thu được: ***strings captured.h264 > rtmp\_attack.log***
* Hãy nhớ rằng tước khi stream, ta đã giấu 1 thông điệp vào video stream.mp4. Kiểm tra xem attacker có thể lấy được thông điệp đó không?

***Cat rtmp\_attack.log | grep “rtmp”***

***TASK 5: Stream sử dụng SRT***

* SRT (Secure Reliable Transport) là một giao thức truyền tải video mã nguồn mở do Haivision phát triển, được thiết kế để truyền dữ liệu trực tuyến (live streaming) với độ trễ thấp, độ tin cậy cao và bảo mật mạnh mẽ qua các mạng không ổn định như Internet.
* Đặc điểm nổi bật của SRT:
  + Bảo mật:
    - Hỗ trợ mã hóa AES-128/AES-256 giúp bảo vệ luồng truyền khỏi nghe lén và can thiệp.
    - Hỗ trợ passphrase để giới hạn quyền truy cập luồng.
  + Độ tin cậy cao:
    - Dựa trên giao thức UDP, kết hợp với cơ chế ARQ (Automatic Repeat reQuest) giúp đảm bảo dữ liệu được truyền chính xác và đầy đủ ngay cả khi mạng không ổn định.
    - Cơ chế packet loss recovery (phục hồi gói tin bị mất) và jitter buffer (bộ đệm độ trễ) giúp cải thiện chất lượng truyền tải.
  + Độ trễ thấp (Low-latency):
    - Tối ưu cho các ứng dụng truyền phát trực tiếp (live), chẳng hạn như hội nghị truyền hình, phát sóng thể thao, và giám sát video.
  + NAT traversal:
    - SRT hỗ trợ chế độ caller/listener và rendezvous, giúp thiết lập kết nối xuyên NAT hoặc firewall mà không cần cấu hình phức tạp.
* Ở đây, streamer sẽ thực hiện phát video trực tuyến cho viewer, và attacker ở giữa sẽ cố gắng thực hiện nghe lén.
* Attacker sử dụng tcp quan sát, bắt các gói tin truyền trong mạng để thực hiện nghe lén: ***sudo tcpdump -i eth0 port 9000 -w sniffed\_srt.pcap -Z root***
* Viewer chờ kết nối stream với streamer bằng SRT ở mode listener: ***ffmpeg -i "srt://:9000?mode=listener&passphrase=SecureReliableTransport" -c copy -bsf:v h264\_mp4toannexb -f h264 record.h264***
* Streamer phát video trực tuyến cho viewer, nhưng sử dụng SRT thay vì RTMP ***:*** ***ffmpeg -re -i /home/ubuntu/steg.h264 -c copy -bsf:v h264\_mp4toannexb -f h264 "srt://<Ip viewer>:9000?pkt\_size=1316&passphrase=SecureReliableTransport"***
* Sau khi stream kết thúc, trên máy viewer ta tiến hành kiểm tra thông điệp được giấu trong video stream:

***strings record.h264 > record.log***

***strings record.h264 | grep "srt"***

* Trên máy attacker, sau khi kết thúc stream của streamer, kết thúc tcpdump. Nhận thấy không quan sát được gì vì SRT có cơ chế mã hóa bằng AES và có passphare

Kết thúc lab:

* Trên terminal khởi động lab, sinh viên sử dụng lệnh:

***Stoplab***

* Khi bài lab kết thúc, một tệp lưu kết quả được tạo và lưu vào một vị trí được hiển thị bên dưới stoplab. Sinh viên cần nộp file .lab để chấm điểm.
* Để kiểm tra kết quả khi trong khi làm bài thực hành sử dụng lệnh:

***checkwork <tên bài thực hành>***

* Sinh viên cần nộp file .lab để chấm điểm.
* Kiểm tra kết quả trong quá trình làm bài:

***checkwork <tên bài lab>***

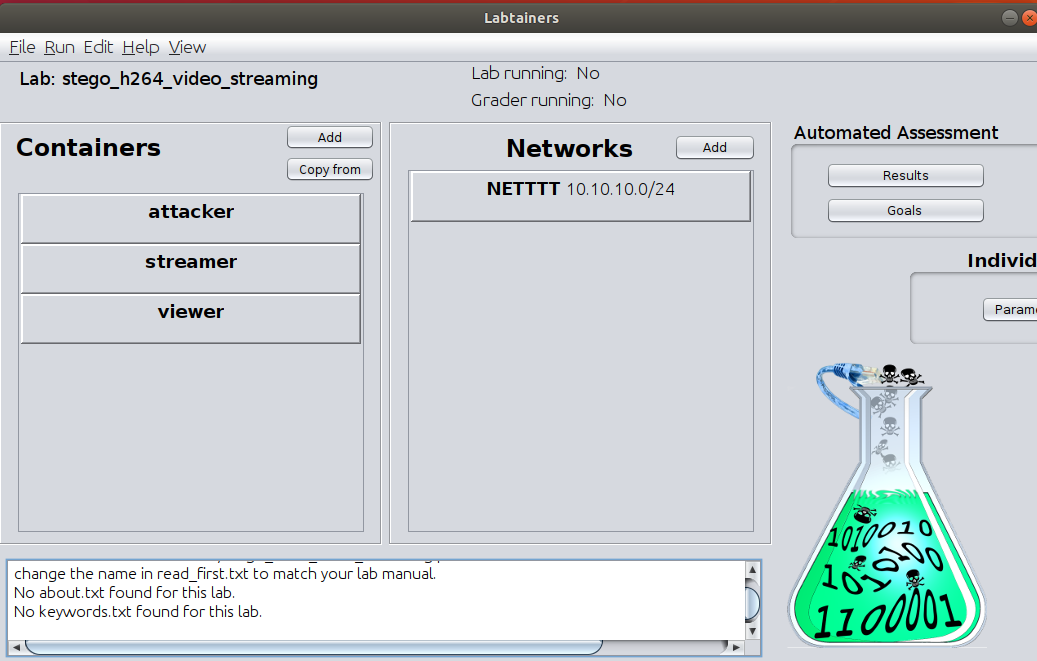
* Khởi động lại bài lab: Trong quá trình làm bài sinh viên cần thực hiện lại bài lab, dùng câu lệnh:

