

**HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG
KHOA AN TOÀN THÔNG TIN**



**BÁO CÁO BÀI THỰC HÀNH
HỌC PHẦN: KỸ THUẬT GIẤU TIN
MÃ HỌC PHẦN: INT14102**

**NHÓM LỚP: D21CQAT01-B
TÊN BÀI: GIẤU TIN TRONG ÂM THANH BẰNG FHSS:
KIỂM TRA ĐỘ BỀN VỚI NHIỀU**

Sinh viên thực hiện:

B21DCAT105 Đặng Thị Thanh Huyền

Giảng viên: PGS.TS. Đỗ Xuân Chợt

HỌC KỲ 2 NĂM HỌC 2024-2025

MỤC LỤC

DANH MỤC CÁC HÌNH VẼ.....	3
DANH MỤC CÁC BẢNG BIỂU.....	3
DANH MỤC CÁC TỪ VIẾT TẮT	4
CHƯƠNG 1: GIỚI THIỆU CHUNG VỀ BÀI THỰC HÀNH.....	5
1.1 Giới thiệu chung về bài thực hành	5
1.2 Nội dung và hướng dẫn bài thực hành	5
CHƯƠNG 2: PHÂN TÍCH YÊU CẦU BÀI THỰC HÀNH.....	9
2.1 Thiết kế bài thực hành	9
2.3 Tích hợp và triển khai	11
CHƯƠNG 3: THỬ NGHIỆM VÀ ĐÁNH GIÁ.....	14
TÀI LIỆU THAM KHẢO	21

DANH MỤC CÁC HÌNH VẼ

Hình 1: Yêu cầu checkwork.....	9
Hình 2: Giao diện Labedit.....	10
Hình 3: Cài đặt phần result.....	10
Hình 4: Nội dung dockerfiles.....	11
Hình 5: Đẩy image bài thực hành lên docker hub.....	11
Hình 6: Trạng thái bài thực hành trên Docker Hub.....	12
Hình 7: tạo thủ công ở thư mục ~/labs bằng lệnh	12
Hình 8: thực hiện đẩy lên github bằng git hoặc thủ công.....	13
Hình 9: Khởi động bài lab.....	14
Hình 10: Sinh viên thực hiện sửa nội dung file message.txt.....	14
Hình 11: sinh viên cần phải chuyển message từ text sang dạng nhị phân	14
Hình 12: Tạo dãy nhảy tần ngẫu nhiên dựa trên số bit	15
Hình 13: Câu lệnh tiến hành giấu tin.....	15
Hình 14: Kiểm tra xem đã có đủ các file chưa.....	15
Hình 15: Kiểm tra và trích xuất thông điệp.....	16
Hình 16: Sử dụng công cụ audacity để thêm nhiễu vào file âm thanh.....	16
Hình 17: Cấu hình audacity.....	17
Hình 18: đặt tên file mới là add_noise.wav.....	17
Hình 19: Kiểm tra xem đã có file âm thanh add_noise chưa:	17
Hình 20: Chỉnh sửa file extract_message.py.....	18
Hình 21: Kiểm tra và trích xuất thông điệp từ stego_audio.wav	18
Hình 22: So sánh thuộc tính âm thanh stego_audio.wav của file với sox.....	19
Hình 23: So sánh thuộc tính âm thanh add_noise.wav của file với sox.....	19
Hình 24: Màn hình checkwork:.....	20

DANH MỤC CÁC BẢNG BIỂU

DANH MỤC CÁC TỪ VIẾT TẮT

Từ viết tắt	Thuật ngữ tiếng Anh/Giải thích	Thuật ngữ tiếng Việt/Giải thích
FHSS	Frequency Hopping Spread Spectrum	Phổ tần nhảy tần ngẫu nhiên
WAV	Waveform Audio File Format	Định dạng tệp âm thanh WAV
Sox	Sound eXchange	Công cụ xử lý và phân tích âm thanh dòng lệnh
PTIT	Posts and Telecommunications Institute of Technology	Học viện Công nghệ Bưu chính Viễn thông
GUI	Graphical User Interface	Giao diện đồ họa người dùng
Python	Python Programming Language	Ngôn ngữ lập trình Python
Audacity	Audacity Audio Editor	Phần mềm chỉnh sửa âm thanh Audacity
Labtainers	Labtainers Cybersecurity Lab Environment	Môi trường thực hành an toàn thông tin Labtainers
Docker	Docker Container Technology	Công nghệ container hóa Docker
Ubuntu	Ubuntu Linux Distribution	Bản phân phối hệ điều hành Linux Ubuntu

CHƯƠNG 1: GIỚI THIỆU CHUNG VỀ BÀI THỰC HÀNH

1.1 Giới thiệu chung về bài thực hành

Bài thực hành này được thiết kế nhằm cung cấp cho sinh viên kiến thức và kỹ năng thực tế về kỹ thuật giấu tin (steganography) trong âm thanh sử dụng phương pháp Frequency Hopping Spread Spectrum (FHSS). FHSS là một kỹ thuật truyền thông trong đó tín hiệu được truyền qua nhiều tần số khác nhau theo một dãy nhảy tần ngẫu nhiên, giúp tăng cường bảo mật và khả năng chống nhiễu. Trong bài lab, sinh viên sẽ thực hành quy trình nhúng thông điệp vào file âm thanh, thêm nhiễu để mô phỏng môi trường thực tế, và trích xuất thông điệp để kiểm tra độ bền của hệ thống.

Bài thực hành được triển khai trên môi trường Labtainers, một nền tảng dựa trên Docker, cho phép sinh viên làm việc trong container Ubuntu với các công cụ hỗ trợ như Python, Audacity, và Sox. Mục tiêu chính là giúp sinh viên hiểu cách hoạt động của FHSS trong giấu tin, phân tích tác động của nhiễu lên tín hiệu âm thanh, và phát triển kỹ năng xử lý dữ liệu âm thanh. Bài lab này không chỉ mang tính giáo dục mà còn tạo cơ hội thực hành các khái niệm an ninh thông tin và xử lý tín hiệu trong môi trường mô phỏng.

