

## MỤC LỤC

MỤC LỤC.....	1
I. Tổng quan .....	3
1.1. Hệ thống phát hiện và ngăn chặn tấn công (IDPS) .....	3
1.1.1. Khái niệm .....	3
1.1.2. Tính năng .....	3
1.1.3. Ưu nhược điểm .....	3
1.1.4. Cấu trúc .....	4
1.1.5. Network-based IDPS (NIDPS) .....	4
1.1.6. Host-based IDPS (HIDPS) .....	5
1.2. Học máy và ứng dụng trong các hệ thống phát hiện xâm nhập .....	5
1.2.1. Phát hiện xâm nhập dưới góc nhìn của bài toán phân lớp .....	5
1.2.2. Đánh giá hệ thống phân lớp (classification) .....	6
1.2.3. Áp dụng mô hình phân loại (classification) cho hệ thống IDS.....	8
II. Một số mô hình học máy cơ bản .....	9
2.1. Logistic Regression .....	9
2.2. Mạng neuron nhiều lớp (Multi-layer Perceptron).....	10
2.3. Cây quyết định (Decision Tree).....	11
2.4. Random Forest.....	12
2.5. Support Vector Machine.....	12
2.6. Naive Bayes.....	13
III. Phân tích bộ dữ liệu KDD99.....	13
3.1. Bộ dữ liệu KDD99 .....	13
3.1.1. Tổng quan .....	14
3.1.2. Các đặc trưng trong bộ dữ liệu.....	14
3.2. Phân tích và xử lý dữ liệu.....	16
3.2.1. Làm sạch dữ liệu.....	16
3.2.2. Trực quan hóa dữ liệu (visualization) .....	18
3.2.3. Phân chia dữ liệu.....	24
3.2.4. Chuẩn hóa dữ liệu .....	24
IV. Huấn luyện mô hình học máy .....	25
4.1. Metrics sử dụng đánh giá mô hình.....	25
4.2. Huấn luyện và đánh giá các mô hình .....	26
4.2.1. Logistic Regression .....	27
4.2.2. Multi-layer Perceptron (MLP).....	29
4.2.3. Cây quyết định (Decision Tree).....	31

4.2.4. Random Forest.....	32
4.2.5. Support Vector Machine (SVM) .....	32
4.2.6. Bộ phân loại Naive Bayes.....	33
IV. Xây dựng mô hình thu thập dữ liệu mạng phục vụ huấn luyện mô hình .....	34
4.1. SDN.....	34
4.1.1. Tổng quan về SDN.....	34
4.1.2. Kiến trúc của SDN .....	35
4.1.2. Mininet .....	36
4.1.3. ONOS.....	37
4.2. Xây dựng topo SDN giả lập.....	39
4.2.1. Khởi động ONOS controller .....	39
4.2.2. Xây dựng topology sử dụng Mininet API .....	40
4.3. Mô phỏng một số tấn công trên hệ thống giả lập .....	46
4.3.1. Một số kiểu tấn công.....	46
4.3.2. Thu thập dữ liệu .....	52
4.4. Trích xuất đặc trưng từ dữ liệu lưu lượng thô.....	55
V. Xây dựng mô hình phát hiện xâm nhập dựa trên dữ liệu tự thu thập.....	58
5.1. Xử lý và phân tích dữ liệu .....	58
5.2. Xây dựng mô hình và đánh giá kết quả.....	60
Tài liệu tham khảo .....	62

# I. Tổng quan

## 1.1. Hệ thống phát hiện và ngăn chặn tấn công (IDPS)

### 1.1.1. Khái niệm

Hệ thống phát hiện và ngăn chặn tấn công (IDPS- Intrusion Detection and Prevention System) hệ thống có khả năng theo dõi, giám sát, phát hiện và ngăn chặn các hành vi tấn công, khai thác trái phép tài nguyên được bảo vệ.

Trong đó, các hành vi tấn công được định nghĩa là các hành vi cố ý gây tổn hại tới hệ thống.

Một kịch bản tấn công cơ bản như sau:

1. Thăm dò, thu thập thông tin
2. Quét, rà soát mục tiêu
3. Giành quyền truy cập
4. Duy trì truy cập và khai thác, tấn công
5. Xóa dấu vết

Yêu cầu đối với hệ thống IDPS:

- Tính chính xác
- Tính kịp thời
- Khả năng tự bảo vệ

### 1.1.2. Tính năng

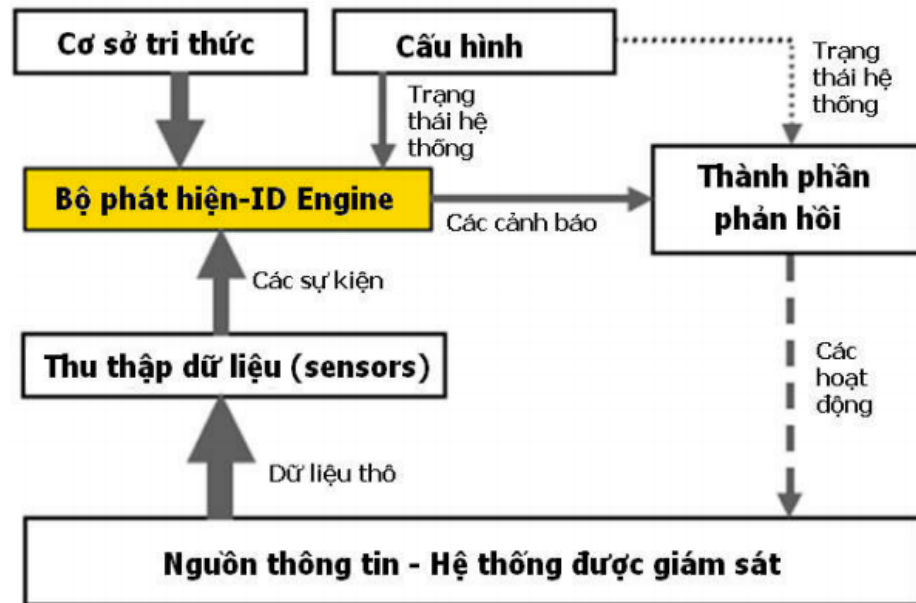
Các tính năng quan trọng của một hệ thống IDPS bao gồm:

- Ngăn chặn (Intrusion Prevention): ngăn cản hoặc giảm xác suất thành công của các hành vi tấn công. Vì vậy, tường lửa (firewall) cũng có thể coi là một dạng IDPS.
- Phát hiện (Intrusion Detection): phán đoán một hành vi có phải là tấn công hay không
- Phản ứng(Intrusion Response):
  - Ghi nhận và phát cảnh báo
  - Cản trở tấn công tiếp diễn
  - Điều chuyển tấn công sang môi trường cách ly và được giám sát
  - chủ động để thu thập thông tin, đặc điểm của cuộc tấn công
  - Điều chuyển tấn công sang môi trường có khả năng chống chịu tốt hơn

### 1.1.3. Ưu nhược điểm

- Ưu điểm:
  - Cung cấp một cách nhìn toàn diện về toàn bộ lưu lượng mạng.
  - Giúp kiểm tra các sự cố xảy ra với hệ thống mạng.
  - Sử dụng để thu thập bằng chứng cho điều tra và ứng cứu sự cố.
- Hạn chế:
  - Có thể gây ra tình trạng báo động nhầm nếu cấu hình không hợp lý.
  - Khả năng phân tích lưu lượng bị mã hóa tương đối thấp.
  - Chi phí triển khai và vận hành hệ thống tương đối lớn .

#### 1.1.4. Cấu trúc



Hình 1. Cấu trúc chung của một hệ thống IDPS

Cấu trúc chung của một hệ thống IDPS bao gồm:

- Bộ cảm biến (Sensor): thu thập dữ liệu từ hệ thống được giám sát.
- Bộ phát hiện : Thành phần này phân tích và tổng hợp thông tin từ dữ liệu thu được của bộ cảm biến dựa trên cơ sở tri thức của hệ thống
- Bộ lưu trữ : Lưu trữ tất cả dữ liệu của hệ thống IDS, bao gồm: dữ liệu của bộ cảm biến, dữ liệu phân tích của bộ phát hiện, cơ sở tri thức, cấu hình hệ thống ... nhằm phục vụ quá trình hoạt động của hệ thống IDS.
- Bộ phản ứng : Thực hiện phản ứng lại với những hành động phát hiện được.
- Giao diện người dùng

Hệ thống IDPS có thể chia làm 2 loại kiến trúc chính:

1. Network-based IDPS (NIDPS)
2. Host-based IDPS (HIDPS)

#### 1.1.5. Network-based IDPS (NIDPS)

- Chức năng: Thu thập và giám sát lưu lượng mạng dựa trên việc triển khai sensor tại nhiều điểm khác nhau trong mạng
  - Sensor có thể là các thiết bị có khả năng phân tích, tổng hợp và thống kê thông tin lưu lượng
  - Sensor phần mềm cài đặt trên một số nút mạng nhất định
- Các loại sensor sử dụng cho NIDPS:
  - Sensor nội tuyến(Inline sensor): đặt tại các vị trí mà lưu lượng mạng bắt buộc phải đi qua
  - Sensor thụ động (Passive sensor): có thể sao chép lưu lượng mạng không bắt buộc lưu lượng mạng phải đi qua

### 1.1.6. Host-based IDPS (HIDPS)

- Chức năng: thu thập và phân tích thông tin để phát hiện tấn công trên nút mạng cụ thể:
  - Lưu lượng đến và đi
  - Trạng thái của hệ thống: các tiến trình, quản lý tài nguyên, truy cập file, log, thay đổi cấu hình...
  - Hoạt động và trạng thái của các ứng dụng
- Mô hình tập trung:
  - Sensor đặt trên các nút mạng để thu thập thông tin
  - HIDPS Server: phân tích thông tin do sensor thu thập và phát cảnh báo tới nút mạng
- Mô hình phân tán: sensor và HIDPS triển khai trên cùng nút mạng

## 1.2. Học máy và ứng dụng trong các hệ thống phát hiện xâm nhập

### 1.2.1. Phát hiện xâm nhập dưới góc nhìn của bài toán phân lớp

Một hệ thống IDPS có thể được hiểu đơn giản, bao gồm sự kết hợp của 2 thành phần chính là IDS (Intrusion Detection System) và IPS (Intrusion Prevention System). Trong đó, muốn ngăn chặn (prevention), yếu tố tiên quyết là cần xác định được khi nào có tấn công/ xâm nhập, và cụ thể chi tiết lưu lượng của tấn công đó từ IDS. Nói cách khác, thành phần tiên quyết trong hệ thống IDPS chính là IDS, có nhiệm vụ phát hiện và giám sát, phân loại lưu lượng tấn công- lưu lượng bình thường và đưa ra chi tiết về các đặc trưng liên quan tới tấn công. Một hệ thống IDS hiệu quả là hệ thống có khả năng phát hiện/phân loại chính xác đâu là lưu lượng đáng ngờ/tấn công, đâu là lưu lượng bình thường.

Trong các thuật toán học máy, có rất nhiều thuật toán giúp giải quyết lớp các bài toán phân loại (classification). Một hệ thống IDS có thể nhìn dưới góc độ một bài toán phân loại, với dữ liệu đầu vào là các đặc trưng về lưu lượng mạng được trích xuất ra, và đầu ra là các nhãn cố định, chẳng hạn như 2 lớp: lưu lượng nguy hiểm và lưu lượng bình thường; hoặc nhiều lớp như: lưu lượng DOS, lưu lượng do thám (Probe), lưu lượng bình thường...

Áp dụng học máy, cụ thể là học có giám sát, có thể giúp ta huấn luyện/tìm ra được mô hình tốt cho việc phát hiện tấn công. Ưu điểm của phương pháp này, là độ chính xác nhờ vào việc tổng hợp và nhận biết dựa trên nhiều đặc trưng phức tạp; hơn thế là sự tự động học dựa trên lưu lượng và nhãn đã biết mà không cần thiết lập các luật kiểm soát một cách thủ công. Các thuật toán học máy ngày nay thể hiện rõ ưu thế và độ hiệu quả so với các phương pháp thủ công khác.

Mô hình hóa bài toán phát hiện xâm nhập sử dụng học máy:

- Giả sử có tập các bộ giá trị  $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$
- Yêu cầu: Ước lượng giá trị  $y^*$  nếu biết  $x^* \in \{x_1, x_2, \dots, x_n\}$ 
  1. B1: Thu thập tập huấn luyện trong đó dữ liệu đã được gán nhãn
  2. B2: Lựa chọn thuật toán phân lớp học tập huấn luyện. Các thuật toán điển hình: SVM, Decision Tree, Naive Bayes, Neural Network,...
  3. B3: Sử dụng tập kiểm tra đã được gán nhãn trước để kiểm tra tính đúng đắn của thuật toán phân lớp ở B2
  4. B4: Hiệu chỉnh thuật toán và lặp lại từ B2.

### 1.2.2. Đánh giá hệ thống phân lớp (classification)

Khi xây dựng một mô hình học máy, chúng ta cần một phép đánh giá để xem mô hình sử dụng có hiệu quả không và để so sánh khả năng của các mô hình. Có rất nhiều cách đánh giá một mô hình phân lớp. Tùy vào những bài toán khác nhau mà chúng ta sử dụng các phương pháp khác nhau. Các phương pháp thường được sử dụng là: accuracy score, confusion matrix, ROC curve, Area Under the Curve, Precision and Recall, F1 score, Top R error, ..

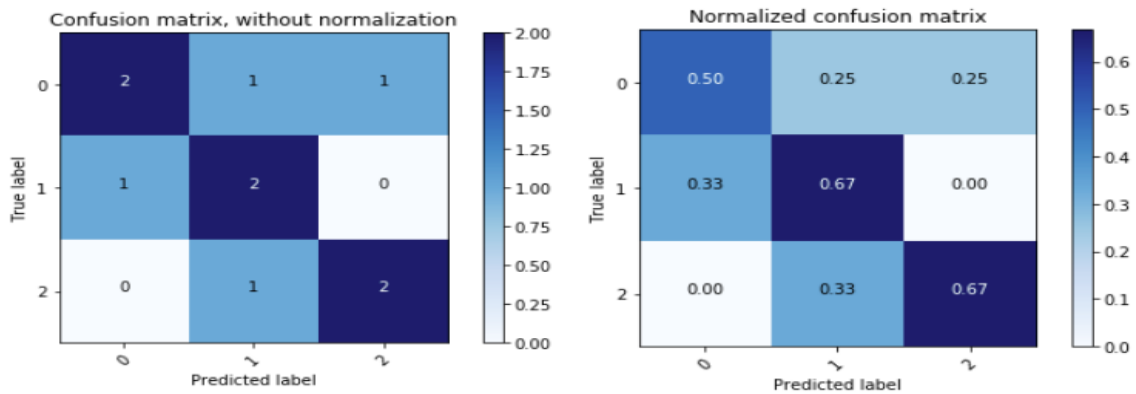
#### 1.2.2.1. Confusion matrix

Một độ đo thông thường được sử dụng trong bài toán phân lớp, đó là độ chính xác (accuracy). Cách đánh giá này đơn giản tính tỉ lệ giữa số điểm được dự đoán đúng và tổng số điểm trong tập dữ liệu kiểm thử, chỉ cho chúng ta biết được bao nhiêu phần trăm lượng dữ liệu được phân loại đúng mà không chỉ ra được cụ thể mỗi loại được phân loại như thế nào, lớp nào được phân loại đúng nhiều nhất, và dữ liệu thuộc lớp nào thường bị phân loại nhầm vào lớp khác. Để có thể đánh giá được các giá trị này, chúng ta sử dụng một ma trận được gọi là confusion matrix.

Về cơ bản, confusion matrix thể hiện có bao nhiêu điểm dữ liệu thực sự thuộc vào một class, và được dự đoán là rơi vào một class. Ví dụ đối với bài toán phân lớp 3 lớp, ta có confusion matrix 3\*3 sau đây:

Toàn bộ: 10	Dự đoán là 0	Dự đoán là 1	Dự đoán là 2
Thực sự là 0	2	1	1
Thực sự là 1	1	2	0
Thực sự là 2	0	1	2

Nó là một ma trận vuông với kích thước mỗi chiều bằng số lượng lớp dữ liệu. Giá trị tại hàng thứ  $i$ , cột thứ  $j$  là số lượng điểm lẽ ra thuộc vào class  $i$  nhưng lại được dự đoán là thuộc vào class  $j$ . Cách biểu diễn trên đây của confusion matrix còn được gọi là unnormalized confusion matrix (ma trận confusion chưa chuẩn hoá). Để có cái nhìn rõ hơn, ta có thể dùng normalized confusion matrix, tức confusion matrix được chuẩn hoá. Để có normalized confusion matrix, ta lấy mỗi hàng của unnormalized confusion matrix sẽ được chia cho tổng các phần tử trên hàng đó. Một mô hình tốt sẽ cho một confusion matrix có các phần tử trên đường chéo chính có giá trị lớn, các phần tử còn lại có giá trị nhỏ. Nói cách khác, khi biểu diễn bằng màu sắc, đường chéo có màu càng đậm so với phần còn lại sẽ càng tốt.



Hình 2. Ví dụ về confusion maxtrix (trái) và normalized confusion matrix (phải)

Cách đánh giá này thường được áp dụng cho các bài toán phân lớp có hai lớp dữ liệu. Cụ thể hơn, trong hai lớp dữ liệu này có một lớp nghiêm trọng hơn lớp kia và cần được dự đoán chính xác. Ví dụ, trong bài toán xác định có bệnh ung thư hay không thì việc không bị sót (miss) quan trọng hơn là việc chẩn đoán nhầm âm tính thành dương tính. Trong bài toán xác định có mìn dưới lòng đất hay không thì việc bỏ sót nghiêm trọng hơn việc báo động nhầm rất nhiều.

#### 1.2.2.2. True/False Positive/Negative

Cách đánh giá này thường được áp dụng cho các bài toán phân lớp có hai lớp dữ liệu. Cụ thể hơn, trong hai lớp dữ liệu này có một lớp nghiêm trọng hơn lớp kia và cần được dự đoán chính xác. Ví dụ, trong bài toán xác định có bệnh ung thư hay không thì việc không bị sót (miss) quan trọng hơn là việc chẩn đoán nhầm âm tính thành dương tính. Nhận xét rằng điều này cũng phù hợp hoàn toàn với bài toán phân loại lưu lượng trong IDS với 2 lớp là “ tấn công” và “bình thường”: việc không bị bỏ sót các hành vi tấn công quan trọng hơn rất nhiều so với việc báo động nhầm hành vi bình thường thành hành vi tấn công (báo động nhầm). Trong những bài toán này, người ta thường định nghĩa lớp dữ liệu quan trọng hơn cần được xác định đúng là lớp Positive (P-dương tính), lớp còn lại được gọi là Negative (N-âm tính). Ta định nghĩa True Positive (TP), False Positive (FP), True Negative (TN), False Negative (FN) dựa trên confusion matrix chưa chuẩn hoá như sau:

	Dự đoán là dương tính (P)	Dự đoán là âm tính (N)
Thực sự là dương tính (P)	True Positive (TP)	False Negative (FN)
Thực sự là âm tính (N)	False Positive (FP)	True Negative (TN)

Từ bảng trên, người ta thường quan tâm đến TPR, FNR, FPR, TNR (R - Rate) dựa trên normalized confusion matrix như sau:

	Dự đoán là dương tính (P)	Dự đoán là âm tính (N)
--	---------------------------	------------------------

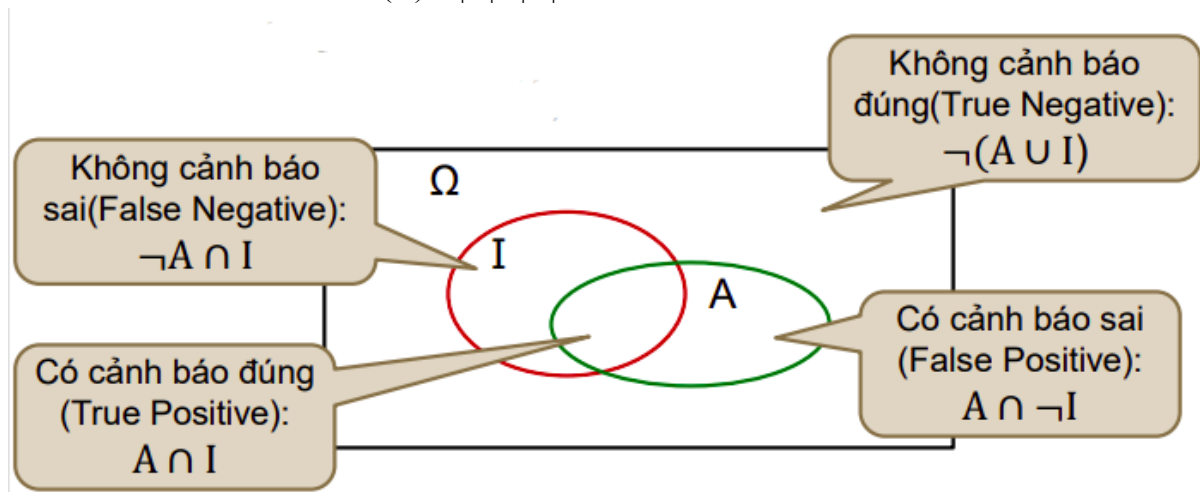
Thực sự là dương tính (P)	$TPR = TP/(TP + FN)$	$FNR = FN/(TP + FN)$
Thực sự là âm tính (N)	$FPR = FP/(FP + TN)$	$TNR = TN/(FP + TN)$

False Positive Rate còn được gọi là False Alarm Rate (tỉ lệ báo động nhầm), False Negative Rate còn được gọi là Miss Detection Rate (tỉ lệ bỏ sót). Thông thường trong bài toán xác định tấn công mạng trong IDS, thà báo nhầm còn hơn bỏ sót, tức là ta có thể chấp nhận False Alarm Rate cao để đạt được Miss Detection Rate thấp.

### 1.2.3. Áp dụng mô hình phân loại (classification) cho hệ thống IDS

Mô hình toán học cho một hệ thống IDS cũng dựa trên lí thuyết về mô hình xác suất thống kê:

- $\Omega$ : Tập các sự kiện có thể xảy ra trên hệ thống
- $I$ : Tập các sự kiện tấn công.
- Xác suất tấn công  $Pr(I) = |I| / |\Omega|$
- $A$ : Tập các cảnh báo tấn công
- Xác suất cảnh báo:  $Pr(A) = |A| / |\Omega|$



Hình 3. Trực quan hóa các trường hợp

- Xác suất phát hiện được tấn công khi có tấn công xảy ra:  
 $TPR = Pr(A|I) = Pr(A \cap I) / Pr(I)$
- Xác suất không phát hiện được tấn công khi có tấn công xảy ra:  
 $FNR = Pr(\neg A|I) = 1 - Pr(A|I)$
- Xác suất cảnh báo đúng khi phát ra cảnh báo:  
 $Pr(I|A) = Pr(A \cap I) / Pr(A)$
- Xác suất phát hiện nhầm khi phát ra cảnh báo:  
 $FPR = Pr(A|\neg I)$

Như vậy, cần cân bằng giữa FPR và FNR. Câu hỏi đặt ra là nên lựa chọn hệ thống có FPR thấp hay FNR thấp?

- Phụ thuộc vào sự mức độ thiệt hại của hệ thống với mỗi dạng lỗi xảy ra và chi phí để khắc phục
- Phụ thuộc vào tỉ lệ tấn công trên thực tế



## II. Một số mô hình học máy cơ bản

Phần này không tập trung đi sâu vào chi tiết lý thuyết/toán học, phương pháp tối ưu đối với từng thuật toán mà chỉ tập trung giới thiệu khái quát ý tưởng và điểm mạnh, điểm yếu cũng như chú ý khi sử dụng các thuật toán học máy.

### 2.1. Logistic Regression

Phương pháp hồi quy logistic (logistic regression) là một mô hình học máy có giám sát (supervised learning). Trong lý thuyết thống kê, một mô hình logistic mô hình hóa xác suất các lớp/ sự kiện nào đó xảy ra. Logistic regression là một mô hình phù hợp với lớp các bài toán mà output hướng tới nằm trong khoảng  $[0, 1]$ . Trái với tên gọi khiến ta liên tưởng tới một mô hình hồi quy (regression), logistic regression là mô hình được sử dụng nhiều trong lớp các bài toán classification: phân loại các đầu vào  $x$  vào các lớp (class)  $y$  tương ứng. Thông thường, ta quan tâm tới 2 class là positive và negative, diễn hình như sự kiện nào đó xảy ra/ không xảy ra, đúng/sai, ..

Có thể mở rộng mô hình ra thành mô hình multi-class logistic regression với nhiều lớp bằng cách kết hợp output của nhiều mô hình logistic regression 2 lớp, bằng các phương pháp như one-vs-one, one-vs-rest.

