BAN CƠ YẾU CHÍNH PHỦ

HỌC VIỆN KỸ THUẬT MẬT MÃ

--------📖--------

BÁO CÁO CHUYÊN ĐỀ KỸ NGHỆ

ĐỀ TÀI:
NGHIÊN CỨU ỨNG DỤNG HỌC SÂU TĂNG CƯỜNG TRONG KIỂM THỬ ỨNG DỤNG WEB

Nhóm sinh viên thực hiện – Nhóm 6:

Tạ Xuân Cường – AT170107

Đỗ Công Minh – AT170634

Giảng viên hướng dẫn:

TS. Nguyễn Mạnh Thắng

Khoa An toàn thông tin - Học viện Kỹ Thuật Mật Mã

Hà Nội, 2023

MỤC LỤC

DANH MỤC HÌNH ẢNH

LỜI NỚI ĐẦU

Trong thời đại số ngày nay, việc bảo vệ thông tin trực tuyến trở thành một thách thức ngày càng lớn khi mà các trang web và ứng dụng web trở thành mục tiêu chính của những kẻ tấn công. Bảo mật web đóng vai trò quan trọng trong việc duy trì tính toàn vẹn và tin cậy của hệ thống thông tin, đặt ra nhiều thách thức và cơ hội mới đối với các chuyên gia bảo mật.

Trong bối cảnh này, nghiên cứu về ứng dụng học sâu tăng cường trong kiểm thử bảo mật web là một bước tiến quan trọng để đối mặt với những thách thức ngày càng phức tạp của môi trường trực tuyến. Học sâu, đặc biệt là các mô hình máy học sâu, đã chứng minh khả năng xuất sắc trong việc phân loại và nhận diện các mô hình tấn công phức tạp.

Đồng thời, kiểm thử bảo mật web, mặc dù đã được thực hiện phổ biến, đang đối mặt với thách thức của việc ngày càng tinh tế và phức tạp hóa các kỹ thuật tấn công. Vì vậy, việc tích hợp học sâu vào quy trình kiểm thử có thể cung cấp một cách tiếp cận hiệu quả hơn để phát hiện và ngăn chặn các mối đe dọa mới.

Đề tài nghiên cứu này tập trung vào khám phá ứng dụng của học sâu tăng cường trong việc cải thiện khả năng kiểm thử bảo mật web, mang lại sự đổi mới và đóng góp cho lĩnh vực ngày càng quan trọng này. Qua đó, chúng ta có thể không chỉ nâng cao chất lượng của quá trình kiểm thử mà còn đảm bảo tính an toàn và bảo mật của thông tin trực tuyến trong môi trường ngày nay.

CHƯƠNG 1 TỔNG QUAN VỀ KIỂM THỬ ỨNG DỤNG WEB VÀ HỌC SÂU TĂNG CƯỜNG

1.1 Tổng quan về học sâu tăng cường

1.1.1 Tổng quan về học máy và học sâu

1.1.1.1 Giới thiệu về học máy

Học máy (Machine Learning – ML) là một lĩnh vực của trí tuệ nhân tạo liên quan đến việc nghiên cứu và xây dựng các kĩ thuật cho phép các hệ thống "học" tự động từ dữ liệu để giải quyết những vấn đề cụ thể. Các thuật toán học máy xây dựng một mô hình dựa trên dữ liệu mẫu, được gọi là *dữ liệu huấn luyện*, để đưa ra dự đoán hoặc quyết định mà không cần được lập trình chi tiết về việc đưa ra dự đoán hoặc quyết định này.

Có bốn loại ML chủ yếu:

- **Supervised Learning (Học có giám sát):** quá trình huấn luyện sẽ sử dụng dữ liệu có nhãn (labeled data), máy sẽ "học" cách phân loại dữ liệu dựa vào các nhãn này.
- **Unsupervised Learning (Học không giám sát):** quá trình huấn luyện sẽ sử dụng dữ liệu không có nhãn (unlabeled data), khi đó máy sẽ tự tìm ra các đặc trưng (pattern) của các nhóm (cluster) của dữ liệu và phân loại dữ liệu vào các nhóm thích hợp.
- **Semi-supervised Learning (Học bán giám sát):** quá trình huấn luyện sử dụng cả dữ liệu có nhãn và dữ liệu không có nhãn. Một lượng nhỏ dữ liệu có nhãn sẽ được sử dụng giúp định hướng trong khi một lượng rất lớn dữ liệu không có nhãn sẽ được dùng để hoàn thành mô hình.
- **Reinforcement Learning (Học tăng cường):** máy sẽ học thông qua hình thức nhận phạt hoặc thưởng sau một loạt các quyết định của mình.

Một số kỹ thuật ML:

- **Classification (Phân loại):** là một quá trình phân loại dữ liệu hoặc đối tượng thành các lớp hoặc danh mục được xác định trước dựa trên các tính năng hoặc thuộc tính của chúng. Trong học máy, phân loại là một loại kỹ thuật học có giám sát trong đó thuật toán được huấn luyện trên tập dữ liệu được gắn nhãn để dự đoán lớp hoặc danh mục dữ liệu mới.

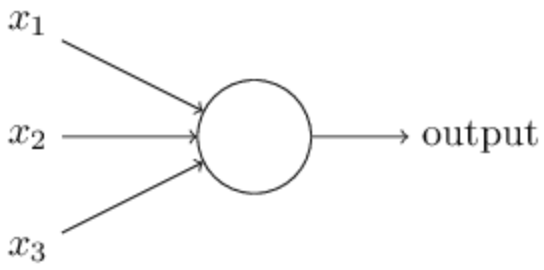
- **Regression (Hồi quy):** Hồi quy trong học máy bao gồm các phương pháp toán học cho phép dự đoán kết quả liên tục (y) dựa trên giá trị của một hoặc nhiều biến dự đoán (x). Hồi quy tuyến tính (Linear Regression) có lẽ là hình thức phân tích hồi quy phổ biến nhất vì tính dễ sử dụng của nó.
- **Recommender systems (Hệ thống gợi ý):** Hệ thống gợi ý là các thuật toán cung cấp các đề xuất được cá nhân hóa phù hợp nhất với từng người dùng. Với sự phát triển mạnh mẽ của các nội dung trực tuyến, người dùng đứng trước vô số lựa chọn. Do đó, điều quan trọng đối với các nền tảng web là đưa ra đề xuất về các mặt hàng cho từng người dùng, nhằm tăng sự hài lòng và mức độ tương tác của người dùng.
- **Clustering (Phân cụm):** Phân cụm có nhiệm vụ chia tổng thể hoặc các điểm dữ liệu thành một hoặc một số nhóm sao cho các điểm dữ liệu trong cùng một nhóm có các đặc tính giống nhau và không giống với các điểm dữ liệu trong các nhóm khác.

1.1.1.2 Giới thiệu về học sâu

1.1.1.2.1 Perceptron

Perceptron là một loại tế bào thần kinh nhân tạo. Perceptron được phát triển vào những năm 1950 và 1960 bởi nhà khoa học Frank Rosenblatt, lấy cảm hứng từ công trình trước đó của Warren McCulloch và Walter Pitts.

Một perceptron có một hoặc nhiều đầu vào nhị phân và tạo ra một đầu ra nhị phân duy nhất:



Trong hình trên, perceptron có ba đầu vào x_1, x_2, x_3. Rosenblatt đề xuất một quy tắc đơn giản để tính kết quả đầu ra. Ông đưa ra khái niệm trọng số w là các số thực thể hiện tầm quan trọng của đầu vào tương ứng đối với đầu ra. Đầu ra của neuron có giá trị 0 hoặc 1, được xác định bằng cách xem xét liệu tổng trọng số $\Sigma_{i=1}^{n}(w_i x_i)$ nhỏ hơn hay lớn hơn "ngưỡng" (threshold). Giống như các trọng số, threshold là một số thực và là tham số của neuron. Công thức đại số tổng quát:

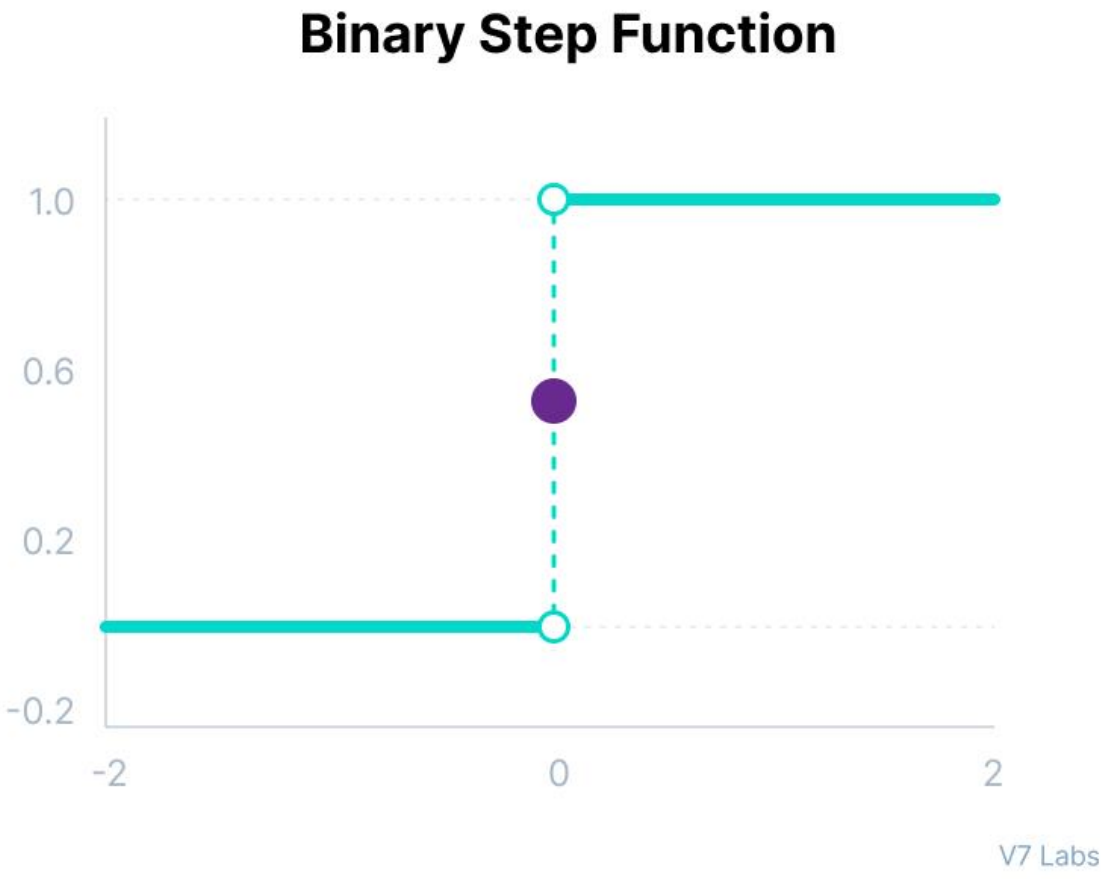
$$output = \begin{cases} 0 \; if \; \Sigma_{i=1}^{n}(w_i x_i) \leq threshold \\ 1 \; if \; \Sigma_{i=1}^{n}(w_i x_i) > threshold \end{cases}$$

Đơn giản hóa công thức trên bằng cách chuyển $threshold$ sang vế bên kia của bất đẳng thức và thay thế nó bằng độ lệch (bias) của perceptron: $b = -threshold$. Công thức trở thành:

$$output = \begin{cases} 0 \; if \; w \cdot x + b \leq 0 \\ 1 \; if \; w \cdot x + b > 0 \end{cases}$$

Trong đó, *w* là ma trận trọng số và *x* là ma trận đầu vào.

Việc dựa vào một ngưỡng để quyết định như trên được gọi là binary step function



Hình 1. 1 Binary Step Funtion

Tuy nhiên hàm này có một số hạn chế. Nó không thể cung cấp một đầu ra đa giá trị và độ dốc của hàm là bằng không, gây trở ngại trong quá trình backpropagation.

1.1.1.2.2 Sigmoid neurons

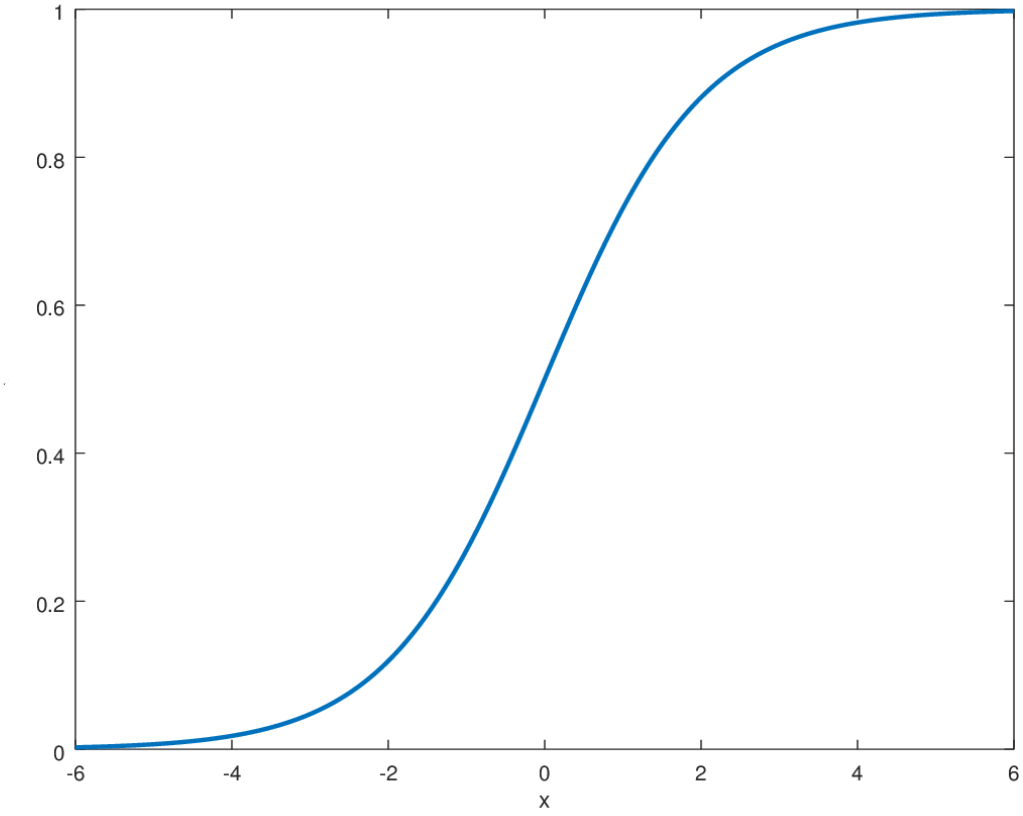
Cũng giống như perceptron, sigmoid neuron cũng có các giá trị đầu vào, trọng số và bias. Tuy nhiên, đầu vào của sigmoid neuron có thể nhận bất kỳ giá trị nào trong khoảng từ 0 đến 1 và đầu ra không bắt buộc phải là 0 hoặc 1 như perceptron. Thay vào đó, nó là $\sigma(w \cdot x + b)$, trong đó σ được gọi là hàm sigmoid và được xác định bởi:

$$\sigma = \frac{1}{1 + e^{-z}}$$

hay

$$\sigma = \frac{1}{1 + exp\left(-\Sigma_j w_j x_j - b\right)}$$

Biểu diễn hình học của hàm sigmoid như sau:



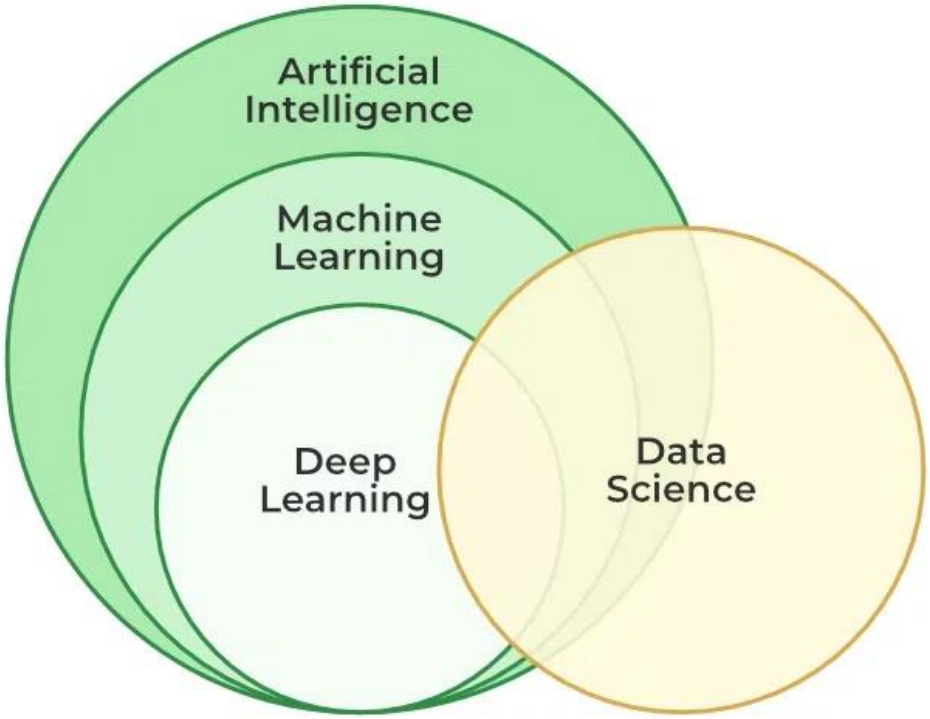
Hình 1. 2 Biểu diễn hình học của hàm sigmoid

Có thể thấy hàm sigmoid là một hàm phi tuyến tính, nó chuyển đổi giá trị đầu vào bất kỳ thành một giá trị trong khoảng (0, 1). Chúng cho phép backpropagation vì giờ đây đạo hàm sẽ liên quan đến đầu vào biết được trọng số nào trong các neuron đầu vào có thể cung cấp dự đoán tốt hơn. Những hàm phi tuyến tính này được gọi là hàm kích hoạt (activation function).

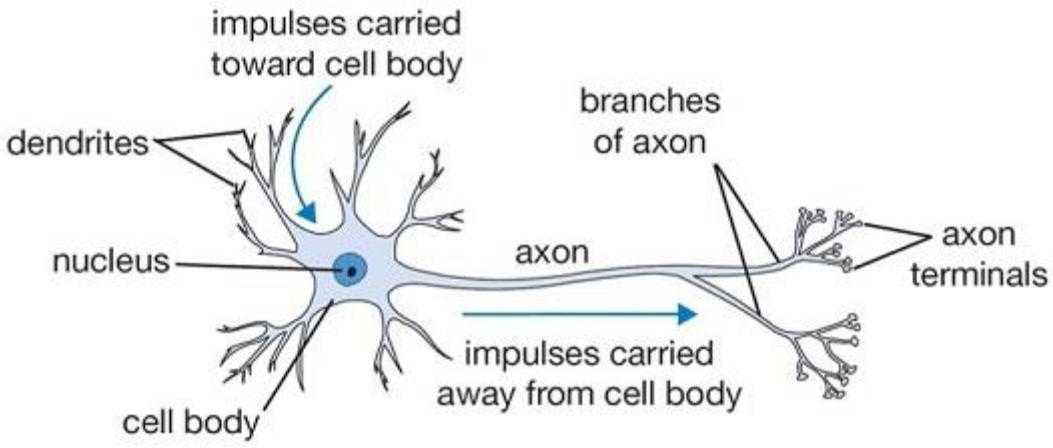
1.1.1.2.3 Mạng thần kinh nhân tạo (Artificial Neural Network – ANN)

Học sâu (Deep Learning – DL) là một kỹ thuật học máy tiên tiến dựa trên mô hình hóa và giải quyết các vấn đề phức tạp thông qua việc sử dụng các mạng neuron có nhiều tầng (deep neural networks). Cách tiếp cận này cho phép máy móc tự động tìm hiểu sâu các đặc trưng từ dữ liệu. Do đó, mô hình DL có thể "học" từ những loại dữ liệu phức tạp như hình ảnh, văn bản, âm thanh và các loại dữ liệu phức tạp khác để đưa ra dự đoán chính xác hơn.

DL là một nhánh của ML với nền tảng là mô hình mạng lưới thần kinh nhân tạo (Artificial Neural Network – ANN).

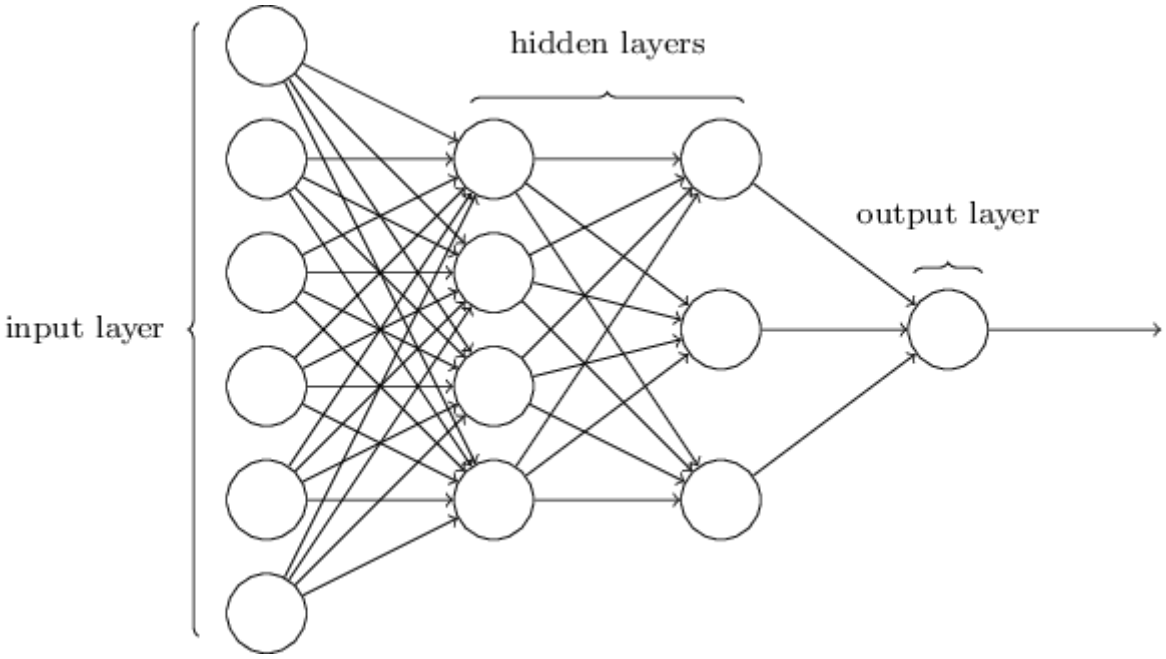


ANN được lấy ý tưởng từ cách mà não người xử lý thông tin. Các neuron sinh học của chúng ta tiếp nhận thông tin từ các sợi nhánh (Dendrite) và đưa vào nhân (Nucleus). Nhân neuron sẽ có trách nhiệm xử lý các thông tin và quyết định xem sẽ gửi tín hiệu gì tiếp theo đến các neuron khác thông qua sợi trục (Axon).