1.2 Nội dung và hướng dẫn bài thực hành

1.2.1 Mục đích

Mục đích của bài thực hành là:

- **Học và áp dụng kỹ thuật giấu tin:** Sinh viên sẽ làm quen với khái niệm giấu tin trong âm thanh bằng phương pháp FHSS, một kỹ thuật tiên tiến trong lĩnh vực an ninh thông tin.
- **Kiểm tra độ bền của hệ thống:** Thực hiện thêm nhiễu vào file âm thanh đã nhúng tin để đánh giá khả năng trích xuất thông điệp trong điều kiện nhiễu, từ đó hiểu rõ ưu điểm và hạn chế của FHSS.
- **Phát triển kỹ năng phân tích:** Sử dụng công cụ Sox để phân tích các đặc tính âm thanh (cường độ, tần số, độ biến thiên) trước và sau khi thêm nhiễu, giúp sinh viên rèn luyện kỹ năng quan sát và so sánh dữ liệu.
- **Tăng cường kỹ năng sử dụng công cụ:** Làm quen với các công cụ như Python (để xử lý nhị phân và nhúng tin), Audacity (để thêm nhiễu), và Sox (để phân tích âm thanh) trong môi trường Labtainers.

- **Đánh giá hiệu quả thực hành:** Đảm bảo sinh viên hoàn thành các nhiệm vụ cơ bản như giấu tin, thêm nhiễu, trích xuất, và so sánh kết quả, đồng thời cung cấp dữ liệu để hệ thống chấm điểm tự động.

1.2.2 Yêu cầu đối với sinh viên

Để hoàn thành bài thực hành một cách hiệu quả, sinh viên cần đáp ứng các yêu cầu sau:

- **Kiến thức cơ bản:** Hiểu các khái niệm cơ bản về giấu tin (steganography), FHSS, và xử lý tín hiệu âm thanh. Sinh viên cần nắm được cách tín hiệu âm thanh được biểu diễn dưới dạng số và cách nhiễu ảnh hưởng đến dữ liệu.
- **Kỹ năng sử dụng công cụ:** Thành thạo các lệnh cơ bản trong terminal Linux (như nano, ls, python3), biết cách sử dụng Audacity để chỉnh sửa âm thanh, và hiểu cách sử dụng Sox để phân tích file âm thanh.

1.2.3 Nội dung thực hành

Khởi động bài lab, tải bài thực hành bằng imodule:

`imodule https://github.com/DTHuyn/steg-fhss-noise/raw/master/steg-fhss-noise.tar`

Vào terminal, gõ:

`labtainer -r steg-fhss-noise`

(chú ý: sinh viên sử dụng mã sinh viên của mình để nhập thông tin email người thực hiện bài lab khi có yêu cầu, để sử dụng khi chấm điểm)

Sau khi khởi động xong, màn hình sẽ xuất hiện 1 terminal với người dùng **ubuntu**. Thực hiện lệnh kiểm tra các thư mục có sẵn trong container

Task1: Tiến hành giấu tin vào file âm thanh ban đầu *original_audio.wav*

(Nếu muốn thực hành giấu tin bằng fhss có thể thay thế bằng bất kỳ nội dung nào, nhưng nếu muốn AC bài này, thì nội dung file message.txt phải có nội dung “*HELLO PTIT!*”)

Sinh viên thực hiện sửa nội dung file message.txt: `nano message.txt`

Để tiến hành giấu tin, sinh viên cần phải chuyển message từ text sang dạng nhị phân.

`python3 convert_message_to_bits.py`

Tạo dãy nhảy tần ngẫu nhiên dựa trên số bit, lưu vào hopping_pattern.txt

`python3 generate_hopping_pattern.py`

Yêu cầu nhập vào số bit của chuỗi thông điệp với ý tưởng cứ mỗi một bit thông điệp được truyền đi bởi 1 tần số khác nhau. Số bit được in ra màn hình ở task1

Câu lệnh tiến hành giấu tin:

`python3 embed_message_to_audio.py`

File âm thanh đã nhúng tin được lưu tại: *stego_audio.wav*

Kiểm tra xem đã có đủ các file chưa bằng câu lệnh:

ls -l

Kiểm tra dung lượng của file audio trước và sau khi giấu tin xem có thay đổi gì không?

Với phương pháp này thì nó sẽ không làm thay đổi dung lượng của file audio trước và sau khi giấu tin.

Kiểm tra định dạng file đầu ra *stego_audio.wav* xem thỏa mãn yêu cầu không?

file stego_audio.wav

Kiểm tra và trích xuất thông điệp từ *stego_audio.wav*

python3 extract_message.py

Thông điệp được trích xuất in ra màn hình và lưu vào *extracted_message.txt*

Task2: Cộng nhiễu vào file âm thanh bằng audacity

Sử dụng công cụ audacity để thêm nhiễu vào file âm thanh

audacity stego_audio.wav

Giao diện audacity lúc này đã hiện ra.

Chọn Ctrl+A, để chọn cả đoạn âm thanh, sau đó ta sẽ chọn Generate → Noise...

Ngoài nhiễu trắng, audacity còn có nhiễu hồng và nhiễu nâu, nhưng ta sẽ tạm thời bỏ qua chúng.

Ở phần Amplitude, ta cần để một giá trị ở mức không quá nhỏ nhưng đủ để tai người khó phát hiện ra, ở đây hãy để giá trị Amplitude là 0.005. Nhiễu ta thêm vào âm thanh sẽ làm xáo trộn bố cục các bit và phổ âm thanh, từ đó phá hủy thông tin giấu trong đó.

Chọn File → Export → Export as WAV để lưu file âm thanh sau khi thêm nhiễu. Lưu file cùng nơi với file âm thanh gốc, đặt tên file mới là *add_noise.wav*

Kiểm tra xem đã có file âm thanh *add_noise* chưa:

ls -l

Task 3: Kiểm tra lại khả năng trích xuất tin giấu

Chỉnh sửa file *extract_message.py* đoạn đầu vào file âm thanh *stego_audio.wav* thành *add_noise.wav*

nano extract_message.py

Kiểm tra và trích xuất thông điệp từ *stego_audio.wav*

python3 extract_message.py

Thông điệp trích xuất ra bị sai, không đúng như ban đầu

Task 4: So sánh thuộc tính âm thanh của file với sox

sox cũng giống như ffprobe là một công cụ cho phép ta so sánh các thuộc tính của file âm thanh. Nhưng khác với ffprobe, sox tập trung vào các thông tin liên quan đến tính chất

của âm thanh hơn là dữ liệu. Hãy dùng lệnh để kiểm tra thuộc tính cho hai file.

```
sox stego_audio.wav -n stat
```

```
sox add_noise.wav -n stat
```

So sánh các thông tin liên quan đến mức cường độ âm, độ biến thiên dải tần số và các tham số có liên quan, ta nhận thấy có một số sự khác biệt nhỏ.

