BAN CƠ YẾU CHÍNH PHỦ
HỌC VIỆN KỸ THUẬT MẬT MÃ

LUẬN VĂN THẠC SỸ

ĐỀ TÀI: NGHIÊN CỨU GIẢI PHÁP VƯỢT QUA HỆ THỐNG PHÁT HIỆN MÃ ĐỘC DỰA TRÊN HỌC MÁY

Chuyên ngành: An toàn Thông tin

Học viên: Nguyễn Vinh Quang

Mã học viên: CHAT2P13

Mã số:

TP.HCM, 2025

BAN CƠ YẾU CHÍNH PHỦ
HỌC VIỆN KỸ THUẬT MẬT MÃ

LUẬN VĂN THẠC SỸ

ĐỀ TÀI: NGHIÊN CỨU GIẢI PHÁP VƯỢT QUA HỆ THỐNG PHÁT HIỆN MÃ ĐỘC DỰA TRÊN HỌC MÁY

Chuyên ngành: An toàn Thông tin

Học viên: Nguyễn Vinh Quang

Mã học viên: CHAT2P13

Mã số:

TP.HCM, 2025

LỜI CAM ĐOAN

Tôi xin cam đoan đây là công trình nghiên cứu khoa học của riêng tôi, được thực hiện dưới sự hướng dẫn của thầy TS. Nguyễn Mạnh Thắng. Các kết quả đạt được trong đề án là sản phẩm của riêng cá nhân, không sao chép lại của người khác. Trong toàn bộ nội dung của đề án, những điều được trình bày hoặc là của cá nhân hoặc là được tổng hợp từ nhiều nguồn tài liệu. Tất cả các tài liệu tham khảo đều có xuất xứ rõ ràng và được trích dẫn hợp pháp. Các số liệu, kết quả nêu trong đề án là trung thực và chưa từng được ai công bố trong bất kỳ công trình nào khác

Học viên

Nguyễn Vinh Quang

LỜI CẢM ƠN

Tôi xin trân trọng cảm ơn các thầy cô giáo công tác trong Học viện Kỹ thuật Mật mã, nhất là Quý thầy cô, cán bộ, giảng viên Khoa An toàn thông tin và Khoa sau đại học đã truyền đạt cho tôi những kiến thức bổ ích trong suốt những năm học vừa qua; giúp đỡ và tạo điều kiện cho tôi hoàn thành đề án này.

Đặc biệt tôi xin gửi lời cảm ơn sâu sắc tới TS. Nguyễn Mạnh Thắng đã định hướng cho tôi trong việc lựa chọn đề tài, đưa ra những nhận xét quý báu và trực tiếp hướng dẫn tôi trong suốt quá trình nghiên cứu và hoàn thành đề án tốt nghiệp.

Tôi xin trân trọng cảm ơn các Thầy (cô) trong Hội đồng khoa học đã đóng góp những ý kiến, những lời khuyên quý giá cho đề án.

Tôi cũng xin trân trọng cảm ơn gia đình, anh em và bạn bè đã quan tâm giúp đỡ, động viên tạo điều kiện thuận lợi hỗ trợ, giúp đỡ tôi trong việc thu thập thông tin, tài liệu trong quá trình học tập, thực hiện hoàn thành đề án. Xin trân trọng cảm ơn!

TP. Hồ Chí Minh, ngày tháng năm 2025

Nguyễn Vinh Quang

MỤC LỤC

MỞ ĐẦU

Tính cấp thiết của đề tài

Trong bối cảnh công nghệ thông tin phát triển mạnh mẽ, các mối đe dọa an ninh mạng, đặc biệt là mã độc (malware), ngày càng trở nên tinh vi và khó lường. Các hệ thống phát hiện mã độc truyền thống dựa trên chữ ký (signature-based) đã bộc lộ nhiều hạn chế khi đối mặt với các biến thể mã độc mới hoặc các kỹ thuật tấn công tiên tiến. Để khắc phục, các hệ thống phát hiện mã độc dựa trên học máy (machine learning) đã được nghiên cứu và ứng dụng rộng rãi, tận dụng khả năng phân tích và nhận diện mẫu dữ liệu phức tạp để phát hiện các mối nguy tiềm tàng. Tuy nhiên, cùng với sự phát triển của các hệ thống này, những kẻ tấn công cũng không ngừng tìm cách vượt qua chúng bằng các kỹ thuật học máy đối kháng (adversarial machine learning), tạo ra thách thức lớn cho ngành an ninh mạng.

Nghiên cứu về việc vượt qua các hệ thống phát hiện mã độc dựa trên học máy là một chủ đề quan trọng và cấp thiết trong bối cảnh hiện nay. Các nghiên cứu gần đây, chẳng hạn như công trình của Search And RetrieVAI of Malware (SARVAM) tại Đại học California, Santa Barbara, đã chỉ ra rằng chỉ cần thay đổi một vài byte trong tệp mã độc, các mô hình học máy có thể bị đánh lừa, phân loại sai mã độc thành phần mềm hợp pháp (goodware). Tương tự, các kỹ thuật như mạng đối kháng tạo sinh (Generative Adversarial Networks - GANs) hay học tăng cường (reinforcement learning) đã được sử dụng để tạo ra các mẫu mã độc đối kháng, có khả năng qua mặt các hệ thống phát hiện tiên tiến mà không cần hiểu rõ cấu trúc bên trong của chúng (black-box attacks). Những phát hiện này cho thấy sự cần thiết phải nghiên cứu sâu hơn về các phương pháp tấn công và phòng thủ trong lĩnh vực này.

Việc lựa chọn đề tài "Nghiên cứu giải pháp vượt qua hệ thống phát hiện mã độc dựa trên học máy" xuất phát từ nhu cầu thực tiễn trong việc đánh giá độ bền vững của các hệ thống bảo mật hiện đại. Kết quả của đề tài dự kiến sẽ cung cấp cái nhìn sâu sắc về các kỹ thuật tấn công, chẳng hạn như sử dụng Foolbox, MalGAN, hay Gym-malware, từ đó giúp các nhà phát triển cải thiện khả năng phòng thủ của hệ thống phát hiện mã độc. Đồng thời, nghiên cứu này mang ý nghĩa khoa học khi góp phần làm rõ các lỗ hổng trong mô hình học máy, thúc

đẩy sự phát triển của các giải pháp bảo mật mạnh mẽ hơn. Trong bối cảnh Việt Nam đang đẩy mạnh chuyển đổi số, việc hiểu và đối phó với các kỹ thuật tấn công học máy đối kháng sẽ đóng vai trò quan trọng trong việc bảo vệ cơ sở hạ tầng số quốc gia.

Đề tài không chỉ có ý nghĩa trong lĩnh vực nghiên cứu học thuật mà còn mang lại giá trị thực tiễn cao, đặc biệt trong việc nâng cao nhận thức về an ninh mạng và hỗ trợ phát triển các công cụ bảo mật tiên tiến. Kết quả nghiên cứu có thể được ứng dụng để kiểm tra và cải thiện các hệ thống phát hiện mã độc, từ đó tăng cường khả năng bảo vệ trước các mối đe dọa an ninh mạng trong tương lai.

Mục tiêu thực hiện đồ án

Mục tiêu của đồ án "Nghiên cứu giải pháp vượt qua hệ thống phát hiện mã độc dựa trên học máy" là xây dựng và đánh giá các phương pháp cụ thể nhằm tạo ra các mẫu mã độc đối kháng (adversarial malware samples) có khả năng qua mặt các hệ thống phát hiện mã độc sử dụng học máy, đồng thời phân tích hiệu quả của các phương pháp này. Đồ án tập trung vào việc đạt được các kết quả cụ thể, bao gồm việc triển khai các kỹ thuật học máy đối kháng, sử dụng các công cụ và khung công tác liên quan, cũng như so sánh hiệu suất của các phương pháp khác nhau. Những kết quả này không chỉ nhằm mục đích hiểu rõ hơn về các lỗ hổng trong hệ thống phát hiện mã độc mà còn cung cấp cơ sở để cải thiện các hệ thống bảo mật trong tương lai. Dưới đây là các mục tiêu cụ thể mà đồ án hướng tới:

Xây dựng các mẫu mã độc đối kháng sử dụng kỹ thuật học máy đối kháng

Mục tiêu đầu tiên là phát triển các mẫu mã độc đối kháng thông qua các kỹ thuật học máy đối kháng, như được mô tả trong các nghiên cứu về việc tấn công mạng nơ-ron nhân tạo (artificial neural networks) và mạng học sâu (deep learning networks). Cụ thể, đồ án sẽ tập trung vào việc sử dụng các phương pháp như Fast Gradient Sign Method (FGSM) và các công cụ để tạo ra các mẫu mã độc có khả năng đánh lừa các bộ phân loại mã độc. Ví dụ, bằng cách thay đổi một số byte trong tệp mã độc hoặc thêm nhiễu được tính toán cẩn thận, đồ án sẽ tạo ra các biến thể mã độc có thể được phân loại sai thành phần mềm hợp pháp (goodware). Kết quả dự kiến là một tập hợp các mẫu mã độc đối kháng,

được kiểm tra để đảm bảo chúng vẫn giữ được tính năng hoạt động trong khi qua mặt được các hệ thống phát hiện.

Sử dụng học tăng cường (Reinforcement Learning) để tối ưu hóa các mẫu mã độc

Mục tiêu thứ hai là áp dụng học tăng cường để tối ưu hóa quá trình tạo mẫu mã độc đối kháng, khắc phục hạn chế của các phương pháp dựa trên GAN, như việc tạo ra các mẫu không hợp lệ. Dựa trên môi trường Gym-malware được phát triển bởi OpenAI, đồ án sẽ thiết lập một tác nhân (agent) học cách sửa đổi các tệp mã độc thông qua các hành động như thêm byte ngẫu nhiên, thay đổi tên section, hoặc loại bỏ chữ ký. Tác nhân này sẽ sử dụng thông tin trạng thái từ tệp mã độc (ví dụ: thông tin tiêu đề, chuỗi) và phần thưởng từ báo cáo của phần mềm diệt virus để tối ưu hóa hiệu suất. Kết quả dự kiến là một tập hợp các mẫu mã độc được tối ưu hóa, có khả năng thoát khỏi sự phát hiện của các hệ thống chống mã độc tiên tiến.

So sánh và đánh giá hiệu quả của các phương pháp

Mục tiêu tiếp theo là so sánh hiệu quả của các phương pháp đã triển khai, bao gồm kỹ thuật học máy đối kháng (FGSM) và học tăng cường (Gym-malware). Đồ án sẽ đánh giá các phương pháp dựa trên các tiêu chí như tỷ lệ thành công trong việc qua mặt hệ thống phát hiện, tính hợp lệ của mẫu mã độc sinh ra, và chi phí tính toán. Kết quả của quá trình so sánh sẽ được trình bày dưới dạng bảng hoặc biểu đồ, sử dụng các công cụ như Matplotlib để trực quan hóa. Mục tiêu này nhằm cung cấp cái nhìn rõ ràng về ưu, nhược điểm của từng phương pháp, từ đó đề xuất giải pháp phù hợp nhất trong các kịch bản thực tế.

Đóng góp vào việc cải thiện hệ thống phát hiện mã độc

Cuối cùng, đồ án hướng tới việc cung cấp các khuyến nghị để cải thiện độ bền của các hệ thống phát hiện mã độc dựa trên học máy. Dựa trên các lỗ hổng được phát hiện thông qua các kỹ thuật tấn công, đồ án sẽ đề xuất các biện pháp như tăng cường dữ liệu huấn luyện, sử dụng các kỹ thuật phòng thủ đối kháng, hoặc cải tiến kiến trúc mô hình học máy. Kết quả này sẽ được trình bày trong báo cáo đồ án, đóng góp vào cộng đồng an ninh mạng bằng cách cung cấp thông tin chi tiết về các rủi ro và giải pháp tiềm năng.

Những mục tiêu trên được thiết kế để đảm bảo đồ án mang lại kết quả cụ thể, có tính ứng dụng cao, và đóng góp vào việc nâng cao hiểu biết về các kỹ thuật tấn công và phòng thủ trong an ninh mạng. Kết quả của đồ án sẽ được trình bày rõ ràng trong báo cáo, kèm theo mã nguồn, dữ liệu thực nghiệm, và phân tích chi tiết.

Đối tượng và Phạm vi nghiên cứu

- Đối tượng nghiên cứu:
    - Đồ án nhắm đến các hệ thống phát hiện mã độc dựa trên học máy, bao gồm các bộ phân loại sử dụng mạng nơ-ron nhân tạo (artificial neural networks) và mạng học sâu (deep learning networks).
    - Các mẫu mã độc đối kháng (adversarial malware samples) được tạo ra để đánh lừa các hệ thống này, tập trung vào các định dạng tệp thực thi như PE (Portable Executable) hoặc PDF.
    - Các kỹ thuật tấn công học máy đối kháng, bao gồm Fast Gradient Sign Method (FGSM) và học tăng cường (reinforcement learning) thông qua môi trường Gym-malware.
    - Đối tượng nghiên cứu chủ yếu được kiểm tra trong các kịch bản tấn công hộp đen (black-box attacks), nơi không cần biết cấu trúc bên trong của mô hình phát hiện mã độc.
- Phạm vi nghiên cứu:
    - Mức độ lý thuyết và thực hành: Đồ án kết hợp cả nghiên cứu lý thuyết và thực hành. Về lý thuyết, nghiên cứu sẽ phân tích các phương pháp tấn công học máy đối kháng dựa trên các tài liệu khoa học và nghiên cứu điển hình như FGSM và Gym-Malware. Về thực hành, đồ án sẽ triển khai các kỹ thuật tấn công bằng Python, sử dụng các công cụ như Pytorch, TensorFlow, và môi trường Gym-malware.
    - Phạm vi ứng dụng: Nghiên cứu tập trung vào việc đánh giá khả năng vượt qua các hệ thống phát hiện mã độc dựa trên học máy, với trọng tâm là các mô hình phân loại nhị phân (benign vs. malicious). Các kết quả có thể được áp dụng để kiểm tra độ bền của các hệ thống chống mã độc thương mại hoặc mã nguồn mở.

- Tập đối tượng cụ thể: Đồ án tập trung vào các mẫu mã độc thực thi (executable malware) và PDF, sử dụng các tập dữ liệu công khai như MNIST (cho mục đích mô phỏng) hoặc các mẫu mã độc từ các nguồn như EMBER.
- Khả năng mở rộng: Nghiên cứu có tiềm năng mở rộng sang các loại hệ thống phát hiện mã độc khác, chẳng hạn như các mô hình dựa trên phân tích hành vi hoặc các hệ thống sử dụng học không giám sát (unsupervised learning). Tuy nhiên, đồ án sẽ giới hạn ở các kỹ thuật tấn công dựa trên học máy đối kháng và học tăng cường, không bao gồm các phương pháp truyền thống như mã hóa (obfuscation).
- Vấn đề nhỏ được tập trung: Trong số các vấn đề liên quan đến việc vượt qua hệ thống phát hiện mã độc, đồ án ưu tiên nghiên cứu các kỹ thuật tạo mẫu mã độc đối kháng có khả năng duy trì tính hợp lệ và chức năng của tệp mã độc, đồng thời đánh giá hiệu quả qua mặt các mô hình học máy.

Phạm vi nghiên cứu được giới hạn để đảm bảo tính khả thi trong khuôn khổ đồ án, nhưng vẫn cung cấp nền tảng cho các nghiên cứu mở rộng trong tương lai, đặc biệt trong việc phát triển các biện pháp phòng thủ chống lại các cuộc tấn công học máy đối kháng.

Kết quả dự kiến

Đồ án "Nghiên cứu giải pháp vượt qua hệ thống phát hiện mã độc dựa trên học máy" hướng đến việc tạo ra và đánh giá các mẫu mã độc đối kháng có khả năng qua mặt các hệ thống phát hiện mã độc sử dụng học máy, đồng thời cung cấp các phân tích và khuyến nghị để cải thiện độ bền của các hệ thống này. Các kết quả dự kiến sẽ được trình bày chi tiết trong báo cáo đồ án, bao gồm mã nguồn, dữ liệu thực nghiệm và các phân tích định lượng, góp phần nâng cao hiểu biết về các lỗ hổng trong hệ thống bảo mật và hỗ trợ phát triển các giải pháp phòng thủ hiệu quả hơn.