Hình 1. 3 Mô hình tế bào thần kinh sinh học

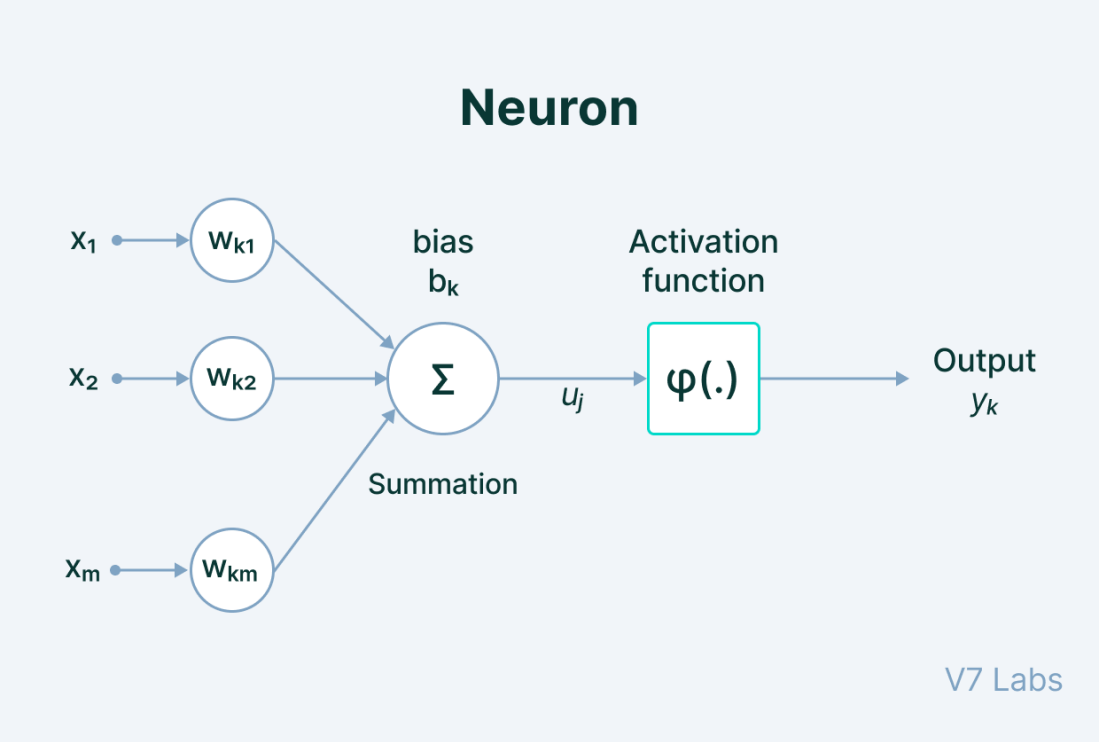
Kiến trúc của một ANN được thiết kế dựa trên mạng lưới thần kinh sinh học, bao gồm ba lớp chính, đó là lớp đầu vào (input layer), lớp ẩn (hidden layer) và lớp đầu ra (output layer).



Hình 1. 4: Mô hình ANN

Neuron nhân tạo

Bên trong các lớp của ANN chứa các neuron nhân tạo được xây dựng luồng xử lý thông tin dựa trên neuron sinh học. Về cơ bản, các neuron trong ANN giống với sigmoid neuron, chỉ khác đầu ra của chúng không chỉ sử dụng hàm sigmoid mà còn có thể sử dụng những hàm phi tuyến khác để phù hợp với từng bài toán cụ thể.



Hình 1. 5 Neurons nhân tạo

Các neuron nhận giá trị từ lớp đầu vào hoặc từ các neuron khác trong mạng. Sau đó, mỗi neuron nhân từng giá trị này với vector trọng số của nó, tính tổng các tích này và cộng với hệ số bias, đưa vào hàm kích hoạt $\boldsymbol{f(z)}$ (hay còn gọi là hàm truyền), trả về giá trị đầu ra $\boldsymbol{a}$. Nếu $\boldsymbol{a}$ vượt quá một ngưỡng nhất định, nó sẽ "kích hoạt" neuron, truyền dữ liệu đến neuron của lớp tiếp theo trong mạng.Công thức tổng quát:

Lớp đầu vào (Input layer): Mỗi đơn vị của lớp đầu vào, theo thứ tự từ trên xuống dưới, truyền giá trị cho mỗi neuron của lớp ẩn đầu tiên.

Lớp ẩn (Hidden layer): Các neuron trong lớp ẩn nhận giá trị từ các đơn vị của lớp đầu vào, thực hiện quá trình tính toán.

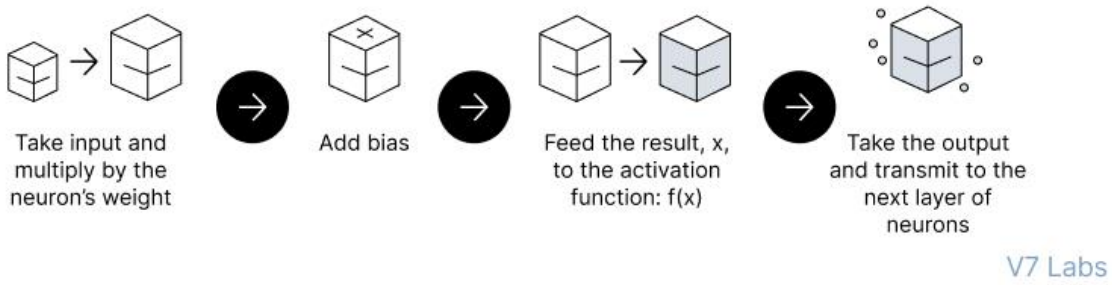
Lớp đầu ra (Output layer): Sau khi đi qua các lớp ẩn trong mạng, một lớp neuron cuối cùng sẽ tính toán và trả về giá trị đầu ra.

Lan truyền thông tin trong ANN

Feedforward Propagation

Feedforward Propagation - Sự lan truyền thông tin xảy ra theo hướng chuyển tiếp. Đầu vào được sử dụng để tính toán một số hàm trung gian trong lớp ẩn, sau đó được sử dụng để tính toán một đầu ra.

Trong quá trình feedforward propagation, hàm kích hoạt đóng vai trò là một "cổng" toán học nằm giữa đầu vào và đầu ra của neuron hiện tại.



Hình 1. 6 Feedfoward Propagation

Backpropagation

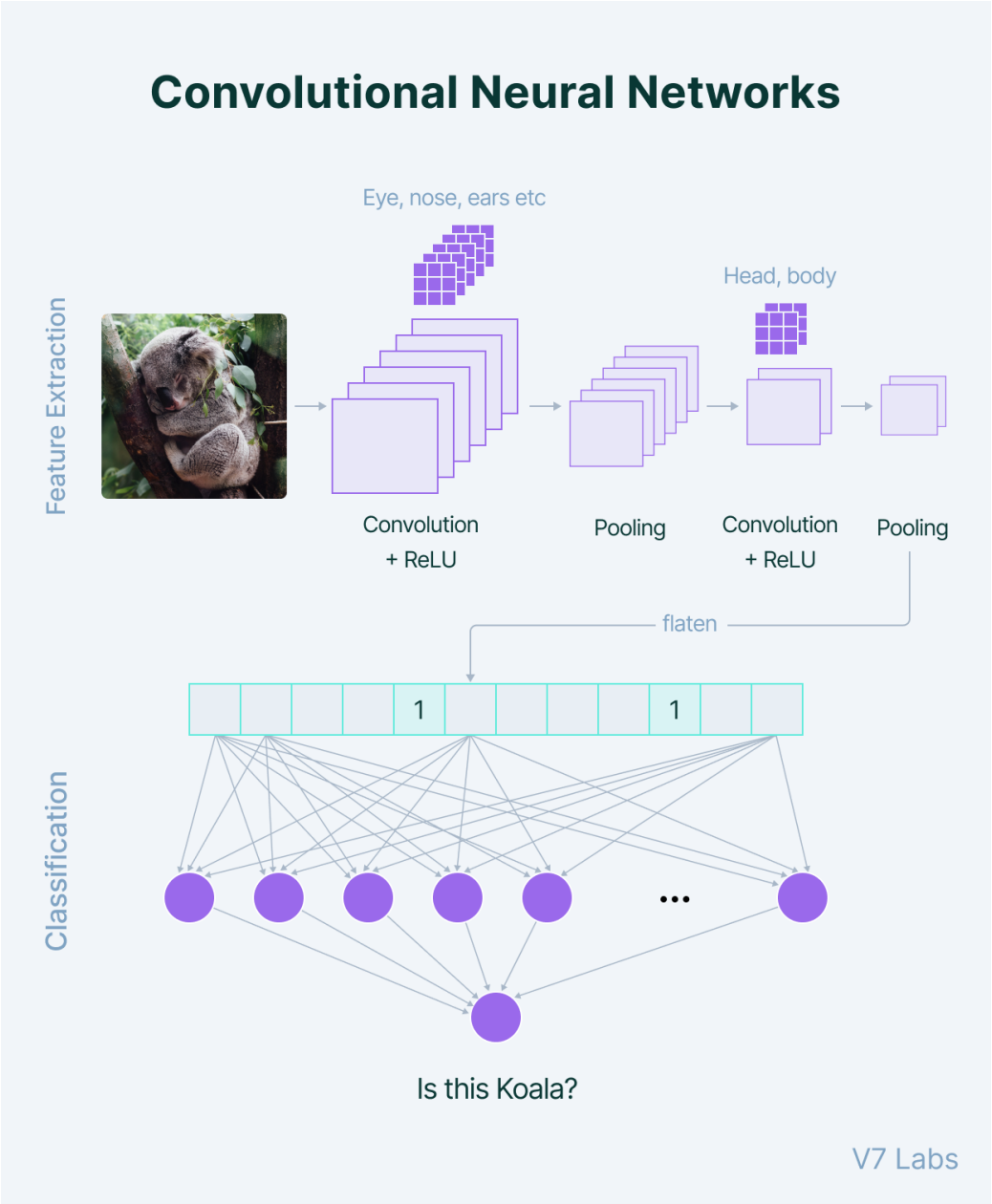
Trong backpropagation, trọng số của các kết nối trong mạng được điều chỉnh lặp đi lặp lại để giảm thiểu sự khác biệt giữa vector đầu ra thực tế của mạng và vector đầu ra mong muốn. Hiểu một cách đơn giản, backpropagation nhằm giảm thiểu hàm chi phí bằng cách điều chỉnh trọng số và độ lệch của mạng. Độ dốc của hàm chi phí quyết định mức điều chỉ đối với các tham số như hàm kích hoạt, trọng số, độ lệch, vv.

1.1.1.3 Các mô hình trong học sâu

1.1.1.3.1 Mạng neuron tích chập (Convolutional Neural Networks - CNN)

Mạng Neural tích chập, chủ yếu được sử dụng cho các nhiệm vụ liên quan đến thị giác máy tính hoặc xử lý hình ảnh.

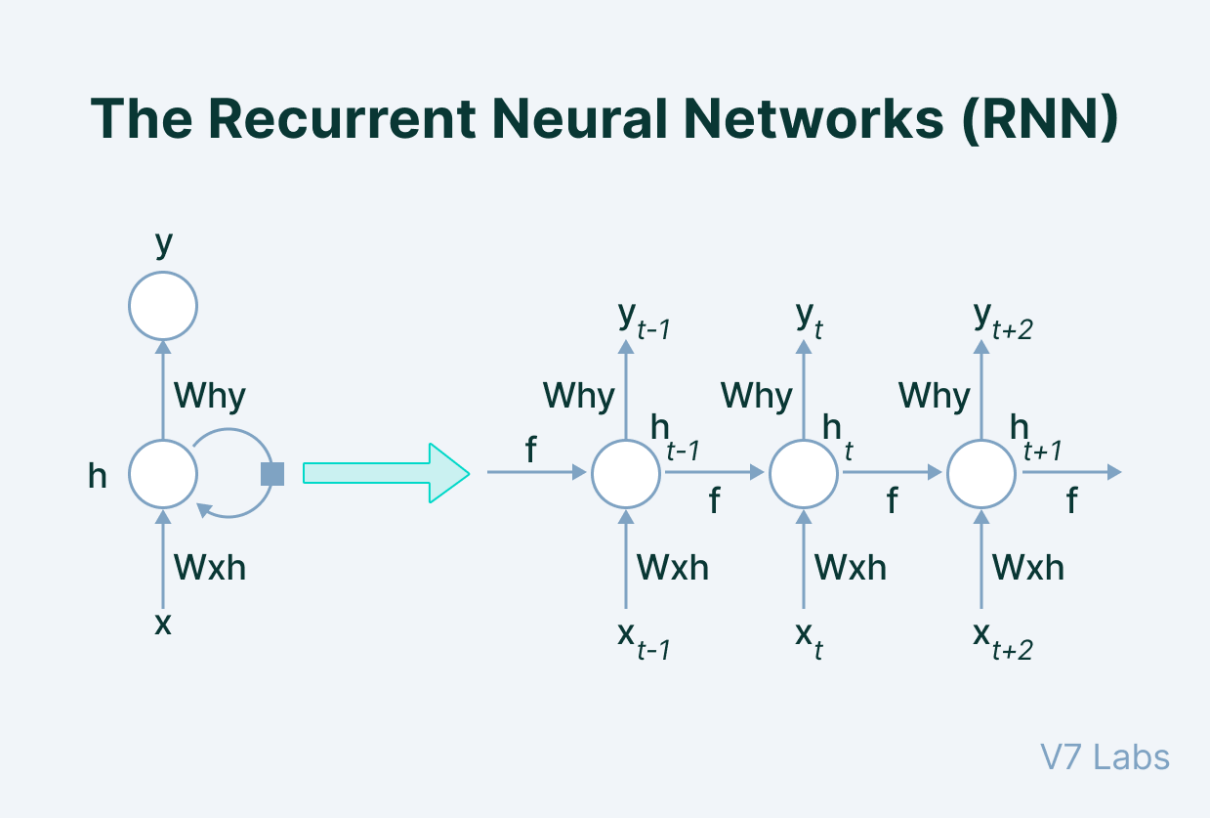
CNNs rất hiệu quả trong việc mô hình dữ liệu không gian như hình ảnh 2D hoặc 3D và video. Chúng có thể trích xuất đặc trưng và mẫu trong một hình ảnh, cho phép các nhiệm vụ như phân loại hình ảnh hoặc phát hiện đối tượng.



Hình 1. 7 Mô hình CNN

1.1.1.3.2 Mạng neural hồi quy (Recurrent Neural Networks - RNN)

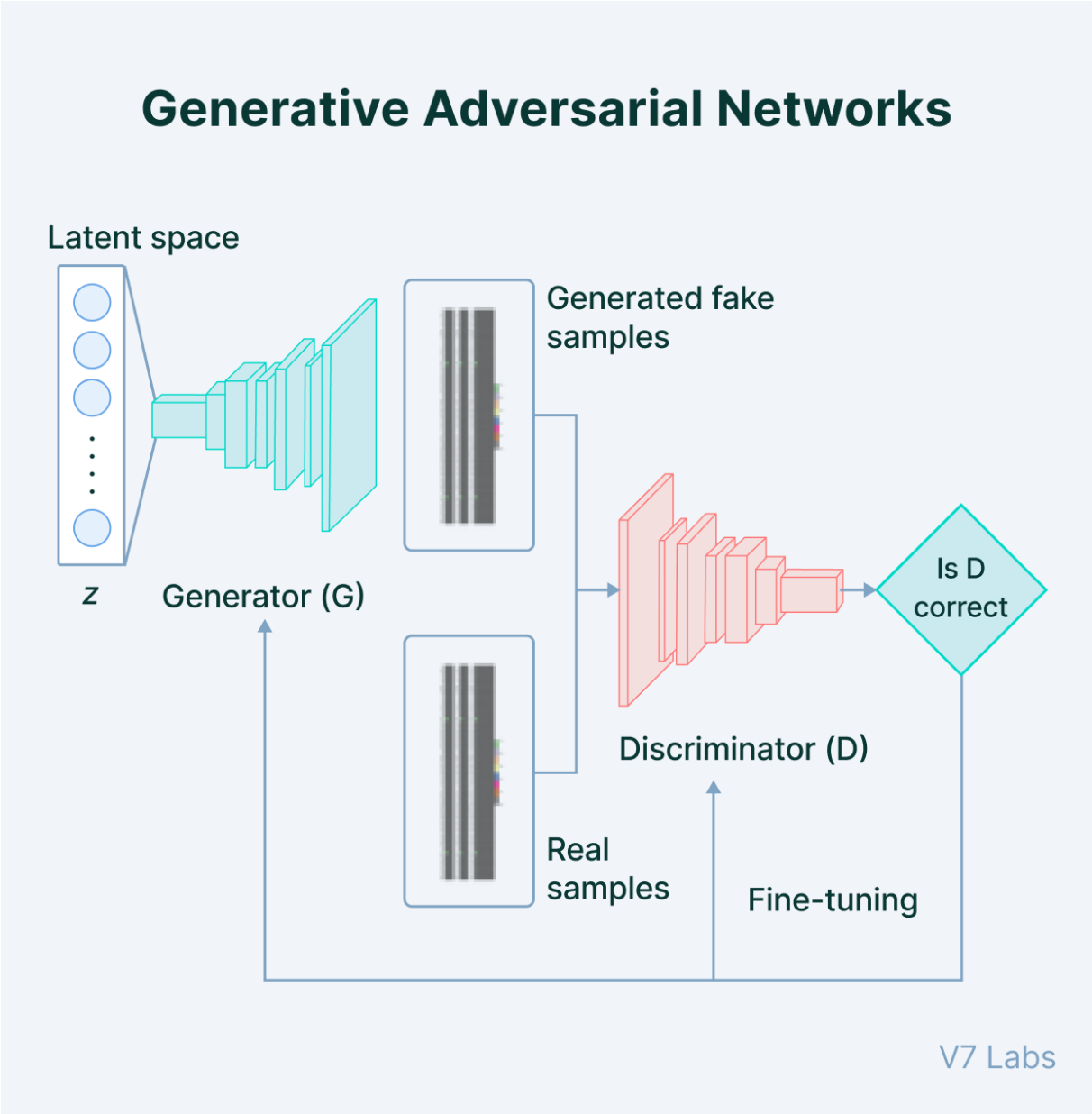
Mạng neural hồi quy, chủ yếu được sử dụng để mô hình dữ liệu tuần tự, như văn bản, âm thanh, hoặc bất kỳ loại dữ liệu nào biểu diễn chuỗi hoặc thời gian. Chúng thường được ứng dụng trong các nhiệm vụ liên quan đến xử lý ngôn ngữ tự nhiên (NLP).



Hình 1. 8 Mô hình RNN

1.1.1.3.3 Mạng tạo sinh đối kháng (Generative adversarial networks - GAN)

Mạng tạo sinh đối địch, hay GANs, là các khung công việc được sử dụng cho các nhiệm vụ liên quan đến học không giám sát. Loại mạng này về cơ bản học cấu trúc của dữ liệu và các mô hình mẫu một cách sao cho nó có thể được sử dụng để tạo ra các ví dụ mới, tương tự như tập dữ liệu gốc.



Hình 1. 9 Mô hình GAN

1.1.2 Tổng quan về học tăng cường

1.1.2.1 Giới thiệu về học tăng cường

Học tăng cường (Reinforcement Learning – RL) là một lĩnh vực con của ML, nghiên cứu cách thức một tác nhân (agent) trong một môi trường (environment) nên chọn thực hiện các hành động (action) nào để cực đại hóa một phần thưởng (reward) nào đó về lâu dài. Các thuật toán học tăng cường cố gắng tìm một chiến lược ánh xạ các trạng thái của môi trường tới các hành động mà agent nên chọn trong các trạng thái đó.

Agent (Tác nhân): Tác nhân (A) thực hiện các hành động ảnh hưởng đến môi trường.

Action (Hành động): Đây là tập hợp của tất cả các hành động mà tác nhân có thể thực hiện. Tác nhân quyết định hành động nào sẽ thực hiện từ một tập hợp các hành động rời rạc (a).

Environment (Môi trường): Tất cả các hành động mà tác nhân học tăng cường thực hiện trực tiếp ảnh hưởng đến môi trường.

State (Trạng thái): Một trạng thái (S) là một tình huống cụ thể mà tác nhân đang gặp phải.

Reward (R): Môi trường cung cấp phản hồi thông qua đó chúng ta xác định tính hợp lệ của các hành động của tác nhân trong mỗi trạng thái. Điều này rất quan trọng trong học tăng cường.

Discount factor (Hệ số giảm): Theo thời gian, hệ số giảm làm giảm ảnh hưởng của các phần thưởng trong tương lai đối với quyết định hiện tại. Hệ số giảm thường được biểu diễn bằng ký hiệu γ (gamma) và nằm trong khoảng từ 0 đến 1.

Policy (π): Nó quyết định hành động nào phải thực hiện trong một trạng thái cụ thể để tối đa hóa thưởng.

Value (V): mô tả giá trị của một trạng thái hoặc cặp trạng thái-hành động trong môi trường.

Q-Value: là thước đo phần thưởng mong đợi tổng thể nếu tác nhân (A) ở trạng thái (s) và thực hiện hành động (a), theo chính sách (π) nào đó.