Trên terminal đầu tiên sử dụng câu lệnh sau để kết thúc bài lab:

```
stoplab
```

Khi bài lab kết thúc, một tệp zip lưu kết quả được tạo và lưu vào một vị trí được hiển thị bên dưới stoplab.

Khởi động lại bài lab:

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

```
labtainer -r steg-fhss-noise
```


CHƯƠNG 2: PHÂN TÍCH YÊU CẦU BÀI THỰC HÀNH

2.1 Thiết kế bài thực hành

Bài lab gồm 1 container là ubuntu. Cấu hình mạng được để random do không cần đến địa chỉ mạng trong bài thực hành.

Cấu hình Docker

- Bài lab chạy với image là .base2
- Cần cài thêm môi trường ảo hóa venv và thư viện pycryptodome trong dockerfiles
- Docs lưu lại hướng dẫn thực hành cho sinh viên

Các nhiệm vụ cần thực hiện để thành công:

- Tiến hành thực hiện giấu và tách tin thành công
- Cộng nhiễu vào file âm thanh sau giấu tin
- Tiến hành tách tin lại và kiểm tra thông điệp tách được
- Kết thúc bài lab và đóng gói kết quả

Để đánh giá được sinh viên đã hoàn thành bài thực hành hay chưa, cần chia bài thực hành thành các nhiệm vụ nhỏ, mỗi nhiệm vụ cần phải chỉ rõ kết quả để có thể dựa vào đó đánh giá, chấm điểm. Do vậy, trong bài thực hành này hệ thống cần ghi nhận các thao tác, sự kiện được mô tả và cấu hình như bảng dưới đây:

```
hide_message = *.stdin : CONTAINS : python3 embed_message_to_audio.py
check_extracted_message = *.stdout : CONTAINS : HELLO PTIT!
noise_addition = ~/.bash_history : CONTAINS : audacity
check_add_noise_file = ls.stdout : CONTAINS : add_noise.wav
compare_audio = ~/.bash_history : CONTAINS : sox add_noise.wav -n stat
```

Hình 1: Yêu cầu checkwork

hide_message : Kiểm tra xem người dùng có chạy câu lệnh giấu tin chưa

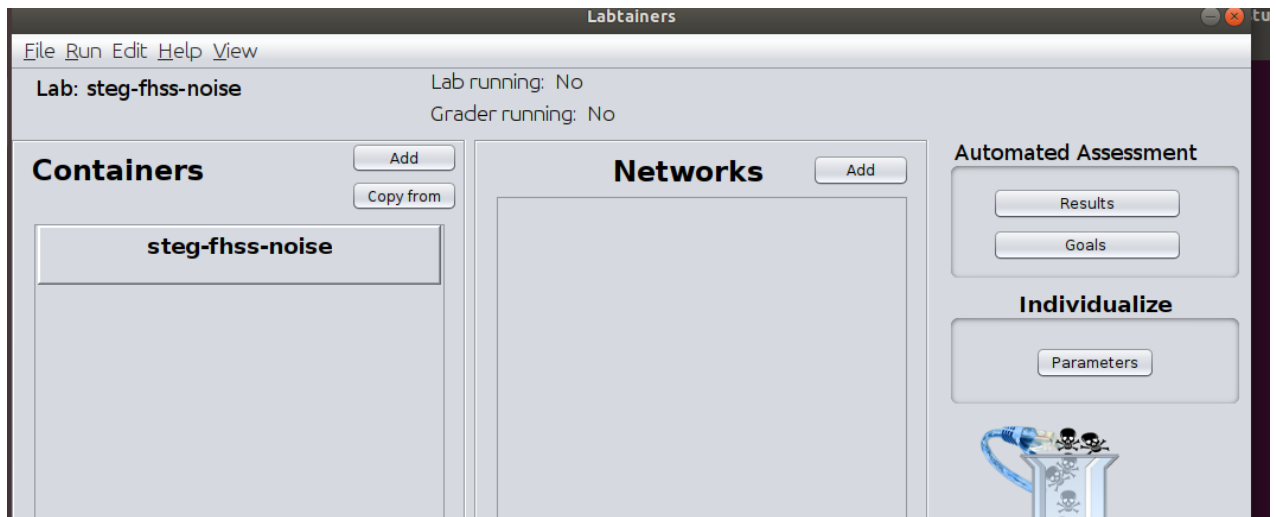
check_extracted_message : Kiểm thông điệp có chính xác không HELLO PTIT!