Một trong những kết quả chính của đồ án là tạo ra các mẫu mã độc đối kháng thông qua các kỹ thuật học máy đối kháng, sử dụng công cụ với phương pháp Fast Gradient Sign Method (FGSM). Các mẫu này được thiết kế để đánh

lừa các bộ phân loại mã độc, khiến chúng nhận diện sai các tệp mã độc thành phần mềm hợp pháp. Các mẫu mã độc sẽ được kiểm tra tính hợp lệ và khả năng thực thi, tập trung vào các định dạng như PE (Portable Executable) hoặc PDF, sử dụng các tập dữ liệu công khai như EMBER, BODMAS

Bên cạnh đó, đồ án sẽ sử dụng môi trường học tăng cường Gym-malware để tối ưu hóa các mẫu mã độc. Một tác nhân (agent) sẽ được thiết lập để thực hiện các hành động như thêm byte ngẫu nhiên, sửa đổi tiêu đề hoặc loại bỏ chữ ký, nhằm tạo ra các mẫu mã độc thoát khỏi sự phát hiện. Kết quả sẽ bao gồm các mẫu mã độc được tối ưu hóa và phân tích hiệu quả của các hành động được chọn.

Để đánh giá các phương pháp, đồ án sẽ so sánh hiệu suất giữa các kỹ thuật: FGSM và học tăng cường (Gym-malware), dựa trên các tiêu chí như tỷ lệ qua mặt thành công, tính hợp lệ của mẫu mã độc và chi phí tính toán. Kết quả so sánh sẽ được trình bày dưới dạng bảng hoặc biểu đồ, sử dụng Matplotlib để trực quan hóa, làm rõ ưu và nhược điểm của từng phương pháp.

Cuối cùng, dựa trên các lỗ hổng được phát hiện, đồ án sẽ đề xuất các giải pháp cải thiện hệ thống phát hiện mã độc, chẳng hạn như tăng cường dữ liệu huấn luyện hoặc áp dụng các kỹ thuật phòng thủ đối kháng. Những khuyến nghị này sẽ góp phần nâng cao độ bền của các hệ thống bảo mật trước các cuộc tấn công học máy đối kháng.

Công cụ nghiên cứu

Đồ án tận dụng các công cụ lập trình và thư viện mã nguồn mở để triển khai các kỹ thuật học máy đối kháng và học tăng cường. Các công cụ được chọn dựa trên tính phổ biến, khả năng tương thích với mục tiêu nghiên cứu và kinh nghiệm lập trình Python của người thực hiện.

Ngôn ngữ lập trình Python là công cụ chính trong đồ án, nhờ vào khả năng hỗ trợ mạnh mẽ trong học máy và an ninh mạng. Python sẽ được sử dụng để viết mã nguồn, xử lý dữ liệu và tích hợp các thư viện cần thiết. Các thư viện và công cụ cụ thể bao gồm:

- **NumPy và Matplotlib**: Sử dụng để xử lý dữ liệu số và trực quan hóa kết quả, chẳng hạn như tạo biểu đồ so sánh hiệu suất của các phương pháp tấn công như FGSM, MalGAN và học tăng cường.
- **TensorFlow, Scikit-learn, PyTorch**: Là các thư viện chính để xây dựng và huấn luyện các mô hình học máy, cần sử dụng xuyên suốt trong quá trình làm đồ án
- **Gymnasium và PyPDF2**: Môi trường Gym-malware, phát triển bởi OpenAI, được sử dụng để áp dụng học tăng cường, cho phép tối ưu hóa các mẫu mã độc thông qua các hành động như thêm byte ngẫu nhiên hoặc sửa đổi tiêu đề. Thư viện **PyPDF2** được cài đặt để xử lý các định dạng tệp thực thi, đảm bảo tính hợp lệ của các mẫu mã độc.

Về nguồn dữ liệu, đồ án sử dụng các tập dữ liệu công khai để kiểm tra và đánh giá các phương pháp:

- **EMBER, BODMAS**: Cung cấp các mẫu mã độc thực thi (PE) và PDF, đại diện cho các kịch bản thực tế trong phát hiện mã độc.
- **MNIST**: Được sử dụng cho các thử nghiệm mô phỏng, đặc biệt để kiểm tra các kỹ thuật tấn công mạng nơ-ron trên dữ liệu hình ảnh, như mô tả trong các nghiên cứu về học máy đối kháng.

Các công cụ và dữ liệu trên được tích hợp để triển khai, kiểm tra và phân tích các phương pháp vượt qua hệ thống phát hiện mã độc. Kết quả thực nghiệm, bao gồm mã nguồn và phân tích, sẽ được trình bày đầy đủ trong báo cáo đồ án, đảm bảo tính minh bạch và khả năng tái tạo.

Dự kiến các Chương, mục

Tổng quan về các kiến thức học máy và mã độc

Chương này giới thiệu các khái niệm cơ bản liên quan đến học máy và mã độc, đặt nền tảng lý thuyết cho nghiên cứu. Nội dung sẽ tập trung vào việc giải thích các kỹ thuật học máy được sử dụng trong phát hiện mã độc và các phương pháp tấn công đối kháng để vượt qua chúng.

Giới thiệu về học máy trong an ninh mạng

- Tổng quan về các loại học máy: học có giám sát, không giám sát và học tăng cường.
- Ứng dụng của học máy trong phát hiện mã độc.

Khái niệm mã độc và các hệ thống phát hiện mã độc

- Định nghĩa và phân loại mã độc (ví dụ: virus, ransomware, trojan).
- Các phương pháp phát hiện mã độc truyền thống và dựa trên học máy.

Học máy đối kháng và các kỹ thuật vượt qua hệ thống phát hiện

- Giới thiệu về học máy đối kháng (adversarial machine learning).
- Tổng quan các phương pháp như Fast Gradient Sign Method (FGSM), mạng đối kháng tạo sinh (GANs) và học tăng cường.

Kết luận Chương 1:

Tóm tắt vai trò của học máy trong phát hiện mã độc và các thách thức khi đối mặt với các cuộc tấn công đối kháng.

Triển khai và các thuật toán xử lý dữ liệu

Chương này trình bày quá trình triển khai các kỹ thuật vượt qua hệ thống phát hiện mã độc, bao gồm việc xử lý dữ liệu, xây dựng mô hình và thực hiện các thuật toán. Nội dung tập trung vào việc áp dụng các công cụ như Foolbox, MalGAN và Gym-malware để tạo ra các mẫu mã độc đối kháng.

Chuẩn bị và xử lý dữ liệu

- Nguồn dữ liệu: EMBER, BODMAS, MNIST và các tập dữ liệu mã độc công khai.
- Quy trình tiền xử lý dữ liệu để phù hợp với các mô hình học máy.

Triển khai các kỹ thuật học máy đối kháng

- Ứng dụng để tạo mẫu mã độc đối kháng với FGSM.

Ứng dụng học tăng cường với Gym-malware

- Cài đặt môi trường Gym-malware và thư viện.
- Tối ưu hóa mẫu mã độc thông qua các hành động như thêm byte hoặc sửa đổi tiêu đề.

Đánh giá và so sánh hiệu quả các phương pháp

- Tiêu chí đánh giá: tỷ lệ qua mặt, tính hợp lệ, chi phí tính toán.
- Sử dụng Matplotlib để trực quan hóa kết quả so sánh.

Kết luận Chương 2:

Tóm tắt các kỹ thuật triển khai và hiệu quả ban đầu của các phương pháp.

Thực nghiệm và kết quả

Chương này trình bày các kết quả thực nghiệm, bao gồm các mẫu mã độc đối kháng được tạo ra, hiệu suất của các phương pháp và các khuyến nghị cải thiện hệ thống phát hiện mã độc. Nội dung sẽ bao gồm mã nguồn, biểu đồ và phân tích chi tiết.

Trình diễn các mẫu mã độc đối kháng

- Giới thiệu các mẫu mã độc được tạo bởi FGSM và Gym-malware.
- Kiểm tra tính hợp lệ và khả năng thực thi của các mẫu mã độc.

Phân tích kết quả thực nghiệm

- Trình bày tỷ lệ qua mặt thành công của từng phương pháp.
- So sánh ưu, nhược điểm giữa FGSM và học tăng cường.

Khuyến nghị cải thiện hệ thống phát hiện mã độc

- Đề xuất các biện pháp như tăng cường dữ liệu huấn luyện hoặc áp dụng kỹ thuật phòng thủ đối kháng.
- Thảo luận về khả năng áp dụng thực tiễn của các kết quả nghiên cứu.

Kết luận Chương 3:

Tóm tắt các kết quả đạt được và đóng góp của đồ án.

DANH MỤC TỪ VIẾT TẮT

SARVAM	Search And RetrieVAI of Malware - Tìm Kiếm Và Truy Xuất Mã Độc
GANs	Generative Adversarial Networks - Mạng Đối Kháng Tạo Sinh
FGSM	Fast Gradient Sign Method - Phương Pháp Dấu Gradient Nhanh
MalGAN	MalGAN - Mạng Tạo Mã Độc Đối Kháng
LightGBM	LightGBM - Mô Hình Gradient Boosting
BODMAS	BODMAS - Bộ Dữ Liệu Mã Độc
Q-Learning	Q-Learning - Thuật Toán Học Tăng Cường
DQN	Deep Q-Network - Mạng Q Sâu

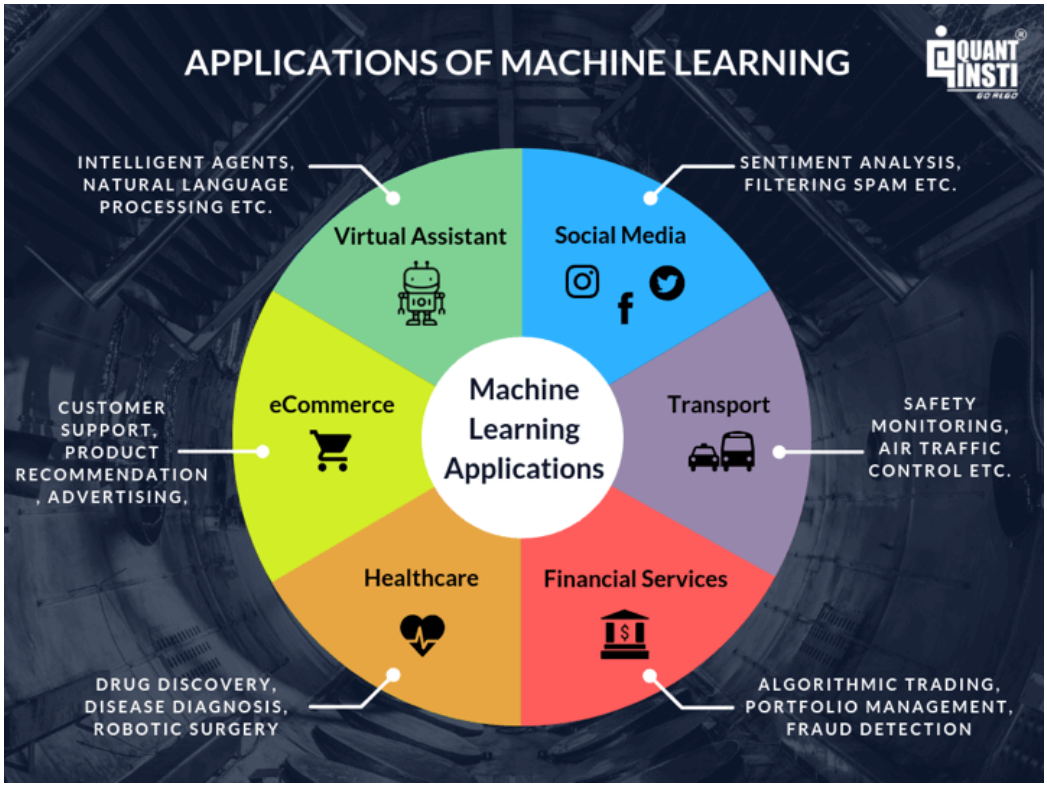
DANH MỤC HÌNH VẼ

CHƯƠNG 1: TỔNG QUAN VỀ HỌC MÁY VÀ MÃ ĐỘC

1.1 Tổng quan về các loại học máy

Học máy (machine learning) là một lĩnh vực quan trọng trong trí tuệ nhân tạo, đóng vai trò cốt lõi trong việc phát triển các hệ thống an ninh mạng, đặc biệt là trong phát hiện và phòng chống mã độc. Trong bối cảnh các cuộc tấn công mạng ngày càng tinh vi, các kỹ thuật học máy như học có giám sát, học không giám sát và học tăng cường đã trở thành công cụ mạnh mẽ để xây dựng các hệ thống phát hiện mã độc hiệu quả.

- Vai trò của học máy trong phát hiện mã độc:
    - Tự động hóa việc phân loại phần mềm độc hại và phần mềm lành tính.
    - Phát hiện các mẫu mã độc mới (zero-day threats) thông qua phân tích hành vi hoặc bất thường.
    - Tăng cường khả năng thích nghi của các hệ thống bảo mật trước các cuộc tấn công tinh vi.
- Các loại học máy được áp dụng:
    - Học có giám sát: Sử dụng dữ liệu có nhãn để huấn luyện mô hình phân loại mã độc.
    - Học không giám sát: Phát hiện các mẫu mã độc bất thường mà không cần nhãn dữ liệu.
    - Học tăng cường: Tối ưu hóa các hành động để phát hiện hoặc né tránh mã độc trong môi trường động.



Hình 1.1: Tổng quan về học máy và các ứng dụng

1.1.1. Học có giám sát

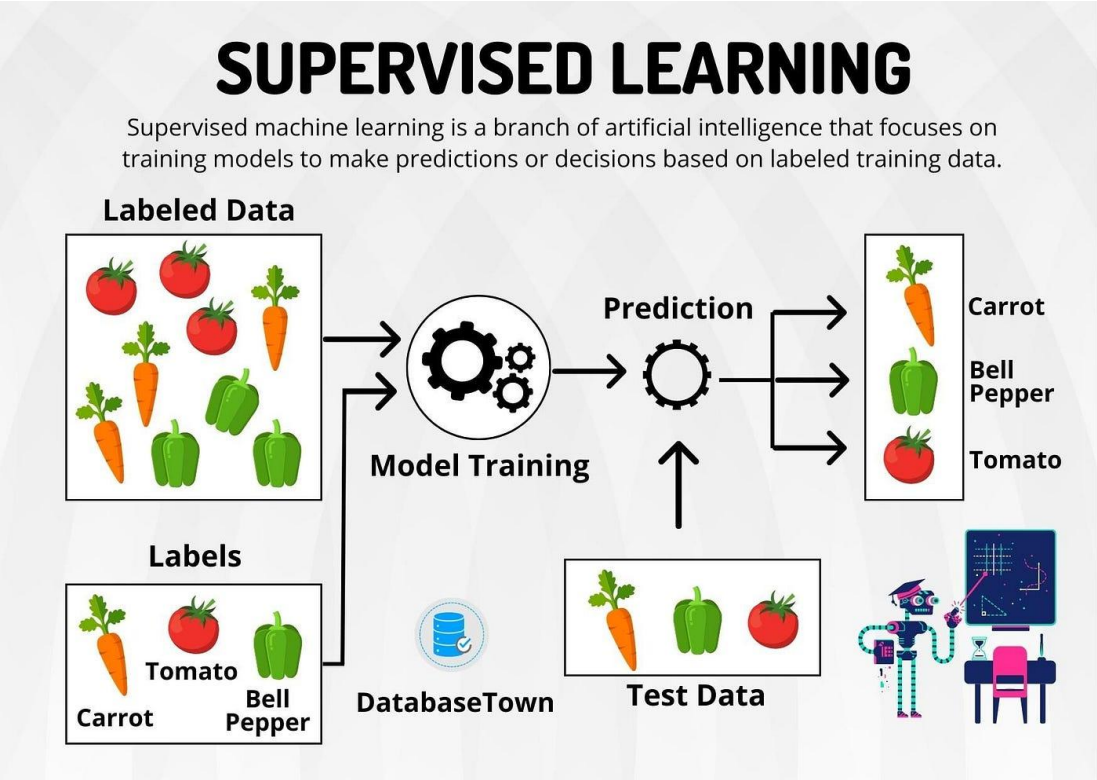
Học có giám sát (supervised learning) là một phương pháp học máy trong đó mô hình được huấn luyện trên một tập dữ liệu có nhãn, tức là mỗi mẫu dữ liệu đầu vào được gắn với một đầu ra cụ thể (nhãn). Mục tiêu của học có giám sát là xây dựng một mô hình có khả năng dự đoán chính xác nhãn của dữ liệu mới dựa trên các mẫu đã học (Goodfellow et al., 2016, trang 105)

- Đặc điểm chính:
    - Yêu cầu tập dữ liệu có nhãn đầy đủ, bao gồm cả đầu vào (features) và đầu ra (labels).
    - Sử dụng các thuật toán như hồi quy tuyến tính, hồi quy logistic, cây quyết định, hoặc mạng nơ-ron nhân tạo.
    - Mô hình được tối ưu hóa thông qua việc giảm thiểu hàm mất mát (loss function), ví dụ: bình phương trung bình (mean squared error) hoặc entropy chéo (cross-entropy).
- Ứng dụng trong phát hiện mã độc:

Học có giám sát được sử dụng rộng rãi trong các hệ thống phát hiện mã độc, đặc biệt là các bộ phân loại nhị phân (binary classifiers) để phân biệt giữa phần mềm độc hại (malware) và phần mềm lành tính (goodware). Một ví dụ điển hình được đề cập trong Chương 9 của *Mastering Machine Learning for Penetration Testing* là việc sử dụng các đặc trưng của tệp thực thi (PE) hoặc hình ảnh thang độ xám (grayscale images) để huấn luyện mô hình phân loại mã độc (Chio, 2018, trang 193). Các đặc trưng này có thể bao gồm thông tin tiêu đề, hàm nhập/xuất, hoặc chuỗi ký tự được trích xuất từ tệp.