***labtainer -r stego\_h264\_video\_streaming***

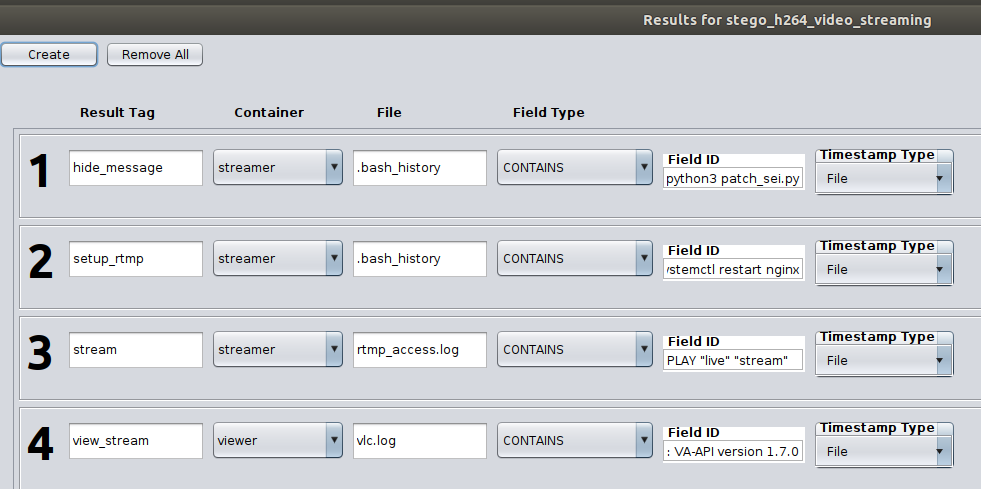
# **Phân tích yêu cầu của bài thực hành**

 Bài thực hành gồm 3 máy tính streamer, viewer và attacker. Để hoàn thành bài thực hành sinh viên cần tiến hành phát được video trực tuyến trên streamer sử dụng RTMP và SRT. Viewer sẽ giải mã được thông điệp giấu trong video khi nhận video stream qua SRT, còn attacker sẽ tấn công vào RTMP đánh cắp thông điệp.