1.1.2.2 Phân loại thuật toán học tăng cường

Model-based algorithms (Thuật toán dựa trên mô hình):

- Thuật toán dựa trên mô hình sử dụng hàm chuyển đổi và phần thưởng để ước tính chính sách tối ưu.
- Chúng được áp dụng trong các tình huống mà chúng ta hiểu đầy đủ về môi trường và cách nó phản ứng với các hành động khác nhau.
- Trong học tăng cường dựa trên mô hình, tác nhân có khả năng truy cập vào mô hình của môi trường. Điều này bao gồm thông tin về hành động cần thực hiện để chuyển từ một trạng thái này sang trạng thái khác, xác suất đi kèm và phần thưởng tương ứng.
- Chúng cho phép tác nhân học tăng cường lập kế hoạch trước bằng cách suy nghĩ về tương lai.

- Trong môi trường ổn định, học tăng cường dựa trên mô hình là phương pháp phù hợp hơn.

Model-free algorithms (Thuật toán không dựa trên mô hình):

- Thuật toán không dựa trên mô hình tìm kiếm chính sách tối ưu với kiến thức rất hạn chế về động lực của môi trường. Chúng không sử dụng bất kỳ hàm chuyển đổi/thưởng nào để đánh giá chính sách tốt nhất.
- Chúng ước tính chính sách tối ưu trực tiếp từ trải nghiệm, tức là sự tương tác giữa tác nhân và môi trường mà không có thông tin trước về hàm thưởng.
- Học tăng cường không dựa trên mô hình nên được áp dụng trong các tình huống liên quan đến thông tin không đầy đủ về môi trường.
- Trong thế giới thực, chúng ta không có một môi trường cố định. Các xe tự lái hoạt động trong môi trường động với các điều kiện giao thông thay đổi, đường đi thay đổi, vv. Trong các tình huống như vậy, các thuật toán không dựa trên mô hình thường hiệu quả hơn so với các kỹ thuật khác.

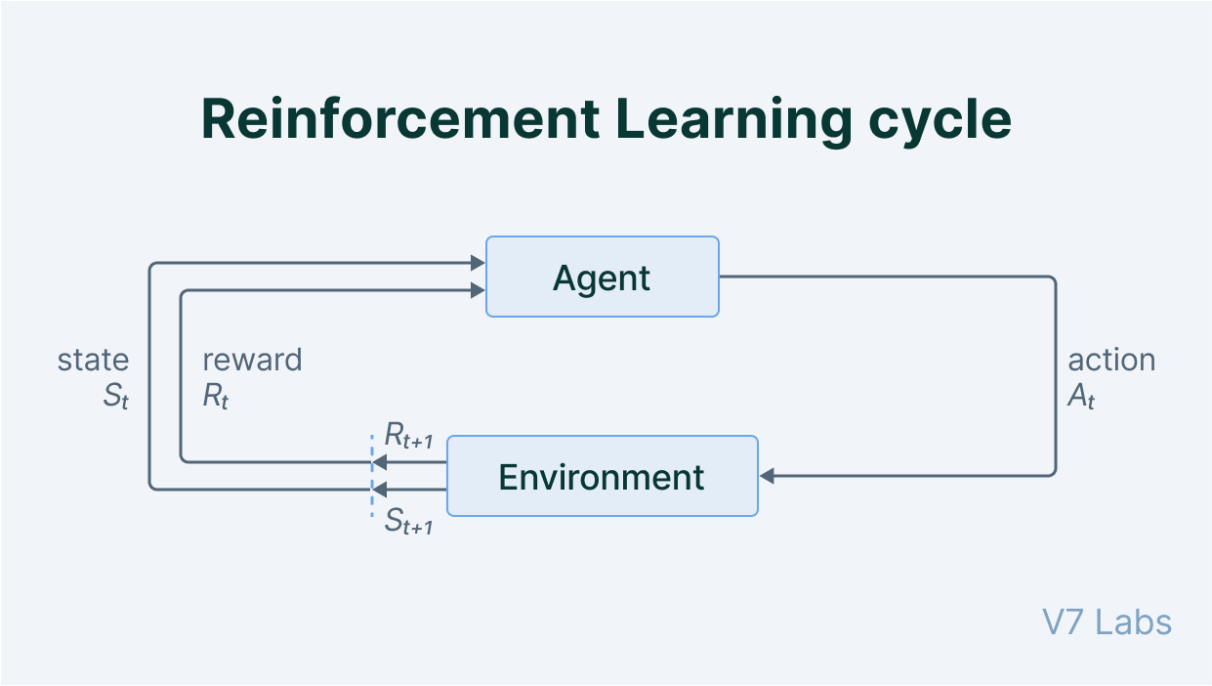
1.2.2.3 Thuật toán phổ biến

Quá Trình Quyết Định Markov (MDP)

Quá trình Quyết định Markov (MDP) là một thuật toán trong học tăng cường, cung cấp một hình thức tiếp cận cho quyết định tuần tự.

Các thành phần chính của MDP bao gồm một tác nhân tương tác với môi trường. Các tương tác này diễn ra theo thời gian dưới dạng chuỗi. Tại mỗi bước thời gian, tác nhân nhận được biểu diễn của trạng thái môi trường và dựa trên nó để chọn hành động. Môi trường chuyển đến trạng thái mới, và tác nhân nhận được một phần thưởng là kết quả của hành động trước đó.

Tóm lại, quá trình lựa chọn hành động, chuyển đến trạng thái mới và nhận phần thưởng xảy ra theo thứ tự và lặp lại. Điều này tạo ra một chuỗi gọi là quỹ đạo, biểu thị trạng thái, hành động và phần thưởng. Trong suốt quá trình này, trách nhiệm của tác nhân là tối đa hóa tổng phần thưởng từ việc thực hiện các hành động trong các trạng thái cụ thể của môi trường. Tác nhân không chỉ muốn tối đa hóa phần thưởng ngay lập tức mà còn là phần thưởng tích lũy nó nhận được trong suốt quá trình toàn bộ.



Hình 1. 10 Thuật toán Reinforcement Learning cycle

Một điểm quan trọng cần lưu ý về quá trình quyết định Markov là nó không quan tâm đến thưởng ngay lập tức mà hướng tới việc tối đa hóa tổng thưởng. Đôi khi, nó có thể ưa thích nhận một phần thưởng nhỏ để đạt được một phần thưởng cao hơn theo thời gian.

Phương trình Bellman

Trước khi đi sâu vào, hãy hiểu một số khái niệm quan trọng của phương trình Bellman.

- Trạng Thái (State): Là cách biểu diễn số học của những gì một tác nhân quan sát tại một điểm cụ thể trong môi trường.
- Hành Động (Action): Là đầu vào mà tác nhân đưa ra cho môi trường dựa trên một chính sách.
- Thưởng (Reward): Là tín hiệu phản hồi từ môi trường đến tác nhân, cho biết cách tác nhân đã thực hiện để đạt được mục tiêu.

Phương Trình Bellman giải quyết những câu hỏi sau:

Tác nhân đang ở trạng thái cụ thể s. Nếu chúng ta thực hiện các hành động tốt nhất trong tất cả các bước tiếp theo, thưởng dài hạn mà tác nhân có thể mong đợi là gì hoặc giá trị của trạng thái mà tác nhân đang ở là bao nhiêu.

Phương Trình Bellman là một loại thuật toán học tăng cường, đặc biệt hữu ích trong môi trường xác định. Nó giúp tác nhân xác định giá trị của một trạng thái cụ thể bằng cách chọn hành động tối ưu nhất trong trạng thái đó. Mục tiêu là tối đa hóa giá trị.

Để làm điều này, tác nhân cần cộng thêm giá trị thưởng của hành động tối ưu 'a' trong trạng thái 's', sau đó nhân với hệ số giảm giá 'γ'. Mỗi khi tác nhân thực hiện hành động, nó quay trở lại trạng thái tiếp theo 's'.

$$V(s) = max_a\big(R(s,a) + \gamma V(s')\big)$$

Thay vì tổng hợp qua nhiều bước thời gian, phương trình này đơn giản hóa việc tính giá trị hàm, cho phép chúng ta tìm ra giải pháp tốt nhất cho một vấn đề phức tạp bằng cách chia nhỏ nó thành các bài toán con đệ quy nhỏ hơn.

Q-Learning

Q-Learning kết hợp chính sách và hàm giá trị, đồng thời nó cho chúng ta biết mức độ hữu ích của một hành động nhất định trong việc đạt được phần thưởng trong tương lai.

Chất lượng (Quality) được gán cho cặp hành động trạng thái là Q (s,a) dựa trên giá trị tương lai mà nó kỳ vọng, với trạng thái hiện tại và chính sách tốt nhất mà tác nhân có. Khi tác nhân học được hàm Q này, nó tìm kiếm hành động tốt nhất tại một trạng thái cụ thể (s) để đạt được chất lượng cao nhất.

Một khi chúng ta có hàm Q tối ưu (Q*), chúng ta có thể xác định chính sách tối ưu bằng cách áp dụng một thuật toán học tăng cường để tìm hành động tối đa hóa giá trị cho mỗi trạng thái.

$$q_*(s,a) = \max_{\pi} q\pi(s,a)$$

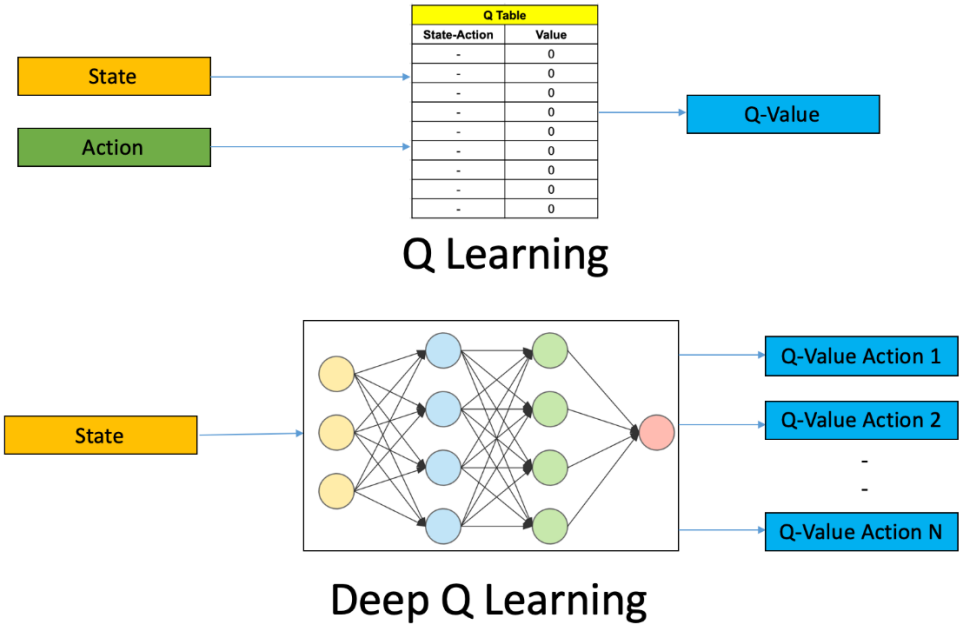
1.1.3 Tổng quan về học sâu tăng cường

1.1.3.1 Giới thiệu

Học tăng cường (Reinforcement Learning) là quá trình quản lý cặp trạng thái-hành động và theo dõi giá trị (thưởng) được gắn kết với một hành động để xác định chính sách tối ưu.

Phương pháp duy trì một bảng giá trị trạng thái-hành động không khả thi trong các tình huống thực tế khi có một số lượng lớn các khả năng.

Thay vì sử dụng một bảng, chúng ta có thể sử dụng mạng neuron để dự đoán giá trị cho các hành động trong một trạng thái cụ thể.



Hình 1. 11 Mô hình Deep Q Learning

1.1.3.2 Ứng dụng của học sâu tăng cường

Sản xuất công nghiệp

Học sâu tăng cường thường được áp dụng rộng rãi trong robot hóa. Các hành động mà robot phải thực hiện là theo đúng trình tự. Các tác nhân học cách tương tác với môi trường động và thay đổi, do đó có ứng dụng trong tự động hóa công nghiệp và sản xuất.

Chi phí lao động, lỗi sản phẩm đang được giảm đi với sự cải thiện đáng kể về thời gian chuyển đổi và tốc độ sản xuất.

Xe tự lái

Các công nghệ học máy là cốt lõi của xe tự lái.Xe tự lái sử dụng lượng lớn dữ liệu hình ảnh và tận dụng khả năng xử lý hình ảnh cùng với kiến trúc mạng neuron.

Các thuật toán học cách nhận diện người đi bộ, đường, giao thông, phát hiện biển báo đường trong môi trường và hành động tương ứng. Nó được huấn luyện trong các tình huống phức tạp và được đào tạo để xuất sắc trong kỹ năng ra quyết định trong các tình huống liên quan đến thiệt hại con người tối thiểu, tuyến đường tốt nhất để đi,…

Giao dịch và tài chính

Chúng ta đã thấy làm thế nào học máy giúp dự đoán thị trường chứng khoán thông qua việc học có giám sát và phân tích chuỗi thời gian. Nhưng không có thuật toán nào giúp chúng ta đưa ra quyết định về điều gì nên làm trong một tình huống cụ thể. Một tác nhân Học Tăng Cường có thể chọn liệu nên giữ, mua hay bán một cổ phiếu. Để đảm bảo mô hình RL hoạt động tối ưu, nó được đánh giá bằng cách sử dụng các tiêu chí đánh giá thị trường.

Xử lý ngôn ngữ tự nhiên

Học tăng cường đang mở rộng lĩnh vực của mình và đã chiếm lĩnh cả xử lý ngôn ngữ tự nhiên (NLP). Các nhiệm vụ NLP khác nhau như trả lời câu hỏi, tóm tắt, triển khai chatbot có thể được thực hiện bởi một tác nhân học tăng cường.

Bot ảo được đào tạo để mô phỏng cuộc trò chuyện. Các chuỗi có tính chất quan trọng về cuộc trò chuyện bao gồm tính nhất quán, tính thông tin và tính đơn giản của phản hồi được thưởng bằng các phương pháp độ dốc chính sách.

Chăm sóc sức khỏe

Học tăng cường trong lĩnh vực chăm sóc sức khỏe là một lĩnh vực nghiên cứu liên tục. Bot trang bị thông tin sinh học được đào tạo một cách kỹ lưỡng để thực hiện các cuộc phẫu thuật đòi hỏi độ chính xác. Bot học tăng cường cũng giúp chẩn đoán bệnh tốt hơn.

1.2 Giới thiệu về kiểm thử bảo mật ứng dụng web

1.2.1 Khái niệm về kiểm thử ứng dụng xâm nhập

Kiểm thử xâm nhập (Penetration Testing – Pentest) là một loại kiểm thử bảo mật mà trong đó người kiểm thử (pentester) sẽ sử dụng các kỹ năng của mình cũng như các công cụ để tấn công mô phỏng thực tế vào hệ thống nhằm mục đích kiểm tra, đánh giá mức độ an toàn của hệ thống cũng như tìm ra được lỗ hổng bảo mật hoặc điểm yếu nếu có mà bọn tội phạm có thể khai thác, chẳng hạn như:

- Cấu hình không phù hợp hoặc không phù hợp.
- Các lỗi phần cứng hoặc phần mềm đã biết và chưa biết.
- Những điểm yếu trong hoạt động trong các quy trình hoặc các biện pháp đối phó kỹ thuật.

Chính cách làm của kiểm thử xâm nhập là thực hiện mô phỏng các cuộc tấn công như những hacker thực sự nên nó đem lại rất nhiều lợi ích. Sau đây là một số lợi ích của công việc pentest mang lại bao gồm:

- Chủ động xác định các mối đe dọa và xác định xác suất tấn công vào tài sản thông tin.
- Kiêm tra toàn diện đảm bảo rằng hệ thống đang hoạt động trong một giới hạn chấp nhận được về rủi ro bảo mật thông tin.
- Giúp xác định các các vectơ tấn công và xác định tác động kinh doanh có thể gây ra của một cuộc tấn công thành công. Từ đó có các bước chuẩn bị có thể được thực hiện để ngăn chặn việc khai thác lỗ hổng.
- Đảm bảo triển khai một cách hiệu quả các biện pháp phỏng thủ (tường lửa, bộ định tuyến và máy chủ web..).
- Đảm bảo tuân thủ các quy định, tiêu chuẩn trên thế giới. Ví dụ: (ISO / IEC 27001: 2013, PCI-DSS, HIPPA, FISMA, v.v.).

1.2.2 Phân loại kiểm thử xâm nhập

Kiểm thử xâm nhập được chia thành 3 loại chính, bao gồm: Black box, White box và Gray box. Sau đây, ta tiến hành phân tích từng loại:

❖ **Black box**

Kiểm thử xâm nhập theo hướng black-box là việc kiểm thử bằng cách tấn công từ bên ngoài vào hệ thống, các cuộc tấn công kiểm thử này sẽ được thực hiện mà không có bất kỳ thông tin nào, pentester sẽ đặt mình vào vị trí của những tin tặc mũ đen và cố gắng bằng mọi cách để thâm nhập vào được hệ thống của khách hàng.

Pentester sẽ mô phỏng một cuộc tấn công thực sự vào hệ thống. Quá trình thử nghiệm bao gồm một loạt ứng các lỗ hổng bảo mật ở cấp dụng được xác định bởi OWASP và WASC, nhắm mục tiêu các lỗ hổng bảo mật nguy hiểm tiềm tàng trong ứng dụng của khách hàng . Quá trình thử nghiệm sẽ tiết lộ các lỗ hổng, thiệt hại khai thác tiềm năng và mức độ nghiêm trọng.

Phân loại kiểm thử black box:

- **Blind Testing:** Mô phỏng các phương pháp của một hacker không có hoặc cực kì ít thông tin được cung cấp cho nhóm kiểm tra thâm nhập, quá trình kiểm thử thường tốn thời gian và công sức thực hiện.

- **Double-Blind Testing:** Rất ít người trong tổ chức biết về kiểm tra thâm nhập đang được tiến hành. Quá trình kiểm thử có liên quan đến kiểm tra bảo mật của tổ chức giám sát, xác định sự cố và các thủ tục ứng phó.

❖ **White box**

Kiểm thử white box là loại kiểm thử mà các thông tin về mạng nội bộ và ngoại sẽ được cung cấp bởi khách hàng, các pentester có thể có quyền truy cập vào hệ thống thông tin từ đó sẽ đánh giá an ninh mạng dựa trên đó.

Phân loại kiểm thử white box:

- **Announced Testing:** Quá trình kiểm thử có sự hợp tác và tham gia cùng của đội quản trị hệ thống thông tin.
- **Unannounced Testing:** Thường chỉ những quản lý cấp cao được biết đến quá trình kiểm thử. Thực hiện kiểm tra việc xử lý, phản hồi các cuộc tấn công của nhân viên quản trị hệ thống thông tin

❖ **Gray box**

Kiểm thử gray box là phương pháp kết hợp cà white-box và black-box testing. Ở đây, pentester được cung cấp một phần thông tin của hệ thống, và giới hạn truy cập một phần vào hệ thống.

Việc lựa chọn phương pháp kiểm thử do tổ chức quyết định, dựa vào yêu cầu, mục đích, thời gian và khả năng tài chính của tổ chức đó. Trong khi kiểm tra bằng phương pháp blackbox đem lại đánh giá chính xác về khả năng tấn công của những hacker thực sự thì whitebox đem lại lợi thế về thời gian, quyền truy cập hệ thống cũng như những hiểu biết chính xác về hệ thống, từ đó tìm ra các điểm yêu tồn tại trên hệ thống.

1.2.3 Quy trình kiểm thử xâm nhập

Kiểm thử xâm nhập được chia thành 6 quá trình chính, mỗi quá trình sẽ đảm nhiệm một nhiệm vụ khác nhau:

❖ **Thu thập và theo dõi thông tin thụ động**

Trong giai đoạn đầu tiên của kiểm thử xâm nhập, pentester phải tiến hành thu thập thông tin về hệ thống mục tiêu trên các nền tảng xã hội. Bước này liên quan đến việc trích xuất các chi tiết có giá trị về cơ sở hạ tầng của hệ thống mục tiêu, chẳng hạn như tên miền, block mạng, router và địa chỉ IP trong phạm vi

của nó. Ngoài ra, mọi thông tin liên quan có thể nâng cao sự thành công của cuộc tấn công, chẳng hạn như dữ liệu nhân viên và số điện thoại, đều phải được thu thập.

Dữ liệu thu được từ các nguồn mở trong giai đoạn này có thể mang lại những chi tiết quan trọng một cách đáng ngạc nhiên. Để đạt được điều này, pentester phải tận dụng nhiều nguồn khác nhau, đặc biệt chú trọng đến trang web và nền tảng mạng xã hội của tổ chức mục tiêu.

❖ **Dò quét và thu thập thông tin chủ động**

Trong giai đoạn này, pentester sẽ tiến hành sử dụng các công cụ dò quét hỗ trợ để tìm và phát hiện được thiết bị chủ động và thụ động nào đang hoạt động trong phạm vi IP, Với sự trợ giúp của thông tin thu được trong quá trình thu thập thụ động này, người thực hiện pentest cần xác định đường đi của mình - họ cần ưu tiên và xác định chính xác những thử nghiệm nào là cần thiết.

Trong giai đoạn này, tin tặc sẽ có thể lấy được thông tin về hệ điều hành, các cổng và dịch vụ đang mở cũng như thông tin phiên bản của chúng trên những hệ thống live.

❖ **Phân tích và kiểm tra**

Ở giai đoạn này, sau khi tìm ra thông tin mà ứng dụng mục tiêu sẽ phản ứng với các lần xâm nhập khác nhau, sẽ cố gắng thiết lập những kết nối hoạt động với các hệ thống mà nó phát hiện là còn hoạt động và cố gắng thực hiện các yêu cầu trực tiếp. Nói cách khác, đây là giai đoạn mà pentester sẽ tương tác với hệ thống mục tiêu bằng cách sử dụng hiệu quả các dịch vụ như FTP, Netcat và Telnet…

❖ **Khai thác**

Ở giai đoạn này, người kiểm thử thâm nhập cố gắng xâm nhập vào hệ thống, sử dụng hệ điều hành chạy trên hệ thống đích, các cổng mở và những dịch vụ phục vụ trên các cổng này cũng như những phương thức khai thác có thể được áp dụng tùy theo phiên bản của chúng. Vì các cổng và ứng dụng dựa trên web bao gồm rất nhiều code và rất nhiều thư viện, nên tin tặc có ác ý sẽ có phạm vi tiếp cận lớn hơn để tấn công. Về mặt này, người kiểm thử thâm nhập tốt nên xem xét tất cả các khả năng và triển khai tất cả mọi hướng tấn công có thể được phép.