- Ví dụ thực tế:
    - Nghiên cứu của đơn vị SARVAM tại Đại học California, Santa Barbara, đã sử dụng hình ảnh thang độ xám để biểu diễn mã độc và huấn luyện mô hình phân loại. Bằng cách thay đổi một số byte (chỉ 0,78% nội dung tệp), mã độc có thể bị phân loại sai thành phần mềm lành tính, minh họa tính dễ bị tổn thương của các bộ phân loại học có giám sát trước các cuộc tấn công đối kháng (Chio, 2018, trang 193).
    - Các bộ phân loại như PDFrate-Mimicus và Hidost được sử dụng trong EvadeML để phát hiện mã độc dựa trên đặc trưng của tệp PDF, nhưng chúng cũng dễ bị vượt qua thông qua các kỹ thuật đối kháng (Chio, 2018, trang 201).
- Né tránh học có giám sát
    - Các kỹ thuật đối kháng (adversarial attacks) thường được sử dụng để vượt qua các bộ phân loại học có giám sát. Một ví dụ nổi bật là việc sử dụng các mẫu đối kháng (adversarial samples), như được trình bày trong nghiên cứu của Goodfellow et al. (2014). Trong nghiên cứu này, một mạng nơ-ron nhận diện hình ảnh đã bị đánh lừa để phân loại sai một hình ảnh gấu trúc thành một con vượn với độ tin cậy cao (99,3%) chỉ bằng cách thêm nhiễu được tính toán cẩn thận (Goodfellow et al., 2014, trang 3).
- Phương pháp né tránh:

- Tấn công dựa trên gradient (Gradient-Based Attacks): Sử dụng gradient của hàm mất mát để tạo ra các mẫu đối kháng, ví dụ như Fast Gradient Sign Method (FGSM) được triển khai trong Foolbox (Chio, 2018, trang 199).
- Tấn công đơn pixel (Single Pixel Attack): Thay đổi một pixel duy nhất để làm sai lệch dự đoán của mô hình (Chio, 2018, trang 199).
- MalGAN: Sử dụng mạng đối kháng sinh tạo (GAN) để tạo ra các mẫu mã độc đối kháng, làm cho bộ phân loại nhận diện sai mã độc thành phần mềm lành tính (Hu & Tan, 2017, trang 4).



Hình 1.2: Học có giám sát

1.1.2. Học không giám sát

Học không giám sát (unsupervised learning) là phương pháp học máy trong đó mô hình được huấn luyện trên dữ liệu không có nhãn. Mục tiêu là tìm ra các mẫu, cấu trúc hoặc mối quan hệ tiềm ẩn trong dữ liệu mà không cần hướng dẫn trước (Hastie et al., 2009, trang 486).

- Đặc điểm chính:

- Không yêu cầu nhãn cho dữ liệu đầu vào.
- Các thuật toán phổ biến bao gồm phân cụm (clustering) như K-means, giảm chiều dữ liệu (dimensionality reduction) như PCA, hoặc mô hình sinh (generative models) như mạng tự mã hóa (autoencoders).
- Thường được sử dụng để khám phá dữ liệu hoặc phát hiện bất thường (anomaly detection).

- Ứng dụng trong phát hiện mã độc

Học không giám sát đặc biệt hữu ích trong việc phát hiện các mẫu mã độc mới hoặc bất thường mà không cần nhãn trước. Trong an ninh mạng, các thuật toán học không giám sát thường được sử dụng để phát hiện các mối đe dọa không xác định (zero-day threats) bằng cách phân tích hành vi bất thường của hệ thống hoặc tệp.

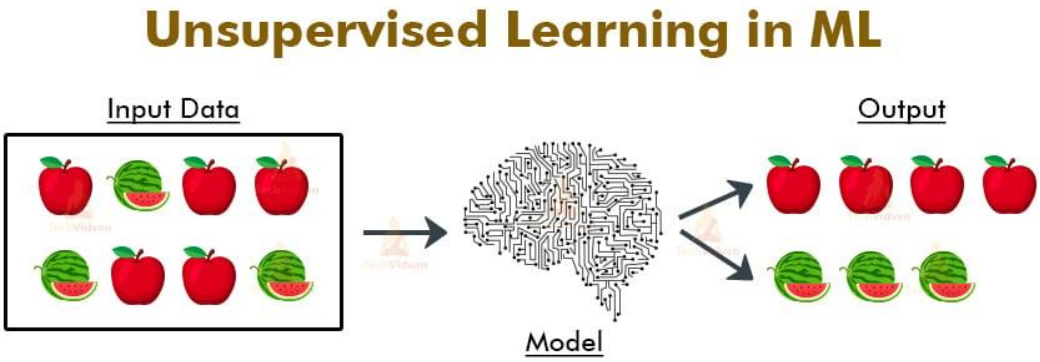
- Ví dụ thực tế:
  - Trong Chương 9, các hệ thống phát hiện mã độc sử dụng học không giám sát để phát hiện các mẫu mã độc không thuộc các lớp đã biết. Ví dụ, các thuật toán phân cụm có thể được sử dụng để nhóm các tệp dựa trên đặc trưng tương tự, từ đó xác định các tệp bất thường có thể là mã độc (Chio, 2018, trang 206).
  - Mạng tự mã hóa (autoencoders) có thể được sử dụng để xây dựng mô hình phát hiện bất thường, trong đó các mẫu không phù hợp với phân phối dữ liệu huấn luyện được coi là mã độc tiềm năng (Goodfellow et al., 2016, trang 356).

- Né tránh học không giám sát

Các hệ thống học không giám sát có thể bị đánh lừa bằng cách làm cho dữ liệu mã độc trông giống dữ liệu lành tính trong không gian đặc trưng. Ví dụ, các kỹ thuật như thêm nhiễu ngẫu nhiên hoặc thay đổi đặc trưng của tệp mã độc (như tên phần, hàm nhập/xuất) có thể làm giảm sự bất thường của chúng trong mắt các thuật toán phân cụm hoặc phát hiện bất thường.

- Phương pháp né tránh:

- Thay đổi đặc trưng tệp: Các hành động như thêm byte ngẫu nhiên hoặc thay đổi tên phần trong tệp PE có thể làm cho mã độc hòa trộn với các tệp lành tính (Chio, 2018, trang 209).
- Sử dụng GAN: Mạng đối kháng sinh tạo có thể được sử dụng để tạo ra các mẫu mã độc có đặc trưng tương tự như phần mềm lành tính, qua đó đánh lừa các hệ thống phát hiện bất thường (Hu & Tan, 2017, trang 5).



Hình 1.3: Học không giám sát

1.1.3. Học tăng cường

Học tăng cường (reinforcement learning) là một phương pháp học máy trong đó một tác nhân (agent) học cách đưa ra quyết định tối ưu bằng cách tương tác với môi trường và nhận phản hồi dưới dạng phần thưởng (reward) hoặc hình phạt (Sutton & Barto, 2018, trang 1). Khác với học có giám sát và học không giám sát, học tăng cường tập trung vào việc tối ưu hóa hành động dựa trên kinh nghiệm thực tế.

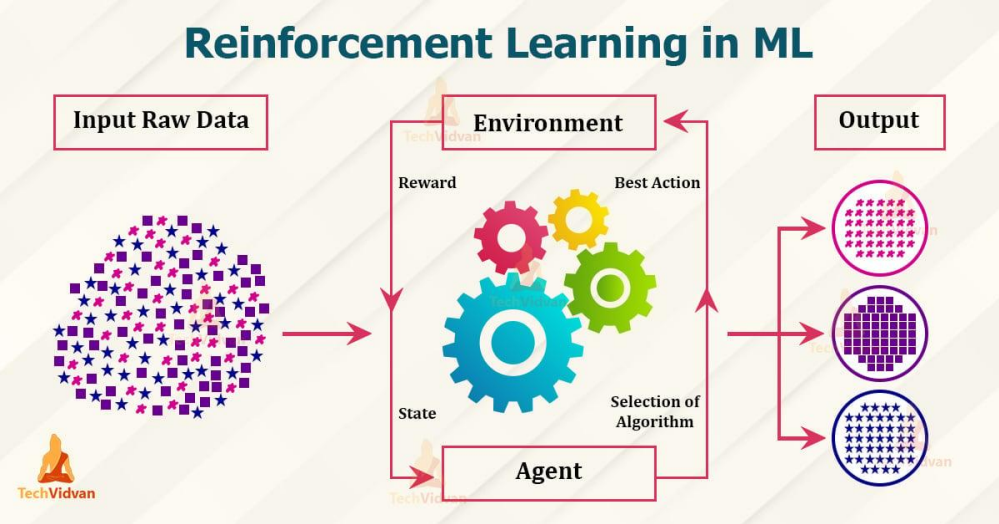
- Đặc điểm chính:
  - Tác nhân học thông qua thử và sai (trial-and-error) trong một môi trường.
  - Sử dụng hàm phần thưởng (reward function) để đánh giá hiệu quả của hành động.
  - Các thuật toán phổ biến bao gồm Q-learning, Deep Q-Networks (DQN), và Proximal Policy Optimization (PPO).
- Ứng dụng trong phát hiện mã độc

Học tăng cường được sử dụng để xây dựng các hệ thống phát hiện mã độc thông minh, có khả năng thích nghi với các cuộc tấn công mới. Trong bối cảnh né tránh mã độc, học tăng cường được áp dụng để tối ưu hóa các hành động sửa đổi mã độc nhằm vượt qua các hệ thống phát hiện.

- Ví dụ thực tế:
    - Gym-malware: Một môi trường học tăng cường được phát triển bởi Endgame, cho phép tác nhân học cách sửa đổi các tệp mã độc (như thêm byte, xóa chữ ký, hoặc nén bằng UPX) để né tránh các hệ thống phát hiện. Tác nhân nhận phần thưởng dựa trên báo cáo từ các công cụ chống virus, tối ưu hóa hành động để đạt được kết quả né tránh tốt nhất (Chio, 2018, trang 209).
    - Trong ví dụ về Atari Breakout, học tăng cường được sử dụng để huấn luyện một tác nhân chơi trò chơi bằng cách tối ưu hóa các hành động dựa trên phần thưởng (ví dụ: phá gạch). Tương tự, trong phát hiện mã độc, tác nhân có thể học cách sửa đổi mã độc để tránh bị phát hiện, với phần thưởng là báo cáo "lành tính" từ hệ thống chống virus (Chio, 2018, trang 207).
- Né tránh học tăng cường

Học tăng cường đặc biệt mạnh mẽ trong việc tạo ra các mẫu mã độc động, vì tác nhân có thể học cách điều chỉnh hành động dựa trên phản hồi từ môi trường. Điều này làm cho các hệ thống phát hiện mã độc dựa trên học tăng cường trở thành mục tiêu khó khăn hơn cho các cuộc tấn công đối kháng.

- Phương pháp né tránh:
    - Sửa đổi môi trường: Tác nhân có thể thực hiện các hành động như thay đổi tên phần, xóa thông tin gỡ lỗi, hoặc thêm các byte ngẫu nhiên để làm cho mã độc trông giống phần mềm lành tính (Chio, 2018, trang 209).
    - Tối ưu hóa hành động: Sử dụng các thuật toán học tăng cường như Q-learning để tìm ra chuỗi hành động tối ưu nhằm né tránh phát hiện, ví dụ: kết hợp nhiều hành động như nén UPX và thay đổi chữ ký (Chio, 2018, trang 209).



Hình 1.4: Học tăng cường

1.2 Mã độc và các phương pháp phát hiện

Mã độc (malware) là một trong những mối đe dọa nghiêm trọng nhất đối với an ninh mạng, gây ra thiệt hại đáng kể về tài chính, dữ liệu và uy tín. Với sự phát triển của các kỹ thuật tấn công mạng ngày càng tinh vi, việc phát hiện và phòng chống mã độc đã trở thành một thách thức lớn. Các hệ thống phát hiện mã độc truyền thống dựa trên chữ ký (signature-based) và quy tắc (rule-based) đã không còn đủ hiệu quả trước các biến thể mã độc mới. Trong bối cảnh này, học máy (machine learning) đã nổi lên như một giải pháp mạnh mẽ, tận dụng các phương pháp như học có giám sát, học không giám sát và học tăng cường để cải thiện khả năng phát hiện mã độc.

1.2.1. Khái niệm và phân loại mã độc

Mã độc là phần mềm được thiết kế với mục đích gây hại, đánh cắp thông tin, phá hoại hệ thống hoặc thực hiện các hành vi trái phép trên các thiết bị máy tính hoặc mạng (Kaspersky, 2020). Mã độc thường được phát tán qua email lừa đảo, tệp đính kèm, hoặc các lỗ hổng phần mềm, nhằm khai thác điểm yếu của hệ thống hoặc người dùng.

- Đặc điểm chính của mã độc:
    - Tính phá hoại: Gây ra thiệt hại cho hệ thống, như xóa dữ liệu hoặc làm gián đoạn hoạt động.

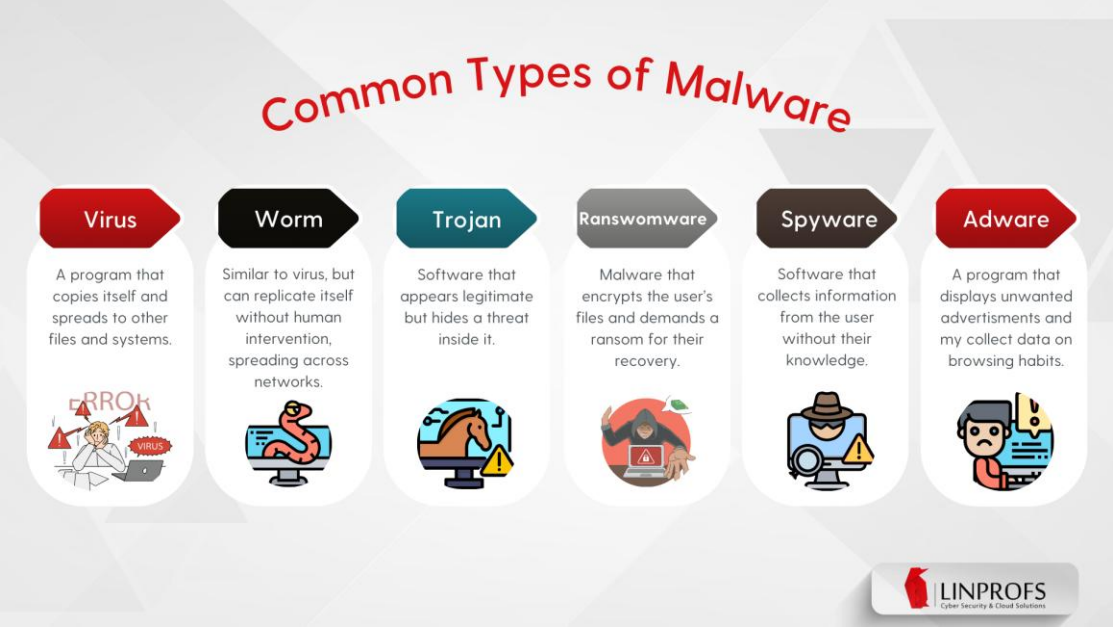
- Tính lén lút: Hoạt động ngầm để tránh bị phát hiện bởi người dùng hoặc hệ thống bảo mật.
- Khả năng tự lan truyền: Một số mã độc có khả năng tự sao chép và lây nhiễm sang các hệ thống khác.
- Tính đa dạng: Có nhiều biến thể với mục đích và phương thức tấn công khác nhau.