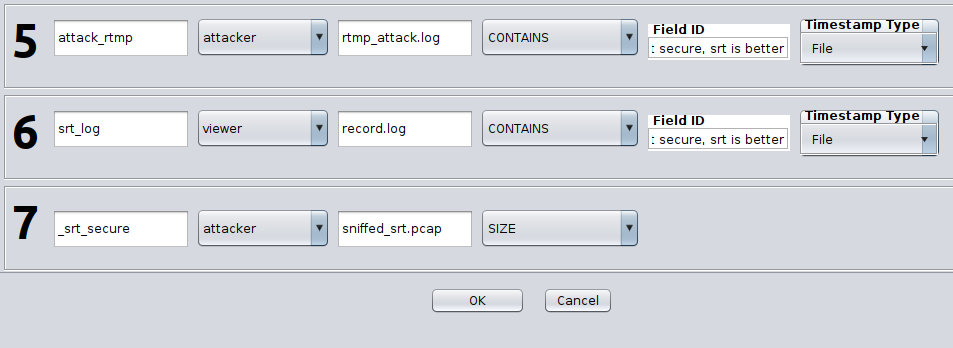
# **Cài đặt và cấu hình máy ảo**



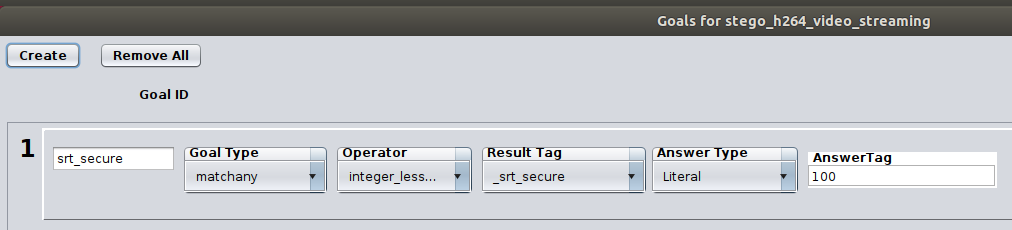
Hình 1: Giao diện Labedit



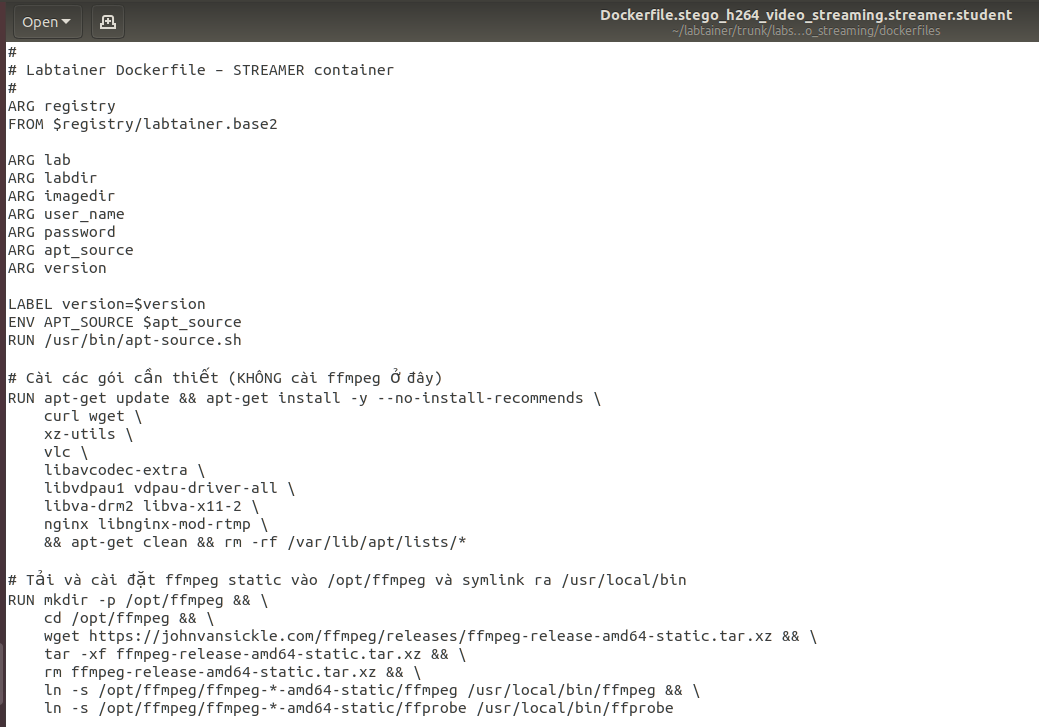
Hình 2: Cấu hình result



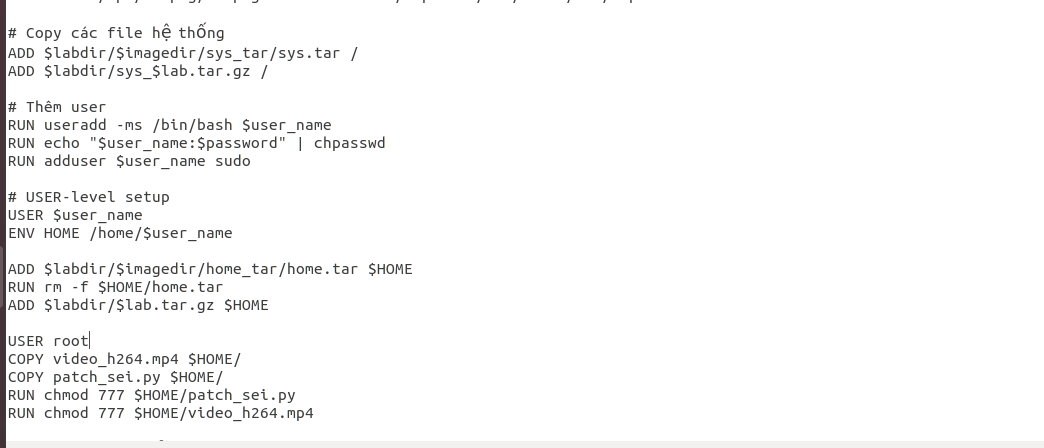
Hình 3: Cấu hình Result



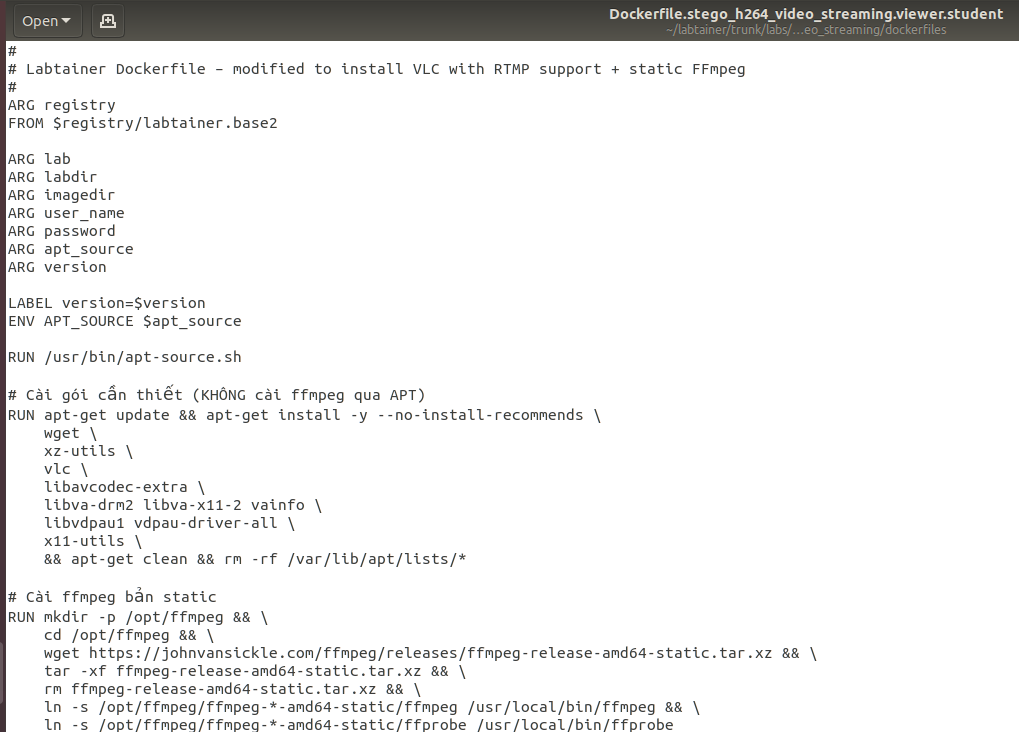
Hình 4: Cấu hình Goal