Mô hình hồi quy tuyến tính (Linear Regression) đơn giản được xây dựng từ giả thuyết rằng đầu ra  $y$  là tổ hợp tuyến tính của các thành phần trong vector đầu vào  $x$ :

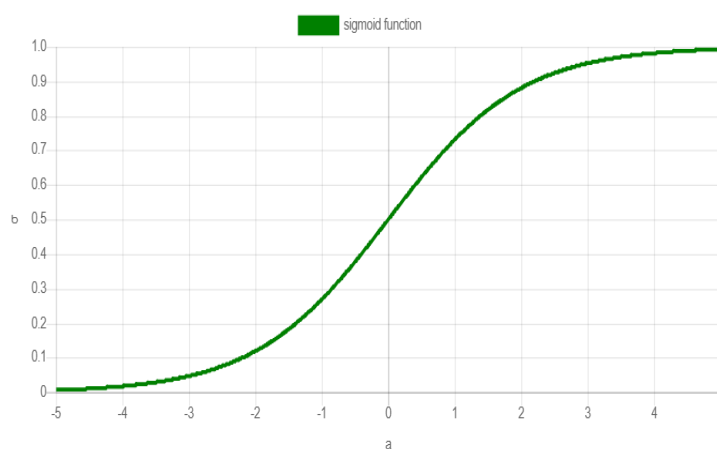
$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$$

Với Logistic Regression, tổ hợp tuyến tính này được đi qua 1 activation function gọi là logistic activation, thường là hàm phi tuyến (non-linear function):

$$f(\mathbf{x}) = \theta(\mathbf{w}^T \mathbf{x})$$

Trong đó, logistic activation thường được chọn là hàm sigmoid:

$$f(s) = \frac{1}{1 + e^{-s}} \triangleq \sigma(s)$$



Hình 1. Đồ thị hàm sigmoid  $\sigma(a)$

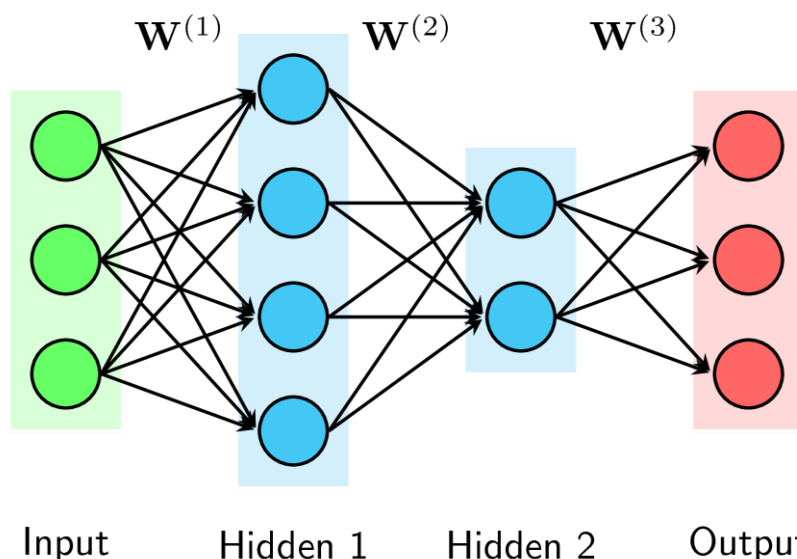
Hình 4. Biểu diễn đồ thị của hàm sigmoid

Ưu nhược điểm:

- Không cần có giả thiết dữ liệu hai class là linearly separable. Tuy nhiên, boundary tìm được vẫn có dạng tuyến tính. Vậy nên mô hình này chỉ phù hợp với loại dữ liệu mà hai class là gần với linearly separable. Một kiểu dữ liệu mà Logistic Regression không làm việc được là dữ liệu mà một class chứa các điểm nằm trong 1 vòng tròn, class kia chứa các điểm bên ngoài đường tròn đó. Kiểu dữ liệu này được gọi là phi tuyến (non-linear).
- Logistic Regression là nó yêu cầu các điểm dữ liệu được tạo ra một cách độc lập với nhau. Mặc dù vậy, để cho đơn giản, khi xây dựng mô hình, người ta vẫn thường giả sử các điểm dữ liệu là độc lập với nhau.

## 2.2. Mạng neuron nhiều lớp (Multi-layer Perceptron)

Ngoài Input layers và Output layers, một Multi-layer Perceptron (MLP) có thể có nhiều lớp ẩn (hidden layers) ở giữa:



Hình 5. Multi-layer Perceptron (neural network) với 2 hidden layer

Mỗi lớp ẩn và lớp output, chứa các node gọi là neuron, là tổ hợp tuyến tính của tất cả các neuron ở layer trước đã đi qua activation function. Vì cách biểu diễn này, nó được gọi là mạng neuron nhiều lớp, với lớp output được biểu diễn là một hàm số thường là phức tạp đối với input (nhờ đi qua các tổ hợp tuyến tính và activation phi tuyến). Việc tối ưu trong huấn luyện mô hình này, người ta sử dụng kỹ thuật lan truyền ngược (backpropagation), dựa trên chain-rule để tính đạo hàm của loss theo giá trị các tham số (weights) của mô hình.

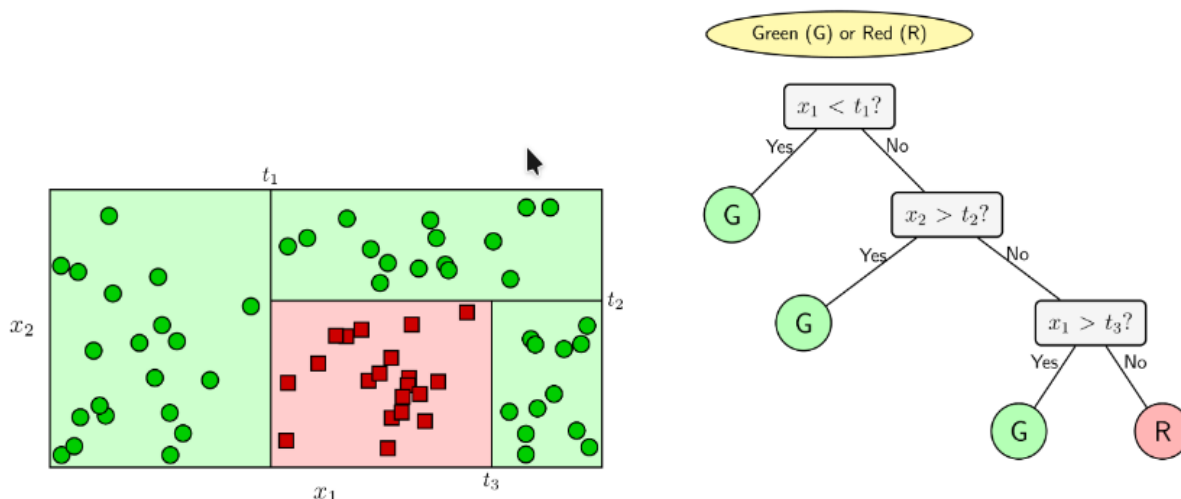
Ưu nhược điểm:

- Người ta đã chứng minh rằng có thể xấp xỉ hầu hết các hàm liên tục bởi một MLP (neural network) với chỉ 1 hidden layer với số neuron đủ lớn và sử dụng non-linear activation function phù hợp.
- Thực nghiệm chứng minh rằng Neural Networks với nhiều hidden layers kết hợp với các nonlinear activation function (đơn giản như ReLU) có khả năng xấp xỉ (khả năng biểu diễn) dữ liệu tốt hơn.

- Khi số lượng hidden layers lớn lên, số lượng hệ số cần tối ưu cũng lớn lên và mô hình sẽ trở nên phức tạp. Sự phức tạp này ảnh hưởng tới hai khía cạnh. Thứ nhất, tốc độ tính toán sẽ bị chậm đi rất nhiều. Thứ hai, nếu mô hình quá phức tạp, có thể gây ra hiện tượng overfitting (mô hình học và thể hiện tốt trên train set nhưng thể hiện kém trên test set).

### 2.3. Cây quyết định (Decision Tree)

Decision tree là một mô hình supervised learning, có thể được áp dụng vào cả hai bài toán classification và regression. Việc xây dựng một decision tree trên dữ liệu huấn luyện cho trước là việc đi xác định các câu hỏi và thứ tự của các đặc trưng dữ liệu được sử dụng. Một điểm đáng lưu ý của decision tree là nó có thể làm việc với các đặc trưng dạng categorical, thường là rời rạc và không có thứ tự. Ví dụ, mưa, nắng hay xanh, đỏ, v.v. Decision tree cũng làm việc với dữ liệu có vector đặc trưng bao gồm cả thuộc tính dạng categorical và liên tục (numeric).



Hình 6. Minh họa cách thức hoạt động của cây quyết định

Ưu nhược điểm:

- Các mô hình cây quyết định là đơn giản, dễ hiểu, và dễ giải thích.
- Thường ít yêu cầu dữ liệu được chuẩn hóa, có thể xử lý cả thuộc tính dạng số và phân loại (categorical). Đồng thời, nó có thể hoạt động tốt ngay cả trong trường hợp dữ liệu đầu vào có phân phối không đúng theo giả định của thuật toán.
- Có thể tạo ra những cây quyết định quá phức tạp (quá sâu,..) dẫn tới hiện tượng overfitting. Tuy nhiên, cũng có một số cơ chế cắt tỉa cây (pruning) cho phép giảm thiểu điều này.
- Cây quyết định có thể không ổn định, vì chỉ một biến thể/nhiều nhỏ trong dữ liệu có thể dẫn tới việc một cây quyết định hoàn toàn khác tạo ra.
- Bài toán tối ưu cây quyết định là bài toán NP khó, chỉ có thể tối ưu cục bộ sử dụng giải thuật tham lam.
- Cây quyết định dễ bị thiên lệch nếu một lớp chiếm số đông hơn phần còn lại rất nhiều (class-imbalance).

## 2.4. Random Forest

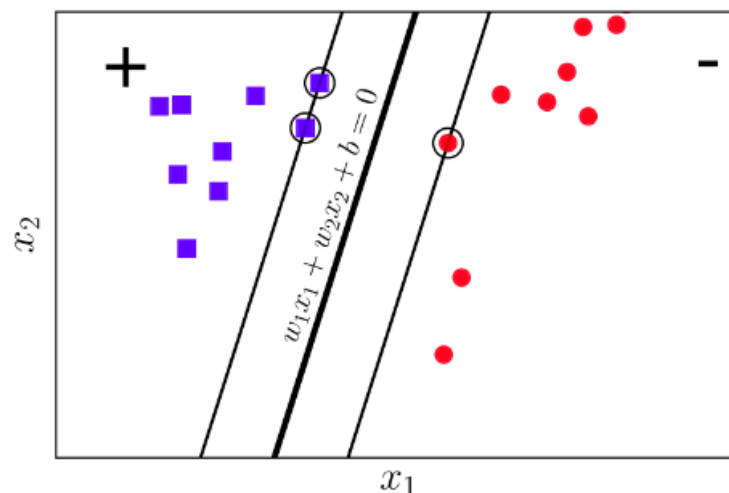
Như tên gọi của nó Random Forest — rừng ngẫu nhiên: đây là phương pháp xây dựng một tập hợp rất nhiều cây quyết định và sử dụng phương pháp voting để đưa ra quyết định về biến target cần được dự báo. Về mặt toán học thuật toán có thể được giải thích như sau: Random Forest là một tập hợp của hàng trăm cây quyết định, trong đó mỗi cây quyết định lại được tạo nên ngẫu nhiên từ việc tái chọn mẫu (chọn ngẫu nhiên một phần của data để xây dựng) và ngẫu nhiên lựa chọn tập đặc trưng là tập con của tập đặc trưng ban đầu.

Ưu nhược điểm:

- Khó bị overfit hơn so với Decision Tree nhờ cơ chế chỉ sử dụng ngẫu nhiên một tập con dữ liệu huấn luyện trong quá trình training mỗi cây quyết định.
- Với cơ chế như vậy, Random Forest cho ta một kết quả chính xác rất cao nhưng đánh đổi bằng việc ta không thể hiểu cơ chế hoạt động của thuật toán này do cấu trúc quá phức tạp của mô hình này — do vậy thuật toán này là một trong những phương thức black-box.
- Thời gian huấn luyện và tính toán lâu hơn (nhược điểm chung của kỹ thuật ensemble learning).

## 2.5. Support Vector Machine

Mô hình SVM là một dạng mô hình phân loại tuyến tính. Bài toán phân loại 2 class có thể được xem là bài toán đi tìm một siêu phẳng ngăn cách 2 class với nhau trong không gian  $n$  chiều ( $n$  là số đặc trưng). Mô hình SVM thường cho kết quả phân loại tốt đối với các bài toán mà dữ liệu là linear separable, nguyên nhân là do khác với mô hình Logistic Regression, SVM có tính chất large margin. Cụ thể, chúng ta cần một đường phân chia sao cho khoảng cách từ điểm gần nhất của mỗi class tới đường phân chia là lớn nhất có thể, và tương đương nhau, khoảng cách như nhau này được gọi là margin (lề). Việc margin rộng hơn sẽ mang lại hiệu ứng phân lớp tốt hơn vì sự phân chia giữa hai classes là rạch ròi hơn. Bài toán tối ưu trong Support Vector Machine (SVM) chính là bài toán đi tìm đường phân chia sao cho margin là lớn nhất.



Hình 7. Minh họa tính large-margin giữa 2 lớp của SVM

Đối với bài toán phân loại đa lớp (multiclass classification), có thể sử dụng phương pháp one-vs-rest giống với Logistic Regression.

## 2.6. Naive Bayes

Xét bài toán:

- Tập huấn luyện  $S = \{S_1, S_2, \dots, S_m\}$
- Mỗi mẫu  $S_i = \{x_1, x_2, \dots, x_n\}$
- Đầu ra  $k$  lớp  $C_1, C_2, \dots, C_k$

Khi biết mẫu  $X$ , giả thuyết  $X$  thuộc một trong  $k$  lớp sẽ tương ứng với xác suất nào cao nhất trong

$P(C_i | X)$

Theo công thức xác suất Bayes:

$$P(C_i | X) = P(X | C_i) \times P(C_i) / P(X)$$

Trong đó,  $P(C_i)$  là xác suất một mẫu thuộc vào lớp  $C_i$ , chính bằng số lượng mẫu thuộc lớp  $C_i$  chia cho tổng số mẫu.  $P(X)$  là hằng số. Ta chỉ cần tìm maximum của  $P(X | C_i)$ .

Thành phần  $P(X | C_i)$  thường rất khó tính toán vì  $x$  là một biến ngẫu nhiên nhiều chiều, cần rất nhiều dữ liệu training để có thể xây dựng được phân phối đó. Mô hình Naive Bayes là mô hình dựa trên xác suất thống kê, trong đó giả thiết rằng các thuộc tính đầu vào là đôi một độc lập. Khi đó:

$$P(X / C_i) = \prod_{t=1}^n P(x_t / C_i)$$

Các xác suất tiên nghiệm  $P(x_t | C_i)$  có thể được tính toán dựa trên thống kê, hoặc sử dụng các kỹ thuật ước lượng như MLE (Maximum Likelihood Estimation) hoặc MAP (Maximum A Posteriori). Tương ứng với các giả định khác nhau về phân phối xác suất  $P(x_t | C_i)$ , sẽ có các mô hình Naive Bayes khác nhau như Gaussian Naive Bayes, Bernoulli Naive Bayes, Multinomial Naive Bayes, ..

Ưu nhược điểm:

- Tốc độ training và test rất nhanh, hiệu quả cao với các bài toán large scale với số lượng mẫu lớn hơn rất nhiều so với số lượng thuộc tính
- Giả thuyết độc lập giữa các thuộc tính thường xuyên không đúng thực tế. Nếu vi phạm quá lớn, dẫn tới kết quả mô hình có thể rất tệ.
- Mô hình Naive Bayes có thể hoạt động với các thuộc tính liên tục (phù hợp với Gaussian likelihood) hoặc rời rạc (phù hợp hơn với Multinomial hoặc Bernoulli likelihood)

## III. Phân tích bộ dữ liệu KDD99

### 3.1. Bộ dữ liệu KDD99

KDD Cup 1999 (KDD99) là bộ dữ liệu được sử dụng cho Cuộc thi khai phá dữ liệu và tri thức quốc tế lần thứ ba (The Third International Knowledge Discovery and Data Mining

Tools Competition). Nhiệm vụ là xây dựng một trình phát hiện xâm nhập mạng, một mô hình dự đoán có khả năng phân biệt giữa các kết nối xấu, được coi là xâm nhập/tấn công và các kết nối bình thường còn lại. Chi tiết tại <http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html>

### 3.1.1. Tổng quan

Bộ dữ liệu KDD99 là dữ liệu đã được sàng lọc và trích xuất từ lưu lượng thô (TCP dump data). Lincoln Labs đã thiết lập một môi trường để thu thập dữ liệu lưu lượng thô trong một mạng cục bộ mô phỏng một mạng LAN điển hình của không quân Hoa Kỳ lúc bấy giờ, trong 9 tuần. Họ sử dụng và vận hành mạng LAN đó như bình thường, tuy nhiên thêm vào các cuộc tấn công đa dạng khác.

Một kết nối (connection) là một chuỗi các gói TCP bắt đầu và kết thúc tại một số thời điểm được xác định, trong đó dữ liệu từ một địa chỉ IP nguồn đến một địa chỉ IP đích theo một giao thức xác định. Mỗi kết nối được gắn nhãn là “bình thường” hoặc là một cuộc tấn công, với chi tiết tên một loại tấn công cụ thể.

Các tấn công có thể được phân chia vào bốn loại chính:

1. **DOS (Denial Of Service)**: từ chối dịch vụ
2. **R2L (Remote to Local)**: truy cập trái phép từ một máy từ xa, ví dụ: đoán mật khẩu;
3. **U2R (User to Root)**: truy cập trái phép vào các đặc quyền super user (root) cục bộ, ví dụ: các cuộc tấn công tràn bộ đệm,..
4. **Probe**: giám sát và thăm dò, ví dụ: quét mạng, quét cổng,..

### 3.1.2. Các đặc trưng trong bộ dữ liệu

Các đặc trưng được trích xuất từ lưu lượng thô, đã bao gồm các đặc trưng cấp cao mang đặc tính domain-knowledge.

- Các đặc trưng “same host” thống kê các kết nối trong 2 giây trước đó mà có cùng địa chỉ đích như kết nối đang xét, và tính toán các số liệu thống kê liên quan đến giao thức, dịch vụ,..
- Các đặc trưng “same service” tương tự chỉ thống kê các kết nối trong hai giây trước đó có cùng dịch vụ với kết nối đang xét.
- Các đặc trưng “same host” và “same service” được gọi chung là các đặc trưng dựa trên thời gian (time-based features).
- Một số cuộc tấn công thăm dò quét máy chủ (hoặc cổng) thường diễn ra trong khoảng thời gian lớn hơn nhiều so với 2 giây, ví dụ 1 lần mỗi phút. Do đó, các bản ghi kết nối cũng được sắp xếp theo địa chỉ đích và các đặc trưng mới được xây dựng bằng cách sử dụng cửa sổ với kích thước 100 kết nối thay vì cửa sổ thời gian. Các đặc trưng mới này được xếp vào nhóm các đặc trưng dựa trên địa chỉ đích (host-based traffic features).
- Khác hẳn với hầu hết các cuộc tấn công của DOS và thăm dò, dường như không có mô hình tuần tự nào thường gặp trong các bản ghi về các cuộc tấn công R2L và U2R. Điều này là do các cuộc tấn công DOS và thăm dò liên quan đến nhiều kết nối đến một số máy chủ trong một khoảng thời gian rất ngắn, nhưng dấu hiệu của các cuộc tấn công R2L và U2R lại được nhúng trong các thành phần dữ liệu (content) của các gói và thường chỉ liên quan đến một kết nối. Stolfo và cộng sự đã sử dụng kiến thức chuyên sâu để thêm các tính năng tìm kiếm hành vi đáng ngờ trong các phần dữ liệu,

chẳng hạn như số lần thử đăng nhập thất bại. Các đặc trưng này gọi là các đặc trưng về nội dung (content features).

Chi tiết các đặc trưng của bộ dữ liệu:

Tên đặc trưng	Mô tả	Kiểu dữ liệu
duration	Thời gian (giây) của một kết nối	Liên tục
protocol_type	Loại giao thức: tcp, udp,	Rời rạc
service	Dịch vụ ở địa chỉ đích: http, telnet, ..	Rời rạc
src_bytes	Số byte gửi từ nguồn tới đích	Liên tục
dst_bytes	Số byte gửi từ đích tới nguồn	Liên tục
flag	Cờ trạng thái của kết nối	Rời rạc
land	1 nếu kết nối có địa chỉ:cổng nguồn và đích giống nhau	Rời rạc
wrong_fragment	Số mảnh (fragment) không hợp lệ	Liên tục
urgent	Số lượng gói tin urgent	Liên tục

*Bảng 1. Các đặc trưng chung của các kết nối TCP*

Tên đặc trưng	Mô tả	Kiểu dữ liệu
hot	Số “hot” indicators	Liên tục
num_failed_logins	Số lần đăng nhập thất bại	Liên tục
logged_in	1 nếu đăng nhập thành công, ngược lại là 0	Rời rạc
num_compromised	Số điều kiện “compromised”	Liên tục
root_shell	1 nếu root shell được lấy thành công, ngược lại 0	Rời rạc
su_attempted	1 nếu phát hiện cố gắng thực hiện lệnh “su root”	Rời rạc
num_root	Số truy cập “root”	Liên tục
num_file_creations	Số lần tạo file	Liên tục
num_shells	Số lượng shell prompt	Liên tục
num_access_files	Số lần truy cập các file điều khiển	Liên tục
num_outbound_cmds	Số câu lệnh outbound trong phiên ftp	Liên tục

is_hot_login	1 nếu phiên đăng nhập thuộc về danh sách “hot”	Rời rạc
is_guest_login	1 nếu lần đăng nhập là đoán	Rời rạc

*Bảng 2. Các đặc trưng về nội dung trong kết nối*

Tên đặc trưng	Mô tả	Kiểu dữ liệu
count	Số lượng kết nối tới cùng host trong 2s vừa qua	Liên tục
<i>Các đặc trưng sau chỉ có ở những kết nối tới cùng host</i>		
serror_rate	Tỉ lệ kết nối có lỗi SYN	Liên tục
rerror_rate	Tỉ lệ kết nối có lỗi REJ	Liên tục
same_srv_rate	Tỉ lệ kết nối tới cùng 1 dịch vụ	Liên tục
diff_srv_rate	Tỉ lệ kết nối tới dịch vụ khác	Liên tục
srv_count	Số kết nối tới dùng dịch vụ trong 2s vừa qua	Liên tục
<i>Các đặc trưng sau chỉ có ở các kết nối tới cùng 1 dịch vụ</i>		
srv_serror_rate	Tỉ lệ kết nối có lỗi SYN	Liên tục
srv_rerror_rate	Tỉ lệ kết nối có lỗi REJ	Liên tục
srv_diff_host_rate	Tỉ lệ kết nối tới host khác	Liên tục

*Bảng 3. Các đặc trưng thống kê theo thời gian trên cửa sổ 2s*

## 3.2. Phân tích và xử lý dữ liệu

Dữ liệu sử dụng để huấn luyện và đánh giá mô hình là bộ dữ liệu đầy đủ có thể tải về từ trang chủ của bộ dữ liệu: <http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html>

### 3.2.1. Làm sạch dữ liệu

Dữ liệu không sạch (có chứa lỗi, nhiễu, không đầy đủ, có mâu thuẫn) có thể làm cho các tri thức khám phá được sẽ bị ảnh hưởng và không đáng tin cậy, sẽ dẫn đến các quyết định không chính xác, hoặc làm cho mô hình bị thiên lệch, học những lỗi trong dữ liệu thay vì học những yếu tố thực sự ảnh hưởng. Do đó, cần gán các giá trị thuộc tính còn thiếu; sửa chữa các dữ liệu nhiễu/lỗi; xác định hoặc loại bỏ các nhiễu, ngoại lệ (outliers); gỡ bỏ, giải quyết mâu thuẫn trong dữ liệu.

Trên thực tế dữ liệu thu có thể, và thường xuyên chứa nhiễu, lỗi, không hoàn chỉnh, có mâu thuẫn, bao gồm:



- Không hoàn chỉnh (incomplete): Thiếu các giá trị thuộc tính hoặc thiếu một số thuộc tính.
- Nhiễu/lỗi (noise/error): Chứa đựng những lỗi hoặc các mang các giá trị bất thường.
- Mâu thuẫn (inconsistent): Chứa đựng các mâu thuẫn về logic, không thống nhất.
- Trùng lặp (duplication): Chứa quá nhiều các bản ghi trùng lặp nhau.

Sau khi tải về và giải nén, ta được dữ liệu dưới dạng csv với kích thước 743 MB. Dữ liệu bao gồm 4898431 dòng, mỗi dòng đại diện cho một connection với 41 đặc trưng đã được trích chọn nêu trên cùng với cột cuối cùng là nhãn tương ứng đánh dấu kết nối đó là bình thường hay tấn công với một phương thức tấn công cụ thể.

```
[7] 1 # data prepairing
    2 df=pd.read_csv(DATA_PATH, names=COLUMNS)
    3 print(df.shape)
    4 df.sample(20)
```

↳ (4898431, 42)

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot
248513	0	tcp	http	SF	218	340	0	0	0	0
3654569	0	tcp	private	S0	0	0	0	0	0	0
1654419	0	icmp	ecr_i	SF	1032	0	0	0	0	0
4619831	0	tcp	private	REJ	0	0	0	0	0	0
2292393	0	icmp	ecr_i	SF	1032	0	0	0	0	0
4541248	2638	udp	other	SF	147	105	0	0	0	0
1049773	0	tcp	http	SF	303	1377	0	0	0	0
3335259	0	icmp	ecr_i	SF	1032	0	0	0	0	0
4798181	0	tcp	http	SF	301	480	0	0	0	0
1974821	0	icmp	ecr_i	SF	1032	0	0	0	0	0
1228969	0	tcp	private	S0	0	0	0	0	0	0
3159659	0	icmp	ecr_i	SF	1032	0	0	0	0	0

Sau khi load dữ liệu, thư viện pandas cung cấp hàm cho phép phân tích nhanh các thông số thống kê tương ứng với từng cột trong bảng. Các thông số thống kê bao gồm: số lượng bản ghi (count), số lượng bản ghi khác nhau (unique), giá trị chiếm ưu thế (số lượng nhiều nhất) trong cột (top), kỳ vọng (mean), độ lệch chuẩn (std), min, max, percentile ở các vị trí 25%, 50%, 75%.

```
[8] 1 df.describe(include='all')
```

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment
count	4.898431e+06	4898431	4898431	4898431	4.898431e+06	4.898431e+06	4.898431e+06	4.898431e+06
unique	NaN	3	70	11	NaN	NaN	NaN	NaN
top	NaN	icmp	ecr_i	SF	NaN	NaN	NaN	NaN
freq	NaN	2833545	2811660	3744328	NaN	NaN	NaN	NaN
mean	4.834243e+01	NaN	NaN	NaN	1.834621e+03	1.093623e+03	5.716116e-06	6.487792e-04
std	7.233298e+02	NaN	NaN	NaN	9.414311e+05	6.450123e+05	2.390833e-03	4.285434e-02
min	0.000000e+00	NaN	NaN	NaN	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
25%	0.000000e+00	NaN	NaN	NaN	4.500000e+01	0.000000e+00	0.000000e+00	0.000000e+00
50%	0.000000e+00	NaN	NaN	NaN	5.200000e+02	0.000000e+00	0.000000e+00	0.000000e+00
75%	0.000000e+00	NaN	NaN	NaN	1.032000e+03	0.000000e+00	0.000000e+00	0.000000e+00
max	5.832900e+04	NaN	NaN	NaN	1.379964e+09	1.309937e+09	1.000000e+00	3.000000e+00

Dữ liệu trùng lặp (giống nhau tất cả các trường) có thể gây ảnh hưởng xấu tới mô hình học máy. Bộ dữ liệu KDD99 chứa một số lượng rất lớn bản ghi trùng lặp nhau (trên 75% số lượng bản ghi là trùng lặp). Cụ thể, có thể gây hiện tượng thiên lệch (classes-imbalance) giữa các lớp dữ liệu. Lúc này, mô hình được huấn luyện theo cách “thiên vị”, có thiên hướng chỉ tập trung dự đoán chính xác các mẫu chiếm số lượng đông đảo nhất. Mặt khác, việc dữ liệu bị trùng lặp còn gây ảnh hưởng tới việc đo lường và đánh giá mô hình.

Thử loại bỏ các dòng giống hệt nhau trong dữ liệu, ta thu được dữ liệu mới với chỉ 1074992 dòng. Điều này cho thấy dữ liệu ban đầu bị trùng lặp quá nhiều (gấp khoảng 4.56 lần so với khi đã loại bỏ trùng lặp).

	Số lượng ban đầu	Số lượng khác nhau	Tỉ lệ giảm
Tấn công	3,925,650	262,178	93.32%
Bình thường	972,781	812,814	16.44%
Tổng	4,898,431	1,074,992	78.05%

### 3.2.2. Trực quan hóa dữ liệu (visualization)

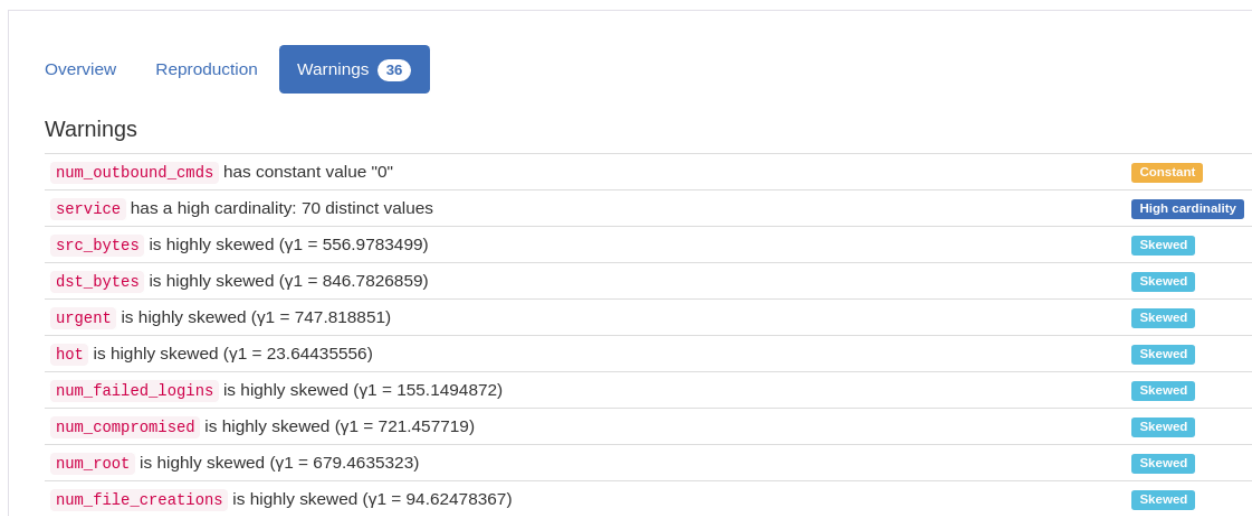
Việc trích xuất các đặc tính thống kê và phân tích về phân phối, giá trị của dữ liệu luôn là bước không thể thiếu trong quá trình phân tích và xây dựng các mô hình học máy trên một bộ dữ liệu cụ thể. Việc nắm bắt các thông tin, hiểu biết về bộ dữ liệu là yếu tố quan trọng để có thể phân tích, lựa chọn cách thức xử lý dữ liệu và lựa chọn mô hình học máy phù hợp với đặc tính của bộ dữ liệu đó. Các công việc này thông thường có nhiều yếu tố lặp đi lặp lại, và tốn thời gian để viết các đoạn mã cho cùng một mục đích, trên các bộ dữ liệu khác nhau.

pandas-profiling (<https://github.com/pandas-profiling/pandas-profiling>) là một thư viện mã nguồn mở viết trên ngôn ngữ lập trình python, tương thích với pandas và cho phép trích xuất các đặc tính thống kê của dữ liệu ở dạng DataFrame. Pandas-profiling hỗ trợ các tính năng

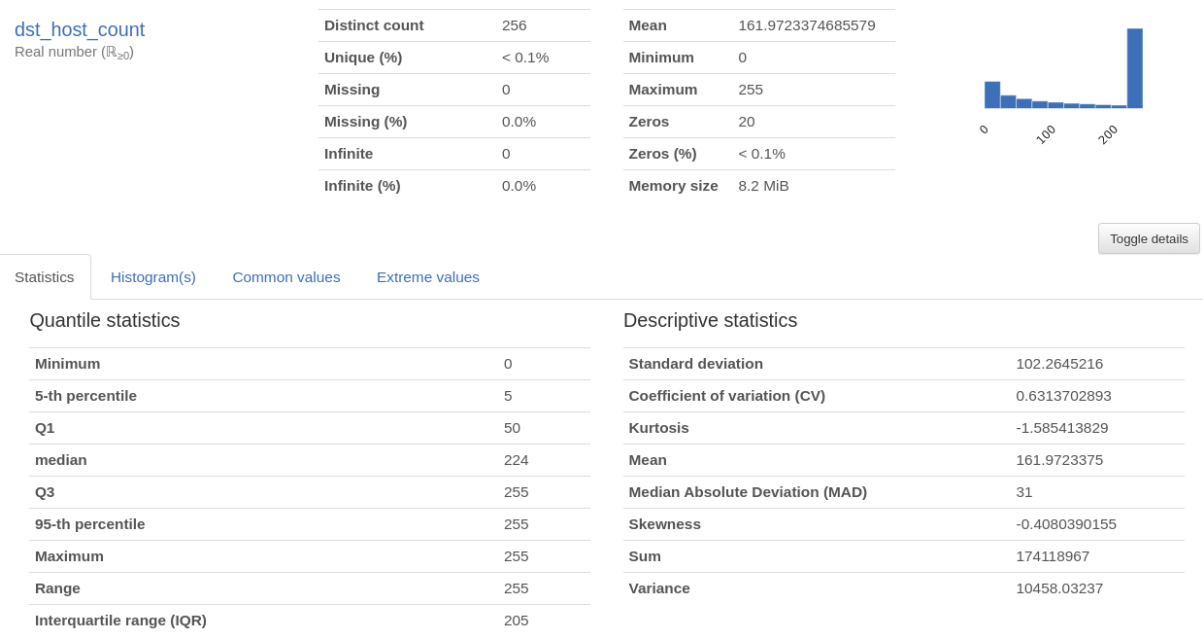
trích xuất thông tin và xuất ra dưới dạng HTML, phù hợp với môi trường ipython notebook mà ta sử dụng trong phân tích và xây dựng mô hình:

- Nhận biết kiểu dữ liệu: nhận biết kiểu dữ liệu của từng cột
- Kiểu dữ liệu, các giá trị khác nhau (unique values), giá trị bị khuyết (missing values)
- Thống kê định lượng: giá trị lớn/nhỏ nhất, Q1, trung vị, Q3, percentile,..
- Thống kê mô tả: kỳ vọng, độ lệch chuẩn, mode, hiệp phương sai, độ xiên (skewness)
- Các giá trị có tần suất xuất hiện cao nhất.
- Histogram trên từng cột
- Các độ đo tương quan: các biến tương quan, ma trận Spearman/Pearson/Kendall
- Heatmap, giá trị khuyết, ..

## Overview



Hình 8. Các cảnh báo từ báo cáo của pandas profiling



Hình 9. Minh họa các đặc tính thống kê của đặc trưng dst\_host\_count

Tiếp đến, phân tích số lượng mẫu của từng dạng tấn công trong dữ liệu:

```
[12] 1 df['label'].value_counts()/df.shape[0]
```

```
normal.      0.756112
neptune.     0.225257
satan.       0.004669
ipsweep.     0.003463
portsweep.   0.003315
smurf.       0.002797
nmap.        0.001446
back.        0.000900
teardrop.    0.000854
warezclient. 0.000831
pod.         0.000192
guess_passwd. 0.000049
buffer_overflow. 0.000028
warezmaster. 0.000019
land.        0.000018
imap.        0.000011
rootkit.     0.000009
loadmodule.  0.000008
ftp_write.   0.000007
multihop.    0.000007
phf.         0.000004
perl.        0.000003
spy.         0.000002
Name: label, dtype: float64
```

Dữ liệu chứa 22 dạng tấn công. Số mẫu thuộc lớp bình thường chiếm tới 75.6% dữ liệu.

24.4% lượng dữ liệu còn lại thuộc lớp tấn công thì có tới 22.5% là kiểu tấn công DOS

neptune. Dữ liệu rất không cân bằng dưới góc nhìn bài toán phân loại nhiều lớp (multi-classes classification), tuy nhiên dưới góc nhìn của bài toán phân loại 2 lớp (binary classification), dữ liệu ở tỉ lệ 75-25 đạt mức độ cân bằng bình thường/chấp nhận được. Tuy nhiên với tần suất các kiểu tấn công trong mẫu như thế, các mẫu thuộc lớp tấn công có thể sẽ mang nhiều đặc trưng của kiểu tấn công DOS neptune hơn.

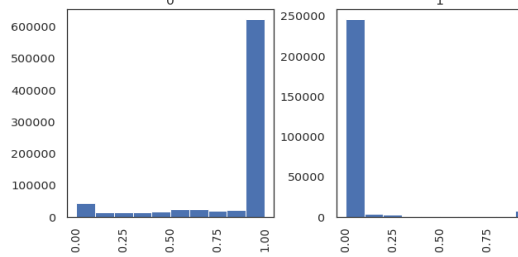
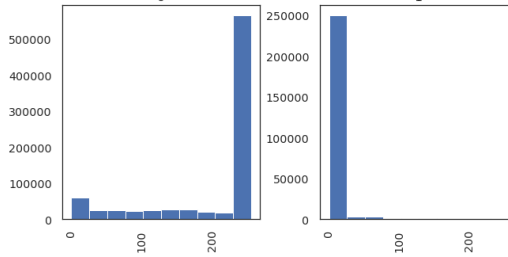
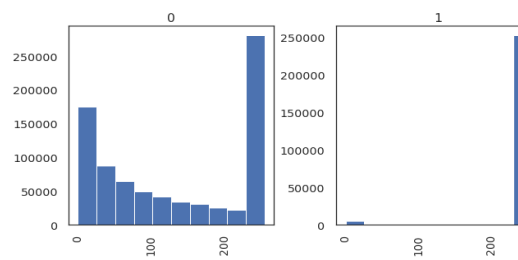
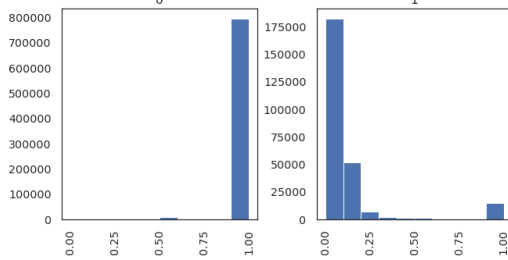
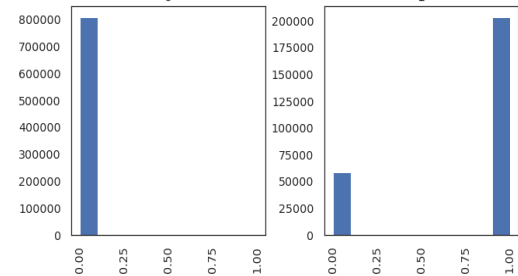
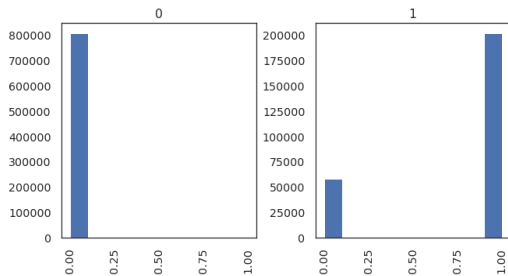
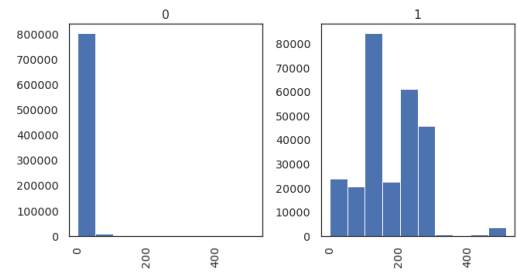
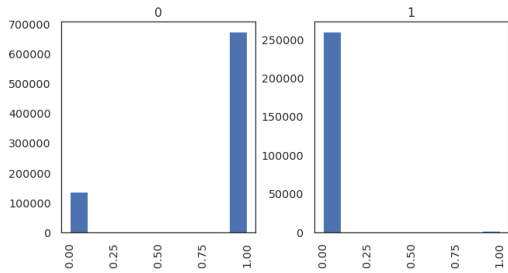
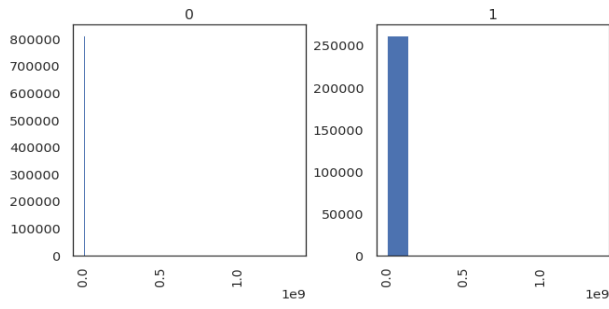
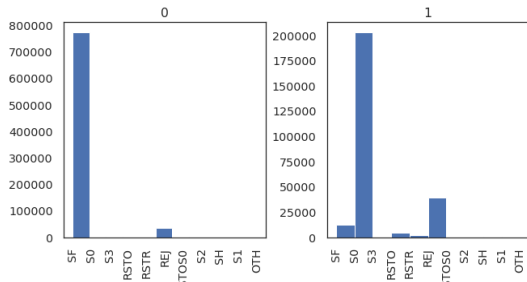
Trực quan hóa phân phối của từng đặc trưng trong bộ dữ liệu dưới dạng biểu đồ histogram:

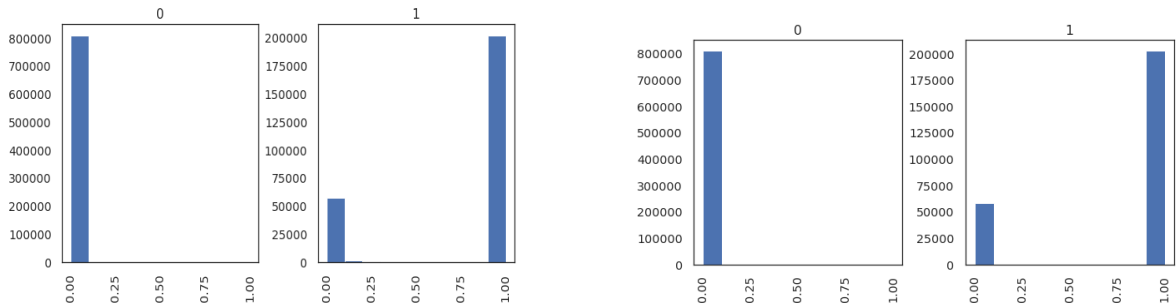


Hình 10. Trực quan hóa phân phối của từng thuộc tính bằng biểu đồ histogram

Nhận xét rằng có rất nhiều đặc trưng có độ lệch (skewness) lớn. Song song với đó, các đặc trưng ở dạng liên tục (continuous features) có phân phối rất khác phân phối chuẩn, hoặc có ngoại lệ rất khác biệt so với phân phối chung của đặc trưng đó (thường lớn hơn rất nhiều- lệch về phía bên phải). Ta có thể đoán được những ngoại lệ này thuộc vào lớp tấn công, và có thể thuộc vào các dạng tấn công có số lượng mẫu quá nhỏ. Lí do thứ hai có thể là do dữ liệu chưa đủ tính bao quát tổng thể (general) mà mới chỉ đang ở mức rải rác với số lượng mẫu ít và độ đa dạng trong giá trị chưa đủ cao.

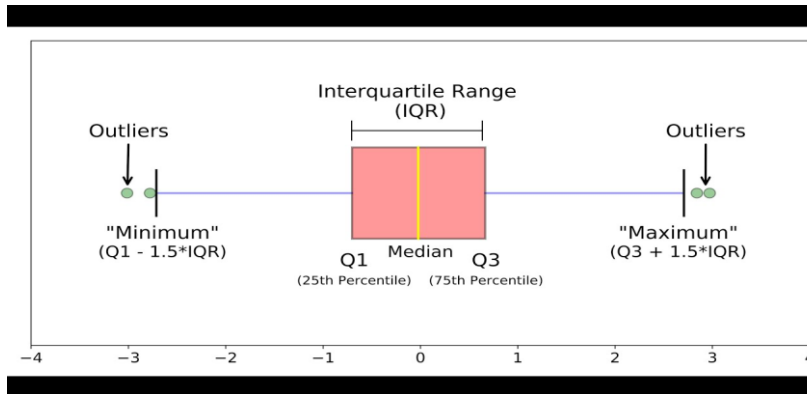
Trực quan hóa phân phối dữ liệu phân theo 2 lớp dưới dạng biểu đồ histogram có thể giúp đoán biết đặc trưng nào là quan trọng hay không có ý nghĩa trong việc phân loại. Một cách heuristic, 2 phân phối tương ứng với 2 lớp càng khác nhau, càng thể hiện rằng đặc trưng đó là quan trọng và có ảnh hưởng nhiều hơn tới kết quả phân loại.





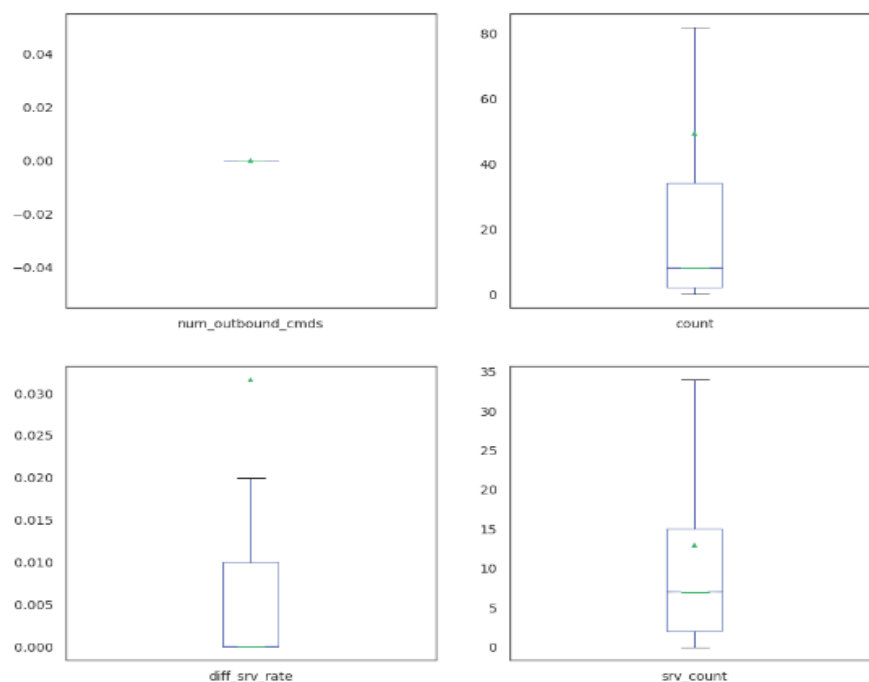
Hình 11. Trái qua phải, trên xuống dưới: *flag*, *src\_bytes*, *logged\_in*, *count*, *error\_rate*, *srv\_error\_rate*, *same\_srv\_rate*, *dst\_host\_count*, *dst\_host\_srv\_count*, *dst\_host\_same\_srv\_rate*, *dst\_host\_error\_rate*, *dst\_host\_srv\_error\_rate*

Trực quan hóa dữ liệu bằng biểu đồ boxplot:



Hình 12. Biểu đồ boxplot minh họa phân phối của dữ liệu

Box plot là biểu đồ trực quan cho ta thấy được phân bố dữ liệu, thông qua biểu diễn các giá trị trung vị, phân vị 25% (Q1) và 75% (Q3). Trực quan hóa dữ liệu dưới dạng boxplot còn giúp ta phát hiện các điểm ngoại lệ nằm ngoài khoảng 2 giá trị  $Q1 - 1.5 \times IQR$  và  $Q3 + 1.5 \times IQR$



Hình 13. Biểu đồ boxplot ứng với các đặc trưng `num_outbound_cmds`, `count`, `diff_srv_rate`, `srv_count`

### 3.2.3. Phân chia dữ liệu

Dữ liệu được chia thành 3 phần một cách ngẫu nhiên sử dụng hàm `train_test_split()` trong thư viện Scikit-learn

- Train set: dữ liệu dùng để huấn luyện mô hình học máy, chiếm ~50% tổng lượng dữ liệu (540183 mẫu)
- Validation set: dữ liệu dùng để đánh giá và chọn các siêu tham số (hyper parameters) trong quá trình huấn luyện mô hình, chiếm ~25% tổng lượng dữ liệu (266061 mẫu)
- Test set: dữ liệu dùng cho việc đánh giá khách quan mô hình, chiếm 25% tổng lượng dữ liệu (268748 mẫu)

### 3.2.4. Chuẩn hóa dữ liệu

Bước xử lý và chuẩn hóa dữ liệu thường là một bước bắt buộc trong đa số bài toán học máy. Điều khó khăn là mỗi thuật toán lại có những giả định về dữ liệu khác nhau nên nó cần những bước chuẩn hóa dữ liệu khác nhau. Nếu chúng ta làm tốt bước tiền xử lý, thuật toán học máy sẽ được cải thiện hiệu quả rõ rệt.

Nhân được gán các giá trị 0,1 tương ứng với 2 lớp tấn công và bình thường.

Các đặc trưng dạng phân loại (categorical features) cần được số hóa dưới dạng vector trước khi đưa vào mô hình học máy. Bởi vì các đặc trưng này là không có thứ tự, phương pháp hiệu quả được sử dụng là đưa từng thuộc tính này về các one-hot vector. Mỗi thuộc tính sẽ được đưa về vector có độ dài m, với m là số lượng loại (category) xuất hiện trong thuộc tính đó. Giả sử với một thuộc tính thuộc loại i, sẽ được encode thành vector trong đó vị trí thứ i là 1, còn lại tất cả các vị trí khác là 0.