1.2.2. Phân loại mã độc

Các loại mã độc:

- Virus: Đây là loại mã độc cổ điển, tự nhân bản bằng cách gắn vào các tệp thực thi hoặc hệ thống. Virus thường lây lan qua việc người dùng vô tình mở tệp bị nhiễm, ví dụ như qua email hoặc USB. Một đặc điểm nổi bật là khả năng làm hỏng dữ liệu hoặc gây gián đoạn hoạt động hệ thống.
- Worm (Sâu máy tính): Khác với virus, worm không cần gắn vào tệp mà tự nhân bản qua mạng. Chúng khai thác lỗ hổng bảo mật để lây lan nhanh chóng, như vụ tấn công worm Conficker năm 2008, ảnh hưởng đến hàng triệu máy tính trên toàn cầu.
- Trojan (Mã độc ngựa Trojan): Ngụy trang dưới dạng phần mềm hợp pháp, Trojan cho phép tin tặc kiểm soát từ xa. Ví dụ, Trojan Zeus được sử dụng để đánh cắp thông tin ngân hàng, gây thiệt hại hàng tỷ USD.
- Ransomware (Mã độc tống tiền): Loại mã độc mã hóa dữ liệu và đòi tiền chuộc, như WannaCry năm 2017, đã tấn công hơn 200.000 máy tính tại 150 quốc gia, gây tổn thất ước tính 4 tỷ USD.
- Spyware (Phần mềm gián điệp): Thu thập thông tin cá nhân mà không được sự cho phép, thường được sử dụng trong các chiến dịch gián điệp công nghiệp.
- Adware (Quảng cáo độc hại): Hiển thị quảng cáo không mong muốn, đôi khi dẫn đến tải thêm mã độc khác.

- Rootkit: Ẩn mình trong hệ thống để tránh phát hiện, thường được dùng để duy trì quyền truy cập của tin tặc sau khi xâm nhập thành công.



Hình 1.5: Các dạng mã độc thường thấy

Mức độ nguy hiểm của mã độc:

- Tác động tài chính: Ransomware như CryptoLocker đã khiến các tổ chức mất hàng triệu USD do phải trả tiền chuộc hoặc khôi phục dữ liệu. Theo báo cáo của Cybersecurity Ventures, thiệt hại toàn cầu từ ransomware dự kiến đạt 20 tỷ USD vào năm 2021.
- Tác động xã hội: Các vụ tấn công như NotPetya (2017) không chỉ nhắm vào doanh nghiệp mà còn gây gián đoạn dịch vụ công cộng, như bệnh viện và giao thông.
- Tác động an ninh quốc gia: Stuxnet, phát hiện năm 2010, được cho là vũ khí mạng nhắm vào cơ sở hạ tầng hạt nhân Iran, cho thấy mã độc có thể trở thành công cụ trong xung đột địa chính trị.
- Tính lan rộng: Worm và virus có khả năng lây lan nhanh chóng, làm tê liệt toàn bộ mạng lưới, trong khi rootkit và spyware thường hoạt động âm thầm nhưng gây tổn hại lâu dài.

Đặc điểm kỹ thuật của mã độc:

- Kỹ thuật ẩn mình: Nhiều mã độc như rootkit sử dụng kỹ thuật hooking để che giấu hoạt động, khiến các công cụ quét thông thường không phát hiện được.
- Tính thích nghi: Malware ngày nay sử dụng kỹ thuật đa hình (polymorphism) và mã hóa để thay đổi cấu trúc, tránh bị phát hiện bởi chữ ký (signature-based detection).
- Khai thác lỗ hổng zero-day: Stuxnet và các mã độc tiên tiến khai thác lỗ hổng chưa được vá, tăng khả năng thâm nhập.
- Tích hợp AI: Một số mã độc mới sử dụng học máy để tối ưu hóa chiến lược tấn công, như tự điều chỉnh dựa trên hành vi người dùng.

Các vụ tấn công mã độc nổi bật trong quá khứ:

- Vụ tấn công ILOVEYOU (2000): Một virus lây lan qua email, gây thiệt hại ước tính 10-15 tỷ USD bằng cách ghi đè tệp và tự gửi bản sao đến danh bạ email. Đây là ví dụ điển hình về tác động của virus trong thời kỳ đầu internet.
- Vụ tấn công Stuxnet (2010): Được thiết kế để phá hoại máy ly tâm hạt nhân tại Iran, Stuxnet là mã độc đầu tiên nhắm vào cơ sở hạ tầng vật lý, đánh dấu bước ngoặt trong chiến tranh mạng.
- Vụ tấn công WannaCry (2017): Tận dụng lỗ hổng Windows, WannaCry đã mã hóa dữ liệu trên toàn cầu, ảnh hưởng đến NHS (Anh) và nhiều tập đoàn lớn, nhấn mạnh tầm quan trọng của bản vá bảo mật.
- Vụ tấn công NotPetya (2017): Ban đầu được cho là ransomware, nhưng thực chất là vũ khí phá hoại nhắm vào Ukraine, gây thiệt hại hơn 10 tỷ USD trên toàn cầu.
- Vụ tấn công SolarWinds (2020): Một chiến dịch tinh vi sử dụng trojan để xâm nhập mạng lưới chính phủ Mỹ, cho thấy sự phát triển của mã độc trong các cuộc tấn công có tổ chức.

Hình 1.6: Mã độc WannaCry nổi tiếng

1.2.3. Các phương pháp phát hiện mã độc truyền thống

Các phương pháp phát hiện mã độc truyền thống dựa trên các kỹ thuật phân tích tĩnh (static analysis) và phân tích động (dynamic analysis), thường sử dụng chữ ký hoặc quy tắc để nhận diện mã độc.

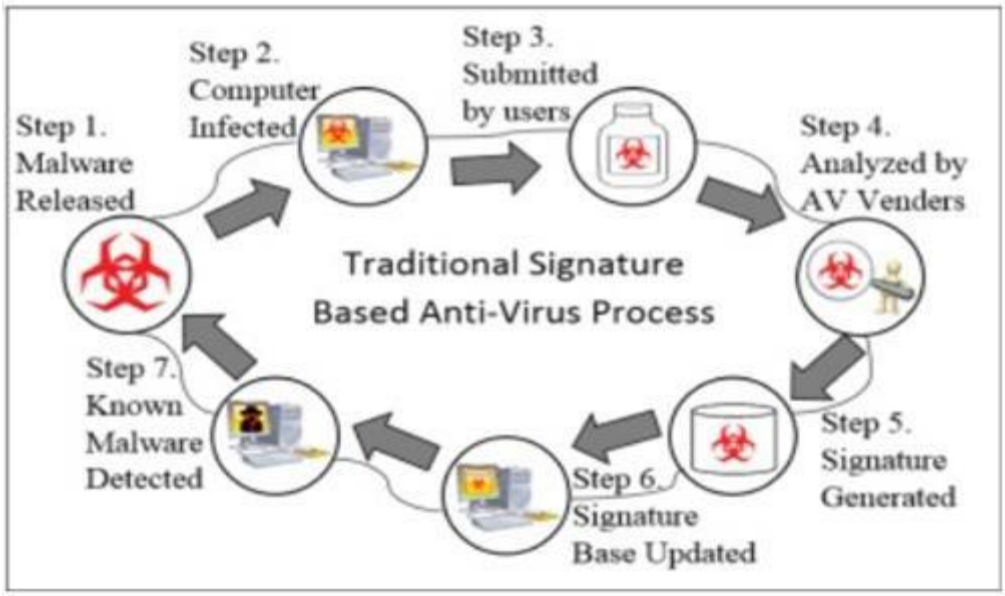
1.2.3.1. Phân tích dựa trên chữ ký

Phân tích dựa trên chữ ký (signature-based detection) so sánh các mẫu mã trong tệp hoặc lưu lượng mạng với cơ sở dữ liệu chữ ký mã độc được cập nhật thường xuyên.

- Ưu điểm:
  - Độ chính xác cao đối với các mã độc đã biết.
  - Tốc độ xử lý nhanh, phù hợp với các hệ thống thời gian thực.
- Hạn chế:
  - Không hiệu quả đối với các mã độc mới hoặc biến thể (zero-day threats).

- Yêu cầu cập nhật liên tục cơ sở dữ liệu chữ ký.

- Ví dụ thực tế:
  - Các phần mềm diệt virus như Norton hoặc Kaspersky sử dụng cơ sở dữ liệu chữ ký để phát hiện các mẫu mã độc như virus ILOVEYOU hoặc WannaCry (Kaspersky, 2020).



Hình 1.7: Phân tích dựa trên chữ ký (signature-based detection)

1.2.3.2. Phân tích dựa trên hành vi

Phân tích dựa trên hành vi (behavior-based detection) theo dõi hoạt động của chương trình trong môi trường sandbox để phát hiện các hành vi đáng ngờ, như truy cập tệp hệ thống hoặc kết nối mạng bất thường.

- Ưu điểm:
  - Có khả năng phát hiện các mã độc mới hoặc không có chữ ký.
  - Hiệu quả trong việc nhận diện các hành vi độc hại, như mã hóa dữ liệu của ransomware.

- Hạn chế:
  - Tốn tài nguyên tính toán và thời gian do cần chạy chương trình trong môi trường mô phỏng.

- Có thể tạo ra cảnh báo sai (false positives) nếu hành vi hợp pháp bị hiểu nhầm.

- Ví dụ thực tế:
  - Cuckoo Sandbox được sử dụng để phân tích hành vi của mã độc, như ghi lại các hoạt động của tệp PE trong môi trường ảo (Chio, 2018, trang 201).

1.2.3.3. Phân tích dựa trên quy tắc

Phân tích dựa trên quy tắc (rule-based detection) sử dụng các quy tắc được định nghĩa trước để phát hiện mã độc dựa trên các mẫu hành vi hoặc đặc trưng cụ thể.

- Ưu điểm:
  - Linh hoạt trong việc tùy chỉnh các quy tắc cho các loại mã độc cụ thể.
  - Hữu ích trong các hệ thống phát hiện xâm nhập (IDS).
- Hạn chế:
  - Yêu cầu chuyên gia thiết kế các quy tắc phức tạp.
  - Dễ bị vượt qua bởi các mã độc sử dụng kỹ thuật che giấu (obfuscation).
- Ví dụ thực tế:
  - Hệ thống phát hiện xâm nhập như Snort sử dụng các quy tắc để phát hiện các mẫu lưu lượng mạng liên quan đến mã độc (Snort, 2020).

1.2.4. Phương pháp phát hiện mã độc dựa trên học máy

Học máy đã cách mạng hóa việc phát hiện mã độc, mang lại khả năng phân tích dữ liệu phức tạp và thích nghi với các mối đe dọa mới. Các phương pháp học máy phổ biến bao gồm học có giám sát, học không giám sát và học tăng cường

1.2.4.1. Học có giám sát

Học có giám sát sử dụng dữ liệu có nhãn để huấn luyện các mô hình phân loại mã độc và phần mềm lành tính. Các đặc trưng như thông tin tiêu đề tệp PE, hàm nhập/xuất, hoặc hình ảnh thang độ xám được sử dụng để xây dựng mô hình.

- Ứng dụng:
    - Phân loại mã độc: Các bộ phân loại như PDFrate-Mimicus và Hidost sử dụng đặc trưng của tệp PDF để phân loại mã độc, như được đề cập trong EvadeML (Chio, 2018, trang 201).
    - Hình ảnh thang độ xám: Nghiên cứu của SARVAM tại Đại học California, Santa Barbara, đã chuyển đổi tệp thực thi thành hình ảnh thang độ xám để huấn luyện mô hình phân loại, đạt hiệu quả cao trong việc nhận diện mã độc (Chio, 2018, trang 193).
- Ví dụ thực tế:
    - Một nghiên cứu minh họa rằng việc thay đổi chỉ 0,78% nội dung của chương trình NETSTAT (88 byte trong 36.864 byte) có thể khiến mô hình học có giám sát phân loại sai mã độc thành phần mềm lành tính (Chio, 2018, trang 193).
    - Các thuật toán như rừng ngẫu nhiên hoặc mạng nơ-ron được sử dụng để phân loại các tệp dựa trên đặc trưng tĩnh, như chuỗi ký tự hoặc cấu trúc tệp.
- Thách thức:
    - Dễ bị tấn công đối kháng, như sử dụng mẫu đối kháng (adversarial samples) để đánh lừa mô hình (Goodfellow et al., 2014, trang 3).
    - Phụ thuộc vào dữ liệu có nhãn chất lượng cao, vốn tốn kém để thu thập.

1.2.4.2. Học không giám sát

Học không giám sát phát hiện mã độc bằng cách phân tích các mẫu hoặc bất thường trong dữ liệu mà không cần nhãn. Các thuật toán như phân cụm K-

means hoặc mạng tự mã hóa được sử dụng để nhận diện các tệp hoặc hành vi bất thường.

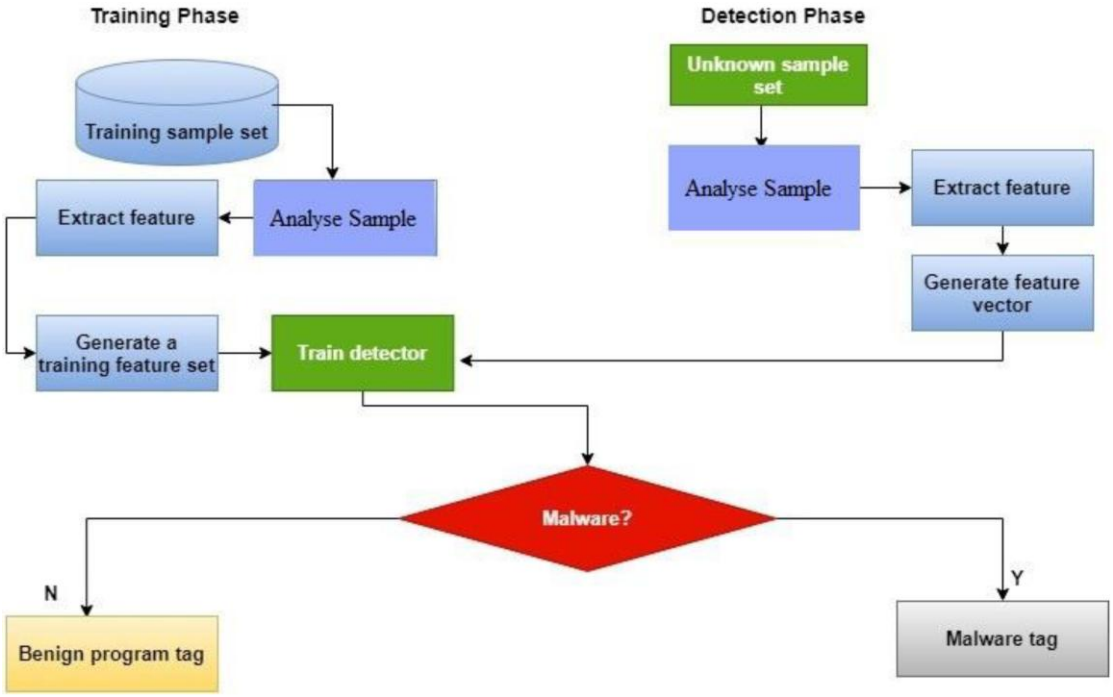
- Ứng dụng:
    - Phát hiện bất thường: Các hệ thống sử dụng mạng tự mã hóa để xác định các tệp không phù hợp với phân phối của phần mềm lành tính, như được đề cập trong Chương 9 (Chio, 2018, trang 206).
    - Phân cụm đặc trưng: Các thuật toán phân cụm nhóm các tệp dựa trên đặc trưng tương tự, giúp phát hiện các cụm mã độc tiềm năng.
- Ví dụ thực tế:
    - Các hệ thống phát hiện mã độc zero-day sử dụng học không giám sát để nhận diện các mẫu hành vi bất thường, như kết nối mạng bất thường hoặc truy cập tệp hệ thống (Chio, 2018, trang 206).
- Thách thức:
    - Độ chính xác thấp hơn học có giám sát do thiếu nhãn hướng dẫn.
    - Dễ bị đánh lừa bằng cách làm cho mã độc trông giống phần mềm lành tính trong không gian đặc trưng.

1.2.4.3. Học tăng cường

Học tăng cường cho phép một tác nhân (agent) học cách tối ưu hóa hành động thông qua tương tác với môi trường, nhận phần thưởng hoặc hình phạt dựa trên kết quả. Trong phát hiện mã độc, học tăng cường được sử dụng để tối ưu hóa các hành động sửa đổi mã độc nhằm né tránh phát hiện.

- Ứng dụng:
    - Gym-malware: Một môi trường học tăng cường của Endgame, cho phép tác nhân thử nghiệm các hành động như thêm byte ngẫu nhiên, xóa chữ ký, hoặc nén tệp bằng UPX để né tránh các hệ thống phát hiện. Phần thưởng được dựa trên báo cáo từ các công cụ chống virus (Chio, 2018, trang 209).
    - Tối ưu hóa hành động: Tác nhân học cách chọn chuỗi hành động tối ưu để làm cho mã độc trông giống phần mềm lành tính (Chio, 2018, trang 209).