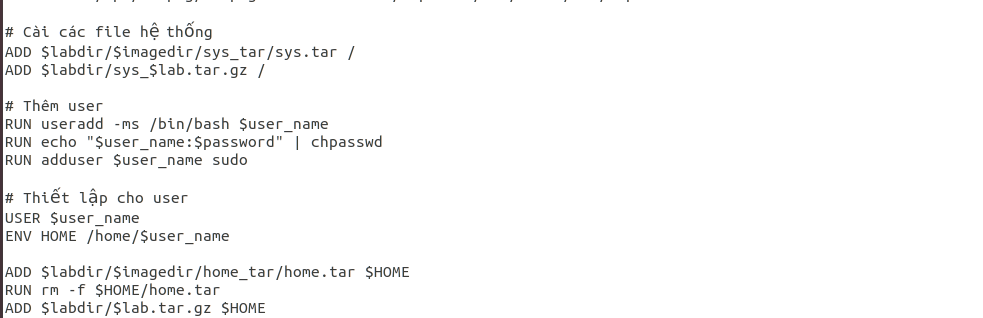
Hình 5: Docker container streamer



Hình 6: Docker container streamer



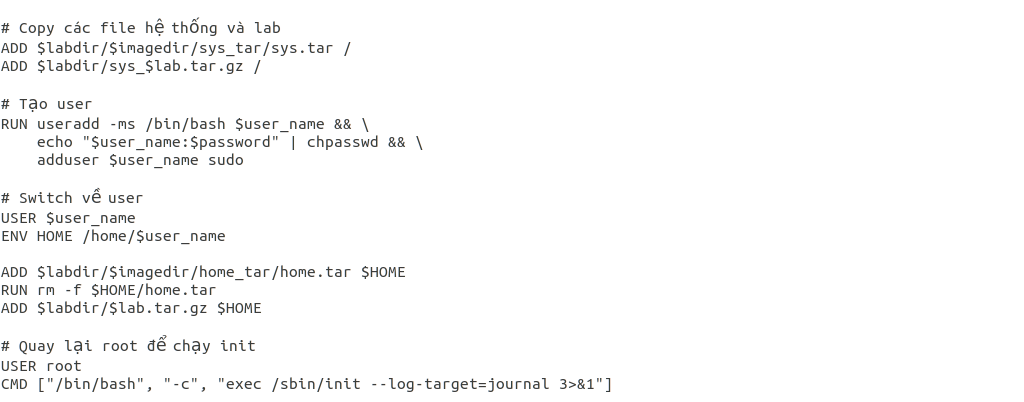
Hình 7: Docker container viewer



Hình 8:Docker container alice



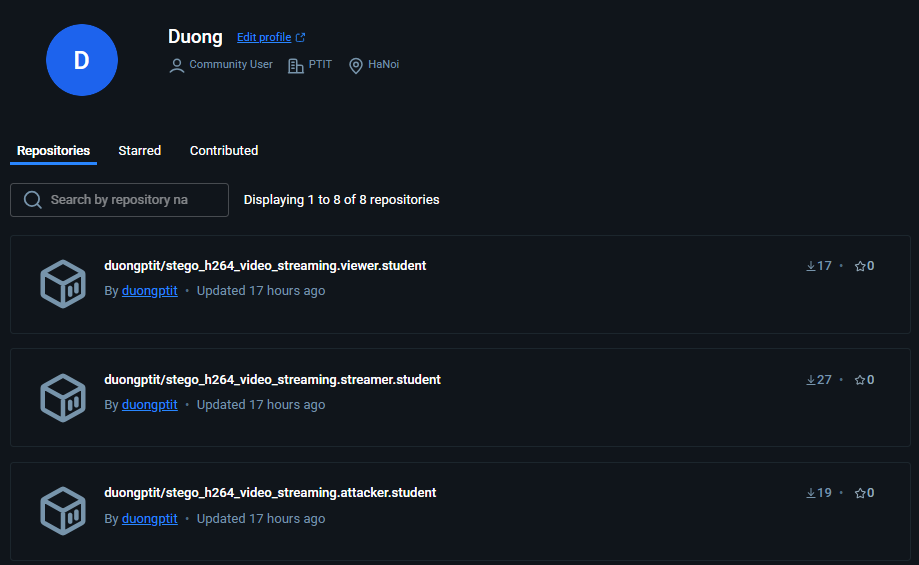
Hình 9: Docker container attacker



Hình 10: Docker container attacker

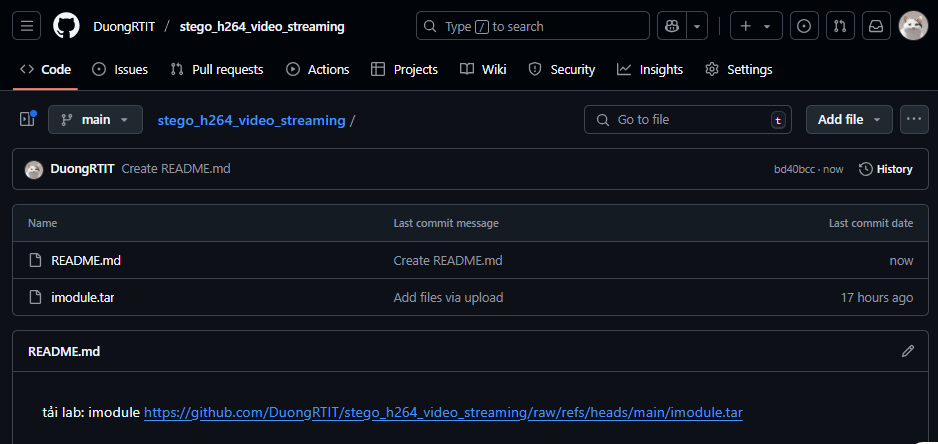