Việc này đòi hỏi chuyên môn và kinh nghiệm cao và đạo đức nghiêm túc để có thể sử dụng thành công và linh hoạt các phương thức khai thác hiện có, không làm hỏng hệ thống và không để lại bất kỳ dấu vết nào trong quá trình tiếp

quản hệ thống. Do đó, giai đoạn kiểm thử thâm nhập này là bước quan trọng nhất.

❖ **Nâng cao đặc quyền**

Ở giai đoạn này, khi người kiểm thử thâm nhập truy cập được vào hệ thống, họ sẽ cần có quyền admin, khai thác các lỗ hổng trong hệ điều hành hoặc môi trường.

Sau đó, họ sẽ nhắm đến việc chiếm giữ các thiết bị khác trong môi trường mạng bằng các đặc quyền bổ sung mà họ đã đạt được và cuối cùng là những đặc quyền người dùng cấp cao nhất như quản trị viên domain hoặc quản trị viên cơ sở dữ liệu.

❖ **Báo cáo**

Đây là giai đoạn cuối của việc kiểm thử xâm nhập. Khi quá trình kiểm thử thâm nhập được hoàn thành, người kiểm thử thâm nhập phải trình bày các lỗ hổng bảo mật mà họ đã phát hiện trong hệ thống đích, những bước tiếp theo và cách họ có thể khai thác các lỗ hổng này cho tổ chức bằng một báo cáo chi tiết. Điều này phải bao gồm các thông tin như ảnh chụp màn hình, code mẫu, giai đoạn tấn công và nguyên nhân lỗ hổng này có thể xảy ra.

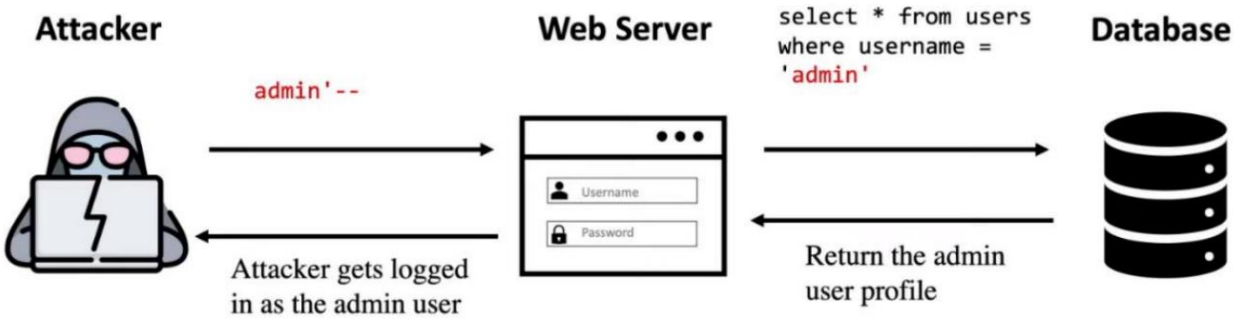
Báo cáo cuối cùng cũng phải bao gồm đề xuất giải pháp về cách thu hẹp từng lỗ hổng bảo mật. Độ nhạy và tính độc lập của các thử nghiệm thâm nhập vẫn còn là một bí ẩn. Hacker mũ trắng không bao giờ được chia sẻ thông tin bí mật thu được ở giai đoạn này và không bao giờ được lạm dụng thông tin này bằng cách cung cấp thông tin sai lệch, vì điều đó thường là bất hợp pháp.

1.3 Tổng quan về các lỗ hổng phần mềm

1.3.1 SQL Injection

1.3.1.1 Định nghĩa SQL Injection

SQL Injection (SQLi) là một loại lỗ hổng bảo mật phổ biến trong ứng dụng web. Nó cho phép kẻ tấn công "tiêm" (chèn) mã SQL vào câu truy vấn mà ứng dụng web sử dụng để tương tác với cơ sở dữ liệu của nó. Nếu ứng dụng không được lập trình để kiểm soát hoặc ngăn chặn điều này, mã SQL độc hại có thể được thực thi bởi máy chủ cơ sở dữ liệu.

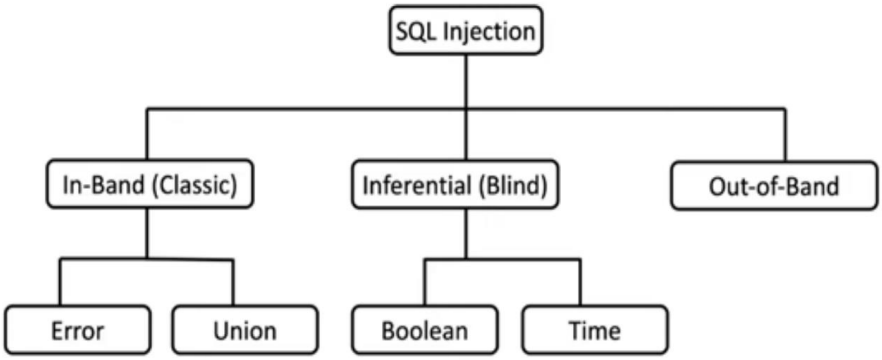


Hình 1. 12 SQL Injection

1.3.1.2 Phân loại SQL Injection

Nhìn chung các cuộc tấn công SQL Injection được chia làm 3 loại chính:

- In-Band SQLi (Classic).
- Inferential SQLi (Blind).
- Out-of-Band SQLi.



Hình 1. 13 Các loại SQL Injection

❖ **In-band SQLi (Classic):**

Trong loại này, kẻ tấn công sử dụng cùng một kênh liên lạc để khởi động các cuộc tấn công và thu thập các kết quả. Tính đơn giản và hiệu quả của In-band SQLi khiến nó trở thành một trong những kiểu tấn công SQLi phổ biến nhất hiện nay. Có hai biến thể phụ của phương pháp này:

- **Error-based SQLi:** kẻ tấn công sẽ thực hiện các hành động làm cơ sở dữ liệu tạo ra thông báo lỗi. Kẻ tấn công có thể dùng dữ liệu được cung cấp bởi các thông báo lỗi này để thu thập thông tin về cấu trúc của cơ sở dữ liệu.

Ví dụ:

Input: www.example.com/index.php?id=’

Output: You have an error in your SQL sytax, check the manual that corresponds to your MySQL server version…

- **Union-based SQLi:** kỹ thuật này lợi dụng toán tử UNION SQL để kết hợp nhiều câu lệnh được tạo bởi Cơ sở dữ liệu để nhận được một HTTP response. Response này có thể chứa dữ liệu mà kẻ tấn công có thể sử dụng.

Ví dụ:

Input: www.example.com/index.php?id=’ UNION SELECT username, password FROM users—

Output: txc3000 16112002cC@ administrator

❖ **Inferential SQLi (Blind)**

Trong loại SQLi này, kẻ tấn công sẽ gửi các data payload đến server và quan sát phản ứng, hành vi của server để tìm hiểu về cấu trúc của nó. Phương pháp này được gọi là blind SQLi vì dữ liệu không được chuyển từ cơ sở dữ liệu trang web đến kẻ tấn công. Do đó kẻ tấn công không thể nhìn thấy thông tin về cuộc tấn trong in-band. Blind SQLi dựa trên phản ứng và các hành vi hoạt động của server. Do đó chúng thường thực thi chậm hơn, nhưng có thể gây ảnh hưởng tương tự. Blind SQLi có thể được phân loại thành 2 dạng.

- **Boolean:** kẻ tấn công gửi một truy vấn SQL đến cơ sở dữ liệu, làm ứng dụng trả về một kết quả. Kết quả có thể khác nhau tùy vào truy vấn đúng hay sai. Dựa trên kết quả, thông tin trong HTTP response sẽ sửa đổi hoặc không. Sau đó, kẻ tấn công có thể tìm hiểu xem thông báo tạo ra kết quả có đúng không.

Ví dụ:

Input 1: www.example.com/index.php?id=’ AND 1=1 –

Input 2: www.example.com/index.php?id=’ AND 1=2 -- 8

Nếu 2 input trên cho 2 phản hồi khác sau thì website đã thực thi mã SQL và có khả năng là dạng boolean-based SQLi.

- **Time-based:** kẻ tấn công sẽ gửi một truy vấn SQL đến Cơ sở dữ liệu, làm cho Cơ sở dữ liệu đợi (trong vài giây) trước khi có thể hoạt động. Sau đó, kẻ tấn công có thể xem từ thời gian cơ sở dữ liệu cần để phản hồi, một truy vấn là đúng hay sai. Dựa trên kết quả, một HTTP repsonse sẽ được tạo ra. Vì vậy kẻ tấn công có thể tìm ra thông báo mà chúng đã sử dụng trả về đúng hay sai, không cần dựa vào dữ liệu từ cơ sở dữ liệu.

Ví dụ:

Input: www.example.com/index.php?id=1’ AND SLEEP(10)

Nếu website tải dữ liệu chậm 10 giây trong việc hiện thị kết quả thì có khả năng website đã bị lỗi time-based SQLi.

❖ **Out-of-band SQLi:**

Trong loại SQLi này, kẻ tấn công chỉ có thể thực hiện hình thức này khi có một số tính năng nhất định được kích hoạt trên server cơ sở dữ liệu được ứng dụng web sử dụng. Hình thức này chủ yếu được dùng để thay thế cho khác kỹ thuật in-band và inferential SQLi. Loại SQLi này được thực hiện khi kẻ tấn công không thể sử dụng cùng một kênh để khởi động tấn công và thu thập thông tin. Hoặc do server quá chậm, không ổn định tấn thực hiện hành động. Các kỹ thuật này dựa vào khả năng server tạo ra các DNS hay HTTP request để chuyển dữ liệu cho kẻ tấn công.

1.3.1.3 Hậu quả của SQLi

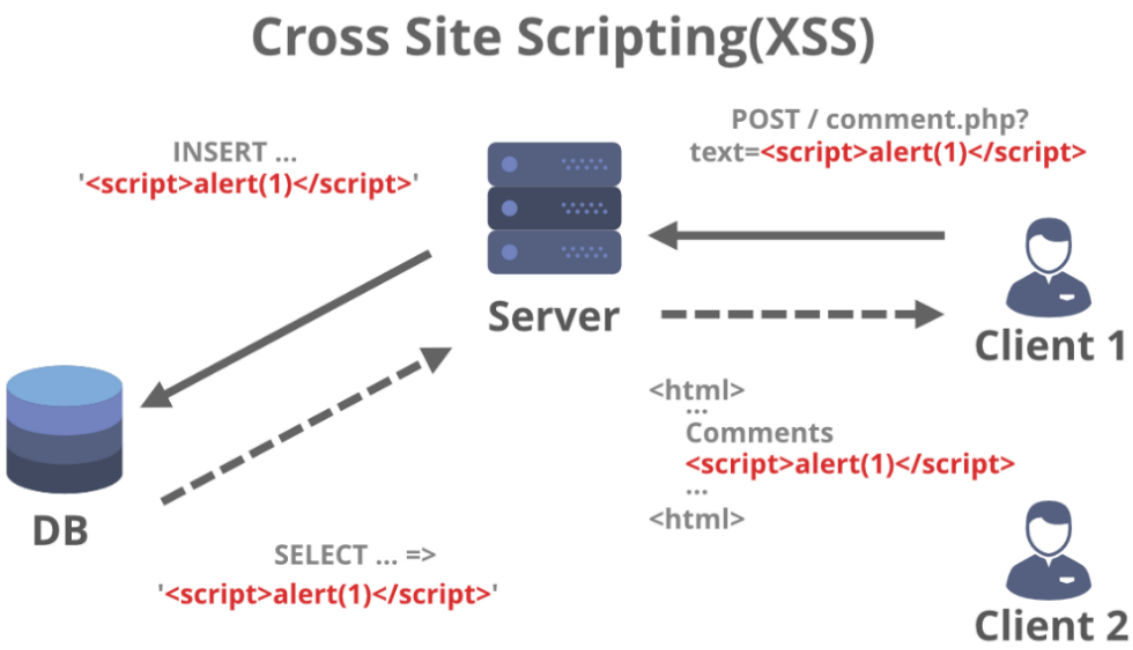
Lợi dụng lỗ hổng SQLi, kẻ tấn công có thể gây ra nhiều loại hậu quả đáng kể cho hệ thống, bao gồm:

- **Đọc dữ liệu không được phép:** Kẻ tấn công có thể sử dụng SQLi để đọc dữ liệu mà họ không nên có quyền truy cập, như thông tin cá nhân của người dùng hoặc dữ liệu kinh doanh bí mật.
- **Thay đổi hoặc xoá dữ liệu:** SQLi có thể cho phép kẻ tấn công thay đổi hoặc xoá dữ liệu trong cơ sở dữ liệu.
- **Thực hiện các hành động như một người dùng khác:** Nếu ứng dụng lưu trữ thông tin về phiên đăng nhập trong cơ sở dữ liệu, SQLi có thể cho phép kẻ tấn công giả mạo đăng nhập như là một người dùng khác.
- **Lấy quyền kiểm soát hệ thống (RCE):** Trong một số trường hợp, SQLi có thể được sử dụng để thực thi mã tùy ý trên máy chủ cơ sở dữ liệu, tiềm ẩn rủi ro cho phép kẻ tấn công kiểm soát hoàn toàn hệ thống.

SQLi có thể ngăn chặn bằng cách áp dụng nhiều phương pháp bảo vệ, như sử dụng câu truy vấn có tham số (parameterized queries), lọc đầu vào của người dùng, và đảm bảo ứng dụng web cập nhật với các bản vá bảo mật mới nhất.

1.3.2 Cross-Site Scripting

1.3.2.1 Định nghĩa Cross-Site Scripting



Hình 1. 14 Cross-Site Scripting

Cross-Site Scripting (XSS) là một loại lỗ hổng bảo mật phổ biến và nghiêm trọng trong ứng dụng web. Nó cho phép kẻ tấn công chèn mã script độc hại vào trang web. Mã này sau đó được thực thi trong trình duyệt của người dùng khi họ truy cập trang web.

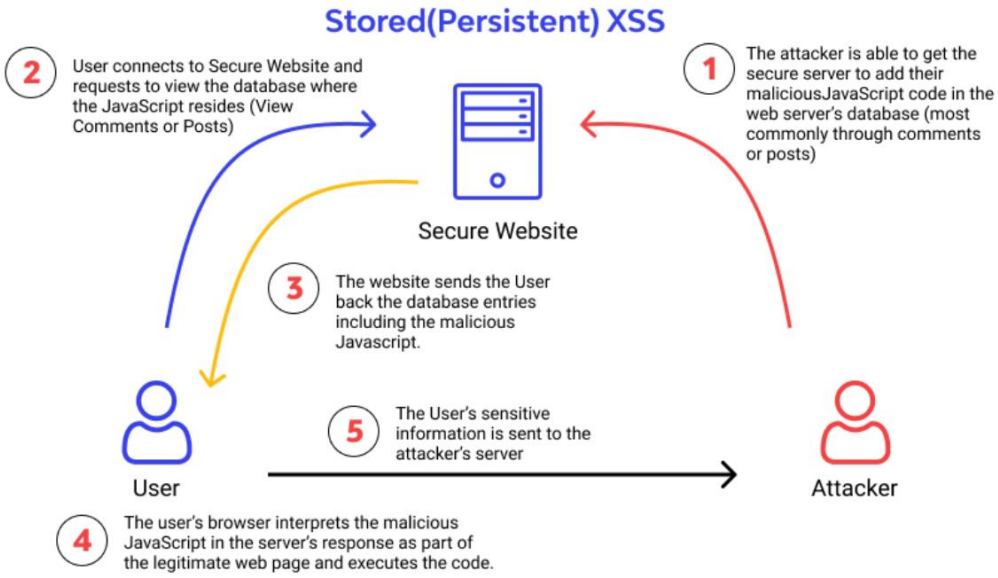
1.3.2.2 Phân loại Cross-Site Scripting

Có thể chia lổ hổng XSS thành 3 loại chính, bao gồm:

❖ **Stored XSS**

Đây là loại tấn công XSS phổ biến và nguy hiểm nhất. Nó là một dạng tấn công mà mã độc hại của kẻ tấn công sẽ được lưu trữ trên máy chủ và được gửi đến trình duyệt của người dùng mỗi khi họ truy cập một trang nhất định. Cách Stored XSS hoạt động:

- Kẻ tấn công bình luận vào một bài viết trên mạng xã hội, trong bình luận đó có chèn đoạn mã JavaScipt độc hại.
- Đoạn mã này sẽ được server lưu trữ trên cơ sở dữ liệu và website sẽ hiện thị bình luận đó khi có người dùng khác xem.
- Khi người dùng khác xem bình luận đó, đoạn mã độc hại sẽ thực thi trên browser của họ, Nó có thể cho phép kẻ tấn công lấy được thông tin đăng nhập hoặc cookie của người dùng, hoặc thực hiện các hành động không mong muốn khác.

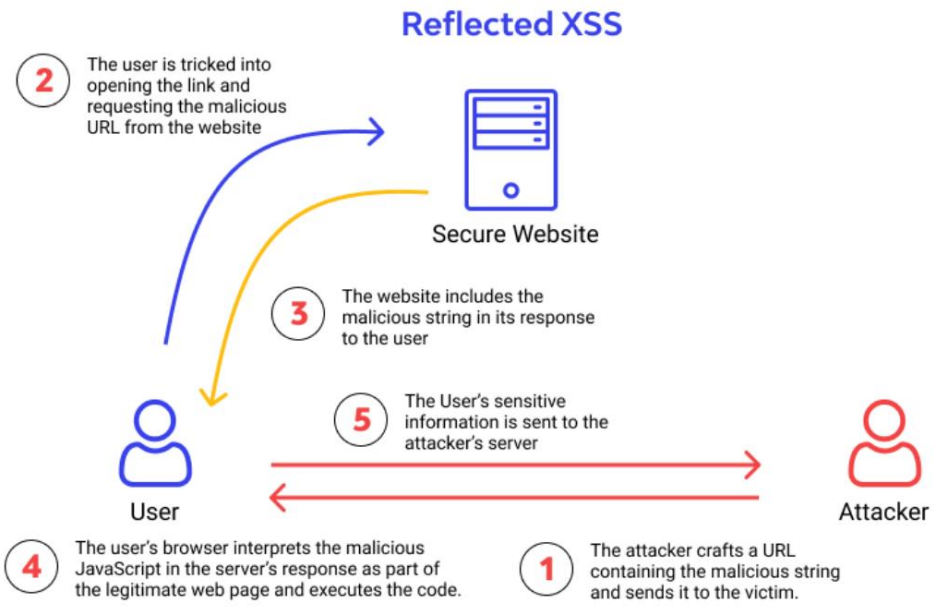


Hình 1. 15 Stored XSS

❖ Reflected XSS

Trong tấn công này, mã độc hại được phản ánh lại từ máy chủ web thay vì được lưu trữ trên đó. Mã độc thường được chèn vào URL và được thực thi khi người dùng nhấp vào liên kết. Cách Reflected XSS hoạt động:

- Kẻ tấn công tạo ra url độc hại, mà trong đó mã JavaScript độc hại sẽ được chèn vào một tham số của url ví dụ như chức năng tìm kiếm.
- Kẻ tấn công sẽ gửi url đó cho nạn nhân thông qua email, tin nhắn hoặc hình thức khác.
- Khi nạn nhân truy cập vào liên kết, yêu cầu sẽ gửi đến trang web và mã JavaScript độc hại sẽ được phản chiếu lại vào phần tìm kiến của trang web và broser sẽ thực thi đoạn mã đó.

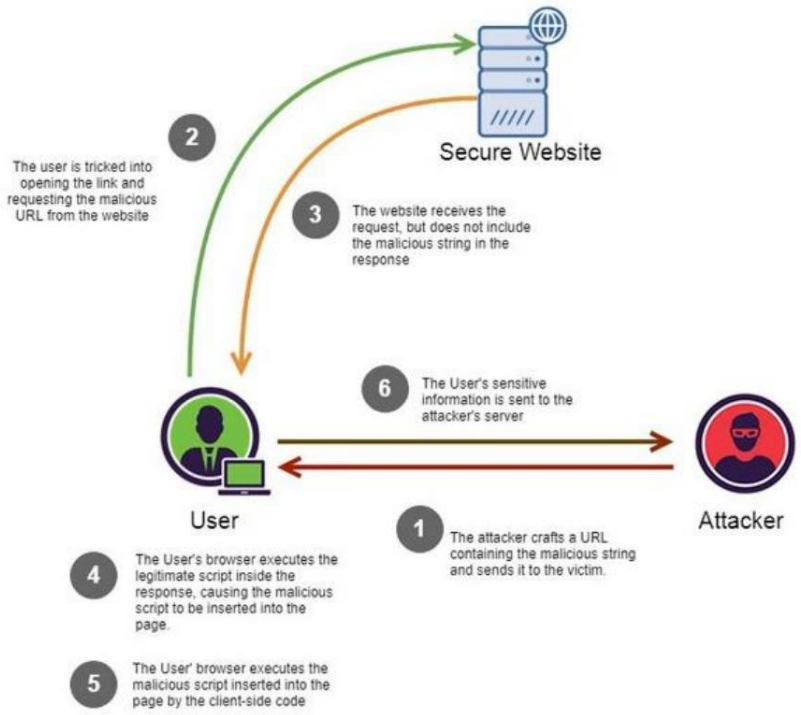


Hình 1. 16 Reflected XSS

❖ **DOM-based XSS**

Đây là loại tấn công XSS phức tạp hơn, trong đó mã độc hại tương tác làm thay đổi Document Object Model (DOM) của trang. Trong đó, DOM là cấu trúc mà browser sử dụng để hiểu và hiện thị nội dung của trang web. Do đó, mã độc sẽ làm thay đổi cách trang hoạt động và gây ra việc thực thi mã độc đó. Cách DOM-based XSS hoạt động:

- Kẻ tấn công sẽ tìm kiếm một điểm yếu trên trang web mà nó có thể tận dụng. Ví dụ như website có thể đọc tham số từ url và thêm vào DOM mà không cần kiểm tra đầu vào.
- Kẻ tấn công sau đó tạo một url độc hại trong đó có chèn một đoạn mã JavaScript vào tham số.
- Khi người dùng nhập vào url độc hại, trang web sẽ lấy tham số độc hại từ url và thêm nó vào DOM. Cuối cùng, mã JavaScript độc hại sẽ được browser thực thi.