noise_addition : kiểm tra có sử dụng audacity để tạo nhiễu không

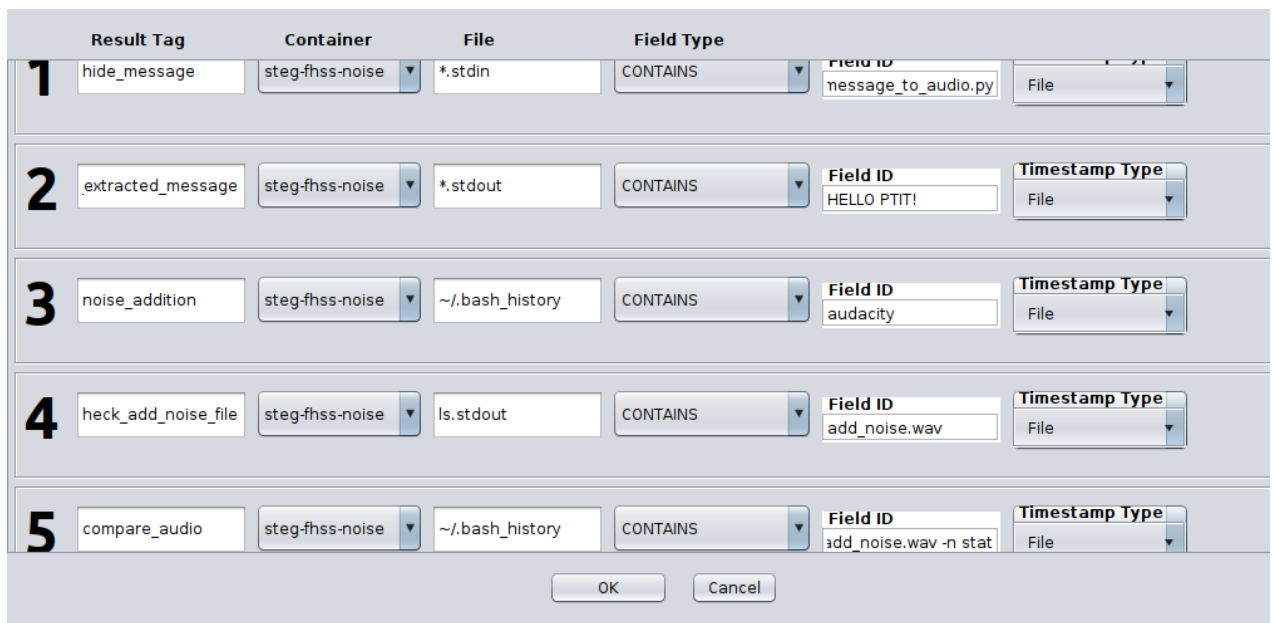
check_add_noise_file : kiểm tra file âm thanh add_noise.wav sau khi cộng nhiễu đã được tạo chưa

compare_audio : kiểm tra người dùng đã sử dụng công cụ sox để so sánh không?

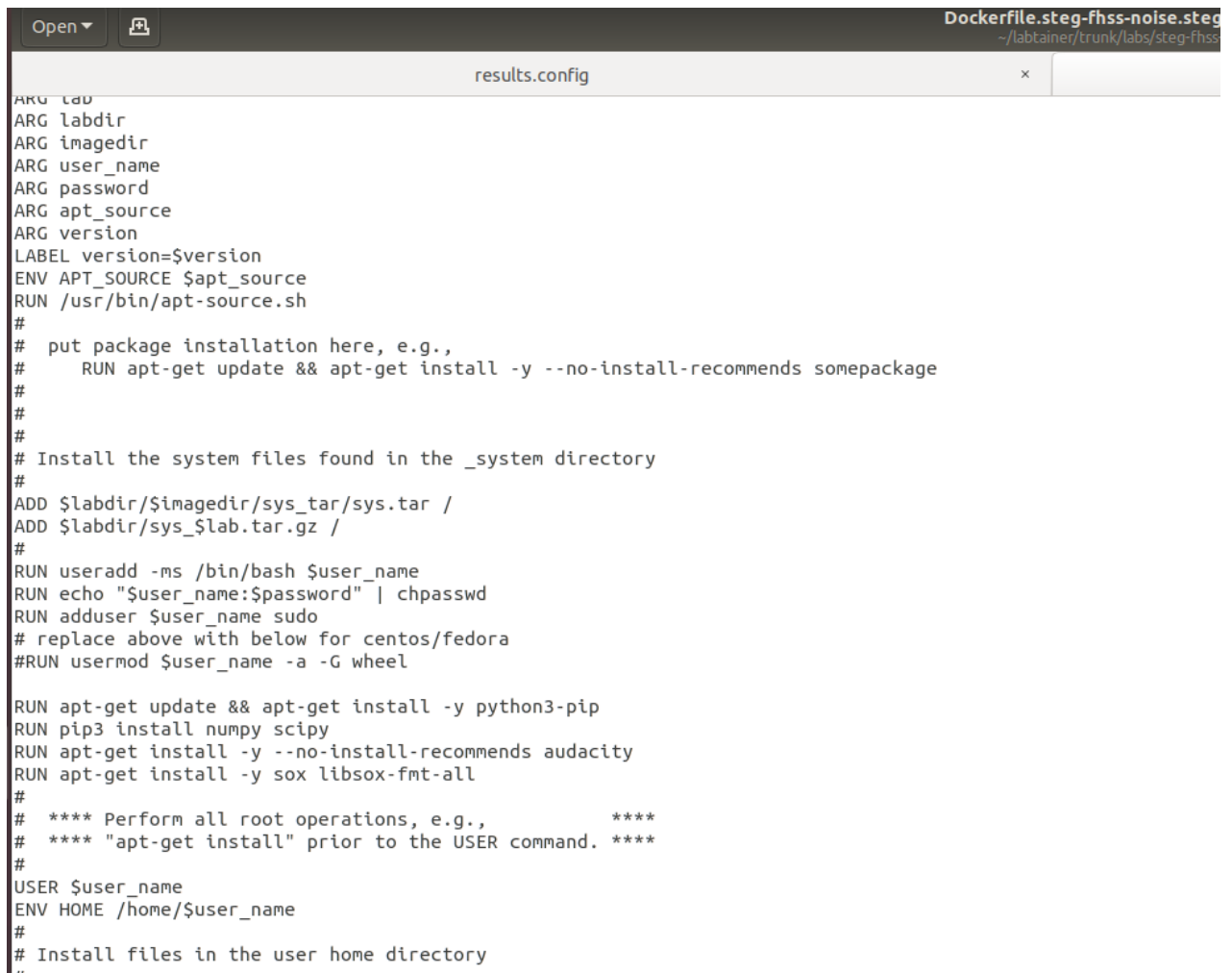
2.2 Cài đặt và cấu hình máy ảo



Hình 2: Giao diện Labedit



Hình 3: Cài đặt phần result

A screenshot of a code editor window titled "results.config" with a close button (x) in the top right corner. The editor shows the content of a Dockerfile. The text is as follows:

```
ARG labdir
ARG imagedir
ARG user_name
ARG password
ARG apt_source
ARG version
LABEL version=$version
ENV APT_SOURCE $apt_source
RUN /usr/bin/apt-source.sh
#
# put package installation here, e.g.,
#   RUN apt-get update && apt-get install -y --no-install-recommends somepackage
#
#
# Install the system files found in the _system directory
#
ADD $labdir/$imagedir/sys_tar/sys.tar /
ADD $labdir/sys_$lab.tar.gz /
#
RUN useradd -ms /bin/bash $user_name
RUN echo "$user_name:$password" | chpasswd
RUN adduser $user_name sudo
# replace above with below for centos/fedora
#RUN usermod $user_name -a -G wheel

RUN apt-get update && apt-get install -y python3-pip
RUN pip3 install numpy scipy
RUN apt-get install -y --no-install-recommends audacity
RUN apt-get install -y sox libsox-fmt-all
#
# **** Perform all root operations, e.g., ****
# **** "apt-get install" prior to the USER command. ****
#
USER $user_name
ENV HOME /home/$user_name
#
# Install files in the user home directory
"
```