- Ví dụ thực tế:
    - Trong môi trường Gym-malware, các hành động như thay đổi tên phần hoặc xóa thông tin gỡ lỗi được tối ưu hóa để giảm khả năng bị phát hiện bởi các hệ thống chống virus (Chio, 2018, trang 209).
    - Phép ẩn dụ Atari Breakout minh họa cách học tăng cường có thể huấn luyện một tác nhân để phá gạch, tương tự như cách tác nhân học sửa đổi mã độc để né tránh phát hiện (Chio, 2018, trang 207).
- Thách thức:
    - Yêu cầu thời gian huấn luyện lâu và tài nguyên tính toán lớn.
    - Hiệu quả phụ thuộc vào việc thiết kế hàm phần thưởng phù hợp.



Hình 1.8: Phát hiện mã độc dựa trên học máy

1.2.5. Các công cụ hỗ trợ phát hiện mã độc dựa trên học máy

Các công cụ học máy đã được phát triển để hỗ trợ việc xây dựng và kiểm tra các hệ thống phát hiện mã độc. Một số công cụ được đề cập trong Chương 9 bao gồm:

- Foolbox: Một hộp công cụ Python để kiểm tra độ bền của các mô hình học máy trước các cuộc tấn công đối kháng. Foolbox hỗ trợ các kỹ thuật

như Fast Gradient Sign Method (FGSM) và Single Pixel Attack, giúp đánh giá khả năng chống chịu của các bộ phân loại mã độc (Chio, 2018, trang 198).

- MalGAN: Một mạng đối kháng sinh tạo (GAN) được thiết kế để tạo ra các mẫu mã độc đối kháng, có khả năng đánh lừa các bộ phân loại trong các kịch bản tấn công hộp đen (Hu & Tan, 2017, trang 4).
- Gym-malware: Một môi trường học tăng cường của OpenAI, cho phép huấn luyện tác nhân để sửa đổi mã độc nhằm né tránh phát hiện, với các hành động như thêm byte hoặc nén tệp (Chio, 2018, trang 209).

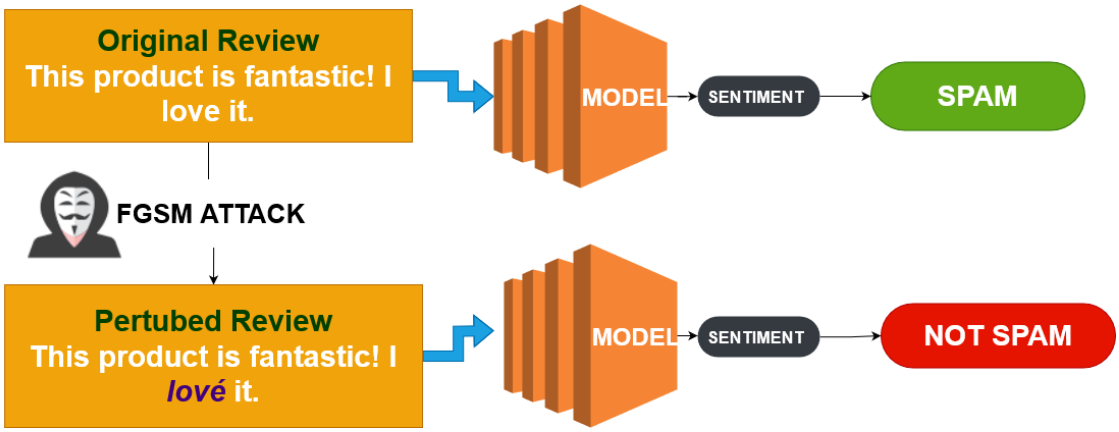
1.2.6. Hạn Chế

- Tấn công đối kháng: Các kỹ thuật như MalGAN và Foolbox cho thấy các mô hình học máy dễ bị đánh lừa bởi các mẫu đối kháng, làm giảm hiệu quả của các hệ thống phát hiện mã độc (Chio, 2018, trang 205).
- Dữ liệu huấn luyện: Thu thập và gắn nhãn dữ liệu mã độc chất lượng cao là một thách thức lớn, đặc biệt đối với học có giám sát.
- Tính hợp lệ của mẫu mã độc: Các mẫu mã độc sinh ra bởi GAN hoặc học tăng cường có thể không hợp lệ về mặt chức năng, đòi hỏi sử dụng sandbox để kiểm tra (Chio, 2018, trang 206).

1.3 Kỹ thuật vượt qua hệ thống phát hiện mã độc

1.3.1 Fast Gradient Sign Method (FGSM)

Fast Gradient Sign Method (FGSM) là một kỹ thuật tấn công đối kháng dựa trên gradient, được đề xuất bởi Goodfellow et al. (2014). Kỹ thuật Fast Gradient Sign Method (FGSM) là một phương pháp quan trọng trong lĩnh vực tấn công đối nghịch (adversarial attack), cho phép tạo ra các mẫu đối nghịch (adversarial examples) nhằm đánh lừa các hệ thống phát hiện mã độc dựa trên học máy. FGSM tạo ra các mẫu đối kháng bằng cách thêm nhiễu vào dữ liệu đầu vào dựa trên gradient của hàm mất mát (loss function) của mô hình, nhằm tối đa hóa lỗi dự đoán.



Hình 1.9: Kỹ thuật Fast Gradient Sign Method (FGSM)

- Nguyên lý hoạt động:
  - FGSM, được đề xuất bởi Goodfellow và cộng sự, dựa trên ý tưởng thêm nhiễu nhỏ (perturbation) vào dữ liệu đầu vào để làm thay đổi dự đoán của mô hình học máy mà không ảnh hưởng đáng kể đến nhận thức của con người. Nhiễu này được tính toán dựa trên gradient của hàm mất mát (loss function) theo hướng ký hiệu (sign) của gradient.
  - Tính gradient của hàm mất mát so với đầu vào (x): (g(x) = \nabla_x L(\theta, x, y)), trong đó (L) là hàm mất mát, (\theta) là tham số mô hình, và (y) là nhãn mục tiêu.

    $$x' = x + \epsilon \cdot \text{sign}(\nabla_x J(\theta, x, y))$$

    Trong đó:
    - ϵ là tham số điều chỉnh độ lớn của nhiễu.
    - $\nabla x J(\theta, x, y)$ là gradient của hàm mất mát J theo x, với θ là tham số của mô hình.
    - sign lấy ký hiệu của gradient để định hướng nhiễu.
  - Thêm nhiễu vào đầu vào: (x' = x + \epsilon \cdot \text{sign}(g(x))), với (\epsilon) là một hệ số nhỏ điều chỉnh mức độ nhiễu.
  - Mục tiêu: Làm cho mô hình phân loại sai (x') mà vẫn giữ (x') giống (x) về mặt chức năng. Phương pháp này nhằm tối đa hóa sai

lệch giữa dự đoán ban đầu và dự đoán trên mẫu đối nghịch, trong khi giữ nhiễu trong giới hạn nhỏ để tránh bị phát hiện thủ công.

- Cơ chế tính gradient và lan truyền ngược
    - Tính gradient: Để áp dụng FGSM, cần tính gradient của hàm mất mát theo đầu vào $ x $. Quá trình này yêu cầu lan truyền ngược (backpropagation) qua mạng nơ-ron, như được mô tả trong tài liệu với đoạn mã triển khai đạo hàm (derivative function). Gradient cung cấp hướng mà nhiễu cần được thêm vào để tăng mất mát.
    - Lan truyền ngược trong FGSM: quá trình lan truyền ngược bao gồm:
        - Tính toán giá trị kích hoạt (activations) qua các lớp của mạng.
        - Tính đạo hàm từng phần của mất mát theo các tham số và đầu vào.
        - Sử dụng ký hiệu gradient để định hướng nhiễu

Ý nghĩa thực tiễn: Gradient cho thấy mức độ nhạy cảm của mô hình với từng đặc trưng, giúp xác định các điểm yếu có thể khai thác.

- Các loại tấn công dựa trên FGSM
    - Tấn công không mục tiêu (Non-targeted attack): Mục tiêu là làm cho mô hình dự đoán sai mà không cần nhắm đến một lớp cụ thể. Ví dụ, với hình ảnh số 2 trong MNIST, FGSM thêm nhiễu để chuyển dự đoán thành một lớp khác (ví dụ: 6), như được minh họa trong hình ảnh đối nghịch trong tài liệu.
    - Tấn công có mục tiêu (Targeted attack): Nhắm đến một nhãn cụ thể (ví dụ: từ 2 thành 6). Công thức được điều chỉnh:
    - $$x' = x - \epsilon \cdot \text{sign}(\nabla_x J(\theta, x, y_{\text{target}}))$$

      Trong đó ytarget là nhãn mục tiêu, và dấu trừ được sử dụng để giảm mất mát theo hướng nhãn mong muốn.

So sánh: Tấn công không mục tiêu dễ thực hiện hơn nhưng ít kiểm soát, trong khi tấn công có mục tiêu đòi hỏi tính toán phức tạp hơn nhưng chính xác hơn.

- Ứng dụng trong phát hiện mã độc:
    - FGSM được sử dụng để tạo ra các mẫu mã độc đối kháng, ví dụ như sửa đổi các byte trong tệp thực thi (PE) để đánh lừa các bộ phân loại học máy (Chio, 2018, trang 199).
    - Trong Foolbox, FGSM được triển khai để kiểm tra độ bền của các mô hình phân loại mã độc, như ResNet50, bằng cách tạo ra các mẫu đối kháng từ các tệp hình ảnh hoặc PE (Chio, 2018, trang 199).
- Ví dụ thực tế:
    - Một nghiên cứu trong Chương 9 đã sử dụng FGSM để tạo ra các mẫu mã độc đối kháng từ tệp hình ảnh, khiến mô hình Keras phân loại sai với độ tin cậy cao (Chio, 2018, trang 199).
    - Việc thêm nhiễu nhỏ vào hình ảnh thang độ xám của tệp NETSTAT đã khiến mô hình phân loại sai thành phần mềm lành tính (Chio, 2018, trang 193).
- Ưu điểm:
    - Tính đơn giản: FGSM chỉ yêu cầu một bước tính gradient, làm cho nó dễ triển khai và tính toán nhanh.
    - Hiệu quả trong môi trường kiểm soát: Phương pháp này hoạt động tốt với các mô hình tuyến tính hoặc mạng nơ-ron nông, như được chứng minh trong các nghiên cứu của Goodfellow (2014).
    - Dễ mở rộng: Có thể tích hợp với các phương pháp khác để tăng cường hiệu quả, như PGD (Projected Gradient Descent).
- Nhược điểm:
    - Hạn chế độ nhiễu: Vì chỉ sử dụng một bước, FGSM có thể không tối ưu hóa hoàn toàn hàm mất mát, dẫn đến tỷ lệ thành công thấp trong các mô hình phức tạp.

- Phụ thuộc vào ϵ: Giá trị ϵ quá lớn có thể làm nhiễu dễ bị phát hiện, trong khi giá trị nhỏ có thể không đủ để đánh lừa mô hình.
- Không hiệu quả với dữ liệu thực tế phức tạp: Trong các tập dữ liệu mã độc thực tế, như BODMAS, FGSM cần điều chỉnh để xử lý không gian đặc trưng cao chiều.

1.3.2 Mạng đối kháng sinh tạo (GANs)

Mạng đối kháng sinh tạo (Generative Adversarial Networks - GANs) là một khung học máy bao gồm hai mạng nơ-ron cạnh tranh: mạng sinh (generator) và mạng phân biệt (discriminator). Mạng sinh tạo ra dữ liệu giả (giả lập mã độc), trong khi mạng phân biệt cố gắng phân biệt giữa dữ liệu thật và dữ liệu giả. Quá trình này giống như một cuộc chơi "min-max", nơi hai mạng cải thiện lẫn nhau.

- Cấu trúc GAN:
  - Bộ sinh (Generator): Nhận đầu vào là vectơ nhiễu ngẫu nhiên (thường từ phân phối chuẩn hoặc đồng nhất). Sử dụng các lớp tích chập ngược (transposed convolution) hoặc lớp tuyến tính để tạo ra dữ liệu có cấu trúc, ví dụ như hình ảnh hoặc đặc trưng mã độc. Được huấn luyện để làm cho mạng phân biệt nhầm dữ liệu giả là thật.
  - Bộ phân biệt (Discriminator): Nhận đầu vào là dữ liệu thật hoặc dữ liệu giả từ mạng sinh. Sử dụng các lớp tích chập (convolutional layers) hoặc lớp tuyến tính để phân loại. Được huấn luyện để cải thiện khả năng phân biệt.
  - Quá trình huấn luyện: Bộ sinh cố gắng đánh lừa bộ phân biệt, trong khi bộ phân biệt cải thiện khả năng phân biệt dữ liệu thật và giả.
- Ứng dụng trong phát hiện mã độc:
  - MalGAN: Một biến thể của GAN được thiết kế để tạo ra các mẫu mã độc đối kháng, có khả năng đánh lừa các bộ phân loại học máy ngay cả trong các kịch bản hộp đen (Hu & Tan, 2017, trang 4). MalGAN lấy đặc trưng của mã độc và nhiễu ngẫu nhiên làm đầu

vào, tạo ra các mẫu mã độc trông giống phần mềm lành tính (Chio, 2018, trang 205).

- Tạo mẫu mã độc đối nghịch: GANs có thể được sử dụng để sinh ra các mẫu mã độc mới, không giống với dữ liệu huấn luyện, giúp kiểm tra độ bền của hệ thống phát hiện. Ví dụ, trong "Master Machine Learning for Pentesters", GANs được đề cập như một công cụ để tạo ra các biến thể của mã độc như NETSTAT.EXE.
- Nâng cao hệ thống phát hiện: Ngược lại, GANs cũng được sử dụng để cải thiện mô hình phân biệt, giúp hệ thống học cách nhận diện các mẫu đối nghịch tốt hơn.

- Ví dụ thực tế:
  - Nghiên cứu của Hu và Tan (2017) đã sử dụng MalGAN để tạo các mẫu mã độc đối kháng, khiến các bộ phân loại nhận diện sai với tỷ lệ cao, ngay cả khi không biết cấu trúc mô hình (Hu & Tan, 2017, trang 4).
  - GAN được huấn luyện trên tập dữ liệu MNIST để tạo các hình ảnh chữ số viết tay giả, minh họa khả năng sinh dữ liệu giống thật, có thể áp dụng tương tự cho các tệp mã độc (Chio, 2018, trang 202).
- Ưu điểm:
  - Khả năng sinh dữ liệu đa dạng: GANs có thể tạo ra các mẫu mã độc độc đáo, không phụ thuộc vào việc sao chép trực tiếp từ dữ liệu thật.
  - Tính linh hoạt: Có thể tích hợp với các kỹ thuật khác như FGSM hoặc Q-learning để tăng cường hiệu quả né tránh.
  - Ứng dụng rộng rãi: Ngoài an ninh mạng, GANs được sử dụng trong hình ảnh, âm thanh, và nhiều lĩnh vực khác.
- Hạn chế:
  - Khó hội tụ: Quá trình huấn luyện GANs thường không ổn định, dễ rơi vào trạng thái "mode collapse" (mạng sinh chỉ tạo ra một số mẫu giới hạn).

- Yêu cầu tài nguyên cao: Huấn luyện GANs đòi hỏi phần cứng mạnh (GPU) và thời gian tính toán dài.
- Phụ thuộc dữ liệu: Hiệu quả của GANs phụ thuộc vào chất lượng và số lượng dữ liệu huấn luyện, vốn có thể bị hạn chế trong lĩnh vực mã độc.

1.3.3 Học tăng cường

Học tăng cường (reinforcement learning) là một phương pháp học máy trong đó một tác nhân (agent) học cách đưa ra quyết định tối ưu bằng cách tương tác với môi trường, nhận phần thưởng hoặc hình phạt dựa trên hành động (Sutton & Barto, 2018, trang 1). Trong bối cảnh phát hiện mã độc, học tăng cường được sử dụng để tối ưu hóa các hành động sửa đổi mã độc nhằm né tránh phát hiện.

- Nguyên lý hoạt động:
  - Học tăng cường khác với học có giám sát (supervised learning) ở chỗ không yêu cầu dữ liệu nhãn sẵn có. Thay vào đó, tác nhân học qua thử và sai, dựa trên phần thưởng nhận được từ môi trường.
  - Tác nhân tương tác với môi trường qua các chu kỳ "state-action-reward", điều chỉnh chính sách để tối ưu hóa hàm mục tiêu, thường là tổng phần thưởng kỳ vọng (expected cumulative reward).
  - Tác nhân nhận trạng thái (state) từ môi trường (ví dụ: thông tin tệp PE) và thực hiện hành động (action) như thêm byte hoặc xóa chữ ký.
  - Môi trường trả về phần thưởng (reward) dựa trên kết quả, ví dụ: báo cáo “lành tính” từ hệ thống chống virus.
  - Tác nhân tối ưu hóa chính sách (policy) để đạt được phần thưởng tối đa.
- Mô hình Markov Decision Process (MDP): Học tăng cường thường được mô hình hóa bằng MDP, bao gồm:
  - Trạng thái (State - S): Mô tả tình trạng hiện tại của môi trường, ví dụ: cấu trúc mã độc hoặc hành vi hệ thống phát hiện.