Hình 1. 17 DOM-base XSS

1.3.2.3 Hậu quả của Cross-Site Scripting

Lỗ hổng Cross-Site Scripting (XSS) có thể dẫn đến nhiều hậu quả nguy hiểm đối với người dùng và trang web bị tấn công. Dưới đây là một số hậu quả tiêu biểu:

- **Đánh cắp thông tin cá nhân:** Kẻ tấn công có thể sử dụng mã độc để đánh cắp cookie hoặc thông tin đăng nhập của người dùng, cho phép họ truy cập vào tài khoản của người dùng.
- **Thực hiện các hành động thay mặt người dùng:** Nếu kẻ tấn công có quyền truy cập vào tài khoản của người dùng, họ có thể thực hiện các hành động thay mặt người dùng, chẳng hạn như gửi tin nhắn, thay đổi thông tin cá nhân, hoặc thực hiện các giao dịch tài chính.
- **Phân phối mã độc:** Kẻ tấn công có thể sử dụng lỗ hổng XSS để phân phối mã độc tới người dùng của một trang web. Mã độc này có thể bao gồm ransomware, keyloggers, hoặc bất kỳ loại phần mềm độc hại nào khác.
- **Pháp lý và tiêu chuẩn tuân thủ:** Nếu thông tin cá nhân của người dùng bị đánh cắp, công ty chủ quản trang web có thể phải đối mặt với các hậu quả pháp lý.

Do đó, rất quan trọng để nhà phát triển web cần hiểu về lỗ hổng XSS và cách ngăn chặn nó để bảo vệ người dùng và trang web của họ.

CHƯƠNG 2 ỨNG DỤNG HỌC TĂNG CƯỜNG VÀO KIỂM THỬ XÂM NHẬP

2.1 Mô tả thực trạng lỗ hổng XSS hiện nay và giải pháp

Trong thời đại công nghệ hiện nay, các lỗ hổng bảo mật nói chung và lỗ hổng XSS nói riêng đang rất phổ biến trên các website và nó rất đa dạng. Vì vậy, đã có nhiều nguyên cứu, bài báo đề xuất và phát triển các phương pháp nhận diện lỗ hổng Reflected Cross-Site Scripting (XSS). Các nghiên cứu đó đã đề xuất chung thành 5 phương pháp để phát hiện lỗ hổng Reflected XSS, bao gồm: static methods, dynamic detection, phương pháp kết hợp, symbolic execution, penetration testing.

Từ 5 phương pháp trên, chúng ta đã tạo ra rất nhiều các công cụ dò quét lỗ hổng XSS. Các công cụ này đã phần sẽ hoạt động bằng cách tiến hành thực tiêm một loạt các payload có trong từ điển của chúng vào các input của website mục tiêu và kiểm tra xem các payload có kích hoạt tấn công XSS không. Và các công cụ này không hề yêu cầu truy cập mã nguồn hay gây ra bất kỳ sự thay đổi nào trong môi trường chạy của ứng dụng web.

Tuy nhiên với sự đa dạng và ngày các phát triển của ứng dụng web, các công cụ này đã gặp phải những hạn chế là:

- Các công cụ dò quét tiêu tốn rất nhiều thời gian để có thể tiến hành tấn công từ điển để tìm ra lỗ hổng XSS thành công.
- Các công cụ dò quét chỉ quan sát các giá trị đầu vào là payload có kích hoạt tấn công XSS mà không hề quan sát các bối cảnh đầu ra, từ đó có thể trả về các kết quả âm tính XSS giả mạo.

Để có thể khắc phục được những hạn chế trên, chúng ta có thể ứng dụng mô hình học sâu tăng cường (Deep Reinforcement Learning - DRL) để điều chỉnh các payload tấn công XSS trong quá trình quan sát các phản hồi trả về từ các lần thử tấn công XSS trước đó, từ đó nó có thế tìm ra được những payload XSS phức tạp mà sẽ kích hoạt tấn công XSS. Chúng ta sẽ tiến hành tạo một công cụ dò quét ứng dụng web tự động có sử dụng một agent DRL để xác định được lỗ hổng.

2.2 Xác định và phân tích vấn đề thách thức

Ta gọi công cụ kiểm thử ứng dụng DRL là Link. Khi xây dựng Link, thì có một số thách thức phát sinh, bao gồm:

- Việc xác định được trạng thái, hành động và hàm sinh phần thưởng của agent trong việc giải quyết các vấn đề.

- Việc đào tạo agent có thể chuyển giao giữa các môi trường đào tạo và kiểm thử khác nhau.

Với vấn đề đầu tiên, agent RL không hội tụ (non-converging agent). Học tăng cường đã được triển khai trong nhiều lĩnh vực khác nhau như trong game, blockchain, robot,… Điều đó chứng minh rằng hiệu quả của RL trong nhiều bối cảnh thực tế khác nhau. Tuy nhiên, việc áp dụng RL vào các vấn đề thực tế là khó khăn do agent không hội tụ. Nguyên nhân chủ yếu của vấn đề này là do các hành động, trạng thái và hàm thưởng được định nghĩa chưa thực sự chính xác, không phản ánh được đúng các vấn đề mục tiêu thực tế. Do đó, một thách thức kỹ thuật ở đây là định nghĩa đúng các hành động, trạng thái và hàm thưởng để đóng góp vào việc hội tụ một agent RL hiệu suất cao có khả năng tìm ra lỗ hổng XSS.

Với vấn đề thứ hai, agent RL có thể chuyển giao môi trường. Trong các nghiên đã từng được đề xuất, việc áp dụng học tăng cường thường có điểm chung là môi trường đào tạo và kiểm thử của một agent RL là giống nhau. Tuy nhiên, để thực hiện các mục tiêu kiểm thử hiệu quả, một agent RL được đào tạo trên các bài kiểm thử nên có khả năng tìm ra lỗ hổng XSS trong các ứng dụng web thực tế và với các bài nghiên cứu trước thường đòi hỏi agent RL phải đào tại lại từ đầu trên các ứng dụng web trong môi trường mới. Vậy nên một thách thức kỹ thuật thứ hai là việc xác định trạng thái và hành động cho một agent RL nên được tổng quát đến mức đủ để bao phủ nhiều loại lỗ hổng XSS phản ánh thực tế.

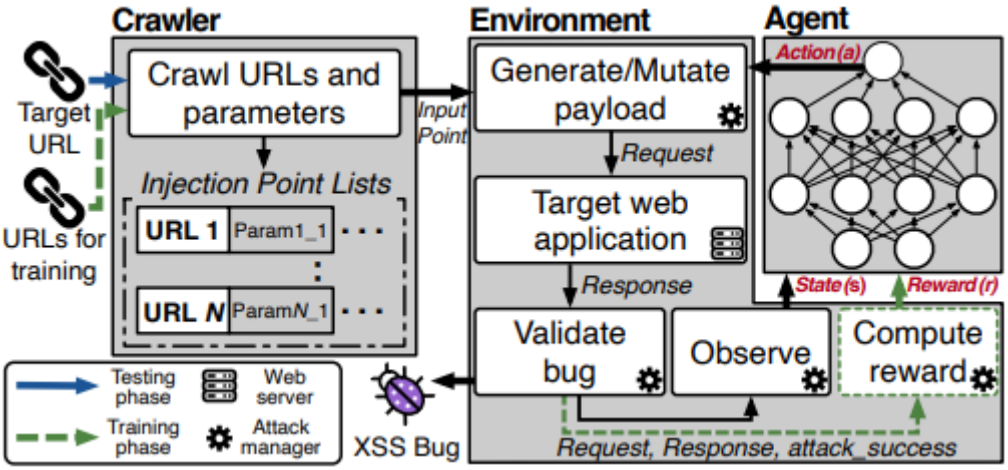
2.3 Xây dựng các giai đoạn và cấu trúc của công cụ

Ta xác định các giai đoạn xây dựng Link, một công cụ do quét tìm lỗ hổng bảo mật Reflected XSS ứng dụng học tăng cường. Công cụ này được chia làm 2 giai đoạn chính là: huấn luyện và kiểm thử.

Trong giai đoạn huấn luyện, Link sẽ tiến hành nhận vào một lượng lớn các đường dẫn URL, trong đó mỗi URL sẽ dẫn đến một ứng dụng web có chứa lỗ hổng Reflected XSS. Sử dụng tập hợp các website này, công cụ sẽ đào tạo một agent RL học một chính sách tối ưu để tạo ra một payload tấn công có thể thích ứng với môi trường website đã cho. Giai đoạn huấn luyện này là quy trình thiết lập một lần duy nhất.

Với sự đào tạo agent RL trong giai đoạn huấn luyện, giai đoạn kiểm thử sẽ thực hiện việc kiểm thử xâm nhập theo hướng black-box đối với mỗi URL cụ thể của website mục tiêu, Khi một quá trình kiểm thử cho website mục tiêu kết

thúc, công cụ sẽ tiến hành báo cáo lại các functional requests mà đã thực hiện thành công việc khai thác lỗ hổng Reflected XSS.



Hình 2. 1: Minh họa cấu trúc Link

Hình trên minh họa kiến trúc của Link. Nó bao gồm ba thành phần: Crawler, môi trường (Environment) và Agent. Crawler thu thập thông tin của tất cả các trang web được liên kết với nhau, từ đó xác định các URL của subdomain và các tham số đầu vào của chúng để sử dụng trong việc đưa payload tấn công vào. Đối với mỗi cặp URL và tham số đầu vào, Agent và môi trường hoạt động song song để thực hiện kiểm tra (hoặc huấn luyện) thông qua việc gửi yêu cầu tấn công có chứa payload. Môi trường cung cấp các giao diện để Agent quan sát các thay đổi của môi trường tương ứng với hành động của nó. Agent xác định một hành động để thực hiện dựa trên các quan sát về môi trường.

❖ **Crawler**

Với các URL được cung cấp, crawler trong công cụ bắt đầu thu thập các subdomain URL, đây là mục tiêu để kiểm tra khả năng thâm nhập. Cụ thể, công cụ sẽ truy cập từng URL, phân tích phản hồi HTML của nó và sau đó trích xuất tất cả các thành phần của URL và chuyển sang truy cập URL tiếp theo. Công cụ lặp lại quá trình này cho đến khi không còn URL nào để truy cập.

Đối với mỗi URL được xác định, công cụ sẽ thu thập các tham số đầu vào để agent sử dụng để đưa payload vào. Nó thu thập các tham số đầu vào từ các yêu cầu GET bằng cách trích xuất các thuộc tính chính trong chuỗi truy vấn của chúng. Ngoài ra, khi truy cập từng trang web, crawler sẽ truy xuất tất cả giá trị thuộc tính của thẻ đầu vào (tag input) trong các phần tử biểu mẫu HTML, trở thành tham số đầu vào cho các yêu cầu POST để thực hiện kiểm tra thâm nhập.

Công cụ tập trung vào việc thực hiện kiểm thử xâm nhập hiệu quả thông qua việc thay đổi linh hoạt các payload tấn công của nó. Chúng tôi đã sửa đổi

Wapiti, một trình quét lỗ hổng web nguồn mở, để tận dụng agent RL trong việc điều chỉnh payload tấn công. Chúng tôi đã mở rộng crawler của Wapiti để xác định URL mục tiêu và thông số đầu vào; cụ thể bao gồm Referer header và URI-based injection cho các nguồn đầu vào bổ sung.

❖ **Môi trường**

Môi trường bao gồm hai thành phần: máy chủ web và trình quản lý tấn công. Máy chủ web có một URL nhất định sẽ chạy ứng dụng mục tiêu. Trình quản lý tấn công là một lớp trừu tượng của máy chủ web cung cấp giao diện cho agent RL để quan sát các thay đổi của môi trường do hành động của agent. Trình quản lý tấn công được thiết kế để hỗ trợ bốn hoạt động chính: (**1**) tạo hoặc thay đổi payload tấn công dựa trên hành động của agent RL, (**2**) kiểm tra xem yêu cầu tấn công có thành công trong việc kích hoạt lỗ hổng XSS được kiểm tra hay không, (**c**) trích xuất quan sát thông tin từ phản hồi đối với yêu cầu tấn công và (**d**) tính toán phần thưởng từ thông tin quan sát được.

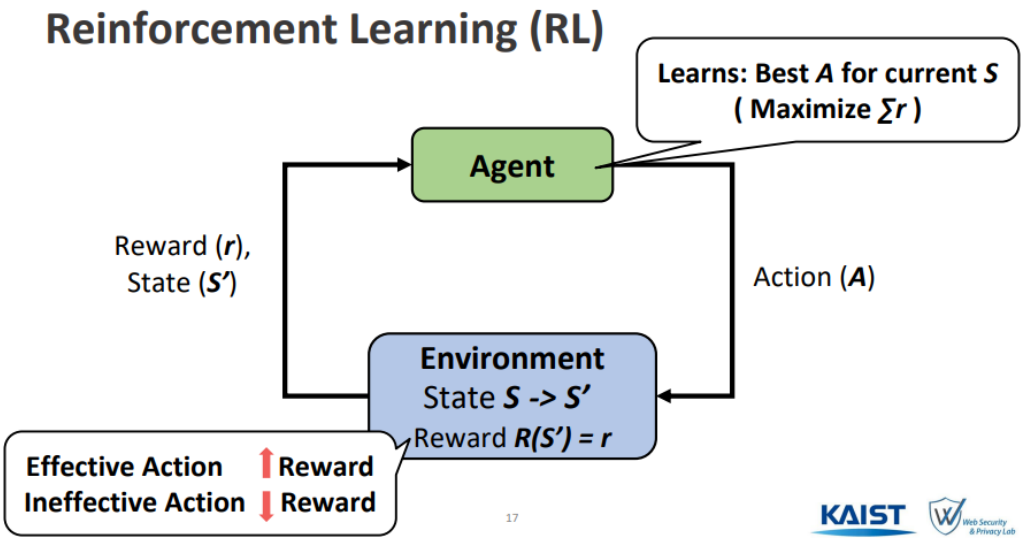
❖ **Agent**

Agent thực hiện chức năng chính của RL. Trong giai đoạn huấn luyện, nó lấy các tham số có thể định cấu hình, bao gồm tốc độ học và hệ số chiết khấu, làm đầu vào và học chính sách tối ưu để tạo ra payload tấn công để thích ứng với môi trường nhất định. Trong giai đoạn thử nghiệm, agent được đào tạo sẽ chọn một cách thích ứng một hành động ở trạng thái hiện tại của môi trường.

2.4 Xây dựng mô hình RL và thuật toán

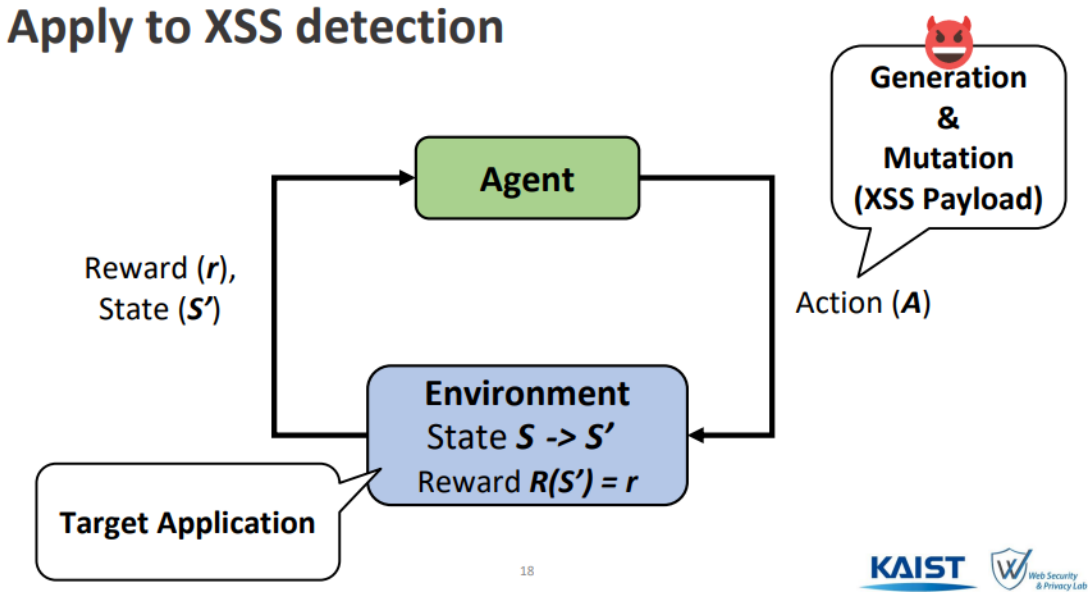
2.4.1 Xây dựng mô hình tổng quát

Theo mô hình chung của học tăng cường (Reinforcement Learning) sẽ huấn luyện agent RL học một chính sách quyết định hành động tối ưu nhằm tối đa phần thưởng. Cụ thể agent RL sẽ thực hiện hành động a và quan sát sự thay đổi trong môi trường E qua bước thời gian t. Tại mỗi bước thời gian t, agent nhận được một trạng thái s_t từ môi trường E và nó sẽ lựa chọn một hành động a_t từ tập hợp các hành dộng có thể theo chiến lược π. Khi a_t được chọn, nếu a_t hiệu quả thì agent sẽ nhận được phần thưởng r_t cao và ngược lại, nếu nếu a_t không hiệu quả thì agent sẽ nhận được phần thưởng r_t thấp. Từ r_t mới nhận được, agent RL sẽ suy ra trạng thái tiếp s_{t+1} thông qua tương tác với E.



Hình 2. 2 Mô hình RL

Dựa theo mô hình chung của học tăng cưởng, ta tiến hành xác định lại các yếu tố mô hình huấn luyện agent RL bao gồm hành động, trạng thái và điểm thưởng sao cho phù hợp với yêu cầu bài toán xây dựng công cụ dò quét lỗ hổng XSS.



Hình 2. 3 Mô hình RL trong Link

Trong mô hình RL ứng dụng vào việc phát hiện lỗ hổng XSS, ta tiến hành xây dựng một bộ chiến lược π, trong bộ chiến lược π có hành động A là tạo và biến đổi XSS payload. Sau đó, ta xây dựng trạng thái S là trạng thái của ứng dụng web, trang thái S được xây dựng từ 3 yếu tố chính:

- Thông tin về giá trị payload đầu vào.
- Thông tin về điểm tiêm payload.
- Thông tin về kết quả tấn công.

Cuối cùng, ta tiến hành xây dựng phần thưởng R là khi agent tấn công thành công sẽ được điểm 1 và khi agent tấn công mà không nhận được kết quả hoặc agent tấn công sử dụng payload cũ thì sẽ được điểm -1.

2.4.2 Phân tích và xây dựng thuật toán

2.4.2.1 Workflow của công cụ

Đầu tiên, tiến hành xác định và xây dựng thuật toán 1 để minh họa toàn bộ quá trình huấn luyện và kiểm tra tổng thể mà công cụ thực hiện xác định các lỗ hổng Reflected XSS.

Algorithm 1: Link Workflow (Highlighted codes are only executed during training).

```
function Train(tgt_set)
    total_step ← 0
    while total_step < MAX_STEPS do
        url, param ← Select (tgt_set)
        found, payload, step ← Episode(url, param)
        total_step ← total_step + step
function Test(tgt_set)
    for url, param ∈ tgt_set do
        found, payload, step ← Episode(url, param)
        if found then
            ReportBug(payload, url, param)
function Episode(url, param)
    // Initialize the payload and the current step t
    payload ← [Initial Payload], t ← 0, found ← false
    response ← SendReqest(url, param, payload)
    if !CheckReflected(payload, response) then
        // If response does not reflect payload, return false.
        return false, payload
    s_t ← Observe(payload, response, found)
    while t < STEPS_PER_EPISODE and !found do
        a_t ← agent.GetAction(s_t) // Get an action a_t given s_t.
        payload ← MutatePayload(payload, a_t)
        response ← SendReqest(url, param, payload)
        found ← BugOracle(payload, response)
        s_{t+1} ← Observe(payload, response, found)
        r_t ← Reward(t, s_{t+1}, payload, found)
        agent.Train(s_t, a_t, s_{t+1}, r_t)
        t ← t + 1;
    return found, payload, t
```

Hình 2. 4 Thuật toán Workflow

2.4.2.1.1 Hàm huấn luyện

Thuật toán hàm huấn luyện (*train*) (dòng 1) bắt đầu với việc cài đặt một bộ các đường dẫn URL và các cặp tham số đầu vào (*tgt_set*) mà crawler đã xác thực. Thuật toán sau đó thực hiện lặp lại hàm Episode cho đến khi tổng số bước (*total_step*) đạt đến *MAX_STEPS*. Đối với mỗi lần lặp, ta có thể gọi là một *episode*, nó lần lượt chọn một URL và một cặp tham số đầu vào trong tập *tgt_set* (dòng 4) và huấn luyện agent RL để tạo ra một kỹ thuật tận dụng cho cặp này thông qua việc thực hiện `Episode()`. Lưu ý rằng *MAX_STEPS* là một siêu tham số quyết định tổng số lượng bước mà agent RL nên được huấn luyện. Chúng tôi đặt giá trị mặc định của *MAX_STEPS* là 3,5 triệu.