Thư viện Scikit-learn hỗ trợ việc chuyển đặc trưng phân loại thành one-hot vector sử dụng `OnehotEncoder()`. Trong bộ dữ liệu KDD99, có 3 đặc trưng phân loại là `protocol_type`,



service và flag. Sau khi chuyển các đặc trưng này thành dạng one-hot vector, ta thu được vector đặc trưng có 120 chiều.

```
[ ] 1 encoder1=OneHotEncoder(handle_unknown='ignore', sparse=False)
2 x_train_1= encoder1.fit_transform(x_train[:, 1:2])
3 x_val_1=encoder1.transform(x_val[:, 1:2])
4 x_test_1=encoder1.transform(x_test[:, 1:2])
5
6 encoder2=OneHotEncoder(handle_unknown='ignore', sparse=False)
7 x_train_2= encoder2.fit_transform(x_train[:, 2:3])
8 x_val_2=encoder2.transform(x_val[:, 2:3])
9 x_test_2=encoder2.transform(x_test[:, 2:3])
10
11 encoder3=OneHotEncoder(handle_unknown='ignore', sparse=False)
12 x_train_3= encoder3.fit_transform(x_train[:, 3:4])
13 x_val_3=encoder3.transform(x_val[:, 3:4])
14 x_test_3=encoder3.transform(x_test[:, 3:4])
15
16 x_train=np.concatenate([x_train[:,0:1], x_train_1, x_train_2, x_train_3, x_train[:,4:]], axis=1)
17 x_val=np.concatenate([x_val[:,0:1], x_val_1, x_val_2, x_val_3, x_val[:,4:]], axis=1)
18 x_test=np.concatenate([x_test[:,0:1], x_test_1, x_test_2, x_test_3, x_test[:,4:]], axis=1)
19
20 print(x_train.shape, x_val.shape, x_test.shape)
```

↳ (540183, 120) (266061, 120) (268748, 120)

Hình 14. chuyển thuộc tính dạng categorical sang dạng onehot vector

Cuối cùng, ta thực hiện chuẩn hóa dữ liệu theo chuẩn Z-score sử dụng StandardScaler() trong thư viện Scikit-learn.

Trong chuẩn hóa z-score, các giá trị của thuộc tính A sẽ được chuẩn hóa dựa trên giá trị trung bình và độ lệch chuẩn của A. Một giá trị v của A sẽ được chuẩn hóa thành giá trị v' bằng công thức sau:

$$v' = \frac{v - \mu_A}{\sigma_A}$$

Trong đó:

- $\mu_A$ : giá trị trung bình của A
- $\sigma_A$ : độ lệch chuẩn của A
- v là giá trị
- v' là giá trị đã được chuẩn hóa

## IV. Huấn luyện mô hình học máy

### 4.1. Metrics sử dụng đánh giá mô hình

Đối với một bài toán phân loại lưu lượng cho IDS, như đã đề cập ở phần 1.2.2, metrics được sử dụng ở đây là:

- **Accuracy**: Độ chính xác tổng thể của mô hình, bằng số điểm dự đoán đúng chia tổng số điểm
- **Confusion matrix**: thể hiện số điểm được phân loại đúng/sai vào mỗi lớp
- **True/False Positive/Negative**
- **ROC curve** (Receiver Operating Characteristic curve) và **AUC** (Area Under the Curve): Dựa trên ROC curve, ta có thể chỉ ra rằng một mô hình có hiệu quả hay

không. Một mô hình hiệu quả khi có FPR thấp và TPR cao, tức tồn tại một điểm trên ROC curve gần với điểm có tọa độ (0, 1) trên đồ thị (góc trên bên trái). Curve càng gần thì mô hình càng hiệu quả. Area Under the Curve hay AUC là đại lượng đại diện cho diện tích nằm dưới ROC curve. Giá trị này là một số dương nhỏ hơn hoặc bằng 1. Giá trị này càng lớn thì mô hình càng tốt.

- **Precision:** tỉ lệ số điểm True Positive trong số những điểm được phân loại là Positive (TP + FP).
- **Recall:** được định nghĩa là tỉ lệ số điểm True Positive trong số những điểm thực sự là Positive (TP + FN).
- **F1-score:** là harmonic mean của precision và recall:

$$\frac{2}{F_1} = \frac{1}{\text{precision}} + \frac{1}{\text{recall}} \text{ hay } F_1 = 2 \frac{1}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}} = 2 \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Dữ liệu có kích thước lớn cùng với việc độ chính xác của mô hình tương đối cao. Các thông số như accuracy, precision, recall, f1-score đều đạt mức xấp xỉ 100% và mang ít ý nghĩa. Ta tập trung vào các metrics có nhiều ý nghĩa hơn như confusion matrix, true/false positive/negative. Đồng thời, khi xem xét kết quả, ta cũng phân tích tới các khía cạnh thống kê như các mẫu sai (miss detection samples) thuộc vào class nào, thuộc vào loại tấn công nào để đưa ra cái nhìn tổng quan về vấn đề đang xảy ra với mô hình.

## 4.2. Huấn luyện và đánh giá các mô hình

Ta sẽ thử nghiệm huấn luyện và đánh giá 6 mô hình phân loại: Logistic Regression, Multi-layer Perceptron, Decision Tree, Random Forest, Support Vector Machine, Naive Bayes. Một số mô hình được huấn luyện sử dụng phương pháp vét cạn tham số ( chỉ trong một tập dữ liệu tổ hợp của các khả năng), sử dụng GridSearchCV() trong thư viện Scikit-learn. Các tham số này là các tham số có kết quả tốt nhất trên validation set. Mô hình được huấn luyện trên dữ liệu train data, việc lựa chọn tham số dựa vào kết quả đánh giá trên tập validation, sau đó sẽ được đánh giá lại trên tập test.

Trước khi đi vào chi tiết quá trình huấn luyện và các thông số khi đánh giá mô hình, bảng tổng kết kết quả sau khi các mô hình đã được huấn luyện và đánh giá trên bộ dữ liệu test như sau:

	Logistic Regression	Multi-layer Perceptron	Decision Tree	Random Forest	SVM (linear kernel)	Naive Bayes
<b>TN</b>	202916	203174	203259	203277	202854	202699
<b>FP</b>	372	114	29	11	434	589
<b>FN</b>	680	149	61	85	517	9818

<b>TP</b>	64780	65311	65399	65375	64943	55642
<b>Accuracy</b>	1.0	1.0	1.0	1.0	1.0	0.96
<b>Precision</b>	0.99	1.0	1.0	1.0	0.99	0.99
<b>Recall</b>	0.99	1.0	1.0	1.0	0.99	0.85
<b>F1-score</b>	0.99	1.0	1.0	1.0	0.99	0.91
<b>Chi tiết (với các tham số tốt nhất)</b>		2 hidden layes với 128 neuron	'criterion': 'gini', 'max_depth': 27, 'min_samples_leaf': 5	'criterion': 'entropy', 'max_depth': 17, 'min_samples_leaf': 5, 'n_estimators': 25	'C': 0.3, 'loss': 'hinge'	Gaussian likelihood

Bảng 4. Kết quả đánh giá trên tập test của các mô hình sau khi huấn luyện

#### 4.2.1. Logistic Regression

##### 4.2.1.1. Mô hình và tham số

Mô hình Logistic Regression được xây dựng và huấn luyện dựa trên framework Keras, sử dụng L2 regularizer (còn gọi là weight decay) với tham số  $1e-3$ .

Trong quá trình huấn luyện, thêm 3 callback function sẽ được thực thi mỗi khi một epoch hoàn thành, có chức năng như sau:

- **ModelCheckpoint:** Lưu lại các tham số của mô hình (weights) vào filesystem sau mỗi epoch mà giá trị validation loss được cải thiện hơn.
- **ReduceLROnPlateau:** Giảm learning rate đi 10 lần nếu phát hiện thấy trong 5 epochs liên tiếp, giá trị validation loss không được cải thiện.
- **EarlyStopping:** Dừng sớm quá trình training, nếu phát hiện trong 8 epochs liên tiếp, validation loss không được cải thiện. Đồng thời, restore lại bộ tham số (weights) cho giá trị validation loss là nhỏ nhất.

Mô hình được training với tối đa 200 epochs. Thực nghiệm cho thấy việc training luôn dừng sớm trước 200 epochs nhờ callback EarlyStopping nêu trên.

Trong quá trình training, ngoài metric mặc định là độ chính xác (accuracy), in ra thêm các metric khác tiện cho việc theo dõi bao gồm precision, recall, f1-score. Chú ý rằng các metric này được tính toán theo batch, cộng lại chia trung bình chứ không phải được tính toán trên toàn bộ dữ liệu.

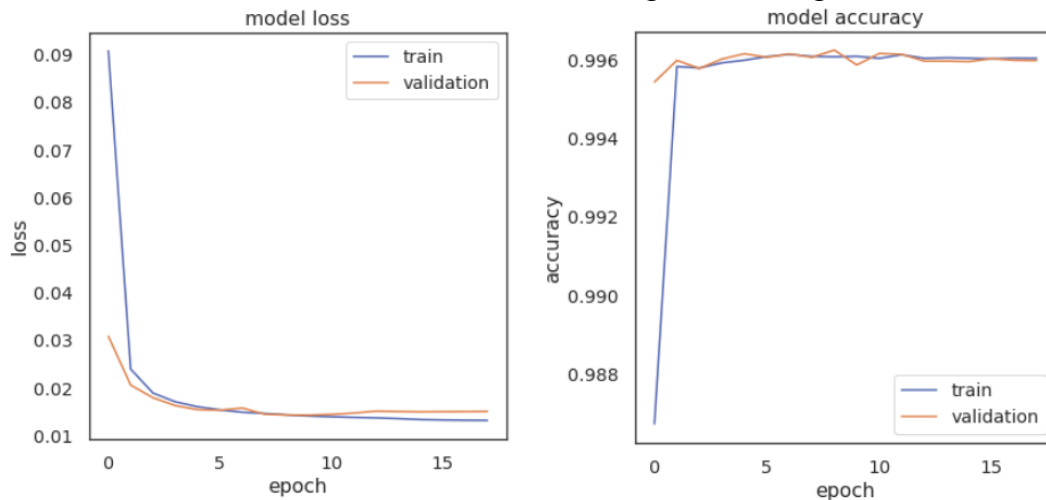
```

1 def recall_m(y_true, y_pred):
2     true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
3     possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
4     recall = true_positives / (possible_positives + K.epsilon())
5     return recall
6
7 def precision_m(y_true, y_pred):
8     true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
9     predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
10    precision = true_positives / (predicted_positives + K.epsilon())
11    return precision
12
13 def f1_m(y_true, y_pred):
14    precision = precision_m(y_true, y_pred)
15    recall = recall_m(y_true, y_pred)
16    return 2*((precision*recall)/(precision+recall+K.epsilon()))

```

Hình 15. Các metrics định nghĩa thêm để theo dõi trong quá trình training

Quá trình training sử dụng Adam optimizer với learning rate khởi tạo là 0.01. Hàm mất mát (loss function) sử dụng là binary crossentropy, batch size là 4096. Quá trình training kết thúc sau 19 epochs với 1 lần giảm learning rate. Theo dõi các thông số như loss, accuracy trên train set và validation set có thể nhận xét mô hình không bị overfitting.



Hình 16. Learning curve khi training mô hình Logistic Regression

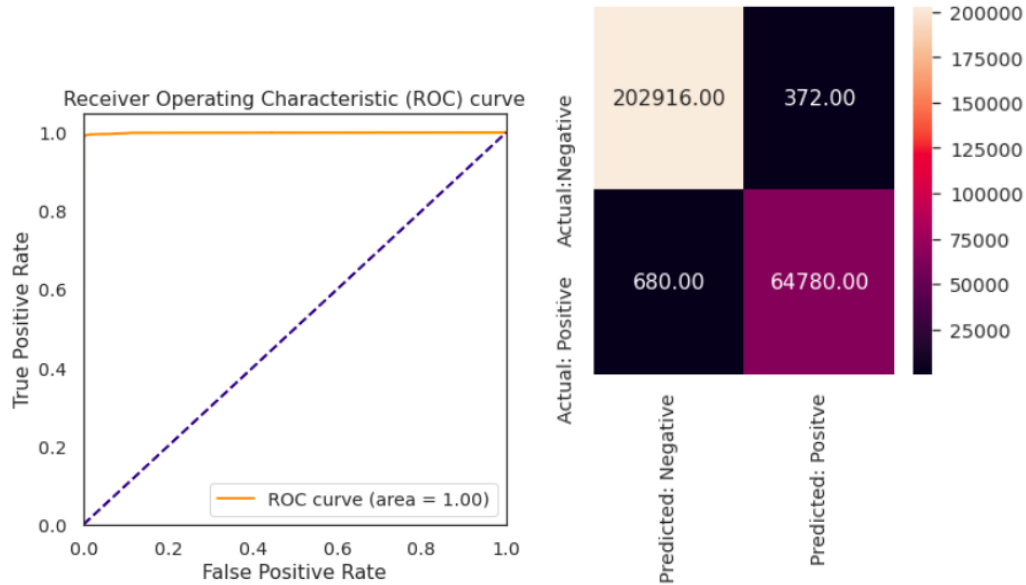
#### 4.2.1.2. Kết quả

Dữ liệu có kích thước lớn cùng với việc độ chính xác của mô hình tương đối cao. Các thông số như accuracy, precision, recall, f1-score đều đạt mức xấp xỉ 100% và mang ít ý nghĩa. Ta tập trung vào các metrics có nhiều ý nghĩa hơn như confusion matrix, true/false positive/negative.

Classification report:					
	precision	recall	f1-score	support	
0.0	1.00	1.00	1.00	203288	
1.0	0.99	0.99	0.99	65460	
accuracy			1.00	268748	
macro avg	1.00	0.99	0.99	268748	
weighted avg	1.00	1.00	1.00	268748	

Hình 17. Classification report của mô hình Logistic Regression

Confusion matrix và ROC curve:



Hình 18. ROC curve (trái) và Confusion matrix (phải) của mô hình Logistic Regression

Có 372 mẫu thuộc lớp normal bị dự đoán sai, trong 680 mẫu thuộc lớp attack bị dự đoán sai, các dạng tấn công bị dự đoán sai như sau:

'back.': 228, 'warezclient.': 195, 'ipsweep.': 40, 'nmap.': 57, 'satan.': 97, 'portsweep.': 14, 'smurf.': 8, 'neptune.': 15, 'rootkit.': 1, 'buffer\_overflow.': 10, 'phf.': 2, 'land.': 7, 'warezmaster.': 1, 'imap.': 1, 'multihop.': 1, 'spy.': 1, 'guess\_passwd.': 2

## 4.2.2. Multi-layer Perceptron (MLP)

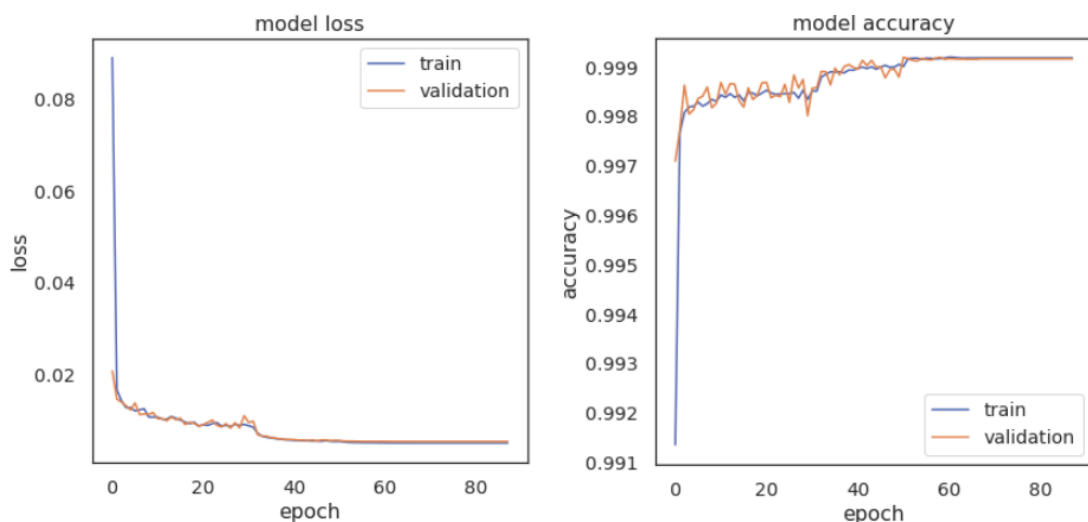
### 4.2.2.1. Mô hình và tham số

Multi-layer perceptron được sử dụng gồm 2 hidden layer với số lượng neuron đều là 128.

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 120)	0
dense_1 (Dense)	(None, 128)	15488
dense_2 (Dense)	(None, 128)	16512
dense_3 (Dense)	(None, 1)	129
Total params: 32,129		
Trainable params: 32,129		
Non-trainable params: 0		

Hình 19. Cấu trúc và số lượng tham số của MLP sử dụng

Chuẩn bị quá trình training và các tham số khác như optimizer, learning rate, batch size giống với khi training mô hình Logistic Regression nêu trên. Quá trình training kết thúc sau 88 epochs với 8 lần giảm learning rate (learning rate cuối cùng là  $9.999998606957661e-11$ ).



Hình 20. Learning curve khi training mô hình MLP

Quan sát thấy rằng mô hình không bị overfitting. Learning curve thể hiện rằng mô hình dần ổn định theo thời gian và learning rate thấp sẽ tăng tính ổn định của việc training.

#### 4.2.2.2. Kết quả

Confusion matrix:

	Dự đoán là 'normal'	Dự đoán là 'attack'
Thực tế là 'normal'	203174	114
Thực tế là 'attack'	149	203174

Bảng 5. Confusion matrix của mô hình MLP

Có 149 mẫu thuộc lớp normal bị dự đoán sai, trong 114 mẫu thuộc lớp attack bị dự đoán sai, các dạng tấn công bị dự đoán sai như sau:

'ipsweep.': 18, 'portsweep.': 16, 'warezclient.': 46, 'satan.': 26, 'nmap.': 9, 'back.': 4, 'smurf.': 10, 'imap.': 2, 'rootkit.': 1, 'buffer\_overflow.': 5, 'land.': 7, 'guess\_passwd.': 2, 'warezmaster.': 1, 'multihop.': 1, 'spy.': 1

MLP là mô hình phức tạp hơn so với Logistic Regression, và kết quả đánh giá cũng tốt hơn rất nhiều trên test set.

### 4.2.3. Cây quyết định (Decision Tree)

#### 4.2.3.1. Mô hình và tham số

Mô hình cây quyết định được xây dựng dựa trên DecisionTreeClassifier, triển khai thuật toán CART trong thư viện Scikit-learn. Việc chọn ra mô hình tốt nhất dựa trên việc vét cạn tham số kết hợp cross validation, sử dụng GridSearchCV. Các tham số quan tâm và các giá trị vét cạn như sau:

```
{
'max_depth': [5, 11, 13, 17, 21, 27, 31],
'criterion': ('gini', 'entropy'),
'min_samples_leaf': (5, 10)
}
```

Trong đó:

- max\_depth: độ sâu tối đa của cây
- criterion: cơ chế đánh giá độ tinh khiết của mỗi khả năng phân nhánh
- min\_samples\_leaf: số lượng tối thiểu sample của lá

Sau khi vét cạn 28 bộ tham số, nếu sử dụng cơ chế cross validation sử dụng tập dữ liệu validation như thông thường, bộ tham số tốt nhất là {'criterion': 'gini', 'max\_depth': 27, 'min\_samples\_leaf': 5}. Nếu sử dụng cơ chế k-fold cross validation (k được chọn bằng 3), bộ tham số tốt nhất là {'criterion': 'entropy', 'max\_depth': 21, 'min\_samples\_leaf': 5}. Kết quả đánh giá mô hình với 2 cơ chế cross validation này là khá tương đồng nhau.

#### 4.2.3.2. Kết quả

Confusion matrix:

	Dự đoán là 'normal'	Dự đoán là 'attack'
Thực tế là 'normal'	203259	29
Thực tế là 'attack'	61	65399

Bảng 6. Confusion matrix của mô hình Decision Tree

Chỉ có 29 mẫu thuộc lớp normal bị dự đoán sai, trong 61 mẫu thuộc lớp attack bị dự đoán sai, các dạng tấn công bị dự đoán sai như sau:

{'warezclient.': 16, 'portsweep.': 5, 'satan.': 10, 'rootkit.': 1, 'buffer\_overflow.': 7, 'pod.': 5, 'phf.': 2, 'land.': 3, 'guess\_passwd.': 3, 'warezmaster.': 1, 'ipsweep.': 2, 'neptune.': 1, 'spy.': 1, 'imap.': 2, 'ftp\_write.': 1, 'nmap.': 1}

Kết quả của mô hình này vượt trội so với các mô hình khác khi số lượng FN chỉ là 61 và FP chỉ là 29. Cây quyết định là mô hình phù hợp cho dữ liệu có thể biểu diễn theo các rẽ nhánh logic if-else.

## 4.2.4. Random Forest

### 4.2.4.1. Mô hình và tham số

Tương tự mô hình cây quyết định, mô hình cũng được xây dựng và vét cạn tham số sử dụng GridSearchCV dựa trên thư viện Scikit-learn. Các tham số vét cạn như sau:

```
{
'n_estimators':[9,25,55],
'max_depth':[5, 11, 13, 17, 21, 27, 31],
'criterion': ('gini', 'entropy'),
'min_samples_leaf': (10,)
}
```

Khác với cây quyết định, mô hình random forest dựa trên cơ chế ensemble learning, kết hợp kết quả dự đoán của nhiều cây quyết định khác nhau. Tham số 'n\_estimators' nêu trên là số cây quyết định sử dụng, các tham số khác là tham số của cây quyết định, có ý nghĩa giống với đã nêu trong phần 4.2.3.1.

### 4.2.4.2. Kết quả

Confusion matrix:

	Dự đoán là 'normal'	Dự đoán là 'attack'
Thực tế là 'normal'	203277	11
Thực tế là 'attack'	85	65375

Bảng 7. Confusion matrix của mô hình Random Forest

Chỉ có 11 mẫu thuộc lớp normal bị dự đoán sai, trong 85 mẫu thuộc lớp attack bị dự đoán sai, các dạng tấn công bị dự đoán sai như sau:

'warezclient.': 37, 'satan.': 15, 'back.': 1, 'nmap.': 3, 'portsweep.': 6, 'normal.': 11, 'imap.': 2, 'rootkit.': 1, 'buffer\_overflow.': 9, 'phf.': 2, 'guess\_passwd.': 3, 'warezmaster.': 1, 'spy.': 1, 'ipsweep.': 2, 'land.': 1, 'ftp\_write.': 1

Là mô hình ensemble learning từ kết quả của nhiều cây quyết định khác, mô hình này cũng cho kết quả rất tốt khi so sánh với các mô hình khác.

## 4.2.5. Support Vector Machine (SVM)

### 4.2.5.1. Mô hình và tham số

Do đặc thù của thuật toán SVM, việc huấn luyện mô hình này với kernel khác kernel tuyến tính (linear kernel) trên bộ dữ liệu với kích thước lớn trên 10000 mẫu là rất tốn thời gian và chi phí, thậm chí là không thể (theo tài liệu Scikit-learn). Do đó, ta chỉ xem xét tới mô hình SVM với kernel tuyến tính sử dụng sklearn.svm. LinearSVC.

Tương tự đối với việc huấn luyện mô hình sử dụng Scikit-learn khác, ta cũng sẽ thực hiện vét cạn tham số sử dụng GridSearchCV. Tổ hợp các tham số được sinh bởi:

```
{
'C':[0.03, 0.1, 0.3, 1, 3, 10],
'loss': ['hinge', 'squared_hinge']
}
```



Sau khi quá trình training hoàn tất, bộ tham số tốt nhất thu được là  $C=0.3$  và loss là hinge loss.

#### 4.2.5.2. Kết quả

Mô hình phân loại kém chính xác hơn so với các mô hình khác, khi precision, recall và f1-score với lớp tấn công (positive) chỉ là 0.99.

Confusion matrix:

	Dự đoán là 'normal'	Dự đoán là 'attack'
Thực tế là 'normal'	202854	434
Thực tế là 'attack'	517	64943

Bảng 8. Confusion matrix của mô hình SVM

Có tới 434 mẫu thuộc lớp normal bị dự đoán sai và trong 517 mẫu thuộc lớp attack bị dự đoán sai, các dạng tấn công bị dự đoán sai như sau:

'back.': 213, 'warezclient.': 139, 'portsweep.': 6, 'satan.': 95, 'smurf.': 9, 'neptune.': 22, 'nmap.': 11, 'rootkit.': 1, 'ipsweep.': 10, 'buffer\_overflow.': 3, 'phf.': 2, 'warezmaster.': 1, 'imap.': 1, 'multihop.': 1, 'spy.': 1, 'guess\_passwd.': 2

#### 4.2.6. Bộ phân loại Naive Bayes

##### 4.2.6.1. Mô hình và tham số

Các mô hình Naive Bayes là các mô hình xây dựng dựa trên lý thuyết thống kê, với giả thiết quan trọng nhất là tính độc lập giữa các thuộc tính của mẫu. Tùy vào giả thiết của phân phối coi như đã biết của các thuộc tính (likelihood), có nhiều mô hình Naive Bayes và chúng cho kết quả phù hợp với các loại dữ liệu khác nhau như Gaussian Naive Bayes, Bernouli Naive Bayes, Categorical Naive Bayes,...

Dữ liệu với các trường bao gồm cả số nguyên và số thực trong khoảng, mô hình Gaussian Naive Bayes với giả thiết các đặc trưng có phân phối Gaussian là mô hình phù hợp. Mô hình được sử dụng là bộ phân loại GaussianNB() hỗ trợ bởi thư viện Scikit-learn.

##### 4.2.6.2. Kết quả

Mô hình Naive Bayes phù hợp cho nhiều bài toán với số lượng đặc trưng ít và dữ liệu nhiều (dữ liệu càng nhiều thì tính chính xác khi thống kê càng cao). Là mô hình đơn giản, Naive Bayes tỏ ra không phù hợp với bài toán và cho kết quả tệ hơn so với các mô hình khác.