- Hành động (Action - A): Các thao tác tác nhân có thể thực hiện, như thay đổi đặc trưng mã độc.
- Chính sách (Policy - π): Quy tắc xác định hành động dựa trên trạng thái, nhằm tối đa hóa phần thưởng.
- Phần thưởng (Reward - R): Phản hồi từ môi trường, ví dụ: thành công né tránh phát hiện.
- Hàm giá trị (Value Function): Đánh giá tổng phần thưởng kỳ vọng trong tương lai.

- Thuật toán Q-learning
  - •Là thuật toán không mô hình (model-free), sử dụng bảng Q (Q-table) để lưu trữ giá trị kỳ vọng của mỗi cặp trạng thái-hành động.
  - Công thức:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$

    Trong đó:
    - α: Tỷ lệ học (learning rate).
    - γ: Hệ số chiết khấu (discount factor).
    - r: Phần thưởng tức thời.
    - s′: Trạng thái tiếp theo.
  - Ứng dụng: Q-learning được dùng để tối ưu hóa hành vi né tránh của mã độc.
- Deep Q-Network (DQN)
  - Kết hợp Q-learning với mạng nơ-ron sâu để xử lý không gian trạng thái lớn.
  - Sử dụng kỹ thuật như experience replay và target network để ổn định huấn luyện.
  - Ví dụ: DQN có thể được áp dụng để học chiến lược né tránh trên dữ liệu mã độc phức tạp.
- Ứng dụng trong phát hiện mã độc:

- Mô phỏng môi trường: Sử dụng thư viện Gym để tạo môi trường ảo, trong đó tác nhân là mã độc và mục tiêu là né tránh hệ thống phát hiện (như trong gym_mal.ipynb).
- Tối ưu hóa hành vi: Tác nhân học cách thay đổi đặc trưng (ví dụ: thêm nhiễu, sửa mã lệnh) để đạt phần thưởng cao nhất, thường là né tránh phát hiện thành công.
- Kiểm tra độ bền: Học tăng cường giúp đánh giá khả năng thích ứng của hệ thống phát hiện trước các chiến lược tấn công linh hoạt.
- Thách thức thực tế: Dữ liệu mã độc phức tạp (như BODMAS với 2381 đặc trưng) đòi hỏi mô hình RL phải được tinh chỉnh để xử lý hiệu quả.

- Ví dụ thực tế:
  - Trong môi trường Gym-malware, các hành động như append_random_bytes hoặc remove_signature được tối ưu hóa để giảm khả năng bị phát hiện bởi các hệ thống chống virus (Chio, 2018, trang 209).
  - Phép ẩn dụ Atari Breakout minh họa cách học tăng cường huấn luyện một tác nhân phá gạch, tương tự như cách tác nhân học sửa đổi mã độc để né tránh phát hiện (Chio, 2018, trang 207).
- Ưu điểm:
  - Tính linh hoạt: RL có thể học chiến lược né tránh mà không cần dữ liệu huấn luyện có nhãn sẵn có.
  - Khả năng thích ứng: Tác nhân có thể điều chỉnh hành vi theo thay đổi của hệ thống phát hiện.
  - Mô phỏng thực tế: Môi trường ảo trong Gym cho phép thử nghiệm an toàn các chiến lược tấn công.
- Hạn chế:
  - Yêu cầu tài nguyên cao: Huấn luyện RL, đặc biệt là DQN, đòi hỏi thời gian và phần cứng mạnh.

- Khó hội tụ: Quá trình học có thể không ổn định, đặc biệt trong môi trường phức tạp.
- Phụ thuộc phần thưởng: Thiết kế phần thưởng không hợp lý có thể dẫn đến hành vi không mong muốn.

Kết luận chương 1

Chương 1 đã trình bày tổng quan về vấn đề nghiên cứu, nhấn mạnh tính cấp thiết của việc vượt qua hệ thống phát hiện mã độc dựa trên học máy trong bối cảnh an ninh mạng ngày càng phức tạp. Các kỹ thuật học máy đối kháng chẳng hạn như sử dụng FGSM để tạo nhiễu đánh lừa mô hình phân loại đã được giới thiệu như nền tảng lý thuyết quan trọng. Mục tiêu của đồ án được xác định rõ ràng, tập trung vào việc xây dựng mẫu mã độc đối kháng và tối ưu hóa bằng học tăng cường, nhằm đánh giá lỗ hổng của các hệ thống phát hiện. Kết quả từ chương này cung cấp cơ sở vững chắc để triển khai các giải pháp thực tiễn trong các chương tiếp theo, đồng thời góp phần nâng cao nhận thức về an ninh mạng ở Việt Nam.

CHƯƠNG 2: TRIỂN KHAI VÀ XỬ LÝ DỮ LIỆU

Trong lĩnh vực an ninh mạng, việc triển khai các kỹ thuật học máy để phát hiện mã độc (malware) đã trở thành một giải pháp mạnh mẽ, cho phép phân tích dữ liệu phức tạp và thích nghi với các mối đe dọa mới. Tuy nhiên, các hệ thống phát hiện mã độc dựa trên học máy cũng dễ bị tấn công bởi các kỹ thuật đối kháng (adversarial attacks), như được đề cập trong Chương 9 của *Mastering Machine Learning for Penetration Testing* (Chio, 2018, Chương 9). Để vượt qua các hệ thống này, quá trình chuẩn bị và xử lý dữ liệu đóng vai trò quan trọng trong việc xây dựng các mẫu mã độc đối kháng

2.1.Triển khai các kỹ thuật vượt qua hệ thống phát hiện mã độc

2.1.1. Tổng quan về quá trình triển khai

Quá trình triển khai các kỹ thuật vượt qua hệ thống phát hiện mã độc dựa trên học máy bao gồm ba giai đoạn chính: chuẩn bị dữ liệu, xây dựng mô hình, và áp dụng các thuật toán đối kháng. Mỗi giai đoạn đều yêu cầu sự cẩn thận để đảm bảo rằng các mẫu mã độc đối kháng được tạo ra không chỉ đánh lừa được mô hình mà còn giữ được tính hợp lệ về mặt chức năng.

- Chuẩn bị dữ liệu: Thu thập, làm sạch, và tiền xử lý dữ liệu để phù hợp với các mô hình học máy.
- Xây dựng mô hình: Sử dụng các công cụ để huấn luyện hoặc kiểm tra các mô hình đối kháng.
- Áp dụng thuật toán: Triển khai các thuật toán như Fast Gradient Sign Method (FGSM), hoặc học tăng cường để tạo ra các mẫu mã độc đối kháng.

2.1.2. Vai trò của các công cụ

- Python – Ngôn ngữ lập trình cốt lõi. Python là một ngôn ngữ lập trình mã nguồn mở, được sử dụng rộng rãi trong lĩnh vực học máy và an ninh mạng nhờ cú pháp đơn giản và hệ sinh thái thư viện phong phú. Phiên bản 3.12.10, được sử dụng trong nghiên cứu này, cung cấp hiệu suất ổn định và hỗ trợ các công cụ hiện đại. Nó cho phép tích hợp dễ dàng với các thư

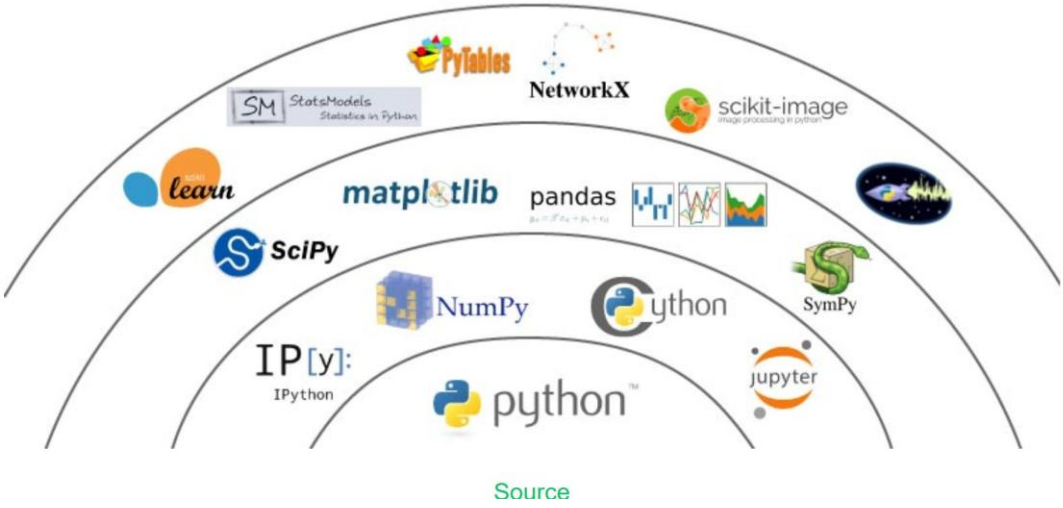
viện như NumPy, PyTorch và LightGBM, tạo điều kiện thuận lợi cho việc phát triển mô hình học máy và tấn công đối kháng (adversarial attack).

- Jupyter Notebook – Môi trường phát triển tích hợp. Jupyter Notebook là một công cụ mã nguồn mở, cho phép viết, thực thi và trực quan hóa mã lập trình trong môi trường tương tác. Nó đặc biệt phù hợp với các dự án nghiên cứu học máy. Khả năng kết hợp mã, văn bản và hình ảnh trong cùng một tài liệu, hỗ trợ trực quan hóa dữ liệu qua các biểu đồ, rất hữu ích cho việc trình bày kết quả, giúpdễ dàng theo dõi quá trình huấn luyện mô hình, phân tích kết quả và điều chỉnh tham số.

Hình 2.1: Jupyter và Python

- NumPy và Pandas – Công cụ xử lý dữ liệu. NumPy: Thư viện cung cấp các mảng đa chiều và các phép toán số học hiệu quả. Pandas: Thư viện hỗ trợ quản lý dữ liệu dạng bảng, đặc biệt hữu ích khi làm việc với các tệp CSV. Tốc độ xử lý nhanh và khả năng tích hợp với các thư viện học máy khác, giúp tối ưu hóa quá trình tiền xử lý dữ liệu – bước quan trọng trong việc xây dựng mô hình phát hiện mã độc.
- Matplotlib – Thư viện trực quan hóa. Matplotlib là một thư viện trực quan hóa dữ liệu phổ biến, cho phép tạo biểu đồ và hình ảnh minh họa. Linh hoạt trong việc tạo các loại biểu đồ khác nhau, hỗ trợ trực quan hóa kết quả nghiên cứu một cách khoa học và chuyên nghiệp.
- LightGBM – Thư viện học máy tăng cường gradient. Là một thư viện học máy hiệu quả, được phát triển bởi Microsoft, tập trung vào việc tăng tốc độ huấn luyện và dự đoán so với các thuật toán truyền thống như XGBoost. Hỗ trợ xử lý dữ liệu lớn, tối ưu hóa bộ nhớ và khả năng chọn lọc các đặc trưng quan trọng (top 10 feature indices), cung cấp cơ sở cho các chiến lược tránh phát hiện.

- PyTorch – Khung học sâu và tấn công đối kháng. PyTorch, phát triển bởi Facebook, là một khung học sâu linh hoạt, được sử dụng rộng rãi trong nghiên cứu tấn công đối kháng (adversarial machine learning). Hỗ trợ tính toán tự động đạo hàm (autograd) và khả năng tùy chỉnh mô hình, rất phù hợp cho việc thử nghiệm các kỹ thuật vượt qua hệ thống phát hiện.
- Gymnasium – Môi trường học tăng cường. Gymnasium, phiên bản cải tiến của OpenAI Gym, là một nền tảng cung cấp môi trường để huấn luyện các tác nhân học tăng cường (reinforcement learning). Hỗ trợ tùy chỉnh môi trường, phù hợp với việc mô phỏng các kịch bản tránh phát hiện trong không gian đặc trưng hạn chế.



Hình 2.2: Python và các thư viện liên quan

2.2.Chuẩn bị và xử lý dữ liệu

2.2.1. Nguồn dữ liệu

Việc thu thập dữ liệu chất lượng cao là bước đầu tiên và quan trọng trong quá trình triển khai các kỹ thuật vượt qua hệ thống phát hiện mã độc. Các nguồn dữ liệu phổ biến bao gồm EMBER, BODMAS, và các tập dữ liệu mã độc công khai.

Các tập dữ liệu mã độc công khai, như BODMAS hoặc EMBER, cung cấp các mẫu mã độc thực tế để huấn luyện và kiểm tra các mô hình học máy.

- Đặc điểm:

- Chứa hàng nghìn mẫu mã độc, bao gồm virus, ransomware, trojan, và worm.
- Cung cấp các đặc trưng tĩnh (static features) như thông tin tiêu đề PE, chuỗi ký tự, hoặc hình ảnh thang độ xám (Chio, 2018, trang 193).
- Thường yêu cầu xử lý thêm để đảm bảo tính đồng nhất và chất lượng.

- Ứng dụng trong phát hiện mã độc:
  - Nghiên cứu của SARVAM tại Đại học California, Santa Barbara, đã sử dụng các tập dữ liệu mã độc công khai để chuyển đổi tệp PE thành hình ảnh thang độ xám, sau đó huấn luyện các mô hình phân loại (Chio, 2018, trang 193).
  - Các tập dữ liệu này được sử dụng để kiểm tra độ bền của các mô hình trước các cuộc tấn công đối kháng, như FGSM hoặc MalGAN.

2.2.2. Quy trình tiền xử lý dữ liệu

Tiền xử lý dữ liệu là một bước quan trọng để đảm bảo rằng dữ liệu phù hợp với các mô hình học máy, đặc biệt trong bối cảnh phát hiện và né tránh mã độc. Quy trình này bao gồm làm sạch dữ liệu, trích xuất đặc trưng, và chuẩn hóa dữ liệu.

2.2.2.1. Làm sạch dữ liệu

Làm sạch dữ liệu đảm bảo rằng dữ liệu không chứa các giá trị bị thiếu, nhiễu hoặc không nhất quán, giúp cải thiện hiệu suất của mô hình học máy.

- Các bước chính:
  - Loại bỏ dữ liệu không hợp lệ: Xóa các tệp bị hỏng hoặc không thể phân tích từ tập dữ liệu mã độc (Chio, 2018, trang 193).
  - Xử lý giá trị bị thiếu: Sử dụng các kỹ thuật như thay thế bằng giá trị trung bình hoặc loại bỏ các mẫu thiếu dữ liệu quan trọng.
  - Loại bỏ nhiễu: Lọc bỏ các đặc trưng không liên quan, như các chuỗi ký tự ngẫu nhiên không mang ý nghĩa.

2.2.2.2. Trích xuất đặc trưng

Trích xuất đặc trưng (feature extraction) chuyển đổi dữ liệu thô thành các đặc trưng có ý nghĩa, phù hợp với các mô hình học máy. Các đặc trưng phổ biến bao gồm đặc trưng tĩnh và đặc trưng động.

- Đặc trưng tĩnh:
    - Thông tin tiêu đề PE: Kích thước tệp, số lượng phần (sections), hoặc thông tin nhập/xuất (imports/exports) (Chio, 2018, trang 193).
    - Chuỗi ký tự: Các chuỗi văn bản được trích xuất từ tệp, như URL hoặc thông tin API.
    - Hình ảnh thang độ xám: Chuyển đổi tệp PE thành hình ảnh 2D, trong đó mỗi byte được biểu diễn bằng một pixel (Chio, 2018, trang 193).
- Đặc trưng động:
    - Hành vi hệ thống: Các hành động như truy cập tệp, sửa đổi registry, hoặc kết nối mạng, được thu thập từ Cuckoo Sandbox (Chio, 2018, trang 201).
    - API calls: Danh sách các hàm API được gọi bởi mã độc, như CreateFile hoặc Connect (Chio, 2018, trang 201).

2.2.2.3. Chuẩn hóa dữ liệu

Chuẩn hóa dữ liệu đảm bảo rằng các đặc trưng có cùng thang đo, giúp cải thiện hiệu quả huấn luyện của các mô hình học máy.

- Các kỹ thuật chuẩn hóa:
    - Chuẩn hóa min-max: Chuyển đổi dữ liệu về khoảng [0, 1] bằng công thức: (x' = \frac{x - \min(x)}{\max(x) - \min(x)}).
    - Chuẩn hóa Z-score: Chuyển đổi dữ liệu về phân phối chuẩn với trung bình 0 và độ lệch chuẩn 1: (x' = \frac{x - \mu}{\sigma}).
    - Chuyển đổi log: Áp dụng hàm log để giảm độ lệch của các đặc trưng có giá trị lớn (Goodfellow et al., 2016, trang 78).

2.3.Các thuật toán xử lý dữ liệu và triển khai

Các thuật toán học máy được sử dụng để xử lý dữ liệu và triển khai các kỹ thuật vượt qua hệ thống phát hiện mã độc bao gồm các phương pháp đối kháng như FGSM và học tăng cường

2.3.1. Fast Gradient Sign Method (FGSM)

2.3.1.1. Giới Thiệu Về Hệ Thống Phát Hiện Mã Độc (Malware Detector)

File PDF (Portable Document Format) là một trong những định dạng file phổ biến nhất bị lợi dụng để phân phối malware, do tính linh hoạt và khả năng che giấu mã độc mà không ảnh hưởng đến giao diện hiển thị. Được phát triển bởi Adobe, PDF cho phép nhúng các yếu tố phức tạp như đối tượng (objects), tham chiếu chéo (cross-references), metadata, và mã JavaScript, tạo điều kiện lý tưởng cho kẻ tấn công ẩn náu payload độc hại. Theo thống kê từ các báo cáo an ninh mạng, hơn 80% các cuộc tấn công qua email liên quan đến file PDF đính kèm chứa malware, vì định dạng này dễ dàng vượt qua các bộ lọc email cơ bản và chỉ kích hoạt mã độc khi người dùng mở file bằng reader như Adobe Acrobat hoặc Foxit.

Các kỹ thuật phổ biến để ẩn malware trong PDF bao gồm:

- Nhúng JavaScript: Sử dụng để khai thác lỗ hổng trong PDF reader, chẳng hạn như CVE-2018-4878, dẫn đến thực thi mã từ xa (remote code execution).
- Embedded Files hoặc Objects: Ẩn payload dưới dạng file nhúng, chỉ kích hoạt khi có hành động cụ thể từ người dùng.
- Thao Túng Metadata: Thay đổi thông tin tài liệu để tránh phát hiện, hoặc sử dụng annotations để che giấu mã.
- Obfuscation Cấu Trúc: Tạo các tham chiếu chéo phức tạp để làm khó phân tích tĩnh.

Để phát hiện malware trong PDF, các hệ thống ML thường ưu tiên phân tích tĩnh (static analysis) dựa trên đặc trưng cấu trúc, thay vì phân tích động để tránh rủi ro thực thi mã độc. Phân tích tĩnh nhanh chóng, an toàn và ít tốn tài

nguyên hơn, nhưng đòi hỏi việc chọn lọc đặc trưng phù hợp để đạt độ chính xác cao.

Trong thí nghiệm của luận án này sử dụng một bộ 7 đặc trưng được trích xuất bằng thư viện PyPDF2. Các đặc trưng này được chọn vì chúng phản ánh các dấu hiệu bất thường thường thấy trong malware PDF:

- Số lượng trang (page count): Malware PDF thường có số trang ít để giảm kích thước và tránh nghi ngờ, nhưng đôi khi thêm trang giả để che giấu mã. Giá trị bất thường (quá ít hoặc quá nhiều) có thể là indicator.
- Sự hiện diện của metadata (metadata presence): Metadata có thể bị thao túng để chứa thông tin độc hại hoặc thiếu hoàn toàn để tránh phân tích.
- Số lượng tham chiếu chéo (xref count): Giá trị cao chỉ ra cấu trúc phức tạp, phổ biến trong file bị obfuscate để ẩn malware.
- Sự tồn tại của JavaScript (has_js): Đây là đặc trưng quan trọng nhất, vì JavaScript thường được dùng để khai thác lỗ hổng, dẫn đến tỷ lệ phát hiện cao lên đến 90% trong một số nghiên cứu.
- Kích thước file (file size): File lớn bất thường có thể chứa embedded payload hoặc mã dư thừa để che giấu.
- Số lượng trường thông tin tài liệu (len(info)): Metadata chi tiết hơn mức cần thiết có thể là dấu hiệu của sự thao túng.
- Số lượng đối tượng nhúng (embedded): Bao gồm embedded files hoặc annotations, thường dùng để ẩn mã độc mà không ảnh hưởng đến nội dung chính.

Quá trình trích xuất bao gồm mở file bằng PyPDF2, đọc các thuộc tính, và xử lý ngoại lệ (exceptions) để đảm bảo tính mạnh mẽ. Sau đó, các đặc trưng được chuẩn hóa bằng cách chia cho giá trị tối đa (ví dụ: 100 cho page count, 1e7 cho file size) và clip trong khoảng [0,1] để phù hợp với input của mô hình ML. Phương pháp này đơn giản nhưng hiệu quả, như được chứng minh trong các nghiên cứu sử dụng chỉ 5-10 features để đạt accuracy trên 95%.

Mô hình detector trong thí nghiệm được xây dựng bằng PyTorch, một framework phổ biến cho deep learning nhờ tính linh hoạt và hỗ trợ gradient

computation – yếu tố quan trọng cho các tấn công đối kháng sau này. Kiến trúc là một mạng nơ-ron feed-forward đơn giản, phù hợp với dữ liệu đặc trưng nhỏ gọn:

- Lớp đầu vào: Linear layer từ input_dim=7 đến 64 nơ-ron, với activation ReLU để giới thiệu tính phi tuyến tính.
- Lớp ẩn: Linear từ 64 đến 32 nơ-ron, cũng với ReLU.
- Lớp đầu ra: Linear từ 32 đến 1 nơ-ron, theo sau bởi Sigmoid để dự đoán xác suất (0: benign, 1: malicious).

Quá trình huấn luyện sử dụng dữ liệu tổng hợp (synthetic data) với 5000 mẫu để simulate môi trường thực tế mà không cần dataset lớn. Features được tạo ngẫu nhiên bằng torch.rand, labels bằng torch.randint (0 hoặc 1). Hàm mất mát là Binary Cross-Entropy Loss (BCELoss), optimizer là Adam với learning rate 0.001. Huấn luyện diễn ra trong 10 epochs, batch size 64, với kiểm tra loss mỗi epoch. Nếu file mô hình đã tồn tại (malware_detector_pdf.pth), nó sẽ được tải; ngược lại, huấn luyện mới và lưu lại.

Ưu điểm của kiến trúc này:

- Đơn giản, dễ triển khai với kiến thức cơ bản về ML.
- Nhanh chóng huấn luyện trên CPU thông thường.
- Dễ dàng mở rộng bằng cách thêm layers hoặc sử dụng transfer learning từ pre-trained models.

Nhược điểm:

- Dễ overfitting với synthetic data.
- Không xử lý tốt dữ liệu thực tế đa dạng.
- Thiếu robustness chống adversarial attacks.

2.3.1.2. Tổng quan về Fast Gradient Sign Method và vai trò của nó trong Adversarial Machine Learning

Fast Gradient Sign Method (FGSM) là một kỹ thuật tấn công đối kháng (adversarial attack) phổ biến trong lĩnh vực học máy (machine learning - ML), được thiết kế để tạo ra các ví dụ đối kháng (adversarial examples) nhằm lừa dối

các mô hình ML mà không làm thay đổi đáng kể input gốc. FGSM thuộc loại tấn công dựa trên gradient (gradient-based attack), nơi kẻ tấn công sử dụng thông tin gradient của hàm mất mát (loss function) để điều chỉnh input theo hướng làm tăng lỗi phân loại của mô hình. Kỹ thuật này đặc biệt hiệu quả đối với các mô hình mạng nơ-ron (neural networks), vốn dễ bị ảnh hưởng bởi các perturbation nhỏ do tính tuyến tính (linearity) trong không gian chiều cao của chúng.

Trong bối cảnh adversarial machine learning, FGSM được coi là một phương pháp "white-box attack", nghĩa là kẻ tấn công cần có kiến thức đầy đủ về kiến trúc mô hình, tham số và hàm mất mát để tính toán gradient. Ý tưởng cốt lõi là thêm một perturbation (nhiễu) nhỏ vào input gốc, với magnitude được kiểm soát bởi tham số epsilon (ε), để làm cho mô hình dự đoán sai mà vẫn giữ input trông giống hệt với mắt người (trong trường hợp hình ảnh) hoặc không thay đổi chức năng (trong trường hợp dữ liệu khác như features từ file). FGSM không chỉ đơn giản mà còn nhanh chóng, vì nó chỉ yêu cầu một bước tính toán duy nhất (one-step attack), khác với các phương pháp lặp lại (iterative attacks) đòi hỏi nhiều vòng lặp.

Vai trò của FGSM trong adversarial ML là rất quan trọng, vì nó giúp làm nổi bật lỗ hổng của các mô hình ML hiện đại, đặc biệt trong các ứng dụng thực tế như nhận diện hình ảnh, xử lý ngôn ngữ tự nhiên và phát hiện mã độc. Ví dụ, trong nhận diện hình ảnh, một perturbation nhỏ có thể làm mô hình nhầm lẫn giữa "panda" và "gibbon" với độ tin cậy cao. Trong an ninh mạng, FGSM có thể được sử dụng để bypass các hệ thống phát hiện malware dựa trên ML bằng cách điều chỉnh features của file mà không làm thay đổi mã độc hại.

Các đặc điểm chính của FGSM bao gồm:

- Tốc độ cao: Chỉ cần một lần forward và backward pass để tính gradient.
- Hiệu quả với epsilon nhỏ: Perturbation có thể rất nhỏ ($\varepsilon \approx 0.01$-0.3) nhưng vẫn gây misclassification.
- Dễ triển khai: Có thể thực hiện bằng các framework như TensorFlow hoặc PyTorch với vài dòng code.

- Hạn chế: Ít stealthy hơn so với các attack phức tạp, dễ bị phát hiện bởi defenses như adversarial training.

FGSM có ứng dụng rộng rãi trong an ninh mạng, đặc biệt trong việc test và bypass các hệ thống dựa trên ML. Trong phát hiện malware, FGSM có thể điều chỉnh features của file (như trong PDF: page count, has_js) để làm detector phân loại sai từ malicious sang benign. Ví dụ, trong chapter 9 của sách "Mastering Machine Learning for Penetration Testing", FGSM được dùng để bypass detector dựa trên features từ PDF.

Các ứng dụng khác:

- Phát hiện intrusion: Perturb network traffic features để bypass IDS/IPS dựa trên ML.
- Phishing detection: Thay đổi text/email để lừa classifier.
- Image-based security: Attack facial recognition trong access control.
- Robustness testing: Sử dụng FGSM trong adversarial training để tăng khả năng chống chịu.

2.3.1.3. Triển Khai Chung Của FGSM Ở Mức Thuật Toán (Pseudocode)

```
function fgsm_attack(model, loss_fn, input_data,
true_label, epsilon): # Bước 1: Cho phép tính gradient trên
input input_data.requires_gradient = True
# Bước 2: Tính forward pass và loss output =
model(input_data) loss = loss_fn(output, true_label)
# Bước 3: Backpropagate để lấy gradient model.zero_grad()
loss.backward() grad = input_data.grad
# Bước 4: Tạo perturbation perturbation = epsilon *
sign(grad)
# Bước 5: Tạo adversarial input và clamp adv_input =
input_data + perturbation adv_input = clamp(adv_input,
min=0, max=1) # Giả sử input chuẩn hóa [0,1]
return adv_input
```

Giải thích pseudocode:

- Bước 1: Kích hoạt gradient tracking để tính đạo hàm đối với input.
- Bước 2: Chạy mô hình và tính loss (ví dụ: BCELoss cho binary classification).

- Bước 3: Xóa gradient cũ và backprop để lấy ∇_x J.
- Bước 4: Sử dụng sign để tạo nhiễu bounded bằng ε.
- Bước 5: Thêm nhiễu và clamp để tránh giá trị ngoài range, đảm bảo tính thực tế.

2.3.1.4. Triển Khai Cụ Thể:

- Định nghĩa hàm fgsm_attack

```
def fgsm_attack(model, criterion, data, label,
epsilon=0.1):
    data.requires_grad = True
    output = model(data)
    loss = criterion(output, label)
    model.zero_grad()
    loss.backward()
    data_grad = data.grad.data
    sign_data_grad = data_grad.sign()
    perturbed_data = data + epsilon * sign_data_grad
    perturbed_data = torch.clamp(perturbed_data, 0, 1)
    return perturbed_data
```

Giải thích code:

- data.requires_grad = True: Kích hoạt gradient tracking cho input data (tensor của features).
- output = model(data): Forward pass để lấy prediction.
- loss = criterion(output, label): Tính loss (ở đây là BCELoss, label=1.0 cho malicious).
- model.zero_grad() và loss.backward(): Xóa gradient và backprop để tính ∇_data loss.
- sign_data_grad = data_grad.sign(): Lấy sign của gradient.
- perturbed_data = data + epsilon * sign_data_grad: Tạo adversarial data.
- torch.clamp(perturbed_data, 0, 1): Giới hạn giá trị trong [0,1] vì features đã chuẩn hóa.

Hàm này là core của FGSM, và có thể được gọi với data từ features PDF.

- Tích hợp với feature extraction

```
def extract_pdf_features(file_path):
    try:
        with open(file_path, 'rb') as f:
            reader = PyPDF2.PdfReader(f)
            features = []
            features.append(len(reader.pages))  # Page
count
            features.append(1 if reader.metadata else 0)  #
Metadata presence
            xref_count = len(reader.xref) if
hasattr(reader, 'xref') else 0
            features.append(xref_count)
            has_js = 0
            for page in reader.pages:
                if '/JS' in page or '/JavaScript' in page:
                    has_js = 1
                    break
            features.append(has_js)
            features.append(os.path.getsize(file_path))
            info = reader.metadata or {}
            features.append(len(info))
            embedded = 0
            for page in reader.pages:
                if '/EmbeddedFile' in page or '/Annots' in
page:
                    embedded += 1
            features.append(embedded)
            features = np.array(features, dtype=np.float32)
            max_values = np.array([100, 1, 1000, 1, 1e7,
10, 100], dtype=np.float32)
            features = np.clip(features / max_values, 0, 1)
            return features
    except Exception as e:
        print(f"PDF parsing error: {e}")
        return None
```

Giải thích code:

- Mở file PDF với PyPDF2 và trích xuất 7 features như đã mô tả ở phần detector.
- Sử dụng loop để check JavaScript và embedded objects.
- Chuẩn hóa features bằng chia max_values và clip [0,1] để phù hợp với model input.
- Xử lý exception để tránh crash khi file lỗi.

Features này sau đó được convert thành tensor để áp dụng FGSM.

- Áp dụng FGSM trên features thực tế

```
features_tensor = torch.from_numpy(features).unsqueeze(0)
label = torch.tensor([[1.0]])
adv_features = fgsm_attack(model, criterion,
features_tensor.clone(), label, epsilon=0.1)
with torch.no_grad():
    adv_pred = model(adv_features).item()
```

Giải thích code:

- Convert features numpy sang tensor và thêm batch dim (unsqueeze(0)).
- Clone tensor để tránh modify gốc.
- Gọi fgsm_attack với criterion=nn.BCELoss().
- Dự đoán trên adv_features với no_grad() để tiết kiệm memory.

- Thử nghiệm trên nhiều Epsilons

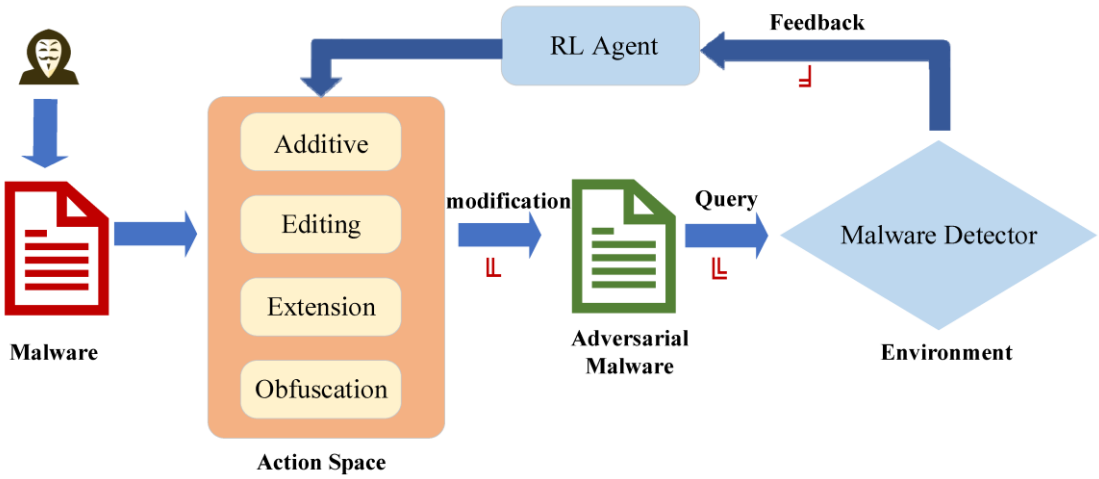
```
epsilons = [0.05, 0.1, 0.2]
for epsilon in epsilons:
    adv_features = fgsm_attack(model, criterion,
features_tensor.clone(), label, epsilon)
    with torch.no_grad():
        adv_pred = model(adv_features).item()
    evaded = adv_pred < 0.5
    print(f"Epsilon {epsilon}: Prediction
{adv_pred:.4f}, Evaded: {evaded}")
```

Giải thích code:

- Loop qua các ε để test sensitivity.
- Kiểm tra evasion nếu pred < 0.5 (benign threshold).