Trong hàm `Episode` (dòng 12), agent thực hiện một loạt các hành động một số lần (*STEPS_PER_EPISODE*) cho cặp URL và tham số đầu vào đã chọn. Cụ thể, đối với mỗi cặp mục tiêu, công cụ khởi tạo một chiến dịch huấn luyện *episode*. Đối với mỗi *episode*, công cụ đầu tiên kiểm tra xem một payload ban đầu có xuất hiện trong trang web mục tiêu không. Để làm điều này, công cụ gửi một yêu cầu HTTP(S) với một payload đầu vào hợp lệ là `[Initial Payload]` (Payload ban đầu) (dòng 15). `[Initial Payload]` là một chuỗi chữ số và ký tự đặc biệt (ví dụ: `">/'>injectionhere1234`). Sau đó, nó kiểm tra xem chuỗi ký tự cụ thể này có được bao gồm trong phản hồi không (dòng 16). Chúng tôi đã chuẩn bị bốn initial payload để thực hiện kiểm thử này. Nếu công cụ không thể xác nhận sự xuất hiện của bất kỳ payload nào, điều này chắc chắn cho thấy rằng việc tấn công cặp tham số đầu vào và URL mục tiêu hiện tại không thể kích hoạt một lỗ hổng. Do đó, công cụ ngay lập tức kết thúc episode hiện tại và bắt đầu một episode mới trên một cặp khác của URL và tham số chèn (dòng 18).

Nếu công cụ xác nhận được payload có xuất hiện, công cụ tiến hành phân tích phản hồi của nó và kiểm tra cách chuỗi đầu vào hợp lệ này xuất hiện trên trang web. Ở dòng 19, công cụ tính toán các đặc trưng của điểm chèn tại vị trí xuất hiện đầu tiên của payload hiện tại. Sau đó, công cụ thực hiện một vòng lặp đào tạo để đào tạo agent RL (từ dòng 20–28). Quá trình đào tạo này lặp lại nhiều bước, và mỗi bước trở thành một đơn vị thực thi mà agent RL thực hiện một hành động.

Đối với mỗi bước, agent sẽ lựa chọn một hành động dựa trên các quan sát (dòng 21). Sau đó, agent tấn công tạo ra hoặc biến đổi một payload tấn công tương ứng với hành động được chọn (dòng 22). Agent tấn công gửi một yêu cầu tấn công với payload vừa được tạo và xác minh xem yêu cầu gửi đi có thành công trong việc kích hoạt một lỗ hổng không (dòng 23–24). Công cụ sau đó quan sát phản hồi này và tính toán một vector đặc trưng của s_{t+1} (dòng 25).

Trong giai đoạn đào tạo, công cụ cũng tính toán một phần thưởng và huấn luyện agent để học một chính sách tối ưu làm tăng phần thưởng dự kiến (dòng 26–27). Khi (1) cuộc tấn công thành công hoặc (2) bước hiện tại đạt đến *STEPS_PER_EPISODE*, agent sẽ kết thúc tập hiện tại.

2.4.2.1.2 Hàm kiểm tra

Công cụ sử dụng hàm Test để tìm lỗ hổng XSS trong một tập hợp *tgt_set* cụ thể. Giai đoạn này tương tự như phần bên trong của hàm Train, ngoại trừ việc kiểm tra từng cặp trong *tgt_set* (dòng 9). Đối với `Episode` trong giai đoạn kiểm thử, công cụ không thực hiện mã được đánh dấu vì công cụ không cần đào tạo agent RL thông qua việc tính toán phần thưởng dự kiến.

2.4.2.1.3 Thuật toán RL

❖ **Actor-Critic (Advantage Actor-Critic / A2C)**

Thuật toán Actor-Critic là một agent học tăng cường kết hợp cả hai phương pháp tối ưu hóa giá trị và tối ưu hóa chính sách. Cụ thể hơn, Actor-Critic kết hợp cả thuật toán Q-learning và Policy Gradient. Thuật toán kết quả ở mức độ cao bao gồm một chu trình chia sẻ đặc điểm giữa:

- **Actor:** Một thuật toán Policy Gradient (PG) quyết định hành động cần thực hiện;
- **Critic:** Một thuật toán Q-learning đánh giá hành động mà Diễn viên đã chọn, cung cấp phản hồi về cách điều chỉnh. Nó có thể tận dụng các chiêu thức hiệu quả trong Q-learning, như đệm nhớ tái phát.

Hơn cả đó, thuật toán còn có hàm Advantage, nó giúp quá trình học ổn định hơn bằng cách nó sẽ thay hàm Giá trị hành động (Acton value funtion) làm Nhân xét. Nó tính toán và lựa chọn hành động tại một trạng thái so với giá trị trung bình của trạng thái đó bằng cách trừ đi giá trị trung bình của trạng thái trong cặp trạng thái-hành động đó.

$$A(s,a) = Q(s,a) - V(s)$$

Nói một cách khác, hàm này tính toán phần thưởng bổ sung mà chúng ta nhận được nếu chọn hành động này ở trạng thái đó so với giá trị trung bình của trạng thái đó.

Phần thưởng bổ sung là phần vượt qua giá trị kỳ vọng của trạng thái đó.

- Nếu $A(s,a) > 0$: gradient của chúng ta được đẩy theo hướng đó.
- Nếu $A(s,a) < 0$ (hành động của chúng ta làm kém hơn giá trị trung bình của trạng thái đó), gradient của chúng ta được đẩy theo hướng ngược lại.

Hàm này yêu cầu có 2 giá trị là hàm $Q(s,a)$ và hàm $V(s)$. Và ta có thể thay $Q(s,a)$ bằng hàm lỗi TD (Temporal Diference) để tính toán

$Q(s,a) \simeq r + \gamma V(s')$

➔ $A(s,a) = r + \gamma V(s') - V(s)$

❖ **Thuật toán RL**

Khi đang huấn luyện agent như được thể hiện ở dòng 27, chúng tôi tận dụng Advanced Actor-Critic (A2C). Chúng tôi huấn luyện hai mạng: mạng diễn xuất (actor) và mạng đánh giá (critic). Mạng diễn xuất ánh xạ mỗi trạng thái s thành một hành động a, đầu ra là một phân phối xác suất $\pi_\theta(a|s)$. Mạng đánh giá ánh xạ mỗi trạng thái s thành một giá trị chất lượng bằng cách tính toán $V_\theta(s)$. A2C huấn luyện $\pi_\theta(a|s)$ và $V_\theta(s)$ cùng một lúc theo hướng tối đa hóa hàm ưu đãi (advantage function), đại diện cho mức độ ưu đãi của việc thực hiện một hành động cụ thể a so với giá trị chất lượng của việc ở trong trạng thái s.

Chúng tôi định nghĩa hàm ưu đãi của chúng tôi như sau: $A(s_t,a_t) = Q(s_t,a_t) - V(s_t)$, trong đó $Q(s_t,a_t)$ tạo ra giá trị chất lượng của việc thực hiện hành động a_t tại trạng thái s_t, và $V(s_t)$ tính toán giá trị chất lượng của việc ở trong trạng thái s_t. Xem xét $Q(s_t,a_t) \simeq r_t + \gamma V(s_{t+1})$, hàm ưu đãi có thể được định nghĩa như sau:

$$A(s_t,a_t) = r_t + \gamma V(s_{t+1}) - V(s_t)$$

Ở đây, r_t là một phần thưởng cho hành động a_t mà agent chọn dựa trên trạng thái hiện tại s_t. Nói cách khác, r_t đại diện cho mức độ phù hợp của a_t đối với s_t để kích hoạt một lỗ hổng XSS. $V_\theta(s)$ đã được huấn luyện chỉ ra mức độ phù hợp của s_t trong việc tìm lỗ hổng XSS mục tiêu, và $\pi_\theta(a|s)$ cung cấp một phân phối xác suất của các hành động, đánh dấu quy tắc biến đổi hoặc tạo payload nào nên được lựa chọn đối với a_t tại s_t để tối đa hóa tổng phần thưởng R_t.

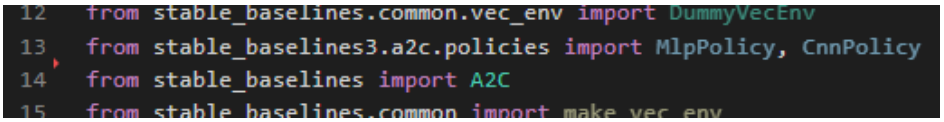
Để có thể huấn luyện mạng lưới diễn xuất và đánh giá, A2C sử dụng một hàm đối tượng *J* và tính toán tốc độ tăng dần của nó như sau:

$$J(\theta) = E[\sum_{\tau=0}^{T} \gamma^t r_t|\pi_\theta], \nabla J_\theta = \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta\,(a_t|s_t) A(s_t,a_t)$$

Với mỗ mạng lưới, chúng tôi sử dụng toàn bộ kết nối mạng lưới neural gồm 3 lớp, mỗi lớp có kích thước là 128. Chúng tôi sử dụng trình tối ưu hóa root mean square propagation (RMSProp) để huấn luyện các mạng.

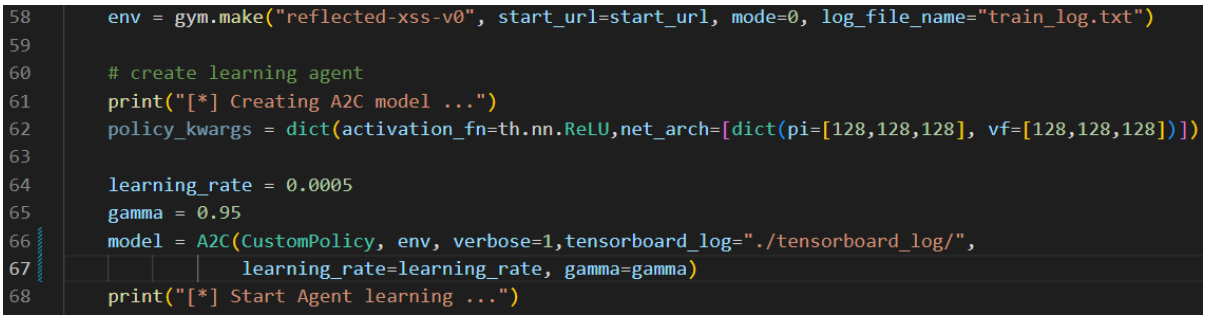
❖ **Phân tích mã nguồn**

Tiến hành phân tích mã nguồn của agent A2C trong công cụ Link.



Hình 2. 5 Mã nguồn A2C trong Link

Đầu tiên chương trình bắt đầu bằng việc khai báo và cấu hình một số thư viện cần thiết, trong đó có thuật toán **A2C** và 2 chính sách là **MlpPolicy (MLP)** và **CnnPolicy (CNN)** từ **module stable_baselines3** để huấn luyện. Tiếp theo, trong hàm main, các tham số dòng lệnh được xử lý để xác định URL khởi đầu (**start_url**) và số lượng bước học (**timesteps**). Môi trường học tập (**env**) được tạo ra thông qua thư viện '**gym**' và môi trường tùy chỉnh "reflected-xss-v0".



Hình 2. 6 Mã nguồn A2C trong Link

Ở đây, một môi trường "reflected-xss-v0" được tạo với các tham số như URL khởi đầu, chế độ (mode), và tên tệp log. Sau đó, một đối tượng A2C model được khởi tạo với chính sách tùy chỉnh (**CustomPolicy**), môi trường học tập, và các tham số khác như tốc độ học (**learning_rate**) và hệ số giảm giá (**gamma**). Mô hình sẽ được huấn luyện thông qua hàm learn với số lượng bước học (**total_timesteps**).

2.4.2.1.4 Huấn luyện tự động

Công cụ tự động hóa quy trình huấn luyện mà không cần sự tham gia của con người. Công cụ tận dụng "bug oracle" kiểm tra sự phản hồi của các payload tấn công và tấn công thành công, loại bỏ sự tham gia của con người.

2.4.2.2 Khởi tạo và biến đổi Payload

Công cụ được xây dựng hỗ trợ 2 loại hành động: khởi tạo (Generation) và biến đổi (Mutation).

Bảng dưới thể hiện tổng cộng 39 hành động mà agent RL có thể thực hiện để tạo ra các payload tấn công. Các hành động này được phân loại thành hai nhóm: 7 hành động đầu của quá trình khởi tạo và 32 hành động sau của quá trình biến đổi.

Type	Category	Action #	Description
Generation	Basic Payload	1	Generate the script tag payload (<script>alert(1);</script>)
		2	Generate a payload with a media tag (img, video, audio, svg) (e.g.,)
		3	Generate a payload in an event attribute (onerror, onload, onclick, onmouseover) (e.g., onerror=alert(1);)
		4	Generate a palyload in an src attribute (e.g., src='http://attack.js')
	JS Component	5–7	Generate the JS snippet (alert(1)), the JS URL (http://attack.js), the JS pseudo protocol (javascript:alert(1))
Mutation	Prefix	8–20	Prepend a symbol from /> , " , ' , "> , '> , > , --> , ;} , */ , '; , "; , enter , or dummy
		21	†Prepend a closing tag for an opening tag that appears immediately ahead of the injection point
	Suffix	22–24	Append <!-- , ' , " to the end of the payload
	Tag	25–27	Replace the tag with other script or media tag / Capitalize the tag / Overlap the tag string (e.g., scrscriptipt, etc.)
	Attribute	28	Capitalize the attribute
	JS snippet	29	Replace the JS snippet with one of the other JS snippets (alert(1), confirm(1), prompt(1))
		30	Split the JS snippet into several JS pieces (e.g., alert(1) ⇒ var A="al"+"er"+"t(1);";eval(A);)
		31–32	Prepend "+ / '+ and append +" / +' to the JS snippet (e.g., alert(1) ⇒ "+alert(1)+" / '+alert(1)+')
		33	Put the JS snippet in onerror, and force an error with throw (e.g., alert(1) ⇒ onerror=alert;throw 1)
	Entire string	34–36	Apply percent encoding / octet encoding / obfuscation with JSFuck [35]
		37–39	Replace a white space with a slash / a single quote with a back quote / a parenthesis (i.e., "(" or ")") with a back quote

† It can be incrementally combined with actions 8–20.

Bảng 2. 1 Bảng các hành động (action) của Link

2.4.2.2.1 Generation

❖ **Mô tả hàm hoạt đông**

Trong hàm chức năng khởi tạo payload của công cụ có hai nhóm hành động chủ yếu, bao gồm: tạo ra payload cơ bản và tạo ra thành phần JS.

Đối với việc tạo payload cơ bản, thành phần quản lý tấn công tạo ra một payload bằng cách sử dụng các phần tử HTML nhúng đoạn mã JS, như các thẻ script, thẻ phương tiện và thuộc tính. Ví dụ, đối với hành động thứ hai trong nhóm, công cụ tạo ra payload `<img src='x' onerror='alert(1);'/>` , chèn thẻ hình ảnh với mã JS. Lưu ý rằng chỉ có một quy tắc tạo ra payload được chọn ngẫu nhiên khi có nhiều quy tắc ứng dụng cho một hành động đã chọn. Ví dụ, khi quản lý tấn công thực hiện hành động thứ hai, một trong bốn thẻ phương tiện (tức là img, video, audio và svg) sẽ được chọn ngẫu nhiên cho việc tạo ra payload.

Đối với việc tạo ra thành phần JS, Link chuẩn bị một đoạn mã JS tấn công mà không có bất kỳ phần tử HTML nào cho phép thực thi đoạn mã JS này. Loại tạo ra này được thiết kế để giải quyết các trường hợp mà điểm chèn nằm trong một môi trường thực thi script (ví dụ: `<script>[payload]</script>`).

Tiếp theo, quá trình chia nhánh vào các hành động cụ thể. Mỗi hành động sẽ ảnh hưởng đến các thuộc tính như '**use_saved_tag**', '**original_tag**', và '**attack_src_attribute**'.

Các hành động cơ bản bao gồm sử dụng thẻ script (USING_SCRIPT_TAG) tạo payload theo mẫu 2 (PATTERN2_PAYLOAD), và tạo payload theo mẫu 3 (PATTERN3_PAYLOAD). Mỗi hành động sẽ gọi đến

các phương thức như action_0, action_1, và action_2 để thực hiện các biến đổi cụ thể.

2.4.2.2.2 Mutation

Thành phần quản lý tấn công biến đổi các thành phần trong payload tấn công trước đó dựa trên một hành động biến đổi mà agent đã chọn. Chúng tôi đã chuẩn bị sáu nhóm hành động biến đổi: tiền tố, hậu tố, thẻ, thuộc tính, đoạn mã JS, và thay đổi toàn bộ chuỗi. Mỗi hành động được thiết kế để thoát khỏi ngữ cảnh phản ánh của một payload (hành động 8-20, 21, 22-24 trong bảng hành động) và tránh sự việc sử dụng lại các yếu tố biến đổi cũ.

Nếu payload tấn công trước đó không chứa các yếu tố đại diện cho loại hành động hiện tại, thành phần quản lý tấn công sẽ không áp dụng hành động cho payload (tức là, dòng 22 của Thuật toán 1 được bỏ qua). Ví dụ, nếu agent quan sát payload `<script>alert(1);</script>` và sau đó chọn hành động của loại thuộc tính, quản lý tấn công sẽ không áp dụng hành động này do sự vắng mặt của một thuộc tính và gửi một yêu cầu với cùng một payload. Trong trường hợp này, công cụ sẽ giảm phần thưởng vì sử dụng lặp lại cùng một payload.

2.4.2.3 Các đặc trưng trạng thái

Type	Category	Feature #	Features	Range
Input Payload	Payload appearance	1–4	The presence of a "script" string / an "alert" string / a "(" or ")" string / a "dummy" string	0–1
		5–6	The presence of an event attribute (`onerror`, `onload`, `onclick`, `onmouseover`) / a URL attribute (`href`, `src`)	0–1
		7–9	The presence of a JS snippet (`alert(1)`, `confirm(1)`, `prompt(1)`) / a HTML comment / a JS comment	0–1
		10–11	The presence of JS pseudo protocol (`javascript:alert(1)`) / a JS URL (`http://attack.js`)	0–1
		12–14	The presence of an HTML tag / a script tag (`<script>`) / a media tag (`<img>`, `<video>`, `<audio>`, `<svg>`)	0–1
		15–21	The presence of a single quote / a double quote / a back quote / a back slash / a bracket / enter / a closing tag	0–1
		22–25	Whether it is `alert(1)`, `confirm(1)`, or `prompt(1)` / percent encoded / octet encoded / obfuscated	0–1
		26–28	The presence of a capitalized tag / a capitalized attribute / an overlapped tag string (e.g., `scrscriptipt`, etc.)	0–1
		29	The presence of a white space (e.g., `<img/src='x'/onerror='alert(1)'/>`)	0–1
	Payload repetitiveness	30	Payload frequency = *(# of times that the current payload is used in the current episode - 1) / TRAINING_EPISODE_SIZE)*	[0, 1]
		31-32	Action number used in the previous step / current step	Action #
Response	Reflected payload appearance	33	The presence of a script tag or a media tag from a payload in the response	-1–1
		34	The presence of a JS snippet, a JS URL, or a JS pseudo protocol from a payload in the response	-1–1
		35	String similarity between a payload and a reflected payload in the response	[0, 5]
	Payload† context information	36	Content type of a target page (HTML, CSS, JSON)	0–3
		37	The input parameter type from its default value (number, string)	0–2
		38	Where the payload is injected (tag, attribute name, attribute value, URL, comment, enclosed tag content, event value)	0–7
		39–40	A character symbol immediately before / after the injection point (', ", < or >, =, :, ;)	0–6
		41	Among the characters (", ', //, /*, <, >, −), the character placed closest before the injection point	0–7
		42	HTML tag type immediately before the point where the payload is injected (link, media)	0–2
		43–46	Reflection of initial payload	0–1
Attack	Attack Result	47	The success of an attempt	0–1

† 0 of the feature values refers to the default value that is assigned if no corresponding value is observed.

Bảng 2. 2 Bảng liệt kê các đặc trưng của trạng thái

Chúng tôi mô hình một trạng thái (s) như một vector đặc trưng ($f_1, f_2, \dots f_n$), mỗi phần tử của nó là một giá trị số biểu thị sự tồn tại hoặc loại của đặc trưng tương ứng. Gồm có 47 đặc trưng mà quản lý tấn công quan sát. Những đặc trưng này được phân loại thành năm nhóm: sự xuất hiện của payload (*appearance payload*), tính lặp lại của payload (*payload repetitiveness*), sự xuất

hiện của payload phản ánh (*reflected payload appearance*), ngữ cảnh của payload (*payload content*) và kết quả của tấn công (*attack result*).

❖ **Payload appearance**

Danh mục này gồm 29 đặc trưng được thiết kế để kiểm tra sự có mặt của các chuỗi cụ thể trong một payload đầu vào thử nghiệm. Mỗi chuỗi được chọn đại diện cho một cách để chèn một đoạn mã JavaScript; bằng cách kiểm tra sự có mặt của nó, Link kiểm tra phương pháp chèn nào đã được sử dụng để trích xuất trạng thái tiếp theo của agent RL. Cụ thể, các đặc điểm này kiểm tra xem một payload đầu vào hiện tại có sử dụng một thẻ cụ thể hay không (ví dụ: `<script>` và `<img>`), chứa một đoạn mã JavaScript (ví dụ: `alert(1)` và `confirm(1)`), hoặc sử dụng các thẻ viết hoa. Giá trị của mỗi đặc điểm là 0 hoặc 1, chỉ ra sự có hoặc không có của chuỗi tương ứng.

❖ **Payload repetitiveness**

Danh mục đặc trưng này đại diện cho sự lặp lại của các payload đầu vào, để kiểm tra việc agent tránh sử dụng cùng một payload tấn công. Đặc trưng thứ 30 đại diện cho tần suất mà payload đầu vào hiện tại đã được sử dụng trong tập hợp hiện tại. Khi đặc trưng này là dương, thành phần quản lý tấn công cung cấp một phần thưởng tiêu cực cho một hành động được chọn. Đặc trưng thứ 31 và 32 đại diện cho chỉ số hành động trước đó tại a_{t-1} và chỉ số hành động hiện tại tại a_t mà agent đã chọn.

❖ **Reflected payload appearance**

Các đặc trưng trong danh mục này mô tả cách payload đã được thử nghiệm được phản ánh trong phản hồi của nó. Các đặc trưng thứ 33 chỉ ra xem một thẻ script (tức là `<script>`) hay thẻ media (ví dụ: <img>) có trong payload hiện tại có xuất hiện như cùng một thẻ trong phản hồi hay không. Đối với điều này, thành phần quản lý tấn công phân tích các phản hồi và sau đó kiểm tra xem thẻ tương ứng có xuất hiện ở điểm chèn không. Công cụ có khả năng xác định chính xác vị trí điểm tiêm thông qua quá trình khởi tạo tập hợp trong dòng 14 đến 19.