# **Tích hợp và triển khai**

## **Docker Hub**



Hình 11: Docker

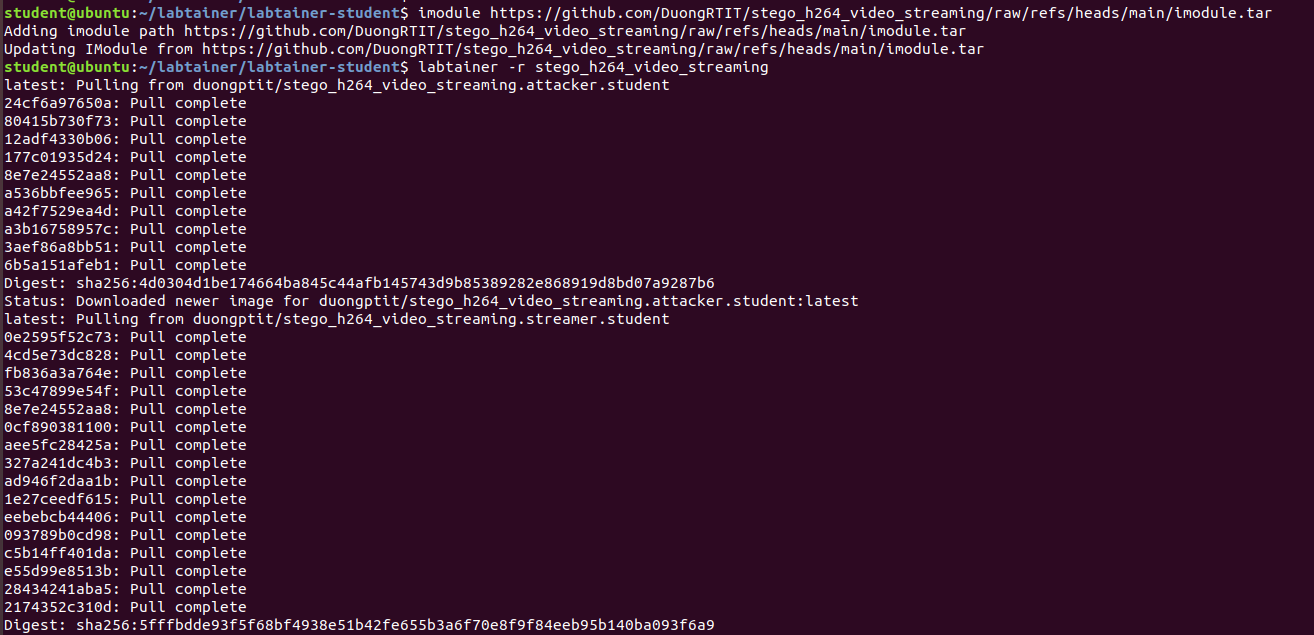
## **GitHub**

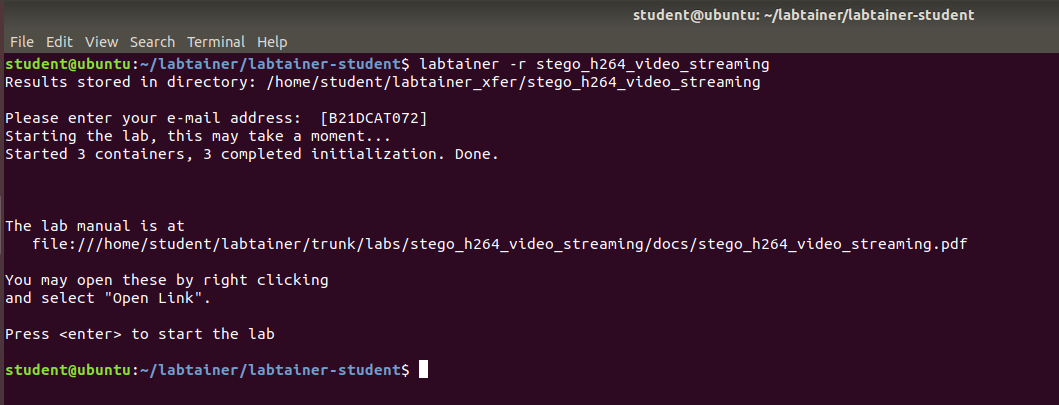


Hình 12: Đẩy lab lên github

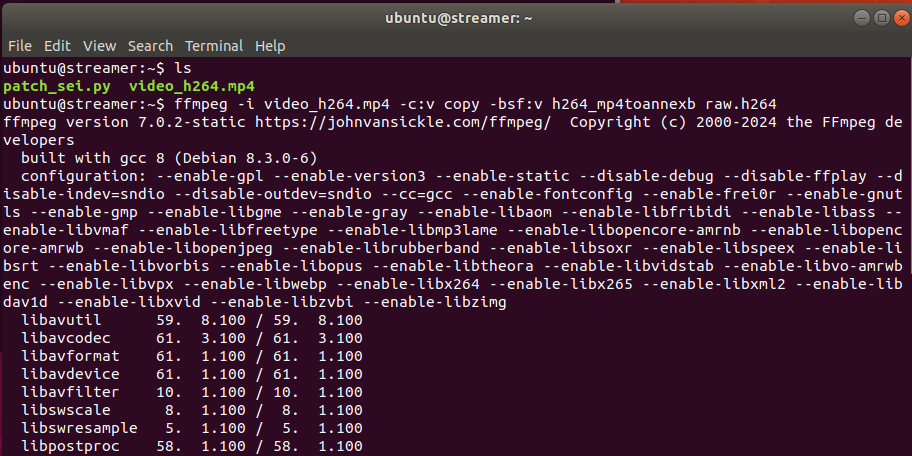
# **Thử nghiệm và đánh giá**

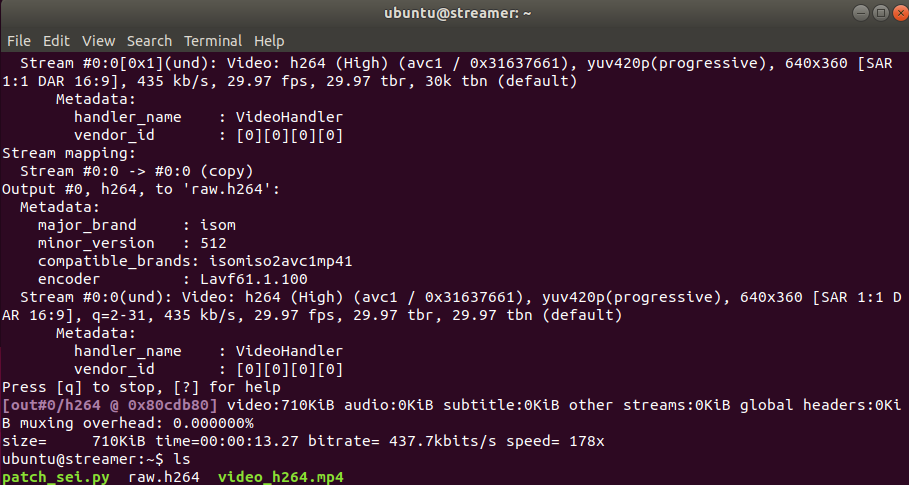
* Tải bài lab về từ Github và chạy bài lab:

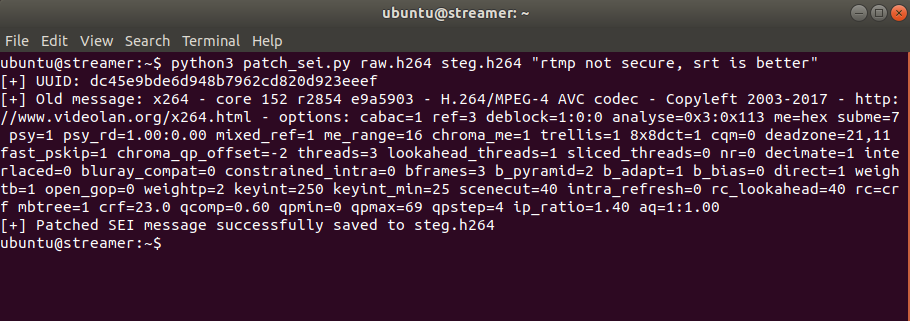


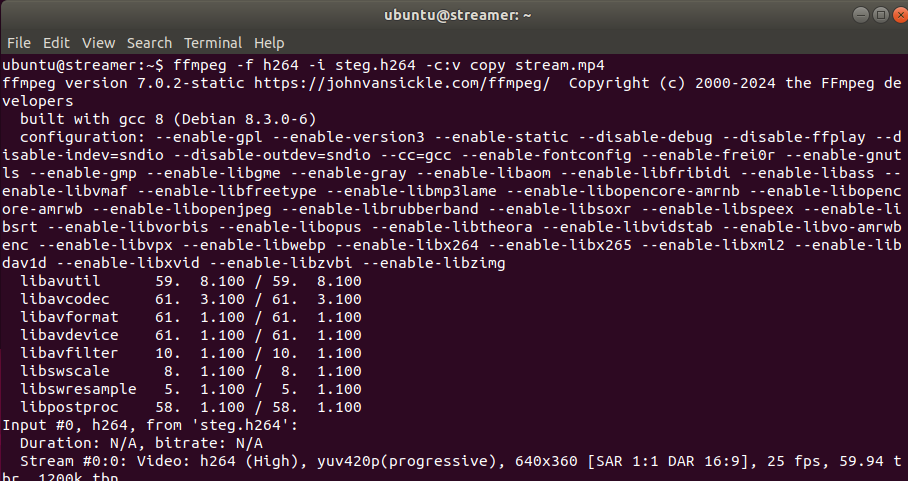


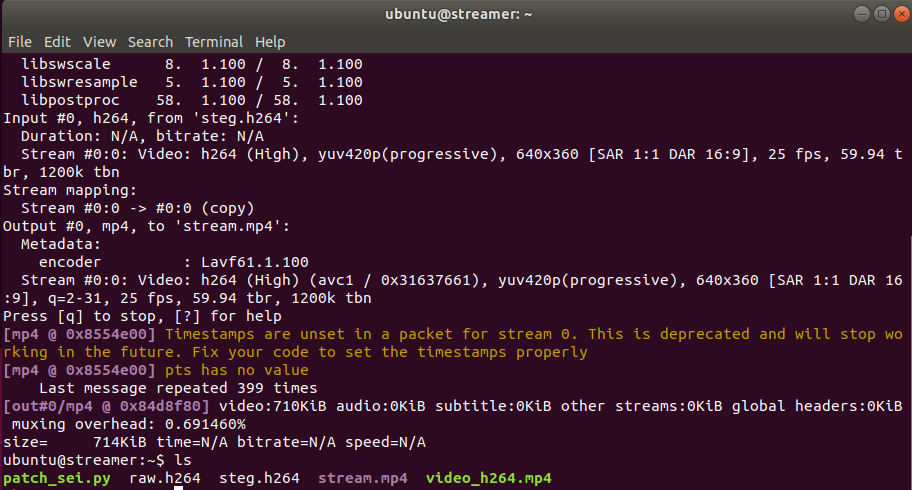
* Thực hiện Task 1:



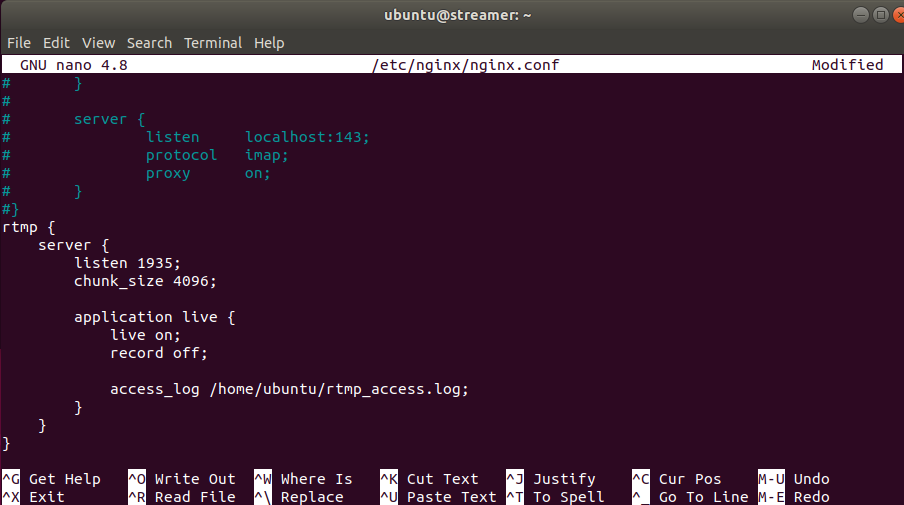


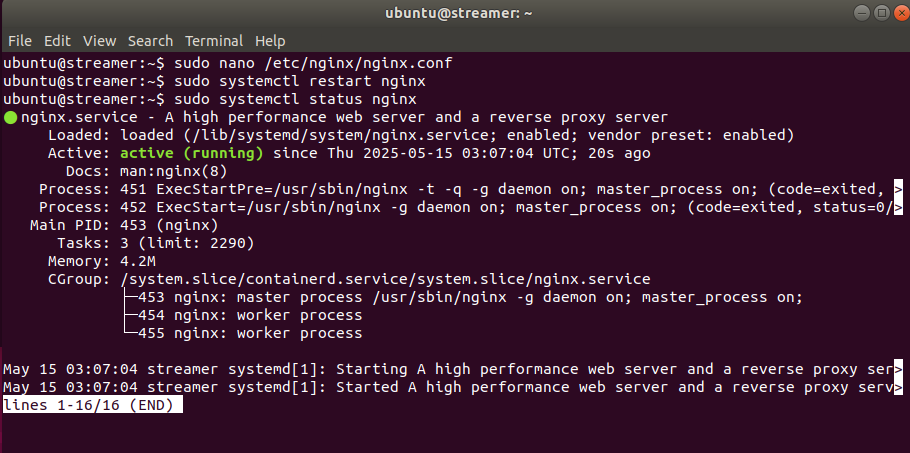




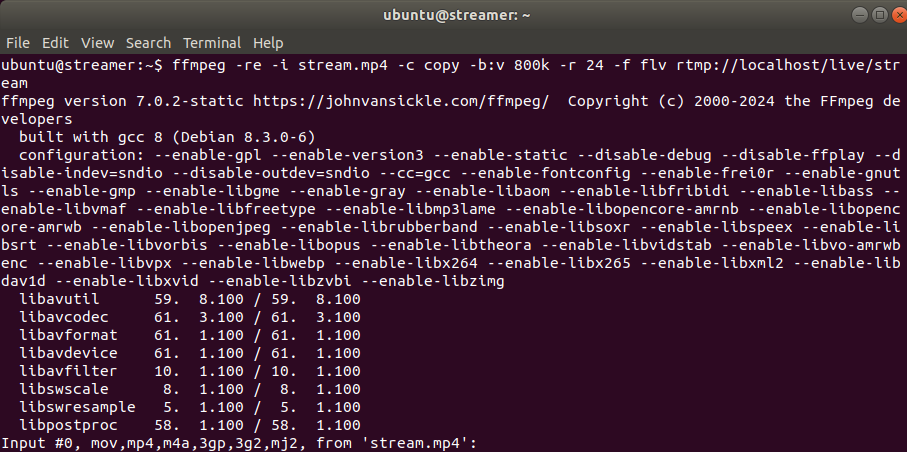


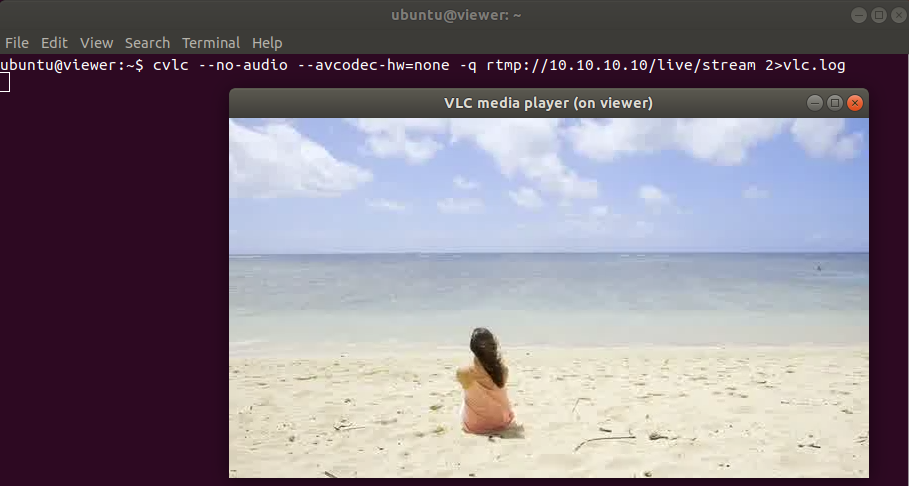
* Thực hiện task 2:

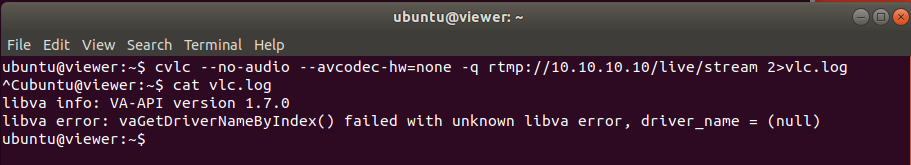


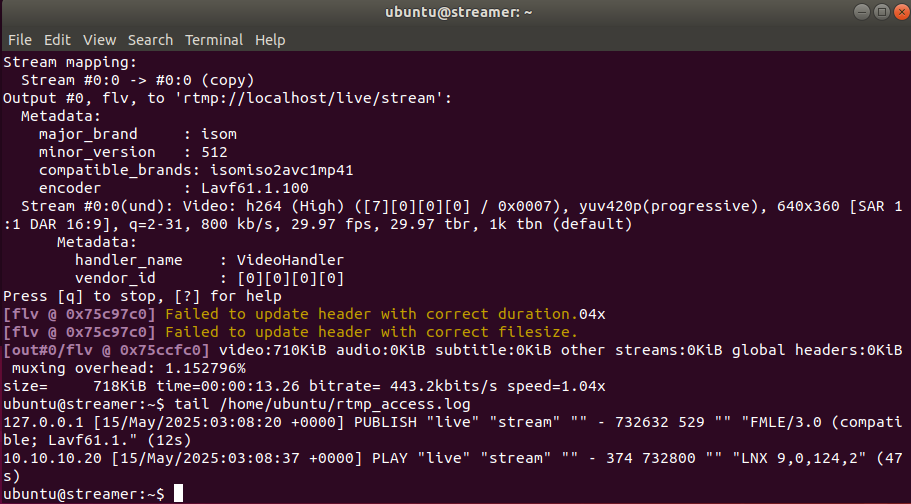


* Thực hiện task 3:

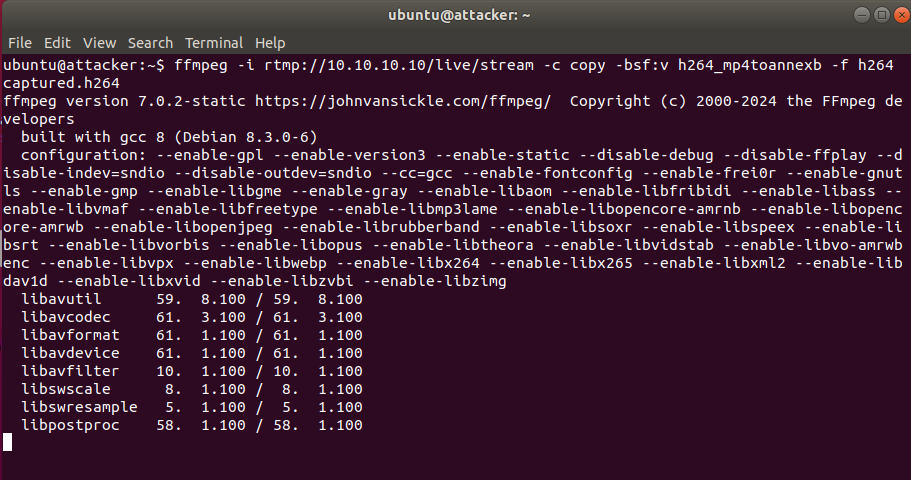


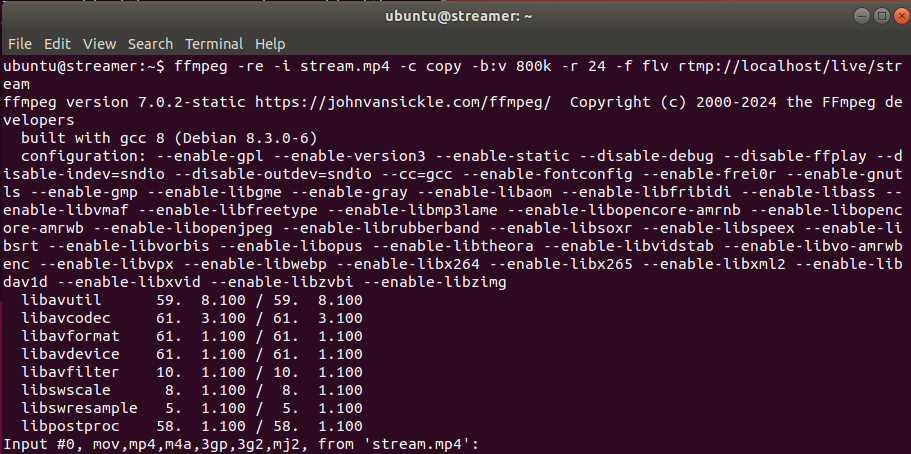


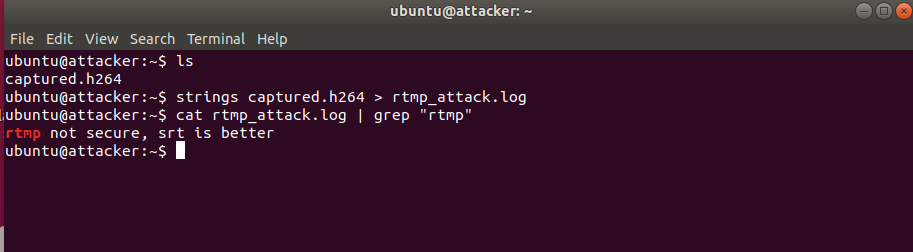




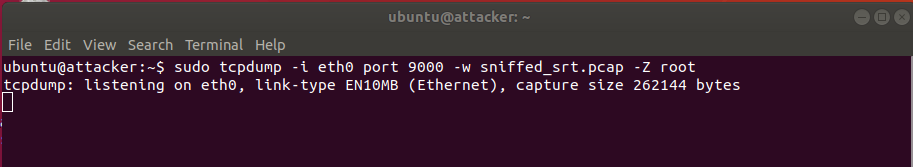
* Thực hiện task 4:

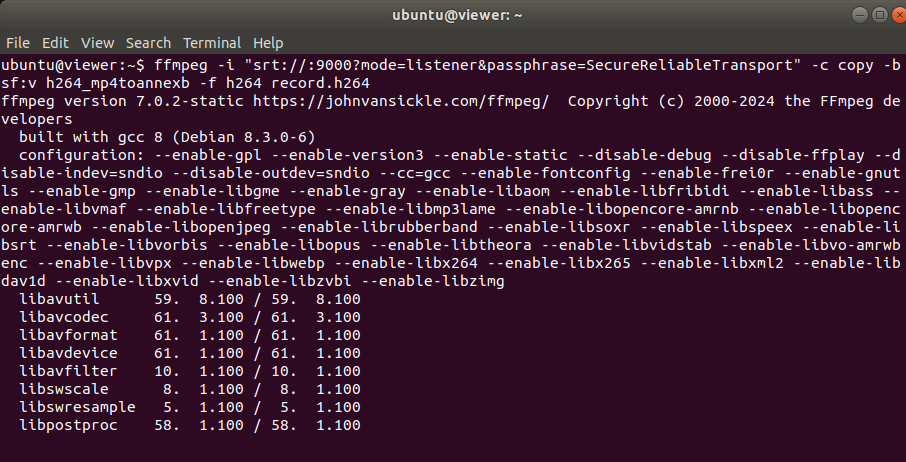


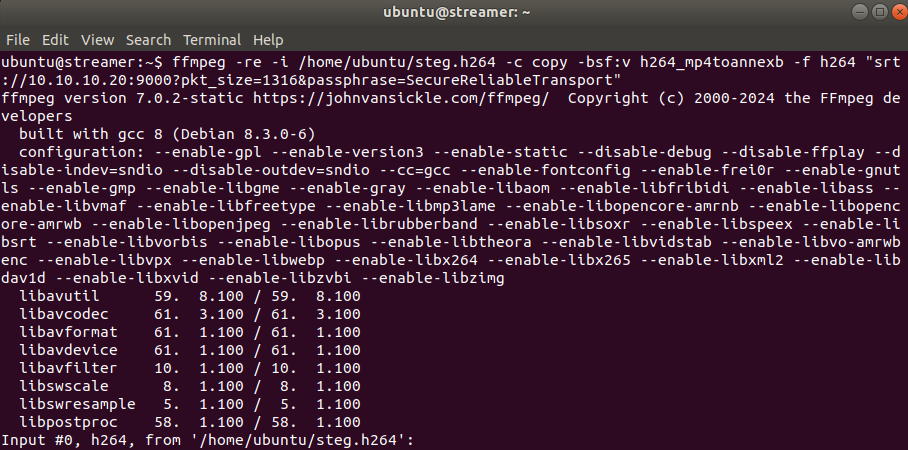




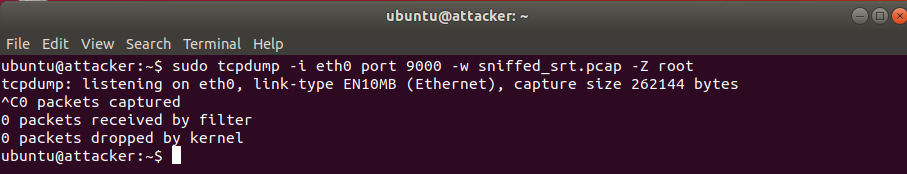
* Thực hiện task 5:











* Checkwork:

