Hướng Dẫn Thực Hành: Giải Mã Echo Hiding Steganography

tháng 5, 2025

1 Giới thiệu

Bài thực hành này hướng dẫn bạn cách trích xuất thông điệp ẩn trong file âm thanh bằng kỹ thuật **Echo Hiding Steganography**, một phương pháp sử dụng tiếng vọng (echo) để ẩn dữ liệu mà không làm thay đổi đáng kể chất lượng âm thanh. Bài lab bao gồm ba tác vụ chính:

1. **Tác vụ 1: Chuẩn bị dữ liệu âm thanh** - Đọc và chuẩn hóa dữ liệu từ file `output.wav`.
2. **Tác vụ 2: Giải mã thông điệp** - Trích xuất thông điệp ẩn từ dữ liệu âm thanh.
3. **Tác vụ 3: Kiểm tra và xác nhận kết quả** - So sánh thông điệp đã giải mã với thông điệp gốc.

Mục tiêu là giải mã chính xác thông điệp ẩn (ví dụ: "HELLO") từ file âm thanh và đánh giá độ chính xác của kết quả. Bài lab sử dụng môi trường Labtainer để đảm bảo tính nhất quán và an toàn.

2 Yêu cầu

- **Labtainer**: Đã cài đặt trên hệ thống (Ubuntu, macOS, hoặc Windows với WSL2). Xem hướng dẫn tại: `https://my.nps.edu/web/c3o/labtainers`.
- **Docker**: Đảm bảo Docker được cài đặt và đang chạy.
- **Python 3 và thư viện**: Python 3, NumPy, SciPy.
- **File đầu vào**: File `output.wav` chứa thông điệp ẩn.
- Kiến thức cơ bản về Python, dòng lệnh và xử lý tín hiệu âm thanh.

3 Thiết lập bài thực hành

Bài thực hành được cấu hình trong môi trường Labtainer với các thông tin sau:

- **Container**: Ubuntu 20.04 (`labtainer.ubuntu20`).
- **Phụ thuộc**: Python 3, NumPy, SciPy.
- **Script**:
  - `prepare_data.py` (Tác vụ 1)
  - `decode_message.py` (Tác vụ 2)
  - `verify_result.py` (Tác vụ 3)

Để bắt đầu:

1. Mở terminal và chuyển đến thư mục bài thực hành.
2. Chạy Labtainer:

```
labtainer echo_hiding_decoding -r
```

3. Container Docker sẽ khởi động với các script và phụ thuộc cần thiết.

4 Hướng dẫn từng tác vụ

4.1 Tác vụ 1: Chuẩn bị dữ liệu âm thanh

Mục tiêu: Đọc và chuẩn hóa dữ liệu âm thanh từ file `output.wav` để chuẩn bị cho quá trình giải mã.

Script: `prepare_data.py`

Các bước:

1. Đảm bảo file `output.wav` đã có trong thư mục làm việc.
2. Chạy script để chuẩn bị dữ liệu:

```
python3 prepare_data.py
```

3. **Quá trình**:
   - Script đọc file `output.wav` và kiểm tra định dạng WAV 16-bit.
   - Dữ liệu được chuẩn hóa sang kiểu float32, biên độ trong khoảng [-1, 1].
   - Kết quả được lưu vào file `decoded_preprocessed.npy`.
4. Kiểm tra đầu ra:
   - Đảm bảo file `decoded_preprocessed.npy` đã được tạo.
   - Định dạng mong đợi: Mảng NumPy, kiểu float32, giá trị trong [-1, 1].

Lưu ý: Nếu gặp lỗi "FileNotFoundError", kiểm tra xem file `output.wav` có trong thư mục làm việc không. Nếu file không phải WAV 16-bit, script sẽ báo lỗi "ValueError".

4.2 Tác vụ 2: Giải mã thông điệp

Mục tiêu: Trích xuất thông điệp ẩn từ dữ liệu âm thanh đã chuẩn hóa.
Script: `decode_message.py`
Các bước:

1. Đảm bảo file `decoded_preprocessed.npy` từ Tác vụ 1 đã có sẵn.
2. Chạy script để giải mã:

```
python3 decode_message.py
```

3. **Quá trình**:
   - Script tải dữ liệu từ `decoded_preprocessed.npy`.
   - Phát hiện echo bằng cách tính năng lượng của từng đoạn âm thanh để xác định bit (0 hoặc 1).
   - Chuyển chuỗi bit thành văn bản ASCII.
   - Lưu thông điệp vào file `decoded_message.txt`.
4. Kiểm tra đầu ra:
   - Đảm bảo file `decoded_message.txt` đã được tạo.
   - Nội dung mong đợi: Thông điệp văn bản (ví dụ: "HELLO").

Lưu ý: Nếu thông điệp không giải mã đúng, thử điều chỉnh tham số `delay` trong script (ví dụ: tăng từ 100 lên 200) để cải thiện khả năng phát hiện echo.

4.3 Tác vụ 3: Kiểm tra và xác nhận kết quả

Mục tiêu: So sánh thông điệp đã giải mã với thông điệp gốc để đánh giá độ chính xác.
Script: `verify_result.py`
Các bước:

1. Đảm bảo file `decoded_message.txt` từ Tác vụ 2 đã có sẵn.
2. Chỉnh sửa script `verify_result.py` để cập nhật thông điệp gốc (nếu cần, mặc định là "HELLO").
3. Chạy script để kiểm tra:

```
python3 verify_result.py
```

4. **Quá trình**:
   - Script đọc thông điệp gốc và thông điệp từ `decoded_message.txt`.
   - Tính tỷ lệ lỗi bit (BER) để đo tỷ lệ ký tự sai.
   - Tính tương quan chuẩn hóa (NC) để đánh giá độ tương đồng.
   - In kết quả so sánh.

5. Kiểm tra đầu ra:
    - Kết quả in ra bao gồm thông điệp gốc, thông điệp giải mã, BER và NC.
    - Kết quả lý tưởng: BER $\approx$ 0%, NC $\approx$ 1.0.

Lưu ý: Nếu BER cao hoặc NC thấp, kiểm tra lại thông điệp gốc hoặc thử điều chỉnh tham số giải mã trong Tác vụ 2.

5 Kết quả mong đợi

Sau khi hoàn thành các tác vụ, bạn sẽ có:

- `decoded_preprocessed.npy`: Dữ liệu âm thanh đã chuẩn hóa từ `output.wav`.
- `decoded_message.txt`: Thông điệp đã giải mã (ví dụ: "HELLO").
- Kết quả kiểm tra: Thông báo so sánh với BER $\approx$ 0% và NC $\approx$ 1.0 nếu giải mã chính xác.

6 Xử lý sự cố

- **Lỗi không tìm thấy file**: Đảm bảo `output.wav` tồn tại cho Tác vụ 1 và `decoded_preprocessed.npy` tồn tại cho Tác vụ 2. Chạy lại các tác vụ trước nếu cần.
- **Lỗi định dạng WAV**: Kiểm tra `output.wav` là file WAV 16-bit. Nếu không, sử dụng file âm thanh đúng định dạng.
- **Thông điệp giải mã không chính xác**: Điều chỉnh tham số `delay` trong `decode_message.py` (ví dụ: tăng lên 200 hoặc 300).
- **Labtainer không khởi động**: Kiểm tra Docker đang chạy và Labtainer được cài đặt đúng. Chạy `labtainer -t` để kiểm tra.

7 Lưu ý

- File `output.wav` phải được tạo từ bài lab mã hóa trước đó với các tham số phù hợp (`delay`, `alpha`).
- Tham số `delay=100` trong `decode_message.py` được tối ưu cho trường hợp đơn giản. Trong thực tế, có thể cần điều chỉnh tùy thuộc vào tham số mã hóa.
- Môi trường Labtainer đảm bảo tính lặp lại. Không sửa đổi phụ thuộc của container ngoài hướng dẫn.