Classification report:

	precision	recall	f1-score	support
0.0	0.95	1.00	0.97	203288
1.0	0.99	0.85	0.91	65460
accuracy			0.96	268748
macro avg	0.97	0.92	0.94	268748
weighted avg	0.96	0.96	0.96	268748

Hình 21. Classification report của mô hình Gaussian Naive Bayes

Recall của lớp “tấn công” là rất thấp, thể hiện rằng mô hình có xu hướng dự đoán tất cả các mẫu là bình thường.

Confusion matrix:

	Dự đoán là ‘normal’	Dự đoán là ‘attack’
Thực tế là ‘normal’	202699	589
Thực tế là ‘attack’	9818	55642

*Bảng 9. Confusion matrix của mô hình Gaussian Naive Bayes*

Có 589 mẫu thuộc lớp normal bị dự đoán sai. Có tới 9818/65460 mẫu thuộc lớp “tấn công” bị dự đoán nhầm thành bình thường. Cụ thể hơn, số mẫu bị dự đoán sai tương ứng với từng loại tấn công như sau:

'back.': 213, 'warezclient.': 139, 'portsweep.': 6, 'satan.': 95, 'smurf.': 9, 'neptune.': 22, 'nmap.': 11, 'rootkit.': 1, 'ipsweep.': 10, 'buffer\_overflow.': 3, 'phf.': 2, 'warezmaster.': 1, 'imap.': 1, 'multihop.': 1, 'spy.': 1, 'guess\_passwd.': 2

## **IV. Xây dựng mô hình thu thập dữ liệu mạng phục vụ huấn luyện mô hình**

### **4.1. SDN**

#### **4.1.1. Tổng quan về SDN**

Mạng định nghĩa bằng phần mềm (SDN) là một công nghệ mới giúp nâng cao hiệu quả và giảm chi phí cho việc vận hành và quản trị hệ thống mạng. Trong một kiến trúc mạng truyền thống data plane và control plane đều cùng nằm trên một thiết bị vật lý (chế độ tự trị) và mỗi thiết bị độc lập với nhau, các chính sách chuyển tiếp lưu lượng nằm riêng trên mỗi thiết bị, vì vậy không có khả năng theo dõi và quản lý thông tin toàn bộ mạng. Các chính sách chuyển tiếp và phân tích quản lý không được tối ưu, thậm chí không thể thực hiện được.

Khi số lượng thiết bị càng nhiều, thì càng gây nên sự phức tạp trong mạng và làm khó khăn cho người quản trị mạng trong quá trình vận hành và điều khiển. Cùng với các thay đổi về mô hình lưu thông, sự gia tăng của các dịch vụ đám mây và nhu cầu phát triển của các nhà khai thác băng thông dịch vụ đưa ra một yêu cầu cần phải phát triển một giải pháp mới. Trước sự cần thiết của một kiến trúc mạng mới, các nhà nghiên cứu đã nghiên cứu và phát triển một kiến trúc mạng mà ở đó nhiệm vụ điều khiển mạng được xử lý bởi các bộ điều khiển và các bộ điều khiển đó có thể thao tác tới phần cứng, bộ nhớ và các chức năng của các thiết bị định tuyến (router), chuyển mạch (switch) để đạt được mục đích của người sử dụng. Do đó, mạng trở nên linh hoạt hơn, hiệu suất sử dụng cao hơn và dễ quản lý hơn.

Mạng định nghĩa mềm hay Software Defined Network (SDN) là một kiểu kiến trúc mạng mới, năng động, dễ quản lý, chi phí hiệu quả, dễ thích nghi và rất phù hợp với nhu cầu mạng ngày càng tăng hiện nay. Kiến trúc này phân tách phần điều khiển mạng (Control Plane) và chức năng vận chuyển dữ liệu (Forwarding Plane hay Data Plane), điều này cho phép việc điều

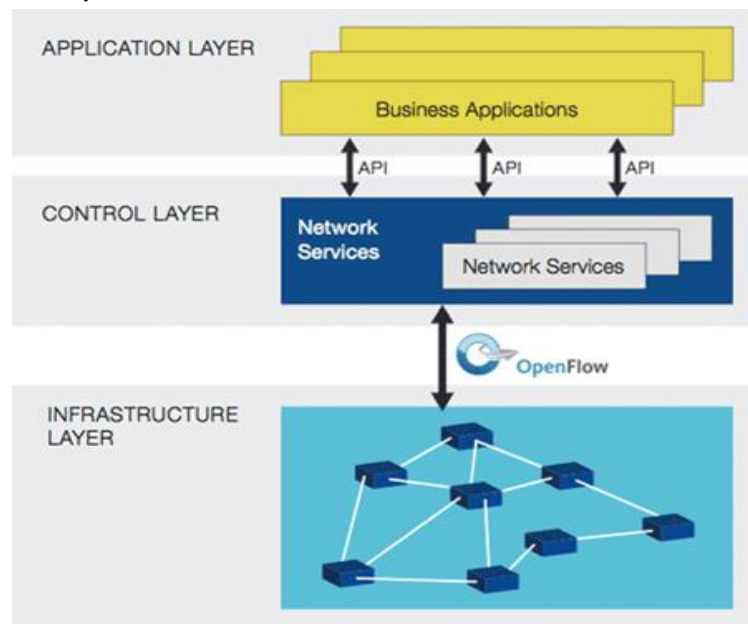
khiến mạng trở nên có thể lập trình được dễ dàng và cơ sở hạ tầng mạng độc lập với các ứng dụng và dịch vụ mạng.

Sự khác biệt cơ bản giữa mạng truyền thống và mạng SDN là:

- Phần điều khiển và phần vận chuyển dữ liệu trên mạng truyền thống đều được tích hợp trong thiết bị mạng trong khi trong mạng SDN, phần điều khiển được tách riêng khỏi thiết bị mạng và được chuyển đến một thiết bị được gọi là bộ điều khiển SDN.
- Phần thu thập và xử lý các thông tin: Đối với mạng truyền thống, phần này được thực hiện ở tất cả các phần tử trong mạng còn trong mạng SDN, phần này được tập trung xử lý ở bộ điều khiển SDN.
- Mạng truyền thống không thể được lập trình bởi các ứng dụng. Các thiết bị mạng phải được cấu hình một cách riêng lẻ và thủ công. Trong khi đối với mạng SDN, mạng có thể lập trình bởi các ứng dụng, bộ điều khiển SDN có thể tương tác đến tất cả các thiết bị trong mạng.

#### 4.1.2. Kiến trúc của SDN

Về cơ bản, SDN được chia làm ba lớp: lớp ứng dụng (Application Layer), lớp điều khiển (Control Layer) và lớp thiết bị hạ tầng (Infrastructure Layer). Các lớp sẽ liên kết với nhau thông qua giao thức hoặc các API.



Hình 22. 3 lớp Application Layer, Control Layer và Infrastructure Layer của SDN

- Lớp ứng dụng (Application Layer): là các ứng dụng được triển khai trên mạng, kết nối tới lớp điều khiển thông qua các API, cung cấp khả năng cho phép lớp ứng dụng lập trình lại (cấu hình lại) mạng (điều chỉnh các tham số trễ, băng thông, định tuyến, ...) thông qua lớp điều khiển lập trình giúp cho hệ thống mạng để tối ưu hoạt động theo một yêu cầu nhất định.
- Lớp điều khiển (Control layer): là nơi tập trung các controller thực hiện việc điều khiển cấu hình mạng theo các yêu cầu từ lớp ứng dụng và khả năng của mạng. Các controller này có thể là các phần mềm được lập trình.

Một Controller (bộ điều khiển) là một ứng dụng quản lý kiểm soát luồng lưu lượng trong môi trường mạng. Để điều khiển lớp cơ sở hạ tầng, lớp điều khiển sử dụng các cơ chế như OpenFlow, ONOS, ForCES, PCEP, NETCONF, SNMP hoặc thông qua các cơ chế riêng biệt. Hầu hết các SDN controller hiện nay dựa trên giao thức OpenFlow.

SDN controller hoạt động như một loại hệ điều hành (OS) cho mạng. Tất cả thông tin liên lạc giữa các ứng dụng và các thiết bị phải đi qua controller. Controller sử dụng giao thức OpenFlow để cấu hình các thiết bị mạng và chọn đường đi tốt nhất cho các lưu lượng ứng dụng. Cùng với chức năng chính, nó có thể tiếp tục được mở rộng để thực hiện thêm các nhiệm vụ quan trọng như định tuyến và truy cập mạng. Tóm tắt lại, lớp điều khiển có vai trò:

- Cung cấp API để có thể xây dựng các ứng dụng cho hệ thống mạng.
- Thu nhận thông tin từ hệ thống mạng vật lý, điều khiển hệ thống mạng vật lý.

## **4.1.2. Mininet**

### **4.1.2.1. Tổng quan về Mininet**

Mininet là một trình giả lập mạng (network emulator), hoặc chính xác hơn là một hệ thống điều phối mô phỏng mạng (network emulation orchestration system). Nó cho phép giả lập và chạy các host, switch và router liên kết với nhau, trên một nhân Linux. Mininet sử dụng công nghệ ảo hóa (virtualization) để làm cho hệ thống xây dựng từ nó giống hệt như một mạng hoàn chỉnh, chạy cùng một nhân, filesystem,... Một host tạo từ Mininet hoạt động giống như một máy tính thực sự, có thể ssh vào nó (nếu đã chạy ssh daemon) và chạy các chương trình tùy ý (bao gồm mọi thứ được cài đặt trên hệ thống Linux bên dưới). Mininet cũng cho phép gửi các gói tin thông qua Ethernet giống như hệ thống card mạng thật, ảo hóa kết nối và đường truyền với băng thông và độ trễ được cấu hình. Nói tóm lại, các máy ảo tạo từ mininet, switch, kết nối mạng và bộ điều khiển của Mininet được tạo bằng phần mềm chứ không phải phần cứng, tuy nhiên các hành vi và tương tác giữa các thành phần với nhau lại tương tự như các phần cứng rời rạc.

### **4.1.2.2. Ưu và nhược điểm**

Một số ưu điểm của Mininet:

- Mã nguồn mở, cộng đồng đông đảo, có tốc độ nhanh
- Khả năng tạo các topology tùy ý, tùy chỉnh các thông số mạng như bộ xử lý, băng thông, độ trễ,..
- Có thể chạy nhiều loại phần mềm một cách thực tế (gần như tất cả các phần mềm mà máy vật lý thật bên dưới có thể chạy)
- Có thể đóng gói và sử dụng ở môi trường máy thật khác một cách dễ dàng.

Một số hạn chế:

- Vẫn sử dụng tài nguyên của máy thật bên dưới, vì vậy có thể hạn chế về tài nguyên
- Máy ảo xây dựng từ Mininet chỉ sử dụng nhân Linux, vì vậy các hệ thống Window, BSD,.. muốn sử dụng phải chạy máy ảo riêng và kết nối nó với Mininet.
- Tách biệt với mạng LAN và kết nối ra ngoài internet, tuy nhiên vẫn có thể cấu hình và sử dụng NAT để kết nối với mạng LAN thực.

- Sử dụng chung filesystem và không gian PID, vì vậy cần thận trọng khi chạy các system service.
- Không giống một hệ thống giả lập, Mininet hoàn toàn dựa trên thực tế, do đó khó có thể giả lập các hệ thống mạng có tốc độ cao hơn thực tế (ví dụ, kết nối 100Gbps).

#### 4.1.2.3. Cài đặt

Mininet có thể dễ dàng tải về và cài đặt theo hướng dẫn trên trang chủ

<http://mininet.org/download/>

Cụ thể, có 3 phương pháp:

- Cài đặt Mininet VM: dễ dàng, có thể cài đặt trên mọi hệ điều hành và môi trường, sử dụng các phần mềm máy ảo như VMware, VirtualBox,... Mininet VM là máy ảo Ubuntu linux 14.04, với môi trường mà một số công cụ đã được cài đặt sẵn.
- Cài đặt Mininet từ source code: Clone source code từ github và build từ đầu, cụ thể trên trang hướng dẫn
- Cài đặt từ kho packages ứng với bản phân phối linux tương ứng. Cách thực hiện này đơn giản, tuy nhiên có thể sẽ không có được phiên bản mới nhất của mininet. Trong quá trình cài đặt, ta sẽ sử dụng phương pháp cài đặt này.

**\$ sudo apt-get install mininet**

sẽ cài đặt mininet và một số packages cần thiết đi kèm. Nếu đã từng cài đặt mininet, hoặc gặp lỗi trong quá trình cài đặt, tham khảo chi tiết tại trang chủ.

Sau khi quá trình cài đặt hoàn tất, kiểm tra lại phiên bản và test thử để chắc chắn mininet có thể hoạt động:

**\$ mn --version**

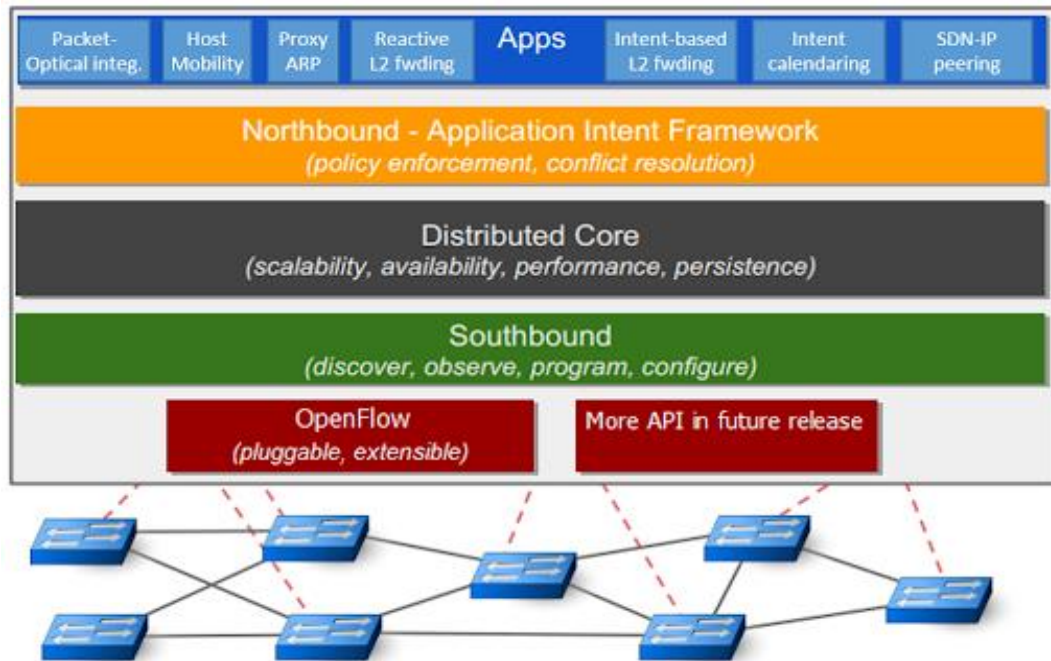
**\$ sudo mn --test pingall**

#### 4.1.3. ONOS

##### 4.1.3.1. Tổng quan

ONOS (Open Network Operating System) là một bộ điều khiển SDN mã nguồn mở để xây dựng các giải pháp SDN/NFV thế hệ tiếp theo.

ONOS được thiết kế để đáp ứng nhu cầu xây dựng các giải pháp cấp nhà mạng, cung cấp tính linh hoạt để tạo và triển khai các dịch vụ mạng động (dynamic network services) mới với giao diện lập trình được đơn giản hóa tới mức tối thiểu. ONOS hỗ trợ cấu hình và điều khiển theo thời gian thực, loại bỏ nhu cầu chạy các giao thức điều khiển định tuyến và chuyển mạch bên trong cấu trúc mạng. Bằng cách chuyển toàn bộ điều khiển vào các bộ điều khiển ONOS, người quản trị có thể dễ dàng tạo các ứng dụng mạng mới mà không cần phải thay đổi hệ thống dataplane.



Hình 23. Mô tả các thành phần của ONOS

Nền tảng ONOS bao gồm:

- Một nền tảng và một tập các ứng dụng, hoạt động như một bộ điều khiển SDN có tính chất phân tán và được module hóa.
- Đơn giản trong quản trị, cấu hình và triển khai phần mềm, phần cứng và dịch vụ mới.
- Một kiến trúc khả mở, cung cấp khả năng phục hồi và mở rộng cần thiết.

#### 4.1.3.2. Cài đặt

Chi tiết cài đặt có thể tìm thấy trên trang chủ

<https://wiki.onosproject.org/display/ONOS/Downloads>

Chi tiết cài đặt trên một máy tính:

<https://wiki.onosproject.org/display/ONOS/Installing+on+a+single+machine>

Trong quá trình cài đặt, ta cài đặt ONOS trên máy 1 máy trạm Ubuntu 18.04 duy nhất, theo hướng dẫn đã nêu trên. Tóm tắt các bước:

- ONOS phiên bản mới nhất hiện tại (Toucan-2.3.0) yêu cầu JDK 11 (Java Development Kit), trước hết kiểm tra lại đã cài đặt JDK11 hay chưa, nếu chưa cần cài đặt JDK11 và chọn JDK11 là môi trường JDK mặc định, cài đặt biến môi trường JAVA\_HOME trỏ tới đường dẫn của JDK 11 (tham khảo trang chủ Oracle JDK hoặc <https://www.javahelps.com/2017/09/install-oracle-jdk-9-on-linux.html> )
- Cài đặt ONOS vào thư mục /opt/onos:  

```
$ cd /opt
$ sudo wget -c http://downloads.onosproject.org/release/onos-\$ONOS\_VERSION.tar.gz
$ sudo tar xzf onos-$ONOS_VERSION.tar.gz
$ sudo mv onos-$ONOS_VERSION onos
```
- Khởi động và kiểm tra hoạt động của ONOS:  

```
$ /opt/onos/bin/onos-service start
```

ONOS cũng cung cấp CLI để thao tác và giao tiếp với onos service:

**\$ /opt/onos/bin/onos**

## 4.2. Xây dựng topo SDN giả lập

Muốn thu thập dữ liệu, ta cần có một hạ tầng mạng giả lập để có thể thu thập lưu lượng (raw TCP dump) giữa các node trong mạng. Sau đây, ta tập trung xây dựng một hệ thống mạng LAN giả lập.

Có rất nhiều cách để có thể xây dựng một mạng LAN giả lập. GNS3 và các công cụ giả lập máy ảo như VMware, VirtualBox có thể giúp xây dựng mô hình mạng, tuy nhiên chi phí tốn kém về mặt tài nguyên phần cứng, hạn chế trong cài đặt ứng dụng và vô cùng phức tạp trong công việc cài đặt. Ngược lại, Mininet là công cụ vô cùng phù hợp cho việc giả lập mạng ảo, chạy ứng dụng trên các node mạng ảo, sử dụng mô hình SDN.

Vì vậy, để tiến hành mô phỏng tấn công và thu thập dữ liệu lưu lượng trong mạng, ta xây dựng một mạng SDN giả lập sử dụng Mininet và controller sử dụng ONOS. Với ưu điểm là khả năng tùy chỉnh cao, thiết lập nhanh chóng, không tốn nhiều tài nguyên, mạng ảo SDN xây dựng từ Mininet và ONOS là phù hợp cho nhu cầu thử nghiệm và thu thập dữ liệu.

Toàn bộ công đoạn xây dựng topo giả lập SDN và mô phỏng tấn công, thu thập dữ liệu sau đây, đều giả thiết rằng:

- Mininet đã được cài đặt trên duy nhất máy trạm thử nghiệm
- ONOS đã được cài đặt thành công tại thư mục /opt/onos

### 4.2.1. Khởi động ONOS controller

Để khởi động ONOS service, thực hiện lệnh

**\$ sudo /opt/onos/bin/onos-service clean**

Từ trình duyệt, truy cập vào địa chỉ <http://localhost:8181/onos/ui/index.html>, ta sẽ thấy giao diện đăng nhập của ONOS.



Hình 24. Giao diện đăng nhập ONOS

Username mặc định là “onos”, password mặc định là “rocks”.

Sau khi đăng nhập thành công, giao diện quản lý của ONOS xuất hiện với các Tab quan trọng như Applications, Topology,.. cho phép quản lý và kiểm tra thông tin của các node trong mạng SDN quản lý bởi ONOS controller.

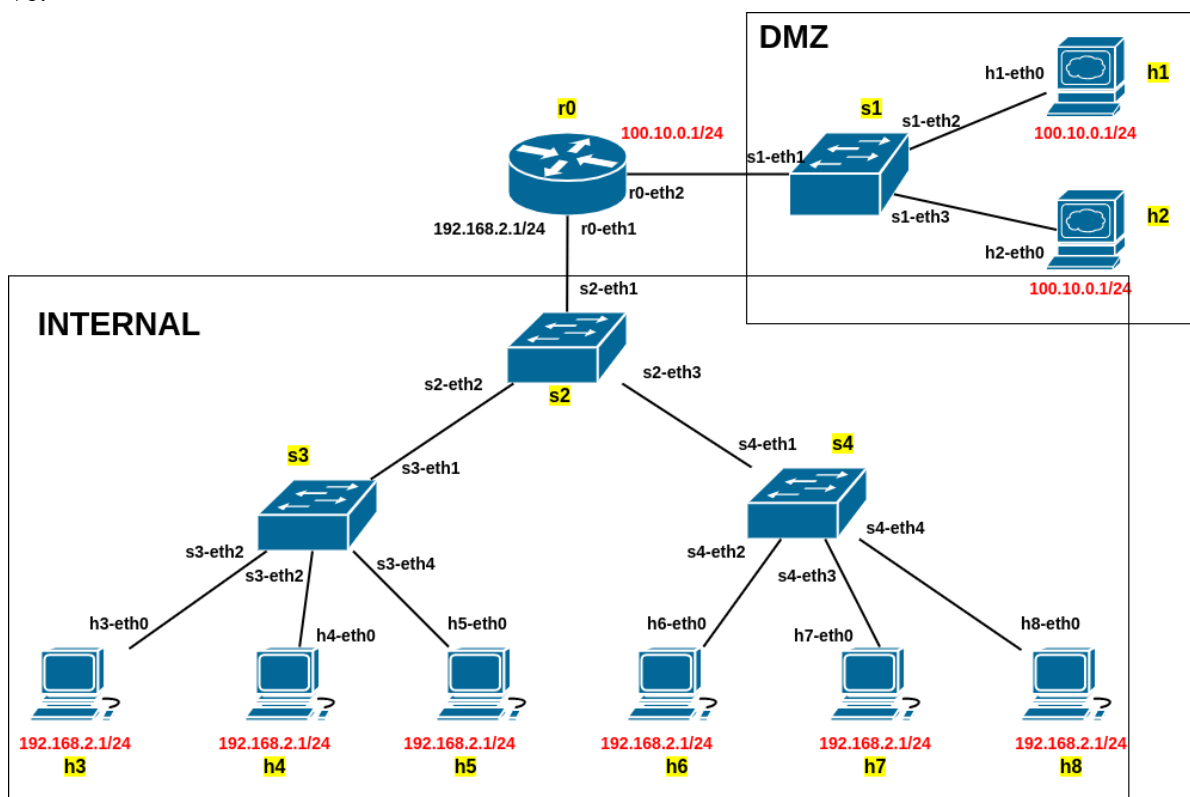
Kích hoạt 2 ứng dụng OpenFlow Provider Suite (kích hoạt OpenFlow, cho phép Mininet có thể kết nối tới) và Reactive Forwarding (định tuyến tự động) như sau:

▼	Title	App ID	Version	Category	Origin
✓	 Default Drivers	org.onosproject.drivers	2.4.0	Drivers	ONOS Community
✓	 Host Location Provider	org.onosproject.hostprovider	2.4.0	Provider	ONOS Community
✓	 LLDP Link Provider	org.onosproject.lldpprovider	2.4.0	Provider	ONOS Community
✓	 ONOS GUI2	org.onosproject.gui2	2.4.0	Graphical User Interface	ONOS Community
✓	 OpenFlow Base Provider	org.onosproject.openflow-base	2.4.0	Provider	ONOS Community
✓	 OpenFlow Provider Suite	org.onosproject.openflow	2.4.0	Provider	ONOS Community
✓	 Optical Network Model	org.onosproject.optical-model	2.4.0	Optical	ONOS Community
✓	 Reactive Forwarding	org.onosproject.fwd	2.4.0	Traffic Engineering	ONOS Community

Hình 25. Giao diện các application trong tab Application

#### 4.2.2. Xây dựng topology sử dụng Mininet API

Ta sẽ xây dựng một mạng đơn giản gồm 1 router, 4 switch và 8 host, chi tiết mô tả như hình vẽ:



Hình 26. Topo mạng mà ta sẽ sử dụng



- Router r0: 2 interface r0-eth1 có địa chỉ 192.168.2.1/24 nối với phân vùng mạng Internal 192.168.2.0/24 và r0-eth2 có địa chỉ 100.10.0.1/24 nối với phân vùng DMZ 100.10.0.0/24
- 4 switch s1, s2, s3, s4
- 6 host h3→ h8 thuộc phân vùng Internal, kết nối với nhau qua hệ thống 3 switch s2, s3, s4 có topo hình cây. Toàn bộ các host này nhận r0 là default gateway (kết nối với interface r0-eth1 có địa chỉ 192.168.2.1/24).
- 2 host h1 và h2 thuộc phân vùng DMZ, kết nối với nhau qua hệ switch s1. Chúng nhận r0 là default gateway (kết nối với interface r0-eth2 có địa chỉ 100.10.0.1/24)

Mininet cung cấp python API, sử dụng để tạo nên các topology tùy biến, các script sử dụng cho việc kiểm thử (testing).. Ta sẽ viết một python script có tên **mytopo.py** để tạo và khởi động một mạng ảo với topology đã nêu ở trên. Nội dung script như sau:

```
#!/usr/bin/env python
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.node import Node, Controller, OVSKernelSwitch, RemoteController
from mininet.log import setLogLevel, info
from mininet.cli import CLI
from mininet.link import TCLink

class LinuxRouter( Node ):
    "A Node with IP forwarding enabled, act like a router."
    def config( self, **params ):
        super( LinuxRouter, self ).config( **params )
        # Enable forwarding on the router
        self.cmd( 'sysctl net.ipv4.ip_forward=1' )

    def terminate( self ):
        self.cmd( 'sysctl net.ipv4.ip_forward=0' )
        super( LinuxRouter, self ).terminate()

class NetworkTopo( Topo ):
    "A simple simulated office network."
    def build( self, n1, n2 ):
        defaultIP = '192.168.2.1/24' # IP address for r0-eth1
        router = self.addNode( 'r0', cls=LinuxRouter, ip= defaultIP )
        s1, s2, s3, s4 = [ self.addSwitch( s ) for s in ( 's1', 's2', 's3', 's4' ) ]
        self.addLink( s2, router, cls=TCLink, bw=20, delay='1ms', intfName2='r0-eth1',
params2={ 'ip' : defaultIP } )
        self.addLink( s1, router, intfName2='r0-eth2', params2={ 'ip' : '100.10.0.1/24' } )
        self.addLink( s3, s2 )
        self.addLink( s4, s2 )

        # DMZ zone contain 2 hosts
        h1 = self.addHost( 'h1', cpu=0.10, ip='100.10.0.2/24', defaultRoute='via 100.10.0.1' )
        h2 = self.addHost( 'h2', cpu=0.10, ip='100.10.0.3/24', defaultRoute='via 100.10.0.1' )
        self.addLink( h1, s1, cls=TCLink, bw=20, delay='1ms' )
        self.addLink( h2, s1, cls=TCLink, bw=20, delay='1ms' )

        # Internal zone
        # n1 host connect to switch s3
        for i in range( n1 ):
            h = self.addHost( 'h%d' % ( i + 3 ), cpu=0.10, ip='192.168.2.%d/24' % ( i + 3 ),
defaultRoute='via 192.168.2.1' )
            self.addLink( h, s3, cls=TCLink, bw=20, delay='1ms' )
```

```

        host=self.addHost('h%s'%(i+3), cpu=0.2/n1, ip='192.168.2.%s/24'%(i+3),
defaultRoute='via 192.168.2.1')
        self.addLink(host, s3, cls=TCLink, bw=15, delay='2ms' )
        # n2 host connect to switch s4
        for i in range(n2):
            host= self.addHost('h%s'%(i+ n1 + 3), cpu=0.2/n2, ip='192.168.2.%s/24'%(i+ n1 + 3),
defaultRoute='via 192.168.2.1')
            self.addLink(host, s4, cls=TCLink, bw=15, delay='2ms')

def run():
    topo = NetworkTopo(3,3)
    net = Mininet(topo=topo, controller=None)
    c0 = net.addController('c0', controller=RemoteController, ip="127.0.0.1", port=6653)
    net.start()
    info( '*** Routing Table on Router:\n' )
    info( net[ 'r0' ].cmd( 'route' ) )
    info('*** Starting some service')
    # ftp service
    net['h1'].cmd('sudo twistd -n ftp -p 21 -r /home/dangnh/ids &> /dev/null &')
    # http web service
    net['h1'].cmd('python -m SimpleHTTPServer 80 &> /dev/null & ')
    # ssh service
    net['h1'].cmd('/usr/sbin/sshd -D &')

    net['h2'].cmd('python -m SimpleHTTPServer 80 &> /dev/null & ')
    net['h2'].cmd('/usr/sbin/sshd -D &')

    info('*** Testing connection: \n')
    info(net.pingAll())

    info( '*** Running services on host h1:\n' )
    info(net['h1'].cmd('netstat -tulpn | grep LISTEN'))
    info( '*** Running services on host h2:\n' )
    info(net['h2'].cmd('netstat -tulpn | grep LISTEN'))

    CLI( net )
    net.stop()

if __name__ == '__main__':
    setLogLevel( 'info' )
    run()

topos = { 'tp': ( lambda n1, n2 : NetworkTopo(n1, n2) ) }

```

Ta sẽ phân tích chi tiết vào nội dung của script trên:

- Class LinuxRouter đại diện cho Router trong topo mạng. Mininet không cung cấp API liên quan tới đối tượng Router, do đó ta cần tạo một router tùy biến bằng việc sử dụng một host bình thường với nhân linux và bật chế độ chuyển tiếp gói tin ip4 forwarding.
- Lớp NetworkTopo kế thừa lớp Topo. Ta cần phải implement phương thức build(self, \*, \*\*). Bên trong nội dung phương thức build() này, ta cần gọi tới các API mà mininet cung cấp để tạo ra topo mạng như mong muốn. Trong script nêu trên, 2 tham số nhận vào là số lượng host liên kết với switch s3 và s4.

- Mininet API khá đơn giản và dễ hiểu. Các phương thức addHost, addSwitch hay addLink hoạt động đúng với cái tên của nó, sẽ tạo ra các host, switch và các kết nối giữa chúng.
- Khi tạo host sử dụng addHost(), ta có thể chỉ ra cụ thể các thông số cấu hình host đó như tỉ lệ phần trăm tối đa CPU mà host đó sử dụng của máy thật, địa chỉ IP, địa chỉ MAC, default gateway,.. Ví dụ ở đây, ta cho phép h1 và h2 sử dụng tối đa 10% CPU máy thật, cài đặt địa chỉ IP và default gateway của chúng,..
- Khi tạo liên kết sử dụng addLink(), ta cũng có thể định nghĩa các thông số mạng giả lập trên những liên kết đó như băng thông (bw), độ trễ (delay), tỉ lệ mất mát gói tin (loss)
- Hàm run() định nghĩa cách thức khởi tạo mạng, cách thức kết nối, chỉ ra controller của SDN được tạo. Ở đây, ta thêm Controller sử dụng API addController(). Controller được thêm có địa chỉ IP là 127.0.0.1:6653, chính là địa chỉ mặc định của ONOS controller mà ta vừa khởi động ở phần trên. Nếu không chỉ ra controller rõ ràng, Mininet sẽ sử dụng controller mặc định được tạo ra.
- Trên 2 host h1 và h2, ta cũng khởi chạy một số service phổ biến như http web server tại cổng 80, ftp server tại cổng 21, ssh service tại cổng 22. Cụ thể:
  - **\$ sudo twistd -n ftp -p 21 -r /home/dangnh/ids &> /dev/null &**  
 Khởi chạy một ftp server đơn giản sử dụng twistd. Việc này yêu cầu Twisted phải được cài đặt trên máy vật lý thật. Twisted là một framework hướng sự kiện sử dụng để lập trình các ứng dụng mạng, chi tiết tại <https://pypi.org/project/Twisted/>, và có thể được cài đặt đơn giản sử dụng trình quản lý gói của python, PIP như sau:  
**\$ sudo pip install Twisted**
  - **\$ python -m SimpleHTTPServer 80 &> /dev/null &**  
 Khởi chạy một http web server đơn giản sử dụng gói SimpleHTTPServer được cung cấp mặc định kèm theo python.
  - **\$ /usr/sbin/sshd -D &**  
 Khởi chạy ssh daemon
- API pingAll() thực hiện ping giữa các node trong mạng với nhau để kiểm tra kết nối. Cũng cần chú ý rằng, giao diện topology của ONOS chỉ hiển thị và nhận biết được các node khi có gói tin đến/đi từ host đó trong mạng, vì vậy điều này còn giúp ONOS controller nhận biết được các host trong mạng.
- Cuối cùng, để kiểm tra thông tin các service mà ta vừa khởi chạy tại h1 và h2 ở trên đã hoạt động chưa, ta in ra thông tin các service đang chạy trên chúng với câu lệnh

**\$ netstat -tulpn | grep LISTEN**

- Dòng cuối cùng

```
topos = { 'tp': ( lambda n1, n2 : NetworkTopo(n1, n2) ) }
```

sẽ định nghĩa class NetworkTopo mà ta nêu ở trên, là một topo tùy biến. Ta có thể tái sử dụng topo đó sau này như một thành phần mở rộng của Mininet, ví dụ:

**\$ sudo mn --controller=remote,ip=127.0.0.1,port=6653 --custom mytopo.py --topo tp,3,3**

sẽ tạo và khởi chạy mạng ảo sinh ra với topo giống hệt như ta định nghĩa trong class với 3 host liên kết với switch s3 và 3 host liên kết với switch s4 và kết nối tới ONOS controller tại 127.0.0.1:6653

Sau khi hoàn thành script, ta sẽ thực thi nó để tạo ra mạng ảo sử dụng Mininet:

**\$ sudo python mytopo.py**

Kết quả thu được như sau:

```
> sudo python mytopo.py
*** Creating network
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8 r0
*** Adding switches:
s1 s2 s3 s4
*** Adding links:
(20.00Mbit 1ms delay) (20.00Mbit 1ms delay) (h1, s1) (20.00Mbit 1ms delay) (20.00Mbit 1ms delay) (h2,
s1) (15.00Mbit 2ms delay) (15.00Mbit 2ms delay) (h3, s3) (15.00Mbit 2ms delay) (15.00Mbit 2ms delay)
(h4, s3) (15.00Mbit 2ms delay) (15.00Mbit 2ms delay) (h5, s3) (15.00Mbit 2ms delay) (15.00Mbit 2ms d
elay) (h6, s4) (15.00Mbit 2ms delay) (15.00Mbit 2ms delay) (h7, s4) (15.00Mbit 2ms delay) (15.00Mbit
2ms delay) (h8, s4) (s1, r0) (20.00Mbit 1ms delay) (20.00Mbit 1ms delay) (s2, r0) (s3, s2) (s4, s2)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 r0
*** Starting controller
c0
*** Starting 4 switches
s1 s2 s3 s4 ... (20.00Mbit 1ms delay) (20.00Mbit 1ms delay) (20.00Mbit 1ms delay) (15.00Mbit 2ms delay)
(15.00Mbit 2ms delay) (15.00Mbit 2ms delay) (15.00Mbit 2ms delay) (15.00Mbit 2ms delay) (15.00Mbit
2ms delay)
*** Routing Table on Router:
Kernel IP routing table
Destination      Gateway         Genmask         Flags Metric Ref    Use Iface
100.10.0.0        0.0.0.0         255.255.255.0   U        0      0        0 r0-eth2
192.168.2.0        0.0.0.0         255.255.255.0   U        0      0        0 r0-eth1
*** Starting some service*** Testing connection:
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7 h8 r0
h2 -> h1 h3 h4 h5 h6 h7 h8 r0
h3 -> h1 h2 h4 h5 h6 h7 h8 r0
h4 -> h1 h2 h3 h5 h6 h7 h8 r0
h5 -> h1 h2 h3 h4 h6 h7 h8 r0
h6 -> h1 h2 h3 h4 h5 h7 h8 r0
h7 -> h1 h2 h3 h4 h5 h6 h8 r0
h8 -> h1 h2 h3 h4 h5 h6 h7 r0
r0 -> h1 h2 h3 h4 h5 h6 h7 h8
*** Results: 0% dropped (72/72 received)
0.0*** Running services on host h1:
tcp        0      0 0.0.0.0:21          0.0.0.0:*        LISTEN      28014/python
tcp        0      0 0.0.0.0:22          0.0.0.0:*        LISTEN      28011/sshd
tcp        0      0 0.0.0.0:80          0.0.0.0:*        LISTEN      28010/python
tcp6       0      0 :::22              :::*             LISTEN      28011/sshd
*** Running services on host h2:
tcp        0      0 0.0.0.0:80          0.0.0.0:*        LISTEN      28012/python
tcp        0      0 0.0.0.0:22          0.0.0.0:*        LISTEN      28013/sshd
tcp6       0      0 :::22              :::*             LISTEN      28013/sshd
*** Starting CLI:
mininet> |
```

Hình 27. Kết quả chạy script mytopo.py

Kết quả cũng hiển thị trên giao diện của ONOS như sau:

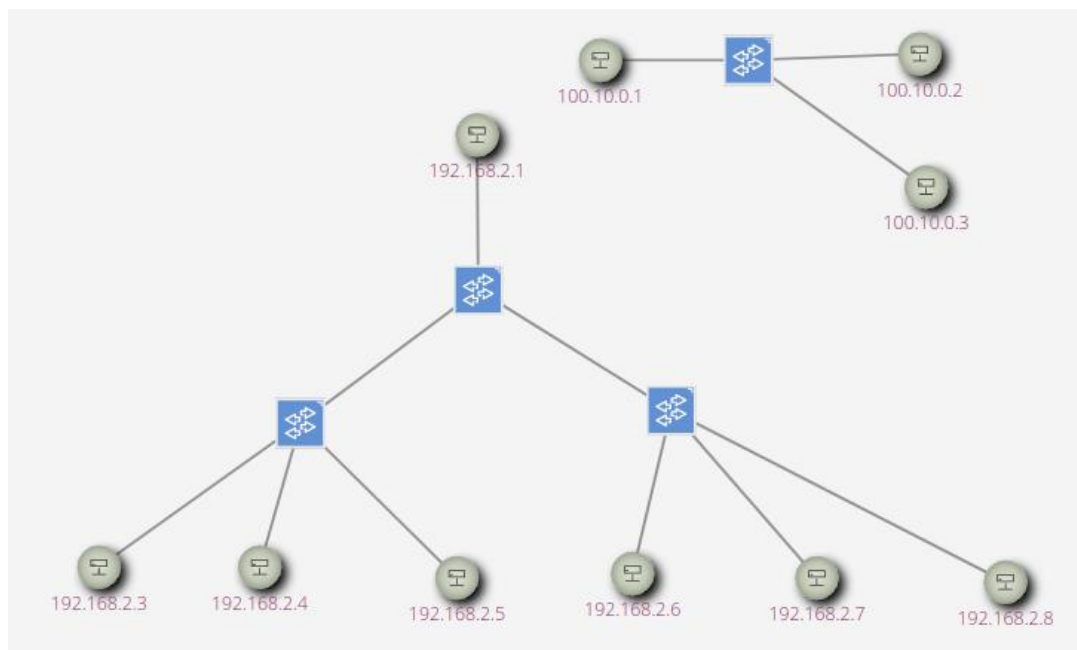
Hosts (10 total)

FRIENDLY NAME	HOST ID	MAC ADDRESS	VLAN ID	CONFIGURED	IP ADDRESSES	LOCATION
192.168.2.8	5A2F:27:F5:61:45/None	5A:2F:27:F5:61:45	None	false	192.168.2.8	of:0000000000000004/4
192.168.2.7	02:65:D1:5D:83:C3/None	02:65:D1:5D:83:C3	None	false	192.168.2.7	of:0000000000000004/3
192.168.2.6	72:66:1F:52:DC:69/None	72:66:1F:52:DC:69	None	false	192.168.2.6	of:0000000000000004/2
192.168.2.5	A2:F2:19:8B:5C:E7/None	A2:F2:19:8B:5C:E7	None	false	192.168.2.5	of:0000000000000003/4
192.168.2.4	1A:82:3F:43:EB:73/None	1A:82:3F:43:EB:73	None	false	192.168.2.4	of:0000000000000003/3
192.168.2.3	B6:9C:3D:81:3E:39/None	B6:9C:3D:81:3E:39	None	false	192.168.2.3	of:0000000000000003/2
192.168.2.1	62:E1:3D:EA:5B:D2/None	62:E1:3D:EA:5B:D2	None	false	192.168.2.1	of:0000000000000002/1
100.10.0.3	E2:7C:F7:D7:6E:B2/None	E2:7C:F7:D7:6E:B2	None	false	100.10.0.3	of:0000000000000001/3
100.10.0.2	A2:86:9B:D9:99:36/None	A2:86:9B:D9:99:36	None	false	100.10.0.2	of:0000000000000001/2
100.10.0.1	7A:B4:0A:B4:61:29/None	7A:B4:0A:B4:61:29	None	false	100.10.0.1	of:0000000000000001/1

Hình 28. Thông tin chi tiết các host trong mạng

	FRIENDLY NAME	DEVICE ID	MASTER	PORTS	VENDOR	H/W VERSION	S/W VERSION	PROTOCOL
✓	of:0000000000000001	of:0000000000000001	192.168.43.126	4	Nicira, Inc.	Open vSwitch	2.9.5	OF_14
✓	of:0000000000000002	of:0000000000000002	192.168.43.126	4	Nicira, Inc.	Open vSwitch	2.9.5	OF_14
✓	of:0000000000000003	of:0000000000000003	192.168.43.126	5	Nicira, Inc.	Open vSwitch	2.9.5	OF_14
✓	of:0000000000000004	of:0000000000000004	192.168.43.126	5	Nicira, Inc.	Open vSwitch	2.9.5	OF_14

Hình 29. Thông tin chi tiết các switch trong mạng



Hình 30. Topo mạng hiển thị trên giao diện ONOS

Như đã chú ý ở phần trước, ta cũng có thể sử dụng mytopo.py như một phần mở rộng cho mininet. Khi đó, ta khởi tạo mạng bằng câu lệnh:

```
$ sudo mn --controller=remote,ip=127.0.0.1,port=6653 --custom mytopo.py --topo tp,3,3 --mac
```

Trong đó:

- controller: chỉ ra controller của SDN được tạo
- custom : chỉ ra file .py mở rộng
- topo: sử dụng topo có tên và tham số đi kèm
- mac: tắt chế độ gán địa chỉ MAC ngẫu nhiên, thay vào đó địa chỉ MAC của các host sẽ dễ theo dõi hơn, ví dụ: 00:00:00:00:00:01, 00:00:00:00:00:02, 00:00:00:00:00:03,..

Tuy nhiên, các thiết lập giống như trong hàm run() trong mã nguồn topo.py sẽ không được thực hiện. Thay vào đó, ta phải thực hiện thủ công. Ví dụ: nếu muốn khởi chạy http web service trên h1, ta phải thực hiện câu lệnh sau trong CLI của mininet:

```
mininet> h1 python -m SimpleHTTPServer 80 &> /dev/null &
```

và tương tự với các câu lệnh khác.

Cũng chú ý rằng, các host trong mininet chia sẻ chung một filesystem và ứng dụng của máy vật lý bên dưới nó. Đồng nghĩa với hầu hết phần mềm/câu lệnh được cài đặt và chạy trên máy thật, cũng có thể thực hiện được trên terminal của host ảo trong mạng. Cú pháp chung là:

```
mininet> {host} {command}
```

sẽ thực hiện command trên host chỉ định. Hoặc có thể tạo cửa sổ xterm cho host trong mạng bằng câu lệnh xterm, đi sau là danh sách các host. Ví dụ:

```
mininet> xterm h1 h2 h3
```

sẽ tạo ra 3 cửa sổ xterm tương ứng với 3 host h1, h2 và h3. Điều này có ích khi thực hiện các câu lệnh có output dài, hoặc các câu lệnh blocking, hoặc đòi hỏi thực thi song song nhau trên các host. Điều này rất hữu ích và sẽ được sử dụng nhiều trong phần sau, đòi hỏi giả lập tấn công và giao tiếp giữa các host trong mạng.

## 4.3. Mô phỏng một số tấn công trên hệ thống giả lập

### 4.3.1. Một số kiểu tấn công

Phần 4.3.1.1 tới 4.3.1.7 đề cập tới các tấn công dạng TCP SYN flood, TCP RST flood, TCP FIN flood, ICMP flood, UDP flood, HTTP flood, HTTP slow, cùng thuộc vào dạng tấn công DOS (Denial Of Service).

Các dạng tấn công do thám sử dụng công cụ Nmap sẽ được đề cập tới trong phần 4.3.1.8.

Phần 4.3.1.9 và 4.3.1.10 đề cập tới các dạng tấn công nhằm mục đích nghe lén man-in-the-middle.

#### 4.3.1.1. TCP SYN flood

TCP SYN flood là một dạng tấn công từ chối dịch vụ (DOS), lợi dụng cách thức hoạt động của kết nối TCP/IP, hacker bắt đầu quá trình thiết lập một kết nối TCP/IP tới mục tiêu muốn tấn công mà không gửi trả gói tin ACK, khiến cho mục tiêu luôn rơi vào trạng thái chờ (đợi gói tin ACK từ phía yêu cầu thiết lập kết nối) và liên tục gửi gói tin SYN ACK để thiết lập kết nối. h3ping là công cụ dễ dàng sử dụng để thực hiện loại tấn công này. Ví dụ:

**\$ hping3 -V -d 100 -S -p 80 --flood 100.10.0.2**

Trong đó:

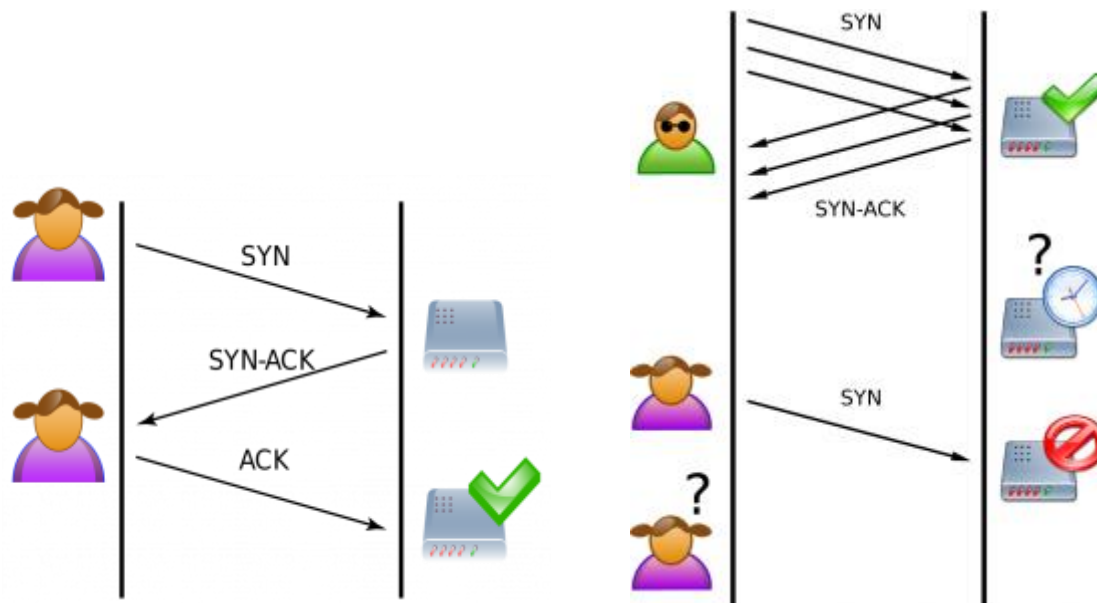
- V: Hiển thị chi tiết (verbose)
- d 100: kích thước gói tin là 100 bytes
- S: loại gói tin là SYN
- p 80: tấn công dịch vụ trên cổng 80 (http)
- flood: gửi gói tin nhanh nhất có thể
- 100.10.0.2: địa chỉ đích muốn tấn công

Một cách khác là giả mạo địa chỉ IP nguồn của gói tin yêu cầu thiết lập kết nối SYN và cũng như trường hợp trên, máy tính đích cũng rơi vào trạng thái chờ vì các gói tin SYN ACK không thể đi đến đích do địa chỉ IP nguồn là không có thật. Ví dụ:

**\$ hping3 -V -d 120 -S -p 80 --flood --rand-source 100.10.0.2**

Trong đó:

- rand-source: địa chỉ nguồn ngẫu nhiên



Hình 31. Minh họa tấn công TCP SYN lợi dụng quá trình bắt tay của giao thức TCP

Kiểu tấn công Land Attack cũng tương tự như SYN flood, nhưng hacker sử dụng chính IP của mục tiêu cần tấn công để dùng làm địa chỉ IP nguồn trong gói tin, đẩy mục tiêu vào một vòng lặp vô tận khi cố gắng thiết lập kết nối với chính nó. Ví dụ:

**\$ hping3 -V -d 100 -S -s 10000 -p 80 -k --flood -a 100.10.0.2 100.10.0.2**

Trong đó:

-k: giữ cùng 1 port khi gửi (là 80)

-a: chỉ ra địa chỉ nguồn là 100.10.0.2

#### 4.3.1.2. TCP RST Flood

Gói tin RST trong kết nối TCP mang ý nghĩa là ngay lập tức hủy kết nối. Gói tin RST thường xuyên được sử dụng trong các trường hợp kết nối gặp lỗi và muốn hủy kết nối ngay lập tức. Nếu kẻ tấn công bằng cách nào đó có thể nghe trộm được lưu lượng đi từ nguồn đến đích (ví dụ, sử dụng thêm các kiểu tấn công nghe lén man-in-the-middle, chúng có thể gửi các gói RST với các giá trị tham số phù hợp (IP nguồn, IP đích, cổng nguồn, cổng đích, sequence number, v.v.) Gói này sẽ hủy ngay lập tức kết nối TCP giữa nguồn và đích trong trường hợp các tham số là hợp lệ (khó đoán biết đúng nhất là giá trị sequence number). Hoặc nếu kẻ tấn công không thể lắng nghe và đoán biết giá trị sequence number này, bằng cách liên tục gửi rất nhiều gói tin TCP RST với các giá trị sequence number khác nhau, có thể may mắn chính xác là làm ngắt kết nối TCP giữa chúng, thậm chí làm cho việc thiết lập lại kết nối là không thể thực hiện được (liên tục bị hủy trong thời gian ngắn). Dễ thấy, nó nguy hiểm khi tấn công vào các ứng dụng đòi hỏi kết nối TCP một cách liên tục, và khả năng chịu lỗi kém khi kết nối bị ngắt quãng giữa chừng. Tương tự TCP SYN, tấn công này cũng có thể thực hiện bằng công cụ hping3. Ví dụ:

**\$ hping3 --flood --rand-source -R -p 22 100.10.0.2**

Trong đó:

-R: gói tin TCP RST

-p 22: tấn công vào dịch vụ trên cổng 22 (thường là dịch vụ ssh)

#### 4.3.1.3. TCP FIN flood

Gói TCP có flag là FIN được bật chỉ được chấp nhận khi máy khách đã thiết lập kết nối TCP với máy chủ. Nếu không, các gói tin này sẽ được bỏ đi. Nếu kẻ tấn công gửi rất nhiều các gói tin FIN tới phía mục tiêu trong khi không thiết lập kết nối, các gói FIN sẽ bị hủy như mong đợi. Tuy nhiên, máy chủ vẫn yêu cầu một lượng tài nguyên để xử lý từng gói, kiểm tra có kết nối được thiết lập hay không (gói tin này có là dư thừa hay không). Các kiểu tấn công này rất dễ thực hiện vì nó chỉ tạo các gói FIN rác và gửi đi. Tương tự như TCP SYN/RST flood, hping3 có thể được sử dụng để thực hiện dạng tấn công này. Ví dụ:

```
$ hping3 --rand-source -F -q -d 80 -p 80 --flood 100.10.0.2
```

Trong đó:

--rand-source: địa chỉ nguồn bất kỳ (mạo danh)

-F: gói tin TCP FIN

#### 4.3.1.4. ICMP flood

ICMP (Giao thức tin nhắn điều khiển Internet) là giao thức không hướng kết nối, như UDP. ICMP được sử dụng để gửi thông báo lỗi và thông tin, trạng thái hoạt động từ các thiết bị mạng. Khi gói tin ICMP request được gửi tới một host, nó sẽ tiêu tốn một chút tài nguyên hệ thống để xử lý và phản hồi lại gói tin ICMP reply tới địa chỉ nguồn. Lợi dụng điều đó, kẻ tấn công gửi hàng loạt gói tin ICMP echo request. Tấn công thường thành công hơn khi kẻ tấn công có băng thông lớn hơn so với mục tiêu. Tấn công dạng này đặc biệt nguy hiểm hơn, khi kiểu tấn công này lợi dụng botnet (phát triển thành dạng DDOS) và có cơ chế che giấu IP thực. Ví dụ cho tấn công ICMP flood sử dụng hping3:

```
$ hping3 --flood --rand-source -1 -p 80 100.10.0.2
```

Trong đó:

-1: gửi gói tin ICMP

#### 4.3.1.5. UDP flood

UDP là một giao thức không hướng kết nối, tức là không cần thiết lập kết nối giữa hai thiết bị như TCP. Mục đích của UDP flood chỉ đơn giản là tạo và gửi một lượng lớn các datagram UDP từ IP giả mạo đến máy chủ đích. Khi một máy chủ nhận được loại lưu lượng này, nó sẽ tiêu tốn một lượng tài nguyên để kiểm tra ứng dụng sử dụng nào sử dụng kết nối UDP đó hay không. Nếu không, nó sẽ gửi lại phía nguồn gói tin ICMP Destination Unreachable. Tuy nhiên với quá nhiều gói tin từ tấn công UDP flood, phía đích tiêu tốn hết tài nguyên, quá tải và không thể phản hồi lại các kết nối khác. Cuộc tấn công UDP flood cũng có thể giả mạo địa chỉ IP, để đảm bảo rằng các gói ICMP trả lại không tiếp cận được với máy của chúng (sử dụng để tấn công) và để ẩn danh cuộc tấn công. Ví dụ:

```
$ hping3 -d 110 --flood --rand-source --udp -p 21 100.10.0.2
```

Trong đó:

--rand-source: địa chỉ nguồn bất kỳ (mạo danh)

--udp: gói tin UDP

#### 4.3.1.6. HTTP flood

HTTP flood là cuộc tấn công phổ biến nhất nhắm vào tầng ứng dụng. Các cuộc tấn công này khó phát hiện hơn các cuộc tấn công vào tầng mạng vì các yêu cầu gửi tới có vẻ như là hợp lệ. Trong khi vẫn hoàn tất quá trình bắt tay 3 bước của kết nối TCP, HTTP flood đánh lừa các



thiết bị và giải pháp chỉ kiểm tra tới tầng giao vận trở xuống. Các kiểu tấn công này thường bao gồm việc gửi một số lượng lớn các yêu cầu HTTP GET hoặc POST đến máy chủ đích. Thông thường, HTTP flood được khởi chạy từ nhiều máy tính cùng một lúc (DDOS). Các ứng dụng phản hồi với các yêu cầu HTTP GET và HTTP POST có thể tiêu tốn nhiều tài nguyên (nằm trên tầng ứng dụng). Tấn công dạng này có thể làm cạn kiệt tài nguyên máy chủ phục vụ, ngăn cản các kết nối hợp lệ khác. Để thực hiện dạng tấn công này, có nhiều công cụ như HULK (HTTP Unbearable Load King), hoặc phần mềm kiểm thử như Apache JMeter cũng có thể sử dụng để thực hiện. Ví dụ đơn giản sử dụng HULK (mã nguồn mở tại <https://github.com/grafov/hulk>) tấn công vào dịch vụ HTTP trên máy chủ 100.10.0.2 như sau:

```
$ python hulk.py http://100.10.0.2
```

#### 4.3.1.7. HTTP slow

Không giống như các cuộc tấn công flood, các cuộc tấn công làm chậm (low and slow) không đòi hỏi băng thông lớn và lượng dữ liệu khổng lồ. Những loại tấn công này tập trung vào các ứng dụng và tài nguyên máy chủ. Chúng cũng khó phát hiện vì lưu lượng truy cập dường như không có gì bất thường.. Slowloris là công cụ có thể được sử dụng để thực hiện các loại tấn công này. Loại tấn công này hoạt động bằng cách mở nhiều kết nối TCP nhất có thể và giữ cho chúng mở càng lâu càng tốt. Nó sẽ gửi một phần các thông điệp/yêu cầu HTTP (ví dụ HTTP header) một cách định kì, và giữ cho không có kết nối nào trong số này sẽ hoàn thành/kết thúc. Nếu máy chủ giới hạn số lượng kết nối, nó sẽ không thể nhận các yêu cầu hợp lệ từ các client khác. Công cụ có thể sử dụng là Slowloris (mã nguồn mở tại <https://github.com/gkbrk/slowloris>). Có thể cài đặt dễ dàng:

```
$ sudo pip3 install slowloris
```

Ví dụ:

```
$ slowloris -v -s 1000 --sleeptime 2 100.10.0.2
```

Trong đó:

- v: hiển thị logging (verbose)
- s 1000: sử dụng 1000 socket
- sleeptime 2: Thời gian chờ (sleep) trước khi tiếp tục gửi phần header tiếp theo để giữ kết nối
- 100.10.0.2: địa chỉ mục tiêu tấn công

#### 4.3.1.8. Các dạng tấn công Nmap scan

Do thám và thu thập thông tin hệ thống là giai đoạn đầu tiên khi tin tặc muốn tấn công vào hệ thống. Trong đó, kẻ tấn công muốn khai thác các thông tin về hệ điều hành, các dịch vụ/cổng đang mở, phiên bản phần mềm của các dịch vụ đang mở này nhằm mục đích xem xét điểm yếu của hệ thống. Có rất nhiều công cụ phục vụ mục đích này, trong đó có Nmap. Nmap là công cụ quét cổng mã nguồn mở hoạt động trên cả Windows/Linux/Mac. Các tính năng chính:

- Host discovery – Xác định các máy (host) trong mạng. Chẳng hạn thông qua việc host có phản hồi gói tin TCP hoặc ICMP
- Port scanning – Liệt kê các cổng mở của một host
- Version detection – Xác định các dịch vụ hoạt động trên host, theo port và phiên bản của dịch vụ đó

- OS detection – Xác định phiên bản hệ điều hành của host hoặc thiết bị mạng

Mục đích của các kỹ thuật quét cổng ứng dụng là để phát hiện các dịch vụ mạng nào đang hoạt động trên máy trạm. Giả sử, quá trình quét cổng cho kết quả cổng 80 đang mở (open), chứng tỏ trên nút mạng đó đang cung cấp dịch vụ Web.

Nmap có thể sử dụng để quét mạng (tìm ra các host hoạt động trong mạng). Ví dụ:

**\$ nmap -sn 192.168.2.0/24 --disable-arp-ping**

Trong đó:

-sn: ping scan (sử dụng gói tin ICMP)

192.168.2.0/24: địa chỉ mạng cần quét

--disable-arp-ping: không thực hiện ARP scanning bằng việc gửi các gói ARP request

Nmap cũng cung cấp các kỹ thuật quét cổng sau:

- TCP Connection Scan: tương tự như TCP SYN Scan. Điểm khác biệt nhỏ là Nmap sẽ gửi lại gói tin ACK để hoàn tất quá trình thiết lập liên kết. Sau đó, Nmap gửi lại ngay các gói tin RST để hủy liên kết này. Ví dụ:

**\$ nmap -sT 100.10.0.2**

**\$ nmap -sT -F 100.10.0.2**

Trong đó:

-sT: sử dụng chế độ TCP connection scan

-F: scan ít port hơn, tốc độ nhanh hơn

- TCP SYN Scan: Nmap gửi lần lượt các gói tin TCP SYN tới các cổng cần quét. Nếu quá trình quét cổng nhận được bất kỳ gói tin TCP SYN/ACK trả lời từ một cổng nào đó, chứng tỏ cổng đó đang ở trạng thái open. Để thực hiện kỹ thuật này, sử dụng option **-sS**
- TCP Null Scan: Nmap gửi lần lượt các gói tin TCP có giá trị Flags trong tiêu đề là NULL tới các cổng cần quét. Nếu nhận được gói tin TCP RST trả lời từ bất kỳ cổng nào, chứng tỏ cổng đó đang ở trạng thái đóng (Close). Tuy nhiên, hiện tại hầu hết các hệ điều hành đã ngăn chặn được kỹ thuật quét cổng này. Để thực hiện kỹ thuật này, sử dụng option **-sN**.
- TCP FIN Scan: Nmap gửi lần lượt các gói tin TCP có giá trị Flags trong tiêu đề là FIN tới các cổng cần quét. Nếu nhận được gói tin TCP RST trả lời từ bất kỳ cổng nào, chứng tỏ cổng đó đang ở trạng thái đóng (Close). Tuy nhiên, hiện tại hầu hết các hệ điều hành đã ngăn chặn được kỹ thuật quét cổng này. Để thực hiện kỹ thuật này, sử dụng option **-sF**.
- TCP Xmas Scan: Nmap gửi lần lượt các gói tin TCP có giá trị Flags trong tiêu đề là FIN, PSH, và URG tới các cổng cần quét. Nếu nhận được gói tin TCP RST trả lời từ bất kỳ cổng nào, chứng tỏ cổng đó đang ở trạng thái đóng (Close). Tuy nhiên, hiện tại hầu hết các hệ điều hành đã ngăn chặn được kỹ thuật quét cổng này. Để thực hiện kỹ thuật này, sử dụng option **-sX**.
- TCP ACK Scan: Kỹ thuật này được sử dụng để phát hiện trên mục tiêu có sử dụng firewall hay không. Nmap gửi lần lượt các gói tin TCP có giá trị Flags trong tiêu đề là ACK tới các cổng cần quét. Nếu nhận được gói tin TCP RST trả lời từ bất kỳ cổng

nào, chúng tỏ mục tiêu có thể đã được bảo vệ bởi firewall. Để thực hiện kỹ thuật này, sử dụng option **-sA**.

- UDP Connection Scan: Gửi hàng loạt gói tin UDP tới các cổng trên mục tiêu. Nếu nhận lại gói tin ICMP Destination Unreachable, chứng tỏ cổng đó đang đóng. Để thực hiện kỹ thuật này, sử dụng option **-sU**.

#### **4.3.1.9. MAC flooding**

Tấn công MAC flooding là một kiểu tấn công mạng lợi dụng khả năng tự học địa chỉ MAC trên switch, trong đó kẻ tấn công kết nối với switch và tạo ra số lượng rất lớn các Ethernet frame khác nhau với địa chỉ MAC nguồn là giả.

Trong một thời gian rất ngắn, bảng địa chỉ MAC của switch đã bị lấp đầy các ánh xạ giữa cổng-địa chỉ MAC giả. Thông thường, bảng địa chỉ MAC của Switch chỉ có một lượng bộ nhớ hạn chế. Switch sẽ không thể lưu thêm bất kỳ địa chỉ MAC nào trong bảng ánh xạ địa chỉ của nó.

Khi bảng địa chỉ MAC của switch đã đầy và nó không thể lưu thêm bất kỳ địa chỉ MAC nào nữa, nó sẽ chuyển sang chế độ mở và bắt đầu hoạt động như một Hub. Các gói tin được quảng bá ra tất cả các cổng (interface) của nó. Như vậy, toàn bộ lưu lượng mạng có thể bị nghe lén trên một node mạng và kẻ tấn công có thể dễ dàng nghe lén, tấn công và thu thập dữ liệu nhạy cảm từ mạng. Để thực hiện kiểu tấn công này, có thể sử dụng công cụ macof, là một phần trong gói dsniff. Để cài đặt:

**\$ sudo apt install dsniff**

Sử dụng:

**\$ macof -i h3-eth0**

Trong đó:

-i h3-eth0: (interface) chọn card mạng h3-eth0 để thực hiện tấn công. Switch kết nối với card mạng này là mục tiêu.

#### **4.3.1.10. ARP Poisoning**

ARP spoofing, ARP cache poisoning, hay ARP poison routing là một kỹ thuật qua đó kẻ tấn công giả thông điệp ARP trong mạng cục bộ, trong đó kết hợp địa chỉ MAC của kẻ tấn công với địa chỉ IP của máy chủ khác, chẳng hạn như cổng mặc định (default gateway), làm cho bất kỳ lưu lượng truy cập nào đi qua địa chỉ IP đó được gửi đến kẻ tấn công.

Giao thức phân giải địa chỉ (ARP) là giao thức được sử dụng bởi giao thức Internet (IP), cụ thể là IPv4, để ánh xạ địa chỉ mạng IP tới các địa chỉ phần cứng MAC được sử dụng bởi giao thức tại tầng liên kết dữ liệu. Giao thức ARP là giao thức phi trạng thái, không có cơ chế kiểm tra, các máy sẽ lưu bất kỳ ARP reply nào mà chúng nhận được, bất kể có ARP request hay không. Đồng thời, các bản ghi ARP mới sẽ ghi đè lên bản ghi cũ. Vì vậy, kẻ tấn công có thể đánh lừa bằng cách gửi các ARP reply giả mạo dù không có bất kỳ ARP request nào. Kịch bản sau đây cho phép kẻ tấn công nghe lén kết nối giữa 2 bên:

1. Attacker gửi gói tin ARP Reply tới gateway với nội dung giả mạo là địa chỉ IP của Victim và địa chỉ MAC của Attacker → Gateway bị lừa rằng Victim đang dùng IP của Attacker

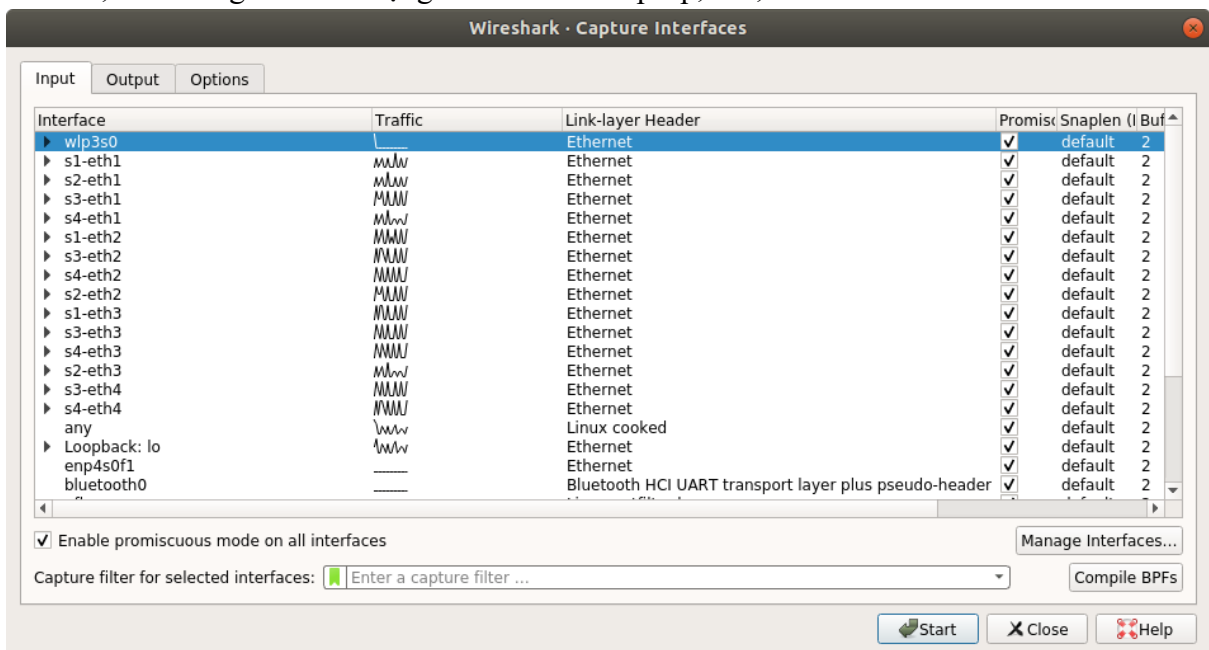
2. Attacker gửi một gói tin ARP Reply tới Victim với nội dung giả mạo là IP của gateway tại địa chỉ MAC của Attacker → Victim bị lừa rằng gateway đang dùng IP của Attacker.
3. Lúc này, Attacker ở giữa 2 bên Gateway và Victim và có khả năng nghe trộm mà không tạo ra dấu hiệu bất thường bằng việc vẫn chuyển tiếp các gói tin qua lại một cách hợp lý.

Có thể sử dụng công cụ Ettercap để thực hiện dạng tấn công này và các tấn công dạng man-in-the-middle. Ta sẽ không đi vào chi tiết cách thức sử dụng công cụ này.

#### 4.3.2. Thu thập dữ liệu

Để thu thập dữ liệu lưu lượng thô tcp dump, ta sử dụng phần mềm Wireshark. Wireshark là một bộ phân tích gói tin miễn phí và nguồn mở. Nó được sử dụng để xử lý sự cố mạng, gỡ lỗi, phân tích, phát triển giao thức phần mềm và truyền thông. Wireshark rất giống với tcpdump, nhưng có giao diện đồ họa (GUI) rõ ràng và dễ sử dụng, đi kèm với một số công cụ, tùy chọn hỗ trợ sắp xếp và lọc gói tin (filter) tích hợp.

Wireshark cho phép lắng nghe lưu lượng mạng đi qua các card mạng (kể cả card mạng ảo) và hiển thị, lưu chúng dưới các dạng khác nhau như pcap, csv,...



Hình 32. Giao diện chọn card mạng để lắng nghe lưu lượng của Wireshark

##### 4.3.2.1. Thu thập dữ liệu tấn công

Dữ liệu lưu lượng cần được gán nhãn một cách thủ công. Đồng thời, các cuộc tấn công phải đa dạng về cách thức, thời gian tấn công, các tham số của quá trình tấn công. Việc tự động hóa công việc thực hiện tấn công và gán nhãn cho lưu lượng thu thập được trong cuộc tấn công đó nên được thực hiện một cách có hệ thống bằng việc viết những script tự động thực hiện công việc này. Ví dụ như sau:

- Liệt kê các dạng tấn công muốn thực hiện và danh sách tổ hợp các tham số muốn sử dụng đối với dạng tấn công đó. Chú ý rằng các tham số nên thay đổi đa dạng và khác nhau. Các tham số như thời gian tấn công cũng nên thay đổi và khác nhau.

- Tự động hóa việc tấn công và thu thập lưu lượng, bằng cách duyệt qua từng phần tử trong tổ hợp các tham số đã định nghĩa ở trên. Ta sử dụng 2 tiến trình chạy song song, 1 tiến trình thực hiện tấn công sử dụng các công cụ như hping3, hulk, slowloris,.. và 1 tiến trình tcpdump ghi lại lưu lượng mạng. Chú ý rằng phải khởi chạy các trình tcpdump với các filter về địa chỉ nguồn-địa chỉ đích, hay loại gói tin để tránh nhầm lẫn ghi lại cả những gói tin thông thường thành gói tin tấn công (ví dụ gói tin OpenFlow trong mạng SDN).

Có thể thấy việc này không hề dễ dàng, ngược lại sẽ gặp nhiều vấn đề trong triển khai: tự động hóa việc tấn công giữa các host khác nhau trong mạng, ... và rất dễ gặp lỗi.

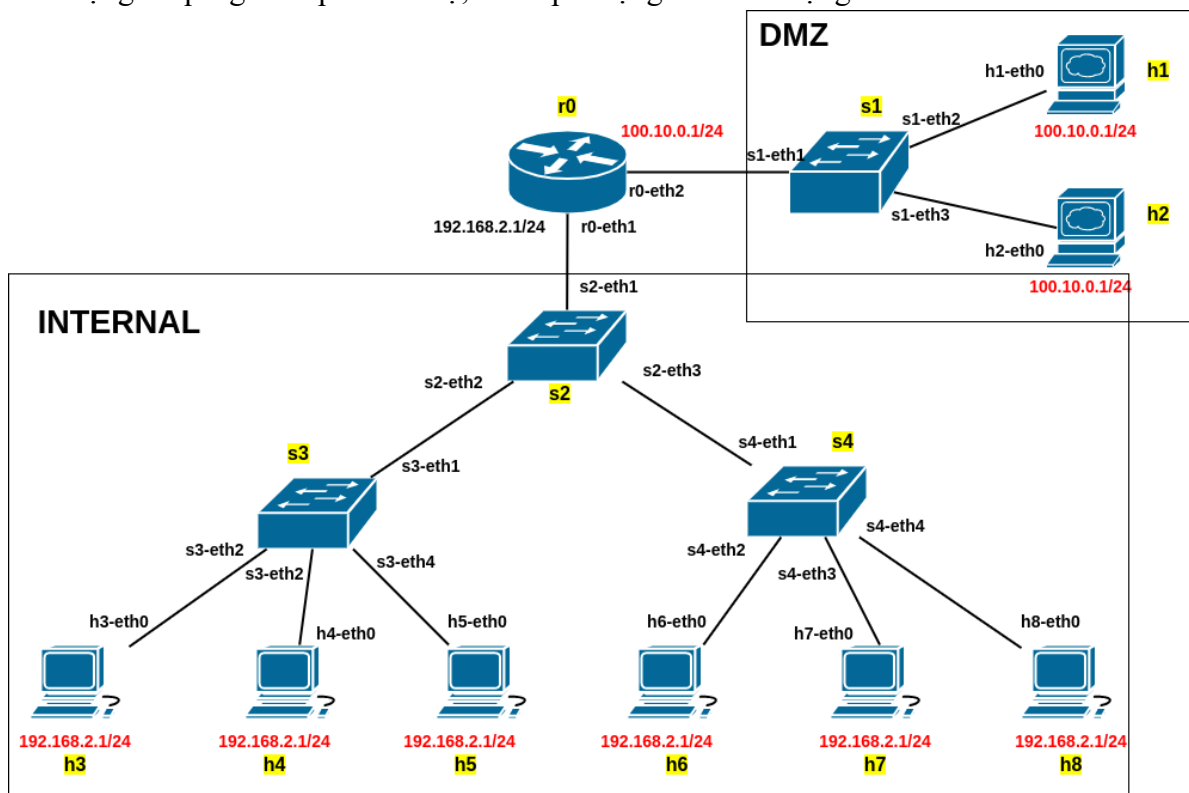
Ở đây, ta chọn cách thức thu thập dữ liệu thủ công để giảm tính phức tạp và tránh gặp lỗi.

Quá trình sau đây giả định một mạng ảo theo topology đã nêu, sử dụng ONOS và Mininet đã được khởi động như trong phần 4.2.2. đã đề cập.

Với mỗi phiên tấn công, lặp lại các bước sau:

- **Bước 1: Khởi động Wireshark lắng nghe trên card mạng phù hợp**

Xác định phiên tấn công là giữa các host nào với nhau để lắng nghe lưu lượng trên network interface tương ứng. Thông thường, lắng nghe trên interface “any” có thể lắng nghe được toàn bộ lưu lượng trên các interface khác. Tuy nhiên, để thực hiện cơ chế này, đã có một chút biến đổi trong cấu trúc và nội dung gói tin, vì vậy công cụ trích xuất đặc trưng mà ta sử dụng (sẽ nêu kĩ ở phần sau) không thể sử dụng cho loại lưu lượng thu từ interface “any”. Do đó, khi thực hiện tấn công, ta buộc phải xác định trước tại interface nào sẽ lắng nghe được toàn bộ lưu lượng đi qua giữa 2 phía. Ví dụ, với topo mạng mà ta sử dụng như sau:



Hình 33. Topo mạng mà ta sử dụng

Giả sử tấn công được thực hiện từ h3 tới h1, lưu lượng giữa 2 phía có thể được lắng nghe trên interface s1-eth2 hoặc s1-eth1 của switch s1. Giữa h4 với h6 có thể lắng nghe trên interface

s2-eth3 hoặc s4-eth1 của switch s4. Để xác định được cần lắng nghe ở interface nào, ta phải xác định đường đi của gói tin giữa 2 phía.

Sau khi xác định được interface cần lắng nghe, khởi động Wireshark và tiến hành lắng nghe trên interface đó.

Đồng thời, để lọc bỏ các gói tin từ bên ngoài (không phải gói tin tấn công), ta thực hiện một số filter trên phần mềm Wireshark để lọc bỏ các gói tin này. Một số filter thường sử dụng như sau:

ip: chỉ lọc các gói tin IP

ip.addr=={IP\_ADDRESS}: lọc các gói tin có nguồn hoặc đích là địa chỉ

IP\_ADDRESS

ip.src=={IP\_ADDRESS}: lọc các gói tin có nguồn là địa chỉ IP\_ADDRESS

ip.dst=={IP\_ADDRESS}: lọc các gói tin có đích là địa chỉ IP\_ADDRESS

tcp: lọc các gói tin TCP

udp: lọc các gói tin UDP

arp: lọc các gói tin ARP

Giữa các biểu thức filter là các toán tử "or" hoặc "and" để kết hợp nhiều biểu thức đơn lẻ thành logic tùy ý. Chi tiết xem thêm tài liệu của Wireshark.

- **Bước 2: Thực hiện tấn công**

Thực hiện tấn công giữa các host bất kì trong mạng. Thông thường sẽ là từ phân vùng Internal tấn công tới phân vùng DMZ (vì h1 và h2 trong DMZ khởi chạy nhiều dịch vụ như ssh, ftp, http). Tấn công cũng có thể thực hiện giữa các host với nhau trong phân vùng Internal. Thực tế, mọi khả năng đều có thể xảy ra khi kẻ tấn công có thể xâm nhập dần vào từng thành phần trong hệ thống.

Các cuộc tấn công phải đa dạng về các tham số. Ví dụ, cùng là tấn công dạng TCP SYN flood, có thể sử dụng công cụ hping3 với các tham số khác nhau:

# TCP SYN flood

```
mininet> h3 hping3 -V -d 100 -S -p 80 --flood h1
```

```
mininet> h4 hping3 -V -d 120 -S -p 22 --faster h1
```

```
mininet> h8 hping3 -V -d 200 -S -p 21 --flood h2
```

# TCP SYN flood với địa chỉ nguồn ngẫu nhiên

```
mininet> h3 hping3 -V -d 120 -S -p 80 --flood --rand-source h2
```

```
mininet> h4 hping3 -V -d 150 -S -p 22 --flood --rand-source h1
```

```
mininet> h3 hping3 -V -d 220 -S -p 80 --faster --rand-source h1
```

# LAND attack

```
mininet> h3 hping3 -V -d 110 -S -s 80 -p 80 -k --flood -a h1 h1
```

```
mininet> h3 hping3 -V -d 110 -S -s 1000 -p 80 -k --flood -a h2 h2
```

```
mininet> h3 hping3 -V -d 150 -S -s 80 -p 22 -k --flood -a h1 h1
```

- **Bước 3: Lưu lại lưu lượng bắt được**

Sau khi hoàn kết thúc tấn công, dùng bắt lưu lượng và lưu lại sử dụng tính năng File → Export Specified Packet.. Sử dụng tính năng này thay vì File → Save để chỉ lưu những gói tin đã được filter.

Lưu các file lưu lượng tấn công dưới tên theo dạng:

**{attack type}-{attack method}-{version}.pcap**

để dễ dàng quản lý và gán nhãn cho bước sau.

Trong đó:

- {attack type}: loại tấn công (dos, probe, mitm,..)
- {attack method}: hình thức tấn công (udp\_flood, tcp\_syn\_flood,..)
- {version}: các thông tin khác để đánh dấu và tránh trùng lặp

Ví dụ: dos-http\_flood-hulk\_1.pcap, dos-http\_slow-slowloris\_1.pcap, probe-udp\_scan-2.pcap, dos-tcp\_syn\_flood-land.pcap,...

#### 4.3.2.2. Thu thập dữ liệu bình thường

Việc thu thập dữ liệu bình thường cũng bao gồm các bước tương tự như khi thu thập dữ liệu tấn công, chỉ khác ở chỗ ta không thực hiện tấn công, mà sử dụng các dịch vụ, giao tiếp một cách bình thường giữa các host trong mạng. Quá trình thực hiện bao gồm các hành vi như sau, được ngẫu nhiên thực hiện với thời gian và trên các host khác nhau:

- Ping giữa các host trong mạng
- SSH tới h1, h2 từ các host khác
- Sử dụng dịch vụ ftp trên h1, h2 từ các host khác
- Sử dụng dịch vụ http trên h1, h2 từ các host khác
- VLC stream để stream video giữa các host bất kỳ trong mạng
- ...

#### 4.4. Trích xuất đặc trưng từ dữ liệu lưu lượng thô

Kết quả sau khi thu thập lưu lượng thô, ta thu được một tập các file lưu lượng tấn công và bình thường. Bước tiếp theo là trích xuất các đặc trưng từ dữ liệu thô đó. Phương pháp trích xuất ta sử dụng là trích xuất theo từng kết nối (connection). Nói cách khác, mỗi mẫu (sample) trong tập dữ liệu mà ta tạo ra, sẽ là một kết nối, giống với tập dữ liệu KDD99:

“Một kết nối là một chuỗi các gói TCP bắt đầu và kết thúc tại một số thời điểm được xác định, trong đó dữ liệu từ một địa chỉ IP nguồn đến một địa chỉ IP đích theo một giao thức xác định. Mỗi kết nối được gán nhãn là bình thường hoặc là một cuộc tấn công, với chi tiết tên một loại tấn công cụ thể.”

Mặc dù KDD99 là bộ dữ liệu nổi tiếng, tuy nhiên cách thức tạo ra bộ dữ liệu này từ dữ liệu lưu lượng thô (tcp dump) lại không được công bố rộng rãi. Để thực hiện công đoạn này, ta sử dụng một công cụ mã nguồn mở, cho phép trích xuất các đặc trưng tương tự như bộ dữ liệu KDD99, tuy nhiên không đầy đủ các đặc trưng như bộ dữ liệu gốc.

Một điểm hạn chế của công cụ này là chỉ hoạt động với các gói tin IPv4, thuộc 1 trong các loại ICMP, UDP, TCP và không hoạt động với lưu lượng được bắt từ interface ảo “any”. Cụ thể hơn, “any” chỉ là một interface ảo mà libcap sử dụng để cung cấp tính năng lắng nghe gói tin trên mọi network interface. Vì đặc thù khác nhau giữa các network interface về mặt giao thức ở tầng liên kết dữ liệu, nếu muốn bắt được tất cả các gói tin trên các interface khác nhau

đó, chúng buộc phải có cùng kiểu giao thức ở tầng liên kết dữ liệu. Các gói tin thu nhận được tại “any” là các gói tin đã được đóng gói dạng “cooked mode”. Có thể chỉ ra điểm khác nhau ở trường Encapsulation Type giữa các gói tin bắt được tại network interface thông thường (chẳng hạn như eth0,..) và “any”. Cũng chính vì lý do công cụ này không hoạt động với dữ liệu thu thập từ “any”, ta buộc phải thực hiện lắng nghe lưu lượng tại các network interface khác nhau nơi có lưu lượng đi qua, tùy vào từng tình huống, như trong phần trước đã đề cập. Đây là công cụ được tạo ra bởi nhóm sinh viên tại đại học Bergen, với mục đích mô phỏng quá trình trích xuất đặc trưng tương tự bộ dữ liệu KDD99. Có thể cài đặt như sau:

```
# Clone mã nguồn
$ git clone https://github.com/AI-IDS/kdd99\_feature\_extractor.git
# Tạo thư mục chứa file tạm khi build
$ mkdir build-files
# Biên dịch cache file
$ cd build-files
$ cmake -DCMAKE_BUILD_TYPE=Debug -G "CodeBlocks - Unix Makefiles" ..
# Biên dịch và build
$ cd ..
$ cmake --build ./build-files --target kdd99extractor -- -j 4
```

Chi tiết tại: [https://github.com/AI-IDS/kdd99\\_feature\\_extractor](https://github.com/AI-IDS/kdd99_feature_extractor)

Kiểm tra hoạt động của file thực thi vừa build được:

```
$ build-files/src/kdd99extractor -h
```





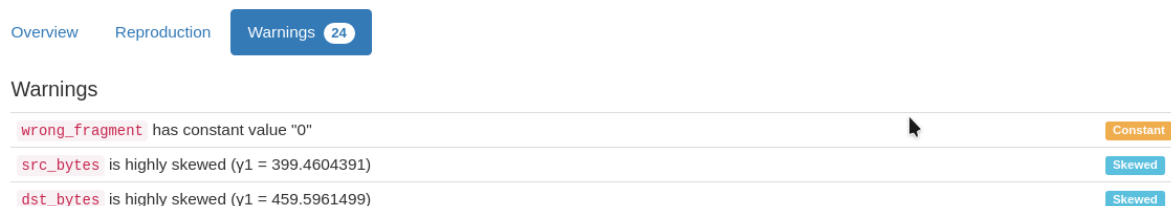
công nếu thuộc loại “tấn công”. Trong số các bản ghi thuộc vào lớp tấn công, kiểu tấn công DOS chiếm phần đa với 94% số bản ghi.

## V. Xây dựng mô hình phát hiện xâm nhập dựa trên dữ liệu tự thu thập

### 5.1. Xử lý và phân tích dữ liệu

Bộ dữ liệu tự tạo và bộ dữ liệu KDD99 có nhiều điểm tương đồng, và cũng được xử lý theo cách tương tự nhau. Sau khi loại bỏ các bản ghi giống hệt nhau, dữ liệu thu được bao gồm 1306829 bản ghi, mỗi bản ghi được biểu diễn bởi 28 đặc trưng.

Khi phân tích các thuộc tính thống kê của dữ liệu sử dụng thư viện pandas profiling, thông tin ta nhận được là đặc trưng `wrong_fragment` luôn là hằng số 0. Vì vậy, thuộc tính này không làm ảnh hưởng tới kết quả phân loại, ngược lại, nếu vẫn sử dụng nó có thể xem là một loại nhiễu gây ảnh hưởng tới khả năng học của mô hình. Vì vậy, ta loại bỏ `wrong_fragment` khỏi danh sách các đặc trưng sử dụng.



The screenshot shows the 'Warnings' section of the pandas profiling interface. It lists three warnings: 'wrong\_fragment' has a constant value of 0, 'src\_bytes' is highly skewed (gamma = 399.4604391), and 'dst\_bytes' is highly skewed (gamma = 459.5961499). Each warning has a corresponding status label: 'Constant', 'Skewed', and 'Skewed'.

Overview	Reproduction	Warnings 24
Warnings		
wrong_fragment	has constant value "0"	Constant
src_bytes	is highly skewed (γ1 = 399.4604391)	Skewed
dst_bytes	is highly skewed (γ1 = 459.5961499)	Skewed

Hình 36. Cảnh báo khi sử dụng pandas profiling

Tỉ lệ số bản ghi thuộc mỗi hình thức tấn công như sau:

```
[ ] 1 df['label'].value_counts()/df.shape[0]
```

normal.	0.641653
tcp_syn_flood	0.162572
udp_flood	0.080793
tcp_rst_flood	0.055403
tcp_fin_flood	0.021499
http_flood	0.018588
other	0.004868
tcp_null_scan	0.004843
tcp_connection_scan	0.004786
udp_scan	0.003844
icmp_ping_scan	0.000944
guest_passwd	0.000136
http_slow	0.000067
ping_of_death	0.000002
icmp_flood	0.000002

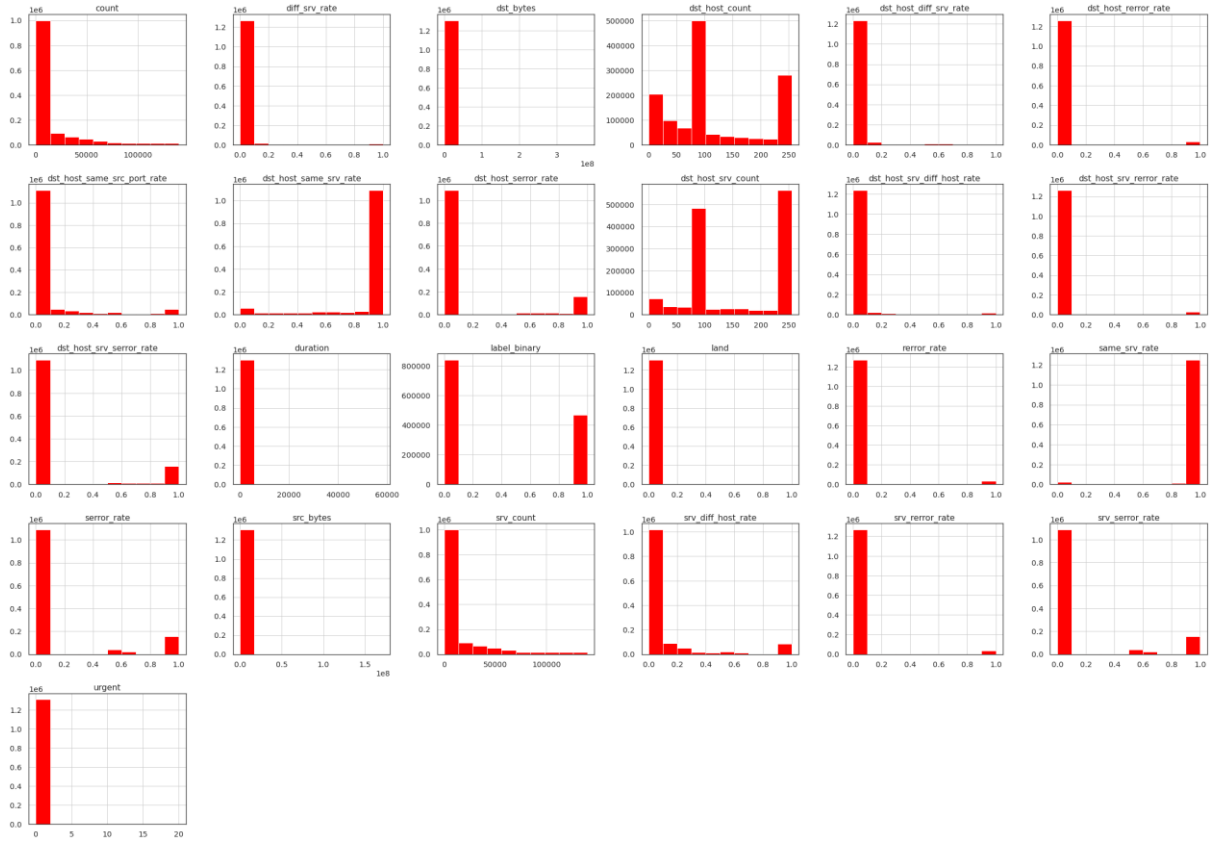
Name: label, dtype: float64

Hình 37. Các hình thức tấn công và tỉ lệ số lượng mẫu tương ứng

Số lượng bản ghi thuộc lớp “bình thường” chiếm khoảng 64% tổng số lượng bản ghi. Trong các bản ghi thuộc lớp tấn công, các dạng tấn công DOS chiếm đa số (94% như đã nêu ở phần

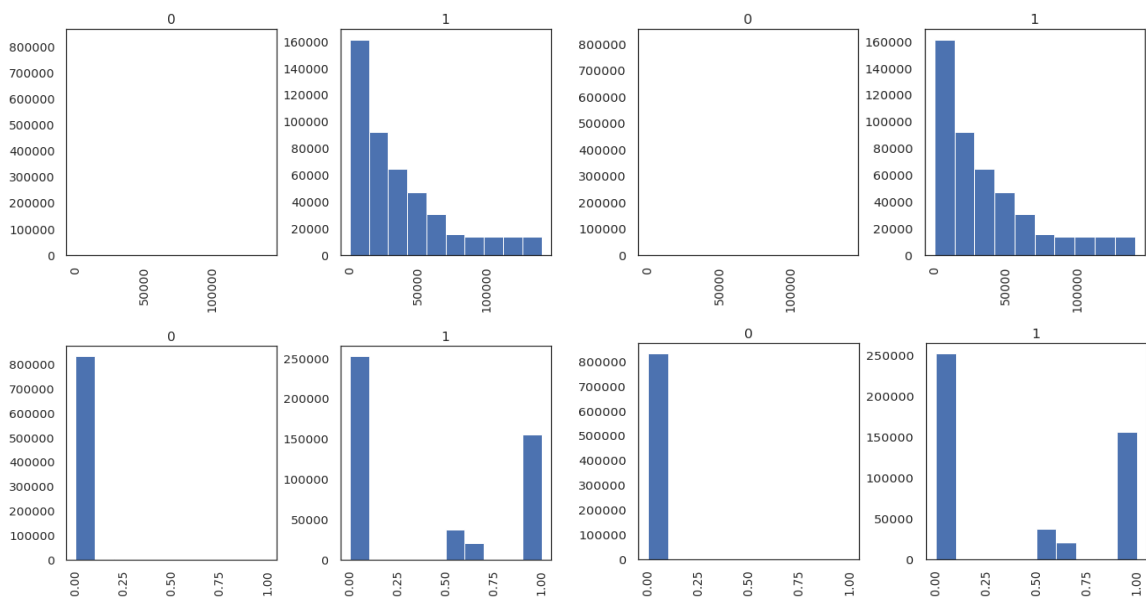
trước). Dữ liệu ở trạng thái cân bằng nếu xét tới bài toán phân loại nhị phân với 2 đầu ra là “bình thường” và “tấn công”.

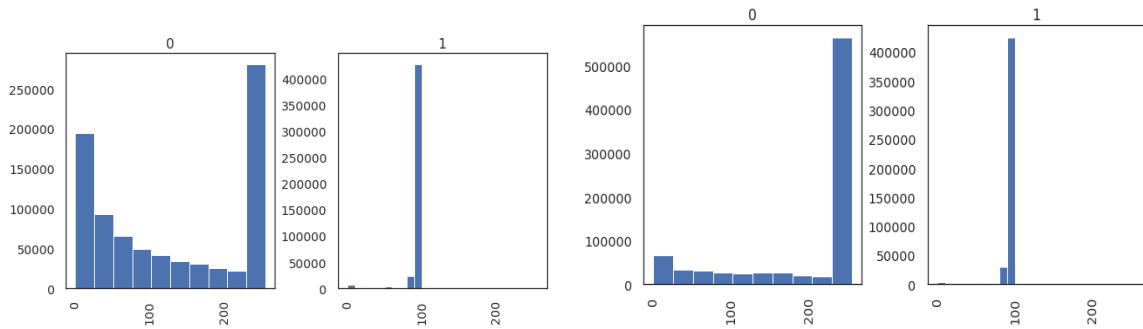
Trực quan hóa phân bố của các đặc trưng trong tập dữ liệu sử dụng biểu đồ histogram trên từng thuộc tính:



Hình 38. Trực quan hóa phân phối của từng đặc trưng sử dụng biểu đồ histogram

Tương tự như với bộ dữ liệu KDD99, ta cũng trực quan hóa phân phối của từng đặc trưng với mỗi nhãn:





Hình 39. Từ trái qua phải, trên xuống dưới: *count*, *srv\_count*, *error\_rate*, *srv\_error\_rate*, *dst\_host\_count*, *dst\_host\_srv\_count*

Dữ liệu cũng được chia thành 3 bộ train set, validation set và test set với tỉ lệ 50%-25%-25%. Các thuộc tính dạng categorical cũng được encode dưới dạng one-hot vector. Sau đó, toàn bộ vector đặc trưng được chuẩn hóa theo chuẩn z-score sử dụng StandardScaler() trong thư viện Scikit-learn. Kết quả cuối cùng, ta thu được vector đặc trưng có 85 chiều.

## 5.2. Xây dựng mô hình và đánh giá kết quả

Quá trình training sử dụng các mô hình và cách chọn/vét cạn tham số tốt nhất sử dụng GridSearchCV giống như khi thực hiện trên bộ dữ liệu KDD99.

Không nêu lại chi tiết quá trình training, bảng dưới đây tổng kết lại kết quả đánh giá các mô hình huấn luyện được, trên tập dữ liệu test.

	<b>Logistic Regressi on</b>	<b>Multi- layer Perceptr on</b>	<b>Decision Tree</b>	<b>Random Forest</b>	<b>SVM (linear kernel)</b>	<b>Naive Bayes</b>
<b>TN</b>	208415	209413	209608	209641	209237	209015
<b>FP</b>	1233	235	40	7	411	633
<b>FN</b>	1693	105	30	30	534	1775
<b>TP</b>	115367	116955	117030	117030	116526	115285
<b>Accura cy</b>	0.99	1.0	1.0	1.0	1.0	0.99
<b>Precisi on</b>	0.99	1.0	1.0	1.0	1.0	0.99
<b>Recall</b>	0.99	1.0	1.0	1.0	1.0	0.98

F1-score	0.99	1.0	1.0	1.0	1.0	0.99
<b>Chi tiết (với các tham số tốt nhất)</b>		2 hidden layers với 128 neuron	'criterion': 'gini', 'max_depth': 31, 'min_samples_leaf': 5	'criterion': 'entropy', 'max_depth': 21, 'min_samples_leaf': 5, 'n_estimators': 55	'C': 10, 'loss': 'squared_hinge'	Gaussian likelihood

*Bảng 10. Kết quả đánh giá trên test set của các mô hình đã huấn luyện*

Không bất ngờ khi mô hình cây quyết định (Decision Tree) có hiệu quả rất tốt, cho kết quả thứ 2 sau Random Forest. Với các bài toán mà dữ liệu train và test không khác nhau nhiều về phân phối, đồng thời dữ liệu có thể được mô hình hóa dạng chuỗi các điều kiện if-else, cây quyết định là lựa chọn tốt. Ở đây, chiều sâu cây tốt nhất tìm được là 31 (rất sâu) mà không xảy ra tình trạng overfit, cho thấy rằng phân phối giữa dữ liệu test và dữ liệu training là rất tương đồng nhau. Điều này cũng dễ hiểu là do dữ liệu thu thập được khá đơn giản và tính đa dạng chưa cao. Mô hình Random Forest là một mô hình ensemble learning dựa trên kết quả dự đoán của nhiều cây quyết định, tất nhiên, nó cũng sẽ cho kết quả tốt hơn 1 cây quyết định, và đúng đầu về kết quả trên tập dữ liệu test.

Các mô hình đơn giản như Naive Bayes, Logistic Regression cũng không tỏ ra hiệu quả rõ rệt, khi số lượng FN là khá cao (1775 và 1693). Mô hình phức tạp hơn Logistic Regression là MLP cho kết quả cải thiện hơn rõ rệt, khi số lượng FN chỉ còn lại 105 và FP chỉ là 235. Mô hình SVM giả định trước rằng dữ liệu giữa 2 class là linear separable, và trong trường hợp này cũng không tỏ ra hiệu quả hơn các mô hình khác.

## Tài liệu tham khảo

- [1] ‘A Detailed Analysis of the KDD CUP 99 Data Set’ - Mahbod Tavallaei, Ebrahim Bagheri, Wei Lu, and Ali A. Ghorbani
- [2] ‘A framework for constructing features and models for intrusion detection systems’, Lee, W. & Stolfo, S. J. (2000), Information and System Security 3 (4) , 227-261.
- [3] ‘A Survey Intrusion Detection with KDD99 Cup Dataset’, Dybey, D. & Dubey, J. (2014), International Journal of Computer Science and Information Technology Research 2 (3), 146-157.
- [4] ‘Relevant Feature Selection Model Using Data Mining for Intrusion Detection System’, Ayman Ismail Madbouly, Amr M. Gody, Tamer Barakat
- [5] ‘Detection of Probe Attacks Using Machine Learning Techniques’, Ch.Ambedkar, V. Kishore Babu
- [6] ‘A review of KDD99 dataset usage in intrusion detection and machine learning between 2010 and 2015’ , Atilla Özgür, Hamit Erdem
- [7] ‘Statistical Analysis Driven Optimized Deep Learning System for Intrusion Detection’, Cosimo Ieracitano, Mandar Gogate, Kia Dashtipour, Francesco Carlo Morabito
- [8] Bài giảng An ninh mạng – ThS. Bùi Trọng Tùng, HUST
- [9] A Survey of Machine Learning Techniques Applied to Software Defined Networking (SDN): Research Issues and Challenges, Junfeng Xie , F. Richard Yu , Tao Huang, Renchao Xie, Jiang Liu, Chenmeng Wang, Yunjie Liu
- [10] Mininet Documentation <https://github.com/mininet/mininet>
- [11] ONOS Documentation <https://wiki.onosproject.org/display/ONOS/ONOS+Documentation>
- [12] KDD main page <http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html>