Đặc trưng thứ 34 biểu thị cho việc liệu một đoạn mã JavaScript (ví dụ: `alert(1)`), URL JavaScript hoặc giao thức JavaScript giả trong payload có xuất hiện trong phản hồi hay không. Đặc điểm thứ 35 cho biết độ tương đồng giữa 2 chuỗi payload đã thử nghiệm và payload được phản ánh trong phản hồi. Để so khớp độ tương tự, chúng tôi đã sử dụng mẫu gestalt khớp với thuật toán trong difflib. Ví dụ: nếu payload được tạo và thử nghiệm là `<script>alert(1);</script>` và payload được phản ánh là

<>alert(1);</>, các tính năng thứ 33, 34 và 35 trở thành -1, 1 và 3.5, tương ứng. Ngay cả khi lỗ hổng XSS không được kích hoạt thông qua hành động hiện tại tại , những các đặc điểm này tiết lộ rằng phản ánh của tải đầu vào đã được đạt được một phần.

❖ **Payload content**

Danh mục đặc điểm này đại diện cho ngữ cảnh của một điểm chèn trong phản hồi, mà công cụ đã xác định trong bước khởi tạo ở dòng 19. Lưu ý rằng các đặc điểm này được thiết kế để thu thập thông tin ngữ cảnh xung quanh một điểm chèn.

Đặc điểm thứ 36 chỉ ra loại phản hồi trong ba định dạng (HTML, CSS và JSON). Đặc điểm thứ 37 chỉ định kiểu giá trị mặc định của tham số đầu vào. Để thu thập thông tin này,trình quán lý tấn công sử dụng kết quả đàm phán để kiểm tra xem tham số có giá trị mặc định hay không. Các đặc điểm từ thứ 39 đến 41 đều đề cập đến thông tin cần thiết cho việc trước khi thoát và sau khi thoát khỏi điểm chèn. Đặc điểm cuối cùng trong danh mục này là sự phản ánh của một payload đầu vào hợp lệ để quyết định kết thúc cặp mục tiêu hiện tại.

❖ **Attack result**

Đặc điểm này chỉ ra xem lỗ hổng có được kích hoạt bởi hành động tại a_t hay không, thông qua việc kiểm tra thông tin được truyền bởi bảng chấm công cụ bug (bug oracle).

2.4.2.4 Tính phần thưởng

Trình quản lý tấn công tính toán một phần thưởng r_t cho một hành động a_t mà agent RL chọn dựa trên trạng thái hiện tại s_t. Hành động hiện tại a_t chi phối payload đầu vào tiếp theo để thử nghiệm và ảnh hưởng đến trạng thái tiếp theo s_{t+1}. Phần thưởng r_t tính đến trạng thái tiếp theo s_{t+1} để đánh giá xem a_t có phải là một hành động tốt tăng cường phần thưởng kỳ vọng hay không.

Algorithm 2: Algorithm for computing a reward r_t at a time step t.

```
previous_payload ← Payload produced in the previous step
function Reward(t, s_{t+1}, payload, found)
    r_t ← 0
    if found then
        r_t ← STEPS_PER_EPISODE − t
    // If the current action (#32) is the same with the previouse action
       (#31), give a negative reward.
    if s_{t+1}[32] == s_{t+1}[31] then
        r_t ← r_t − 1
    if payload == previous_payload then
        r_t ← r_t − 1
    // If the input payload is frequently used (#30), give a negative
       reward.
    r_t ← r_t − s_{t+1}[30]
    return r_t
```

Hình 2. 7 Thuật toán tính phần thưởng

Thuật toán 2 mô tả cách chương trình tính toán một r_t. Chúng tôi đã thiết kế hai loại phần thưởng: (1) một phần thưởng tích cực để chỉ độ chương trình kiểm thử kết thúc sớm đến mức nào và (2) một phần thưởng tiêu cực để chỉ sự lặp lại của một hành động được chọn. Cụ thể, khi a_t kích hoạt thành công khai thác, phần thưởng trở thành sự khác biệt giữa số bước tối đa (STEPS_PER_EPISODE) đến thời điểm hiện tại t, do đó thúc đẩy agent tìm kiếm một payload tấn công phù hợp càng sớm càng tốt. Ngược lại, dongf7-10, thuật toán giảm r_t đi một khi a_t là giống như a_{t-1} hoặc khi payload đầu vào hiện tại giống payload đầu vào trước đó, hướng dẫn agent tránh hành động giống nhau.

❖ **Bug oracle.**

Hàm BugOracle theo dõi các payload phản ánh đầu vào trong các phản hồi để xác định liệu các payload đầu vào có kích hoạt thành công lỗ hổng XSS phản ánh hay không. Chúng tôi thiết kế BugOracle để kiểm tra sự có mặt của đoạn mã JavaScript thực sự được chèn, thay vì các thẻ HTML phản ánh không có ý nghĩa, giảm thiểu các kết quả dương giả. Đặc biệt, thuật toán xác thực thay đổi tùy thuộc vào hành động thử nghiệm cuối cùng trong nhóm tạo ra (hành động 1–7 trong bảng hành động). Ví dụ, khi agent thực hiện các hành động 1, 20 và 25, BugOracle sử dụng kỹ thuật xác thực cho hành động 1. Lý do là trong khi tất cả các hành động biến đổi được thiết kế để hoặc bypass logic vệ sinh không

chính xác hoặc thoát khỏi các giá trị đầu vào bị ràng buộc, mỗi hành động tạo ra quyết định cách mã JS chèn vào sẽ được thực thi.

❖ **Hành động 1–2**

Hành động đầu tiên và thứ hai được thiết kế để chèn thẻ script và thẻ media, tương ứng. Đối với hành động đầu tiên, BugOracle kiểm tra xem có thẻ script nào chứa đoạn mã JS `alert(1);` không. Đối với hành động thứ hai, nó kiểm tra xem có thẻ media nào chứa cặp giá trị thuộc tính `onerror=alert(1);` không.

❖ **Hành động 3–4**

Hành động thứ ba và thứ tư được thiết kế để xử lý phản ánh của payload đầu vào dưới dạng <[thẻ] [payload]. BugOracle xác định xem thẻ nào được sử dụng trước điểm chèn. Sau đó, nó kiểm tra xem thẻ này có cặp giá trị thuộc tính nào đó, trong đó giá trị là `alert(1);` và thuộc tính nằm trong `onerror`, `onclick`, `onmouseover`, và `onload` hay không. Nếu không, nó kiểm tra xem giá trị src của thẻ này có phải là `http://attack.js` hay không.

❖ **Hành động 5–7**

Các hành động 5–7 được thiết kế để chèn mã JS dưới dạng <[thẻ] [thuộc tính]=[payload] hoặc <script>[payload]</script>. Để kiểm tra hành động đầu tiên, quản lý tấn công truy xuất [thẻ] tại điểm chèn. Sau đó, nó kiểm tra xem thẻ được xác định này có thuộc tính sự kiện nào đó (ví dụ: `onerror`, `onload`, `onclick`, hoặc `onmouseover`) cho hành động 5 hoặc thuộc tính URL (src hoặc href) cho hành động 6–7. Sau đó, nó xác nhận xem payload đầu vào có được chèn vào giá trị của thuộc tính này hay không. Để xác thực hành động thứ hai, quản lý tấn công quét qua mỗi thẻ script trên toàn bộ phản hồi và sau đó kiểm tra xem nội dung của thẻ script được xác định có chứa payload hay không.

2.4.3 Đánh giá kết quả

Chúng tôi đánh giá Link trong việc tìm các lỗ hổng XSS reflected bằng cách so sánh nó với các công cụ quét hiện có. Chúng tôi cũng phân tích khả năng của Link trong việc phát hiện lỗ hổng XSS phản ánh trong 12 ứng dụng thực tế.

2.4.3.1 Thiết lập môi trường thử nghiệm

❖ **Benchmarks**

Chúng tôi đã chọn các bộ kiểm thử của các ứng dụng máy chủ có lỗ hổng XSS reflected từ Firing-Range WAVSEP, và các bộ kiểm thử của OWASP. Lưu ý rằng những bộ kiểm thử này đã được sử dụng để kiểm tra khả năng của các

công cụ quét web trong việc tìm kiếm các lỗi bảo mật khác nhau. Trong số các ứng dụng thuộc các bộ kiểm thử này, chúng tôi đã chọn các ứng dụng có ít nhất một lỗ hổng XSS reflected. Bộ kiểm thử của chúng tôi bao gồm 351 ứng dụng máy chủ trong PHP và Java Server Pages (JSP); 45, 60 và 246 ứng dụng lần lượt từ Firing Range, WAVSEP và OWASP.

Chúng tôi cũng đã chuẩn bị 12 ứng dụng web PHP được liệt kê trong cột đầu tiên của Bảng 3 để thể hiện hiệu suất của Link trong việc tìm kiếm lỗ hổng trong các ứng dụng thực tế. Chúng tôi đã lựa chọn những ứng dụng này từ hai nguồn: (1) các ứng dụng web mã nguồn mở có lỗ hổng XSS được công cụ quét Netsparker xác định; và (2) các bộ dữ liệu đánh giá từ các nghiên cứu trước đó.

❖ **Huấn luyện và kiểm thử**

Để huấn luyện agent, chúng tôi đã sử dụng các bộ kiểm thử Firing-Range, WAVSEP và Filter Evasion. Chúng tôi đã sử dụng các thuật toán huấn luyện Proximal Policy Optimization (PPO), Deep Q-Network (DQN), và A2C để đánh giá so sánh và thiết lập A2C làm cơ sở cho Link. Đối với các đánh giá còn lại, chúng tôi đã huấn luyện mô hình với các siêu tham số được đặt là 3.5 triệu MAX_STEPS, 500 STEPS_PER_EPISODE, hệ số giảm 0.95, và tốc độ học 0.0005, cho hiệu suất tốt nhất. Tổng thời gian cần cho việc huấn luyện là 39.5 giờ. Trong giai đoạn kiểm thử, chúng tôi đã thiết lập STEPS_PER_EPISODE bằng 30 dựa trên quan sát thực nghiệm, hạn chế số lần thử tấn công của agent.

2.4.3.2 So sánh với các công cụ quét kiểm thử khác

Chúng tôi đã so sánh khả năng của Link trong việc tìm lỗ hổng XSS reflected trong các bộ kiểm thử Firing-Range, WAVSEP, OWASP và Filter Evasion với bốn công cụ quét web hiện đại: Burp Suite Pro, Wapiti, OWASP ZAP, và Black Widow (tức BW). Chúng tôi cũng thiết lập Link với một chính sách ngẫu nhiên (Link-R), tức là chọn hành động ngẫu nhiên thay vì hành động của agent để thể hiện hiệu suất của agent. Để có sự so sánh công bằng, chúng tôi đặt các công cụ khác để chỉ tìm các lỗ hổng XSS reflected, ngăn chúng gửi yêu cầu cho các loại lỗ hổng khác. Vì Link sử dụng một chính sách ngẫu nhiên, cho mỗi thiết lập thử nghiệm, chúng tôi đã thực hiện năm giai đoạn kiểm thử và báo cáo giá trị.

Benchmark	# of Bugs	Tools	TP	FP	FN	Time	# of Attempts
Firing-Range	45	**Link**	**45**	0	0	8s	459
		Link-R	20	0	25	55s	1,351
		Wapiti	37	0	8	8s	447
		Burp	38	0	7	20s	4,344
		ZAP	43	0	2	17s	1,530
		BW	23	0	22	39s	218
WAVSEP	60	**Link**	**60**	0	0	5s	718
		Link-R	28	0	32	55s	1,871
		Wapiti	30	0	30	11s	1,782
		Burp	54	0	6	5s	13,859
		ZAP	52	0	8	11s	4,717
		BW	13	0	47	5m 36s	577
OWASP Benchmark	246	**Link**	**213**	0	33	1m 17s	11,912
		Link-R	178	0	68	5m 47s	16,457
		Wapiti	137	0	109	59s	6,451
		Burp	186	0	60	1m 58s	121,311
		ZAP	186	0	60	1m 30s	29,483
		BW	157	0	89	137m 44s	10,759
Filter Evasion	25	**Link**	**25**	0	0	4s	334
		Link-R	6	0	19	30s	885
		Wapiti	17	0	8	3s	505
		Burp	22	0	3	1s	1,852
		ZAP	6	0	19	5s	857
		BW	12	0	13	8m 4s	602
Total	376	**Link**	**343**	0	**33**	1m 34s	13,423
		Link-R	232	0	144	8m 7s	20,564
		Wapiti	221	0	155	1m 21s	9,185
		Burp	300	0	76	2m 24s	141,366
		ZAP	287	0	89	2m 3s	36,587
		BW	205	0	171	152m 3s	12,156

Bảng 2. 3 Bảng so sánh kết quả thử nghiệm của Link với các công cụ khác

Bảng trên tóm tắt kết quả thử nghiệm. Link đã tìm thấy 343 kết quả dương tính có kích hoạt XSS thực sự (**TPs**) (chiếm 91,2%) với 33 kết quả âm tính sai (**FNs**) trong tất cả các bộ kiểm thử, đạt được hiệu suất tốt nhất so với tất cả các công cụ khác về việc tối đa hóa số lượng **TPs** và giảm thiểu số lượng **FNs**. Trong số 343 kết quả dương tính thực sự (**TPs**), chúng tôi đã phân tích 181 **TPs** mà các công cụ khác không thể xác định và báo cáo chúng là **FNs**. Hiệu suất vượt trội được báo cáo của Link xuất phát từ ba nguyên nhân:

- (1) Link có thể tạo ra các payload khai thác thích hợp mà các công cụ khác không thể tạo ra.
- (2) Link sử dụng bảng chấm điểm lỗi chính xác hơn để xác định các payload phản ánh

- (3) Các công cụ khác bỏ qua một số loại tham số đầu vào do các heuristics của chúng.

Reason	Wapiti	Burp	ZAP	BW	Total
Reason (1): ineffective input payloads	53	29	49	73	204
Reason (2): incomplete bug oracle	13	6	7	0	26
Reason (3): unidentified input parameters	56	8	0	65	129
Total	122	43	56	138	181

Bảng 2. 4 Bảng nguyên nhân của FNs của các công cụ kiểm thử

Bảng trên trình bày nguyên nhân gốc của FNs mà các công cụ khác báo cáo. Chúng tôi quan sát thấy rằng các công cụ quét khác bỏ qua nhiều lỗ hổng do nguyên nhân (1) được mô tả trước đó, điều này chứng minh rằng khả năng của RL agent tạo ra các đầu vào payloads đóng góp lớn vào việc tìm kiếm nhiều lỗ hổng. Chúng tôi cũng lưu ý rằng Wapiti đã tạo ra 56 FNs do nguyên nhân (3). Kết quả này cho thấy rằng mặc dù Link được triển khai trên nền tảng của Wapiti, thuật toán lựa chọn tham số đầu vào của Link đóng góp lớn vào việc tìm kiếm nhiều lỗ hổng hơn..

Hiệu suất của Link nổi bật so với các công cụ quét khác đối với bộ kiểm thử Filter Evasion. Điều này xuất phát từ sự hiệu quả của agent, kiểm tra cách mà payload đầu vào được phản ánh trong phản hồi và đề xuất các hành động tối ưu để vượt qua logic làm sạch đầu vào không đúng. Ngược lại, các công cụ quét khác thường có hiệu suất tương đối thấp vì chúng không có các payload khai thác chức năng trong từ điển của mình để xác định FNs. Một giải pháp là đưa các payload này vào từ điển của các công cụ quét hiện tại, nhưng các payload này là hiếm khi xuất hiện vì chúng đặc biệt cho các bộ kiểm thử mục tiêu. Điều này cuối cùng dẫn đến một chiến dịch quét không có khả năng mở rộng được.

❖ **Hiệu suất**

Cột thứ bảy và thứ tám của bảng thể hiện thời gian thực thi và số lượng yêu cầu thử nghiệm cho mỗi công cụ, tương ứng. Chúng tôi quan sát thấy rằng Wapiti có số lần thực thi thấp nhất vì nó có một số lượng payload được định nghĩa trước nhỏ và bỏ qua một số loại tham số đầu vào. Những payload và tham số hạn chế này đáng kể đã làm suy giảm khả năng tìm kiếm lỗ hổng của Wapiti so với các công cụ quét khác. Black Widow tập trung vào việc tối đa hóa phạm vi thu thập dữ liệu thay vì tạo ra các payload thích hợp. Do đó, nó sử dụng một kích thước nhỏ cho từ điển payload, dẫn đến việc báo cáo một số lượng TP nhỏ. Chúng tôi lưu ý rằng Link vượt trội hơn các phương pháp khác về thời gian thực thi, số lượng yêu cầu và số lỗ hổng được tìm thấy. Link chỉ cần 13,423 yêu cầu

để xác định 343 TPs, gửi ít yêu cầu hơn rất nhiều so với công cụ quét hiệu suất tốt thứ hai (tức là Burp). Chúng tôi cũng quan sát thấy rằng Wapiti và Burp gửi nhiều yêu cầu tấn công với các payload khác nhau cho một số ứng dụng. Ngược lại, Link tạo ra các payload hoạt động chỉ sau một số lần thử nghiệm. Ví dụ, đối với một bộ kiểm thử Filter Evasion, Link, Wapiti và Burp thử nghiệm lần lượt 9, 37 và 26 payloads khác nhau. Đối với ứng dụng này, ZAP báo cáo một false negative. Điều này có nghĩa là Link có khả năng tạo ra các payload thích hợp mà không cần thử tất cả các payload trong từ điển tấn công.

❖ **Tác động của các đặc trưng trạng thái**

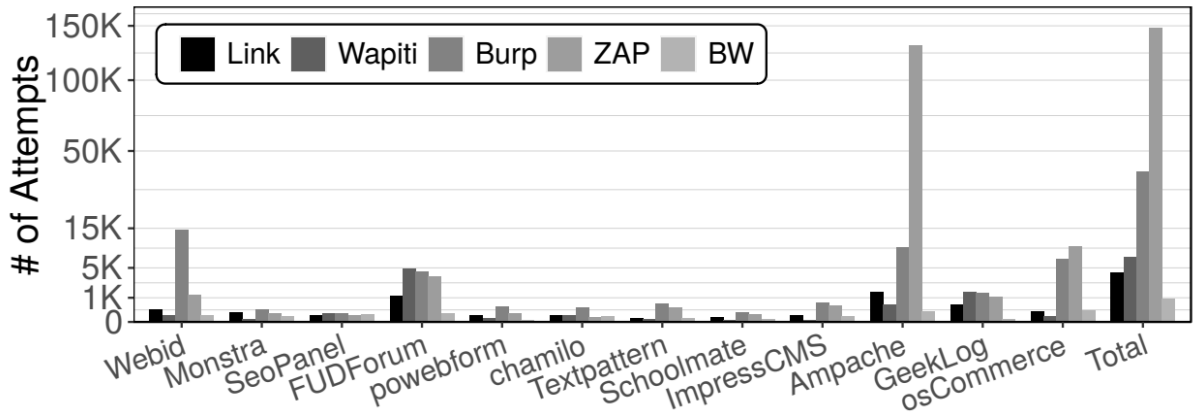
Chúng tôi đã tiến hành các đánh giá tầm quan trọng của mỗi đặc trưng của trạng thái. Đối với mỗi đặc trưng, chúng tôi đã buộc Link không sử dụng nó bằng cách cố định giá trị của nó thành giá trị ban đầu, sau đó đo lường số lượng TPs. Chúng tôi quan sát thấy rằng các đặc trưng thứ 32, thứ 7, thứ 41 và thứ 38 đóng góp nhiều nhất vào việc giảm số lượng TPs lần lượt là 48%, 18%, 12% và 9%. Hành động hiện tại, việc sử dụng đoạn mã JS trong payload, việc chuyển đổi ký tự đặc biệt (escaping characters) và loại điểm nhập mà đặc trưng ảnh hưởng chủ yếu đến việc tìm kiếm số lượng TPs.

2.4.3.3 Kiểm thử lỗ hổng trong môi trường thực tế

Chúng tôi đã đánh giá khả năng của Link trong việc tìm lỗ hổng trong 12 ứng dụng thực tế có tổng cộng 49 lỗ hổng XSS. Giống như phần trước, trong thí nghiệm này, chúng tôi so sánh khả năng phát hiện của Link so với bốn công cụ quét web hiện tại với lượng thời gian là ba giờ. Bảng 3 và Hình 3 thể hiện kết quả phát hiện và số lượng yêu cầu mà mỗi công cụ thử nghiệm đã thử, tương ứng. Link có số lượng TPs thứ hai cao nhất (tức là 43 so với 33 của Wapiti, 46 của Burp, 36 của ZAP và 25 của Black Widow), trong khi báo cáo số lượng yêu cầu tấn công thấp nhất thứ hai (tức là 4,105 so với 7,175 của Wapiti, 38,622 của Burp, 147,595 của ZAP và 879 của Black Widow).

Application (Version)	Link			Wapiti			Burp			ZAP			BW		
	TP	FP	FN	TP	FP	FN	TP	FP	FN	TP	FP	FN	TP	FP	FN
Webid (1.2.2)	10	0	0	10	0	0	10	0	0	10	0	0	0	0	10
Monstra (3.0.4)	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0
SeoPanel (4.8.0)	2	0	0	2	0	0	2	0	0	2	0	0	2	0	0
FUDForum (3.1.0)	3	0	0	3	0	0	3	0	0	3	1	0	1	0	2
powebform (1.0.3)	13	0	0	12	0	1	13	0	0	12	0	1	12	0	1
chamilo (1.11.14)	1	0	0	1	0	0	0	0	1	1	0	0	1	0	0
Textpattern (4.5.5)	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1
Schoolmate (1.5.4)	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0
ImpressCMS (1.3.10)	5	0	3	0	0	8	8	0	0	0	0	8	3	0	5
Ampache (4.4.2)	4	0	2	1	0	5	5	2	1	4	0	2	2	0	4
GeekLog (2.2.1)	3	0	0	2	0	1	3	1	0	2	0	1	2	0	1
osCommerce (2.3.3)	0	0	0	0	0	0	0	0	0	0	18	0	0	0	0
Total	43	0	6	33	0	16	46	3	3	36	19	13	25	0	24

Bảng 2. 5 Bảng so sánh số lỗ hổng thực tế tìm được với các công cụ kiểm thử



Bảng 2. 6 Bảng số lượng yêu cầu giữa các công cụ trên các ứng dụng

Chúng tôi quan sát thấy rằng Link đã tìm thấy nhiều lỗ hổng thực tế với ít yêu cầu hơn. Trong đó agent của Link được huấn luyện trên các bộ kiểm thử Firing-Range, WAVSEP và Filter Evasion, khác biệt so với các ứng dụng đang được thử nghiệm; và Link vẫn báo cáo số lượng TPs thứ hai cao nhất.