Hình 4: Nội dung dockerfiles

2.3 Tích hợp và triển khai

Bài thực hành được triển khai như sau:

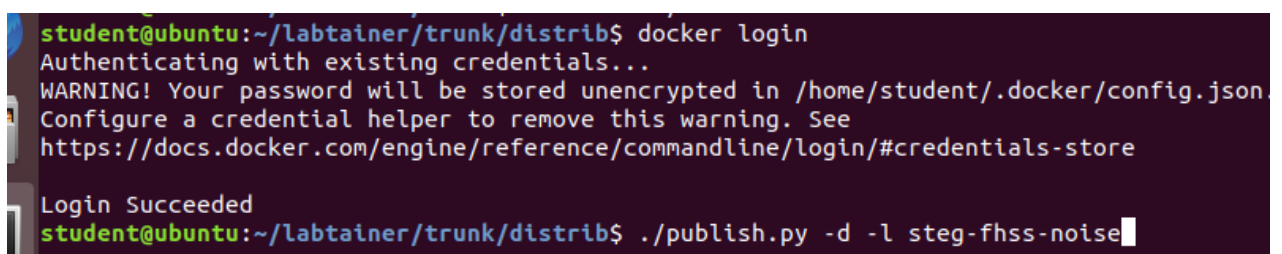
Docker

Đường dẫn: <https://hub.docker.com/repositories/dthuyn>

Thêm registry cho bài thực hành

Truy cập vào thư mục trunk/distrib gõ lệnh: *docker login* đăng nhập tài khoản DockerHub với *Username: dthuyn* và *password: *****

Sử dụng lệnh *./publish.py -d -l steg-fhss-noise* để đẩy images của bài thực hành lên DockerHub

A screenshot of a terminal window with a dark background. The prompt is "student@ubuntu:~/labtainer/trunk/distrib\$". The user enters "docker login". The output shows "Authenticating with existing credentials..." followed by a warning: "WARNING! Your password will be stored unencrypted in /home/student/.docker/config.json. Configure a credential helper to remove this warning. See https://docs.docker.com/engine/reference/commandline/login/#credentials-store". Then "Login Succeeded" is displayed. The user then enters the command ". /publish.py -d -l steg-fhss-noise".

```
student@ubuntu:~/labtainer/trunk/distrib$ docker login
Authenticating with existing credentials...
WARNING! Your password will be stored unencrypted in /home/student/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
student@ubuntu:~/labtainer/trunk/distrib$ ./publish.py -d -l steg-fhss-noise
```

Hình 5: Đẩy image bài thực hành lên docker hub

dthuyn/steg-fhss-noise.steg-fhss-noise.student

Last pushed about 3 hours ago • Repository size: 664.4 MB

Add a description  

Add a category  

General

Tags


Image Management BETA

Collaborators





Webhooks

Settings

Tags

 DOCKER SCOUT INACTIVE
[Activate](#)

This repository contains 2 tag(s).

Tag	OS	Type	Pulled	Pushed
 base_image739eee7...		Image	less than 1 day	about 3 hours
 latest		Image	less than 1 day	about 3 hours

[See all](#)

Hình 6: Trạng thái bài thực hành trên Docker Hub

Github

Đường dẫn: <https://github.com/DTHuyn/steg-fhss-noise>


Ở đường dẫn \$LABTAINER_DIR/distrib, tạo file tar bằng create-imodules.sh hoặc tạo thủ công ở thư mục ~/labs bằng lệnh:

```
tar -cvf steg-fhss-hide.tar steg-fhss-noise
```

```
student@ubuntu:~/labtainer/trunk/labs$ tar -cvf steg-fhss-hide.tar steg-fhss-hide
steg-fhss-hide/
steg-fhss-hide/config/
steg-fhss-hide/config/start.config
steg-fhss-hide/config/parameter.config
steg-fhss-hide/config/steg-fhss-hide-home_tar.list
steg-fhss-hide/dockerfiles/
steg-fhss-hide/dockerfiles/Dockerfile.steg-fhss-hide.steg-fhss-hide.student
steg-fhss-hide/instr_config/
steg-fhss-hide/instr_config/pregrade.sh
steg-fhss-hide/instr_config/goals.config
steg-fhss-hide/instr_config/results.config
steg-fhss-hide/docs/
steg-fhss-hide/docs/read_first.txt
steg-fhss-hide/steg-fhss-hide/
steg-fhss-hide/steg-fhss-hide/_bin/
steg-fhss-hide/steg-fhss-hide/_bin/fixlocal.sh
steg-fhss-hide/steg-fhss-hide-home_tar
```


Hình 7: tạo thủ công ở thư mục ~/labs bằng lệnh



Sau đó, thực hiện đẩy lên github bằng git hoặc thủ công



 **steg-fhss-noise** Public

Pin Unwatch

master 1 Branch 0 Tags + <> Code

 **DTHuyn** Create README.md 4a44a92 · 2 hours ago 3 Commits

 README.md	Create README.md	2 hours ago
 steg-fhss-noise.tar	Add steg-fhss-noise lab .tar file	2 hours ago

 **README** 

imodule <https://github.com/DTHuyn/steg-fhss-noise/raw/master/steg-fhss-noise.tar>

Hình 8: thực hiện đẩy lên github bằng git hoặc thủ công

CHƯƠNG 3: THỬ NGHIỆM VÀ ĐÁNH GIÁ.

Khởi động bài lab, tải bài thực hành bằng imodule:

`imodule https://github.com/DTHuyn/steg-fhss-noise/raw/master/steg-fhss-noise.tar`

Vào terminal, gõ:

`labtainer -r steg-fhss-noise`

(chú ý: sinh viên sử dụng mã sinh viên của mình để nhập thông tin email người thực hiện bài lab khi có yêu cầu, để sử dụng khi chấm điểm)

Sau khi khởi động xong, màn hình sẽ xuất hiện 1 terminal với người dùng **ubuntu**. Thực hiện lệnh kiểm tra các thư mục có sẵn trong container

```
student@LabtainerVMware:~/labtainer/labtainer-student$ imodule https://github.com/DTHuyn/steg-fhss-noise/raw/master/steg-fhss-noise.tar
Adding imodule path https://github.com/DTHuyn/steg-fhss-noise/raw/master/steg-fhss-noise.tar
Updating IModule from https://github.com/DTHuyn/steg-fhss-noise/raw/master/steg-fhss-noise.tar
student@LabtainerVMware:~/labtainer/labtainer-student$ labtainer -r steg-fhss-noise
non-network local connections being added to access control list

Please enter your e-mail address: [B21DCAT105]
Started 1 containers, 1 completed initialization. Done.

The lab manual is at
file:///home/student/labtainer/trunk/labs/steg-fhss-noise/docs/steg-fhss-hide.pdf

You may open these by right clicking
and select "Open Link".

Press <enter> to start the lab
```

Hình 9: Khởi động bài lab

Task1: Tiến hành giấu tin vào file âm thanh ban đầu `original_audio.wav`

(Nếu muốn thực hành giấu tin bằng fhss có thể thay thế bằng bất kỳ nội dung nào, nhưng nếu muốn AC bài này, thì nội dung file `message.txt` phải có nội dung “*HELLO PTIT!*”)

Sinh viên thực hiện sửa nội dung file `message.txt`: `nano message.txt`

```
ubuntu@steg-fhss-noise: ~
GNU nano 4.8 message.txt
HELLO PTIT!
```

Hình 10: Sinh viên thực hiện sửa nội dung file `message.txt`

Để tiến hành giấu tin, sinh viên cần phải chuyển message từ text sang dạng nhị phân.

`python3 convert_message_to_bits.py`

```
ubuntu@steg-fhss-noise:~$ python3 convert_message_to_bits.py
Chuỗi bit: 0100100001000101010011000100110001001111001000000101000001010100010010010101
010000100001
Kích thước thông điệp: 88 bit
Chuỗi bit đã được lưu vào message_bits.txt
ubuntu@steg-fhss-noise:~$
```

Hình 11: sinh viên cần phải chuyển message từ text sang dạng nhị phân

Tạo dãy nhảy tần ngẫu nhiên dựa trên số bit, lưu vào hopping_pattern.txt

python3 generate_hopping_pattern.py

```
ubuntu@steg-fhss-noise:~$ python3 generate_hopping_pattern.py
Nhập số bit của thông điệp: 88
88
Dãy nhảy tần đã được lưu vào hopping_pattern.txt
ubuntu@steg-fhss-noise:~$
```

Hình 12: Tạo dãy nhảy tần ngẫu nhiên dựa trên số bit

Yêu cầu nhập vào số bit của chuỗi thông điệp với ý tưởng cứ mỗi một bit thông điệp được truyền đi bởi 1 tần số khác nhau. Số bit được in ra màn hình ở task1

Câu lệnh tiến hành giấu tin:

python3 embed_message_to_audio.py

```
ubuntu@steg-fhss-noise:~$ python3 embed_message_to_audio.py
File âm thanh đã nhúng tin được lưu tại: stego_audio.wav
ubuntu@steg-fhss-noise:~$
```

Hình 13: Câu lệnh tiến hành giấu tin

File âm thanh đã nhúng tin được lưu tại: *stego_audio.wav*

Kiểm tra xem đã có đủ các file chưa bằng câu lệnh:

ls -l

```
ubuntu@steg-fhss-noise:~$ ls -l
total 892
-rwxrwxr-x 1 ubuntu ubuntu 1017 May  5 02:31 convert_message_to_bits.py
-rwxrwxr-x 1 ubuntu ubuntu 2540 May  5 02:32 embed_message_to_audio.py
-rwxrwxr-x 1 ubuntu ubuntu 2378 May  5 02:32 extract_message.py
-rwxrwxr-x 1 ubuntu ubuntu  837 May  5 02:32 generate_hopping_pattern.py
-rw-rw-r-- 1 ubuntu ubuntu  175 May 10 14:13 hopping_pattern.txt
-rw-rw-r-- 1 ubuntu ubuntu   12 May 10 14:12 message.txt
-rw-rw-r-- 1 ubuntu ubuntu   88 May 10 14:12 message_bits.txt
-rw-rw-r-- 1 ubuntu ubuntu 441044 May  5 02:32 original_audio.wav
-rw-rw-r-- 1 ubuntu ubuntu 441044 May 10 14:13 stego_audio.wav
ubuntu@steg-fhss-noise:~$
```

Hình 14: Kiểm tra xem đã có đủ các file chưa

Kiểm tra dung lượng của file audio trước và sau khi giấu tin xem có thay đổi gì không?

Với phương pháp này thì nó sẽ không làm thay đổi dung lượng của file audio trước và sau khi giấu tin.

Kiểm tra định dạng file đầu ra stego_audio.wav xem thỏa mãn yêu cầu không?

file stego_audio.wav

Kiểm tra và trích xuất thông điệp từ stego_audio.wav

python3 extract_message.py

```
ubuntu@steg-fhss-noise:~$ python3 extract_message.py
Thông điệp trích xuất: HELLO PTIT!
Thông điệp đã được lưu vào: extracted_message.txt
ubuntu@steg-fhss-noise:~$
```