Burp tìm thấy nhiều lỗ hổng hơn so với Link do khả năng thu thập url của Burp, bao gồm đoán tên và suy luận từ các quy ước đặt tên. Việc thiết kế một crawler tiên tiến hơn để tìm URL và các tham số đầu vào không phải là một mục tiêu mà Link đang giải quyết. Thay vào đó, chúng tôi tập trung vào việc điều chỉnh các payload bằng cách sử dụng RL. Ngoài ra, việc tận dụng crawler của Burp không khả thi cho Link vì Burp là một công cụ thương mại và mã nguồn của nó không có sẵn để kiểm tra và sửa đổi.

Khi chỉ xem xét các tham số đầu vào và URL mà crawler của Link đã tìm thấy, Link có khả năng tìm thấy mọi lỗ hổng mà Burp cũng đã tìm thấy, nhưng với số lượng yêu cầu ít hơn nhiều. Những kết quả thử nghiệm này chứng minh rằng việc tận dụng RL để điều chỉnh các payload tấn công là một phương pháp triển vọng trong việc thực hiện kiểm thử xâm nhập.

❖ **Lỗ hổng zero-day**

Với các lỗ hổng zero-day, chúng tôi tiến hành thêm kiểm thử đối với các phiên bản mới nhất của các ứng dụng đã có báo cáo CVE trong quá khứ. Link báo cáo ba lỗ hổng mới trong Geeklog (v2.2.1sr1) và một trong PESCMS (v2.3.3) (CVE-2021-44884).

2.4.4 Hạn chế trong kiểm thử

Chúng tôi giải quyết vấn đề điều chỉnh các đầu vào payloads bằng cách sử dụng RL. Tuy nhiên, còn nhiều khía cạnh khác quyết định hiệu suất của các công cụ quét web black-box: (1) vấn đề phạm vi thu thập URL mục tiêu để tấn công, (2) xác định tất cả các tham số đầu vào để chèn payloads, và (3) ưu tiên hóa các tham số đầu vào để tấn công. Vì Link được xây dựng trên nền tảng của Wapiti, crawler và các mô-đun để xác định và ưu tiên hóa các tham số đầu vào tương tự. Tuy nhiên, như chúng tôi đã chứng minh trong quá trình đánh giá, việc tạo ra các payloads bằng cách sử dụng RL đã đóng góp vào việc Link tìm thấy 12 TPs bổ sung mà tất cả các công cụ quét trong đánh giá không thể tìm thấy.

Chúng tôi nhấn mạnh khả năng của Link trong việc giảm số lần thử tấn công để kích hoạt các lỗ hổng mục tiêu. Thường xuyên, các chiến dịch kiểm thử của các công cụ quét web black-box mất rất nhiều thời gian thực thi. Do đó, việc đạt được cả hai mục tiêu là giảm thiểu số lượng yêu cầu và tối đa hóa số lượng kết quả dương là các nhiệm vụ quan trọng đối với những công cụ này. Tuy nhiên, các công cụ có sẵn hiện nay tập trung vào việc triển khai các thuật toán heuristics của riêng mình. Chúng tôi lập luận rằng cách tiếp cận sử dụng RL là một hướng đi mới và hiệu quả để giải quyết vấn đề này, đồng thời tránh triển khai các quy tắc heuristics tùy ý.

Chúng tôi kiểm tra hiệu suất của việc tận dụng một trình duyệt Chrome headless để kiểm tra việc thực thi script. Tuy nhiên, chúng tôi quan sát thấy rằng việc sử dụng một trình duyệt headless đã làm tăng thêm 20% thời gian huấn luyện và kiểm thử. Vì lý do này, chúng tôi chọn phân tích các phản hồi HTML và JS vì phân tích không chỉ được yêu cầu để xác định việc khai thác thành công mà còn để trích xuất thông tin quan sát. Lưu ý rằng Link tận dụng một agent RL hoàn toàn tự động trong quá trình huấn luyện và kiểm thử. Khác với các phương

pháp trước đó, Link không đòi hỏi sự tham gia của con người để đánh giá các hành động được chọn trong môi trường cụ thể.

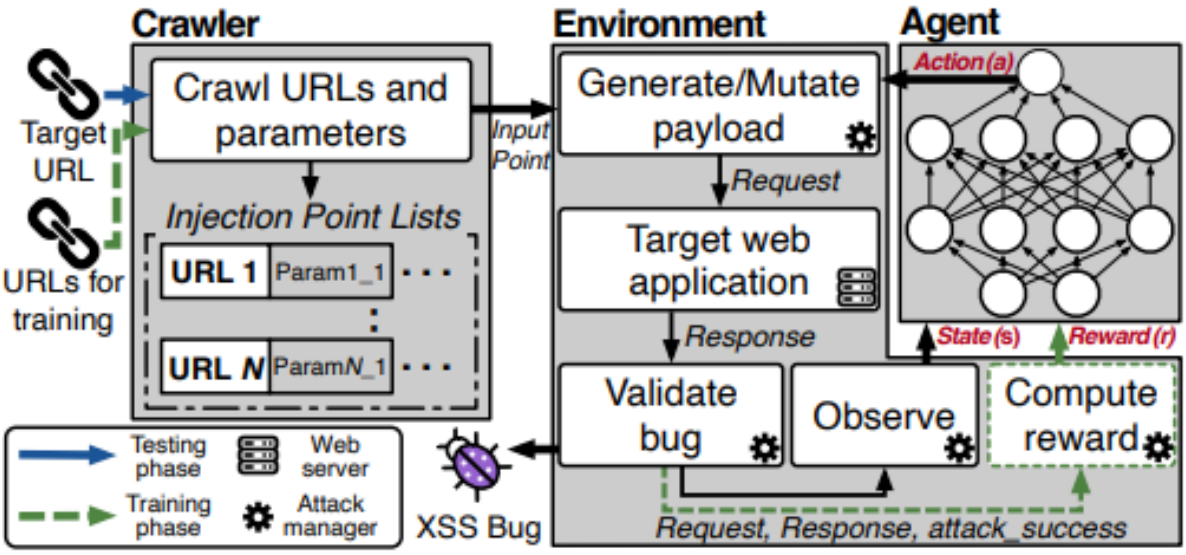
2.5 Kết luận chương 2

Link được coi là công cụ quét web black-box đầu tiên sử dụng một agent RL không đòi hỏi sự tham gia của con người. Link sử dụng RL để điều chỉnh các payload tấn công của mình dựa trên ngữ cảnh mục tiêu có lỗ hổng, giúp tránh yêu cầu tấn công không hiệu quả và giảm số lượng các phản hồi âm tính giả (FNs). Link thể hiện hiệu suất xuất sắc của mình trong việc giảm số lượng yêu cầu tấn công trong khi tìm thấy nhiều TPs hơn và ít FNs hơn so với công cụ cơ sở của nó, Wapiti. Kết quả thử nghiệm của chúng tôi chứng minh rằng sử dụng RL để xác định lỗ hổng XSS phản ánh là một hướng đi triển vọng cho kiểm thử xâm nhập có khả năng mở rộng và chính xác.

CHƯƠNG 3 THỰC NGHIỆM MÔ HÌNH VÀ KẾT QUẢ

3.1 Mô tả thực nghiệm

Mục đích thực nghiệm: Áp dụng và đánh giá được hiệu quả của việc phát hiện lỗ hổng phần mềm XSS ánh xạ sử dụng học sâu tăng cường, tiến hành đánh giá khả năng của phương pháp này trong việc phát hiện các lỗ hổng bảo mật XSS, đồng thời đánh giá khả năng ổn định và chính xác của hệ thống.



Hình 3. 1 Luồng hoạt động của công cụ

Quy trình thực nghiệm được mô tả chung thành 8 bước sau:

Bước 1: Tiến hành truyền vào URL của website mục tiêu hoặc một danh sách các URL huấn luyện, Crawler sẽ tiến hành dò quét, phân tích và thu thập thông tin, tìm ra các URL con khác hoặc các tham số ẩn từ URL đã nhận được.

Bước 2: Sau khi tìm được URL con hoặc tham số, Crawler sẽ chuyển tất cả các URL và tham số tìm được thành dãy điểm tiêm và lưu chúng vào một danh sách. Sau khi quá trình dò quét và thu thập hoàn tất, Crawler sẽ tiến hành lần lượt từng điểm tiêm vào môi trường.

Bước 3: Môi trường tiến hành xử lý, phân tích URL chứa điểm tiêm đã được nhập vào sau đó gửi yêu cầu URL đó tới website mục tiêu.

Bước 4: Khi nhận được phản hồi từ website, môi trường sẽ tiến hành phân tích phản hồi đó và xác nhận lỗi. Sau đó, nó sẽ gửi kết quả phân tích phản hồi đó tới hàm quan sát và hàm tính toán phần thưởng (giai đoạn huấn luyện).

Bước 5:

- Hàm quan sát dựa vào kết quả để tính toán ra lựa chọn ra đặc trưng trạng thái và tính ra trạng thái s rồi gửi đến agent.
- Nếu trong giai đoạn huấn luyện huấn luyện, hàm tính phần thưởng dựa vào kết quả để tính toán ra phần thưởng r cho agent rồi gửi tới agent.

Bước 6: Khi agent nhận được trạng thái s và phần thưởng r, sẽ tiền hành phần tích, tính toán ra hành động a và gửi tới hàm tạo hoặc đột biến payload.

Bước 7: Khi hàm khởi tạo hoặc đột biến nhận được hành động được gửi từ agent sẽ tiền hành xử lý và chèn vào điểm tiêm. Sau đó gửi yêu cầu tới website.

Bước 8: Quá trình huấn luyện diễn ra lặp đi lặp lại cho tới khi môi trường xác nhận được lỗi có chứa trong phản hồi trả về từ URL. Nó sẽ ghi lại lỗi và chuyển sang URL tiếp theo nếu có.

Kịch bản thực nghiệm: Tiến hành huấn luyện agent trên môi trường thực nghiệm Anaconda và nhận URL của webserver mục tiêu huấn luyện OWASP Benchmark. Từ đó phât tích kết quả và đánh giá.

Yêu cầu thức nghiệm:

- Sử dụng công cụ lập trình Python để tiến hành thực nghiệm.
- Cài đặt môi trường thực nghiệm:
  - Sử dụng công cụ Anaconda.
  - Webserver huấn luyện OWASP Benchmark Java.
  - Python phiên bản 3.7.16.
  - Gym phiên bản 0.26.2.
  - Tensorflow phiên bản 1.14.0.
  - Stable_baselines phiên bản 2.10.0.
  - Stable_baselines3 phiên bản 2.0.0.
  - Jdk và gói maven.

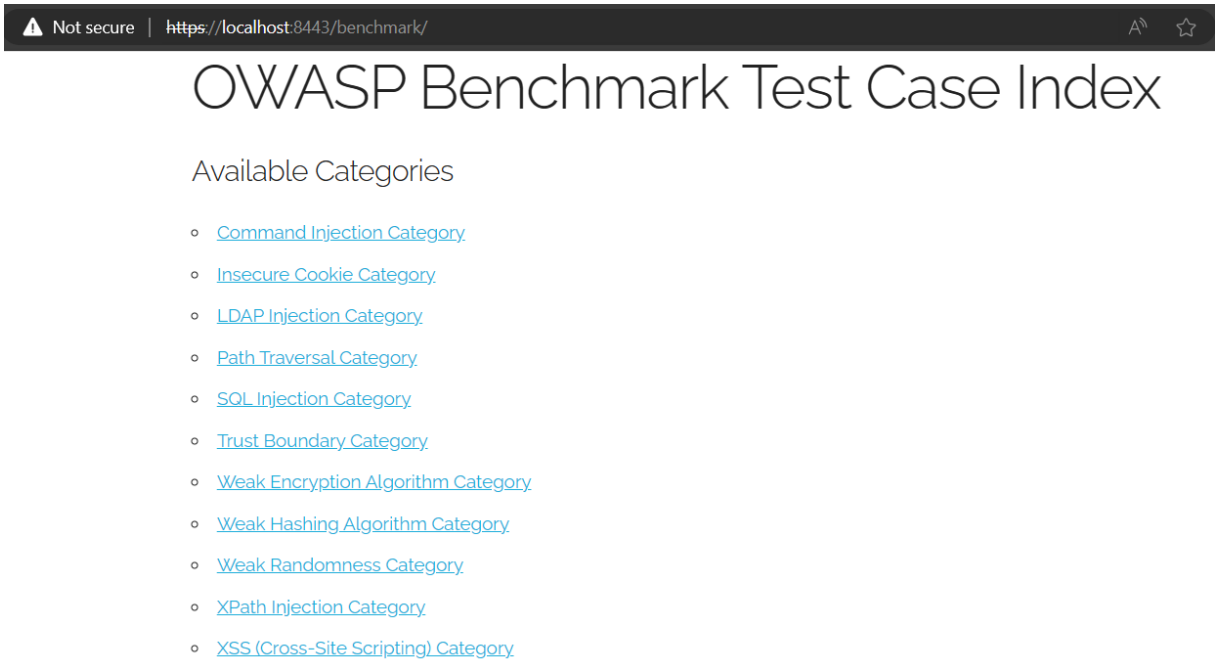
3.2 Bộ dữ liệu huấn luyện công cụ

Bộ dữ liệu huấn luyện được chứa trong webserver thuộc dự án OWASP Benchmark được phát triển bằng Java. Dự án này tạo ra nhằm mục đích đánh giá tính chính xác, phạm vi và tốc độ của các công cụ phát hiện lỗ hổng phần mềm tự động. Dự án này cung cấp một bộ kiểm tra chạy hoàn toàn có thể chạy được, chứa hàng ngàn trường hợp kiểm thử có thể khai thác, mỗi trường hợp được ánh xạ với các lỗ hổng cụ thể (CWEs). Vì vậy, ta tiến hành sử dụng webserver để làm môi trường huấn luyện cho mô hình

Yêu cầu:

- JDK phiên bản 11.0.20.
- Apache Maven.

Tiến hành cài đặt webserver huấn luyện lên máy thực nghiệm (Xem tại Mục 1 Phần Phụ Lục). Sau khi cài đặt xong, tiến hành khởi chạy server, ta có thể truy cập vào bằng đường dẫn sau: http://localhost:8443/benchmark/.



Hình 3. 2 Webserver OWASP Benchmark

3.3 Kết quả kịch bản thực nghiệm

3.3.1 Kịch bản training

3.3.2 Kịch bản testing

Thực hiện testing trên url mục tiêu:

https:// public-firing-range.appspot.com/reflected/index.html

Kết quả thực nghiệm:

Crawler của công cụ tìm đuoowjc 49 URL trên môi trường đó và tiến hành load model đã được training từ trước rồi bắt đầu hoạt động

```
[*] Scanning complete
[*] Saving scan state, please wait...
[*] Wapiti found 49 URLs and forms during the scan

DevTools listening on ws://127.0.0.1:42718/devtools/browser/ba235432-a1d2-4add-8ba3-7dbd441f62d1
[*] Launching module xss
[*] Attack module configuration complete
[*] Loading A2Cmodel ...
Loading a model without an environment, this model cannot be trained until it has a valid environment.
[*] Start Agent working ...
```

Tiếp theo Link tiến hành tiêm payload vào các điểm tiêm mà crawler đã tìm được và phân tích phản hồi, nếu có kích hoạt tiến hành in ra.

Vậy là Link đã sử dụng model đã được training để tiến hành kiểm thử thành công.

KẾT LUẬN

Hiện nay, việc phát hiện các lỗ hổng phần mềm vẫn chưa được giải quyết triệt để. Thể hiện bằng việc vẫn có nhiều lo hỏng được báo cáo hàng ngày. Sự phát triển nhanh chóng của công nghệ và kỹ thuật tấn công đòi hỏi.Cần phải có nhiều nghiên cứu về phương pháp tối ưu hơn.

Thực tế đã cho thấy rằng không thể tránh khỏi các lỗ hỏng, điều quan trọng là phát hiện chúng càng sớm càng tốt. Một phương pháp kiểm thử lỗ hổng phần mềm hiệu quả là phải phát hiện được nhiều loại lỗ hổng nhất có thể và càng ít phụ thuộc vào con người càng tốt.

Thực tế đã cho thấy rằng không thể tránh khỏi các lỗ hỏng, điều quan trọng là phát hiện chúng càng sớm càng tốt. Một phương pháp kiểm thử lỗ hổng phần mềm hiệu quả là phải phát hiện được nhiều loại lỗ hổng nhất có thể và càng ít phụ thuộc vào con người càng tốt.

Báo cáo đã tập trung nghiên cứu về ứng dụng học sâu để phát hiện các lỗ hổng phần mềm và đạt được một số kết quả sau:

- Giới thiệu tổng quan về kiểm thử lỗ hổng phần mềm.
- Tổng quan về học máy và học sâu tăng cường.
- Giới thiệu về mô hình kiểm thử ứng dụng học sâu tăng cường.
- Tiến hành thực nghiệm và đánh giá kết quả.

Bài báo cáo đã cho thấy việc sử dụng học sâu tăng cường trong kiểm thử phần mềm mang lại hiệu quả hơn so với phướng pháp truyền thông.

TÀI LIỆU THAM KHẢO

WSP-LAB/Link: Link: Black-Box Detection of Cross-Site Scripting Vulnerabilities Using Reinforcement Learning (github.com)

Link: Black-Box Detection of Cross-Site Scripting Vulnerabilities Using Reinforcement Learning | Proceedings of the ACM Web Conference 2022

Deep Reinforcement Learning: Definition, Algorithms & Uses (v7labs.com)

Deep Learning 101: Introduction [Pros, Cons & Uses] (v7labs.com)

Activation Functions in Neural Networks [12 Types & Use Cases] (v7labs.com)

Stable-Baselines3 Docs - Reliable Reinforcement Learning Implementations — Stable Baselines3 2.2.1 documentation

Gym Documentation (gymlibrary.dev)

Keras: Deep Learning for humans

Link: Black-Box Detection of Cross-Site Scripting Vulnerabilities Using Reinforcement Learning, Soyoung Lee School of Computing, KAIST Seongil Wi School of Computing, KAIST Sooel Son School of Computing, KAIST

Wapiti Web Application Scanner (github.com)

Stack Overflow - Where Developers Learn, Share, & Build Careers

Giới thiệu về blog | Reinforcement Learning (anavuongdin.github.io)

AI with Misa

Deep Learning (deeplearningbook.org)

NumPy Documentation

pandas documentation — pandas 2.1.4 documentation (pydata.org)

Matplotlib documentation — Matplotlib 3.8.2 documentation

PHỤ LỤC

Máy thực hiện thực nghiệm sử dụng hệ điều hành Windows 11.

1 Xây dựng webserver mục tiêu huấn luyện

1.1 Công cụ hỗ trợ

- **JDK phiên bản 11.0.20**

Bước 1: Tiến hành tải JDK phiên bản 11.0.20 tại đường dẫn sau:

https://www.oracle.com/java/technologies/javase/jdk11-archive-downloads.html#license-lightbox

Bước 2: Sau khi tải về xong tiến hành truy cập vào file vừa tải và cài đặt.

Bước 3:

- Mở Edit the system enveiroment variable.
- Nhấp vào nút "Environment Variables...".
- Ở phần "System variables", tìm biến có tên là Path và nhấp vào "Edit...".
- Nhấp vào "New" và thêm đường dẫn đến thư mục bin của JDK 11.0.20 mà đã cài đặt.

- Gói Maven

Bước 1: Tiến hành tải gói Maven tại đường dẫn sau:

https://dlcdn.apache.org/maven/maven-3/3.9.6/binaries/apache-maven-3.9.6-bin.zip

Bước 2: Sau khi tải xong, tiến hành giải nén file.

Bước 3:

- Mở Edit the system enveiroment variable.
- Nhấp vào nút "Environment Variables...".
- Ở phần "System variables", tìm biến có tên là Path và nhấp vào "Edit...".
- Nhấp vào "New" và thêm đường dẫn đến thư mục bin của Maven đã giải nén.

1.2 Xây dựng webserver mục tiêu huấn luyện

Mục tiêu huấn luyện agent là một webserver thuộc dự án OWASP Benchmark

Bước 1: Tiến hành tải mã nguồn của webserver tại đường dẫn sau:

https://codeload.github.com/OWASP-Benchmark/BenchmarkJava/zip/refs/heads/master

Bước 2: Sau khi tải về xong, tiến hành giải nén file.

Bước 3: Tại thư mục vừa giải nén, ta mở một giao diện dòng lệnh cli và nhập lệnh sau để tiến hành cài đặt và chạy webserver.

```
./ runBenchmark.bat
```

Bước 4: Sau khi cài đặt và đang chạy, có thể truy cập vào bằng đường dẫn http://localhost:8334/benchmark/

2 Xây dựng môi trường thực nghiệm

Ta sử dụng công cụ Anaconda để xây dựng môi trường thực nghiệm.

Bước 1: Ta tiến hành tải về Anaconda tại đường dẫn sau:

https://repo.anaconda.com/archive/Anaconda3-2023.09-0-Windows-x86_64.exe

Bước 2: Sau khi tải xong, tiến hành cài đặt công cụ

Bước 3: Khi cài đặt thành công, truy cập và mở giao diện dòng lệnh cmd của anaconda và tiến hành tải về các module bằng lệnh sau:

```
pip install <tên_module>==<phiên_bản>
```