Hình 15: Kiểm tra và trích xuất thông điệp

Thông điệp được trích xuất in ra màn hình và lưu vào *extracted_message.txt*

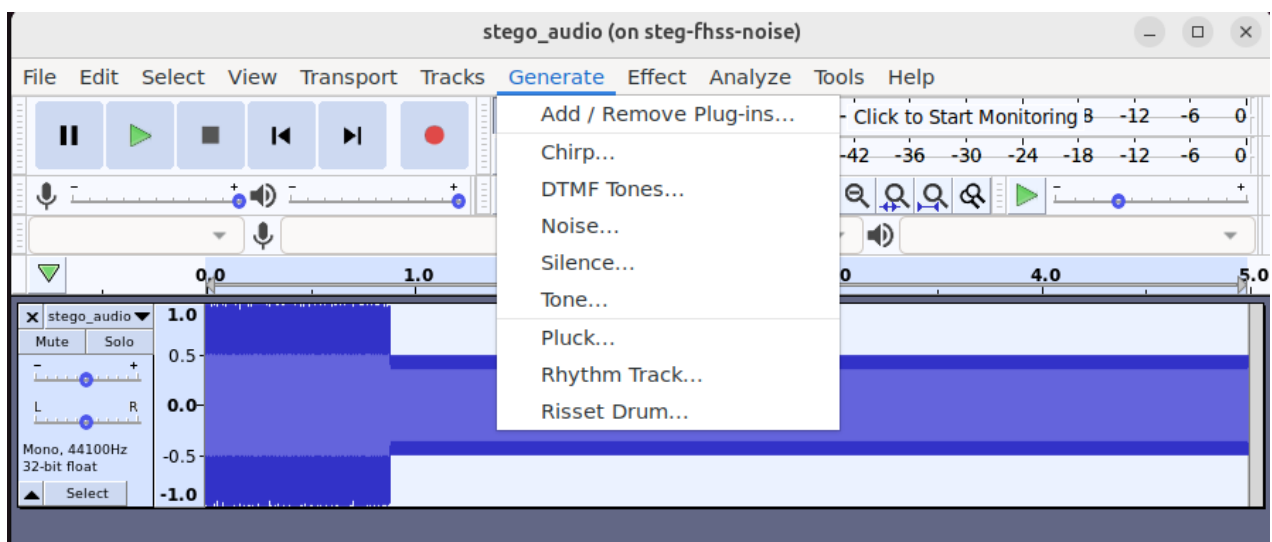
Task2: Cộng nhiễu vào file âm thanh bằng audacity

Sử dụng công cụ audacity để thêm nhiễu vào file âm thanh

audacity stego_audio.wav

Giao diện audacity lúc này đã hiện ra.

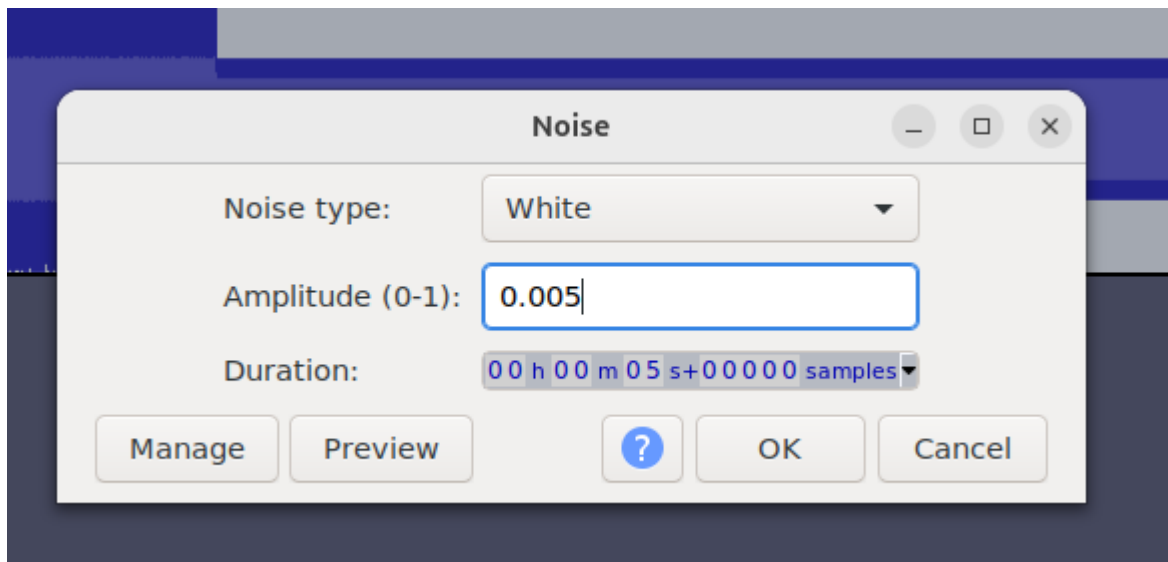
Chọn Ctrl+A, để chọn cả đoạn âm thanh, sau đó ta sẽ chọn Generate → Noise...



Hình 16: Sử dụng công cụ audacity để thêm nhiễu vào file âm thanh

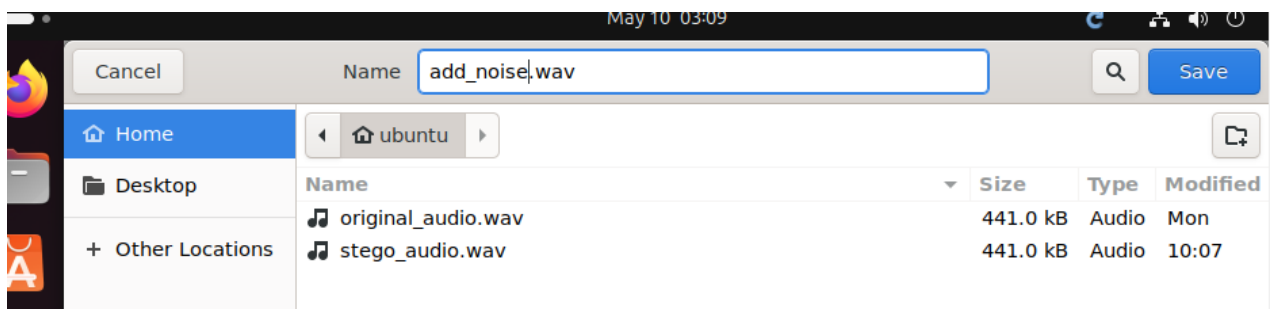
Ngoài nhiễu trắng, audacity còn có nhiễu hồng và nhiễu nâu, nhưng ta sẽ tạm thời bỏ qua chúng.

Ở phần Amplitude, ta cần để một giá trị ở mức không quá nhỏ nhưng đủ để tai người khó phát hiện ra, ở đây hãy để giá trị Amplitude là 0.005. Nhiễu ta thêm vào âm thanh sẽ làm xáo trộn bố cục các bit và phổ âm thanh, từ đó phá hủy thông tin giấu trong đó.



Hình 17: Cấu hình audacity

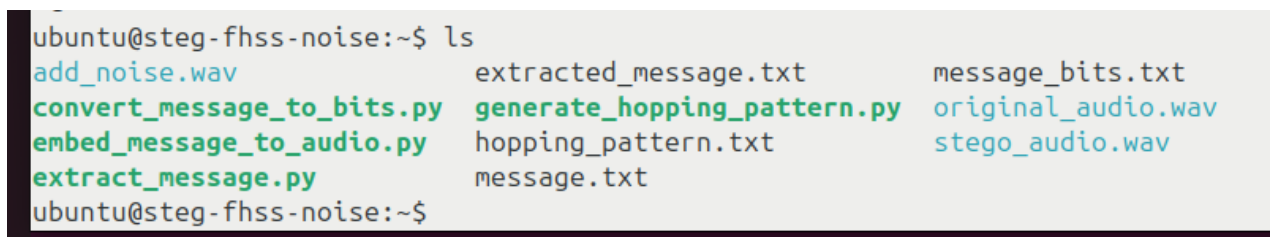
Chọn File → Export → Export as WAV để lưu file âm thanh sau khi thêm nhiễu. Lưu file cùng nơi với file âm thanh gốc, đặt tên file mới là add_noise.wav



Hình 18: đặt tên file mới là add_noise.wav

Kiểm tra xem đã có file âm thanh add_noise chưa:

`ls -l`



Hình 19: Kiểm tra xem đã có file âm thanh add_noise chưa:

Task 3: Kiểm tra lại khả năng trích xuất tin giấu

Chỉnh sửa file extract_message.py đoạn đầu vào file âm thanh stego_audio.wav thành add_noise.wav

`nano extract_message.py`

```

if __name__ == "__main__":
    # ^p ^mc file m thanh ^q nh
    fs, stego_audio = read("add_noise.wav")
    if stego_audio.ndim > 1: # Chuy ^cn sang mono n
        stego_audio = stego_audio[:, 0]

```

Hình 20: Chỉnh sửa file `extract_message.py`

Kiểm tra và trích xuất thông điệp từ `stego_audio.wav`

`python3 extract_message.py`

```

ubuntu@steg-fhss-noise:~$ nano extract_message.py
ubuntu@steg-fhss-noise:~$ python3 extract_message.py
Thông điệp trích xuất: `^~¼!àÝ.ç)
Thông điệp đã được lưu vào: extracted_message.txt
ubuntu@steg-fhss-noise:~$

```

Hình 21: Kiểm tra và trích xuất thông điệp từ `stego_audio.wav`

Thông điệp trích xuất ra bị sai, không đúng như ban đầu

Task 4: So sánh thuộc tính âm thanh của file với sox

sox cũng giống như `ffprobe` là một công cụ cho phép ta so sánh các thuộc tính của file âm thanh. Nhưng khác với `ffprobe`, sox tập trung vào các thông tin liên quan đến tính chất của âm thanh hơn là dữ liệu. Hãy dùng lệnh để kiểm tra thuộc tính cho hai file.

`sox stego_audio.wav -n stat`

`sox add_noise.wav -n stat`

```

ubuntu@steg-fhss-noise:~$ sox stego_audio.wav -n stat
Samples read:          220500
Length (seconds):      5.000000
Scaled by:            2147483647.0
Maximum amplitude:     0.999939
Minimum amplitude:     -0.999939
Midline amplitude:     0.000000
Mean   norm:          0.333492
Mean   amplitude:     -0.000000
RMS    amplitude:     0.383370
Maximum delta:         0.215454
Minimum delta:         0.000000
Mean   delta:         0.035093
RMS    delta:         0.053811
Rough   frequency:     985
Volume adjustment:     1.000

```

Hình 22: So sánh thuộc tính âm thanh stego_audio.wav của file với sox

```

ubuntu@steg-fhss-noise:~$ sox add_noise.wav -n stat
Samples read:          220500
Length (seconds):      5.000000
Scaled by:            2147483647.0
Maximum amplitude:     0.005157
Minimum amplitude:     -0.005157
Midline amplitude:     0.000000
Mean   norm:          0.002502
Mean   amplitude:     0.000004
RMS    amplitude:     0.002888
Maximum delta:         0.010101
Minimum delta:         0.000000
Mean   delta:         0.003341
RMS    delta:         0.004090
Rough   frequency:     9941
Volume adjustment:     193.893
ubuntu@steg-fhss-noise:~$

```

Hình 23: So sánh thuộc tính âm thanh add_noise.wav của file với sox

So sánh các thông tin liên quan đến mức cường độ âm, độ biến thiên dải tần số và các tham số có liên quan, ta nhận thấy có một số sự khác biệt nhỏ.

Trên terminal đầu tiên sử dụng câu lệnh sau để kết thúc bài lab:

stoplab

Khi bài lab kết thúc, một tệp zip lưu kết quả được tạo và lưu vào một vị trí được hiển thị bên dưới stoplab.

Màn hình checkwork:

```
student@LabtainerVMware:~/labtainer/labtainer-student$ checkwork steg-fhss-noise
Results stored in directory: /home/student/labtainer_xfer/steg-fhss-noise
Successfully copied 405kB to steg-fhss-noise-igrader:/home/instructor/B21DCAT105.steg-fhss-noise.lab
Successfully copied 2.05kB to /home/student/labtainer_xfer/steg-fhss-noise
Labname steg-fhss-noise

Student          |   hide_message | check_extracted |   noise_addition | check_add_noise |   compare_audio |
=====|=====|=====|=====|=====|=====|
B21DCAT105      |               Y |               Y |               Y |               Y |               Y |
What is automatically assessed for this lab:
```

Hình 24: Màn hình checkwork:

Khởi động lại bài lab:

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

labtainer -r steg-fhss-noise

TÀI LIỆU THAM KHẢO

[1] Bài giảng Các kỹ thuật giấu tin, PGS. TS Đỗ Xuân Chợt