- Trong file notebook, các thư viện chính bao gồm:
  - PyTorch: Để xây dựng model, tính gradient (torch.nn, torch.optim).
  - PyPDF2: Trích xuất features từ PDF.
  - NumPy và OS: Xử lý array và file size.
  - Torch utilities: Như relu, sigmoid cho model.

2.3.2. Học tăng cường Re-inforcement Learning



Hình 2.3: Học tăng cường trong phát hiện mã độc

Gymnasium là một thư viện mã nguồn mở được sử dụng rộng rãi để phát triển và so sánh các thuật toán học tăng cường . Đây là phiên bản fork từ thư viện Gym gốc của OpenAI, được duy trì bởi Farama Foundation từ năm 2022, nhằm khắc phục các vấn đề về bảo trì và tích hợp các tính năng mới . Lịch sử của Gymnasium bắt nguồn từ Gym, được OpenAI phát hành năm 2016 như một giao diện chuẩn (API) cho môi trường RL, giúp các nhà nghiên cứu dễ dàng thử nghiệm mà không cần xây dựng môi trường từ đầu . Đến năm 2022, do OpenAI không còn hỗ trợ tích cực, cộng đồng đã fork thành Gymnasium, thêm hỗ trợ cho Python hiện đại, vectorized environments (cho phép chạy song song nhiều môi trường), và tích hợp tốt hơn với các thư viện như Stable Baselines3 hoặc RLlib .

Đặc điểm chính của Gymnasium bao gồm:

- Giao diện chuẩn hóa: Môi trường (Environment) được định nghĩa với các phương thức như reset() (khởi tạo trạng thái), step(action) (thực hiện hành động và trả về phần thưởng, trạng thái mới), và render() (hiển thị). Không gian hành động (Action Space) và quan sát (Observation Space) được hỗ trợ đa dạng, từ rời rạc (Discrete) đến liên tục (Box) .
- Tích hợp với các thuật toán RL: Hỗ trợ các mô hình như Q-Learning, DQN (Deep Q-Network), PPO (Proximal Policy Optimization), giúp dễ dàng huấn luyện agent.

- Mở rộng: Người dùng có thể tạo môi trường tùy chỉnh bằng cách kế thừa lớp gym.Env, như trong nghiên cứu này, chúng ta xây dựng môi trường mã độc để agent học cách biến đổi đặc trưng.

Trong lĩnh vực an ninh mạng, Gymnasium được áp dụng ngày càng phổ biến để mô phỏng các tình huống tấn công và phòng thủ. Ví dụ, MininetGym sử dụng Gymnasium để mô hình hóa mạng SDN (Software-Defined Networking) và huấn luyện RL chống lại các cuộc tấn công DDoS hoặc xâm nhập . Trong phát hiện và vượt qua mã độc, Gymnasium cho phép tạo môi trường nơi agent học cách biến đổi mã độc để tránh phát hiện, như trong mô hình "gym-malware" hoặc "MininetGym" mở rộng cho an ninh mạng . Các nghiên cứu gần đây (2023-2025) cho thấy RL dựa trên Gymnasium đạt tỷ lệ vượt qua từ 20-50% đối với các mô hình phát hiện tĩnh , cao hơn so với phương pháp ngẫu nhiên nhờ khả năng học thích nghi.

2.3.2.1. Thuật Toán LightGBM Trong Học Máy

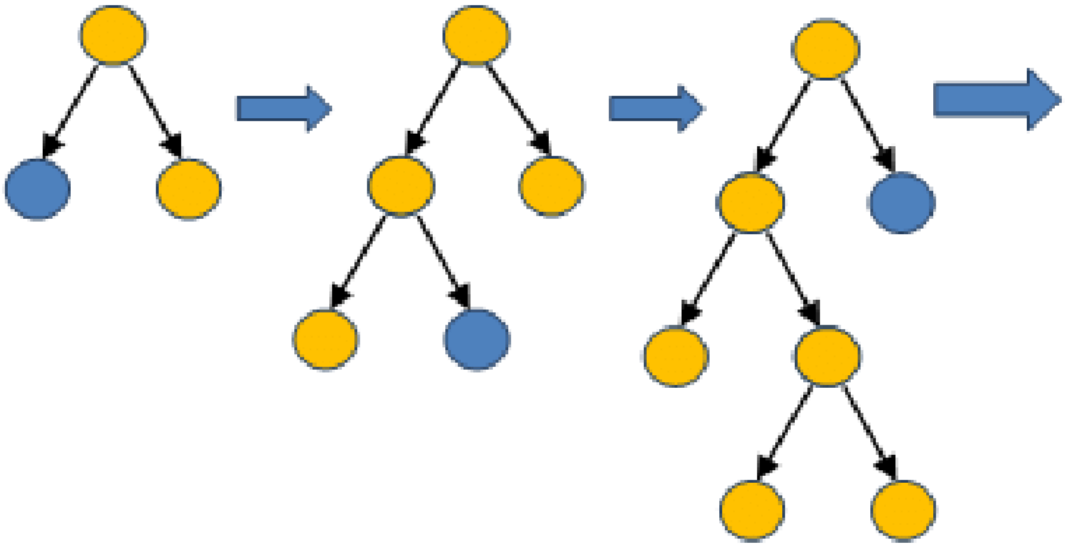
LightGBM (Light Gradient Boosting Machine) là một khung tăng cường gradient (Gradient Boosting Framework) hiệu quả, được phát triển bởi Microsoft và phát hành mã nguồn mở năm 2016 . Lịch sử của LightGBM bắt nguồn từ nhu cầu xử lý dữ liệu lớn trong học máy, nơi các thuật toán truyền thống như GBM (Gradient Boosting Machine) chậm và tiêu tốn bộ nhớ. LightGBM sử dụng kỹ thuật histogram-based để phân chia dữ liệu, giảm thời gian tính toán và bộ nhớ so với XGBoost (một thuật toán tương tự) lên đến 20-30 lần . Đến năm 2023-2025, LightGBM đã được cập nhật với hỗ trợ GPU, xử lý dữ liệu thưa thớt tốt hơn, và tích hợp với các công cụ như Scikit-learn, làm cho nó trở thành lựa chọn hàng đầu trong các cuộc thi Kaggle và ứng dụng thực tế .

Các đặc điểm kỹ thuật chính:

1. Leaf-wise Tree Growth: Xây dựng cây theo chiều sâu lá thay vì mức độ, giảm lỗi nhanh hơn nhưng có thể dẫn đến overfit nếu không điều chỉnh.
2. Histogram-based Splitting: Phân loại đặc trưng thành bin, giảm chi phí tính toán O(data) thay vì O(data * features).

3. Hỗ trợ song song: Sử dụng GOSS (Gradient-based One-Side Sampling) và EFB (Exclusive Feature Bundling) để xử lý dữ liệu lớn.
4. Tham số chính: num_leaves (số lá tối đa), learning_rate (tỷ lệ học), feature_fraction (tỷ lệ đặc trưng sử dụng) .

Trong phát hiện mã độc, LightGBM nổi bật nhờ tốc độ và độ chính xác cao. Các nghiên cứu cho thấy nó đạt độ chính xác 98-99% trên dữ liệu PE (Portable Executable), vượt trội hơn Random Forest hoặc SVM trong việc xử lý đặc trưng cao chiều . Ví dụ, một mô hình LightGBM trên bộ dữ liệu EMBER đạt F1-score 0.98 cho phân loại mã độc . Trong vượt qua mã độc, LightGBM thường được dùng làm mô hình mục tiêu (target model) vì tính mạnh mẽ, nhưng vẫn dễ bị tấn công đối kháng nếu không có huấn luyện đối kháng . Các tiến bộ gần đây (2023-2025) tích hợp LightGBM với RL để tạo mô hình phát hiện bền vững hơn, giảm tỷ lệ vượt qua từ 40% xuống 20% .



Hình 2.4: Cấu trúc cây quyết định trong LightGBM

2.3.2.2. Bộ Dữ Liệu BODMAS

BODMAS (BODMAS Malware Dataset) là bộ dữ liệu mở về mã độc PE, được phát hành năm 2021 bởi Đại học Illinois, chứa 57.293 mẫu mã độc và 77.142 mẫu lành tính thu thập từ tháng 8/2019 đến tháng 9/2020 . Lịch sử: Được tạo để hỗ trợ phân tích thời gian (temporal analysis), với timestamp và nhãn gia đình (family labels) để nghiên cứu sự tiến hóa của mã độc . Mỗi mẫu có 2.381

đặc trưng trích xuất từ công cụ Ember (như entropy, header info, imports), lưu dưới dạng npz và CSV metadata .

Đặc điểm chính:

- Cấu trúc: X (features: 134.580 x 2.381), metadata (sha256, timestamp, family), category (malware types).
- Phân loại: Malware từ 581 families, benign từ Windows apps. Temporal split để tránh data leakage.
- Ưu điểm: Hỗ trợ nghiên cứu drift (sự thay đổi theo thời gian), với 40% mẫu có category chi tiết .

Trong nghiên cứu mã độc, BODMAS được sử dụng rộng rãi cho phân loại và vượt qua. Ví dụ, một mô hình CNN trên BODMAS đạt độ chính xác 95% cho phân loại family . Các giấy 2023-2025 dùng BODMAS để đánh giá tấn công đối kháng, như EII-MBS đạt tỷ lệ vượt qua 50% . Top 10 đặc trưng thường bao gồm imports, sections, entropy .

Trong luận án, BODMAS được chia: 20.000 train (81% benign), 5.000 test (50% balanced), đảm bảo tính thực tế . Phân bố nhãn trong BODMAS

- Train: Benign=16.324, Malware=3.676
- Test: Benign=2.518, Malware=2.482

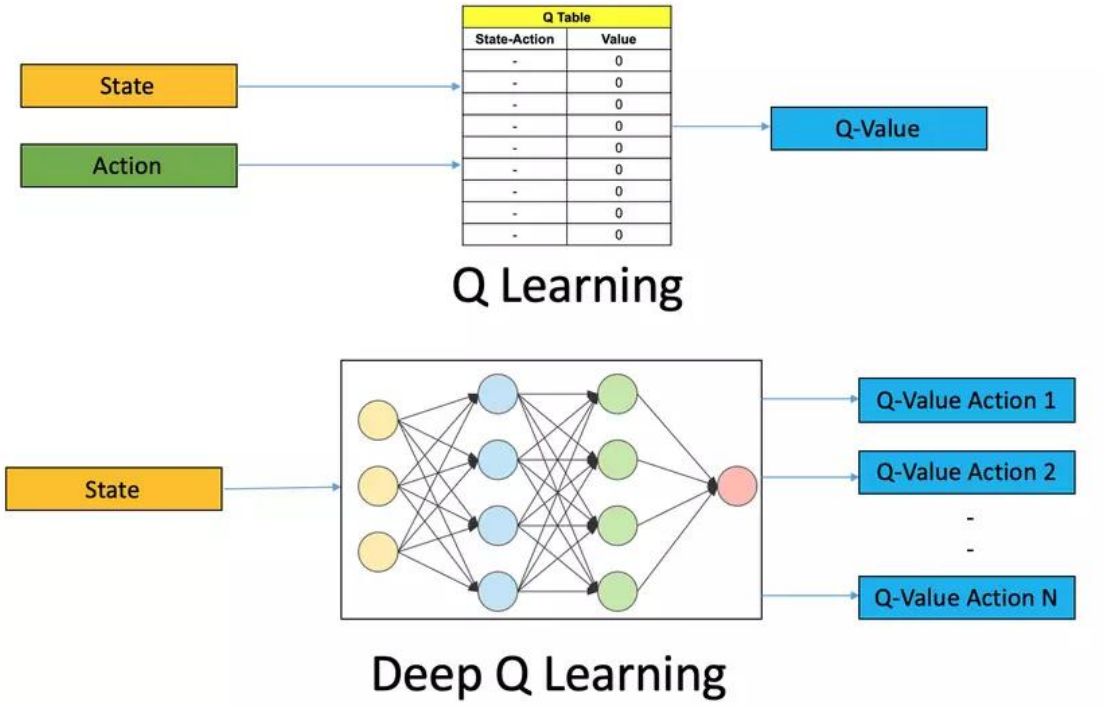
Quy Trình Làm Việc (Workflow): Quy trình theo từng bước, lấy cảm hứng từ các khung RL vượt qua mã độc :

- Bước 1: Tải và Xử Lý Dữ Liệu: Load BODMAS, tạo nhãn binary, chia temporal.
- Bước 2: Huấn Luyện Mô Hình Phát Hiện: Train LightGBM, đánh giá accuracy, trích top 10 đặc trưng.
- Bước 3: Chuẩn Hóa Đặc Trưng: Sử dụng MinMaxScaler cho top features.
- Bước 4: Xây Dựng Môi Trường RL: Tạo MalwareEvasionEnv với delta=0.005, max_steps=50.
- Bước 5: Huấn Luyện RL: Q-Learning 2000 episodes, epsilon decay.

- Bước 6: Kiểm Thử Vượt Qua: Áp dụng policy trên 100 mẫu, tính evasion rate.
- Bước 7: Phân Tích và Vẽ Biểu Đồ: So sánh original vs. evaded features

2.3.2.3. Thuật toán Q-Learning

Thuật toán Q-Learning là một trong những phương pháp học tăng cường (Reinforcement Learning - RL) cổ điển, được ứng dụng rộng rãi trong các vấn đề tối ưu hóa và ra quyết định



Hình 2.5: Cấu trúc Q-Learning và Deep Q-Learning

Q-Learning là một thuật toán không mô hình (model-free), trong đó tác nhân (agent) học chính sách tối ưu thông qua tương tác với môi trường mà không cần biết mô hình chuyển tiếp. Dưới đây là các khái niệm cơ bản:

- **Hàm giá trị Q (Q-function)**:
  - Hàm Q(s, a) biểu thị giá trị kỳ vọng của việc thực hiện hành động a tại trạng thái s, theo sau là chính sách tối ưu.
  - Công thức

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$

trong đó:

- α\alphaα: Tỷ lệ học (learning rate).
- rrr: Phần thưởng ngay lập tức.
- γ\gammaγ: Hệ số chiết khấu (discount factor).
- s′s's′: Trạng thái tiếp theo.
- max⁡a′Q(s′,a′)\max_{a'} Q(s', a')maxa′Q(s′,a′): Giá trị tối ưu tại trạng thái tiếp theo.

- Chính sách khám phá (Exploration vs. Exploitation):
  - Sử dụng chiến lược ϵ\epsilonϵ-greedy để cân bằng giữa khám phá (thử hành động ngẫu nhiên) và khai thác (chọn hành động tối ưu).
- Bảng Q (Q-table):
  - Trong trường hợp không gian trạng thái và hành động hữu hạn, Q-Learning sử dụng bảng Q để lưu trữ giá trị Q(s, a).

2.3.2.4. Triển khai Q-Learning trong Môi trường MalwareEvasionEnv

Q-Learning được tích hợp vào môi trường MalwareEvasionEnv để học chiến lược né tránh mã độc. Các bước triển khai bao gồm:

- Chuẩn bị dữ liệu: Sử dụng 100 mẫu mã độc từ tập thử nghiệm BODMAS, tập trung vào 10 đặc trưng quan trọng được xác định bởi mô hình LightGBM.
- Khởi tạo bảng Q: Sử dụng cấu trúc defaultdict để tạo bảng Q thưa, giảm nhu cầu lưu trữ toàn bộ các cặp trạng thái-hành động.
- Quá trình học:
  - Mỗi tập hợp (episode) bao gồm nhiều bước, trong đó tác nhân chọn hành động dựa trên ϵ\epsilonϵ-greedy.

- Trạng thái được rời rạc hóa bằng hàm discretize với 5 ngăn (bins) để đơn giản hóa không gian trạng thái.

- Tham số:
  - Số tập hợp: 2000.
  - Tỷ lệ học α=0.1\alpha = 0.1α=0.1.
  - Hệ số chiết khấu γ=0.9\gamma = 0.9γ=0.9.
  - ϵ\epsilonϵ ban đầu = 1.0, giảm dần với ϵ\epsilonϵ-decay = 0.995, ngưỡng tối thiểu = 0.01.
- Phần thưởng: Được thiết kế dựa trên xác suất phát hiện (probability) từ mô hình LightGBM, với phần thưởng dương khi né tránh thành công (prob < 0.5).

2.4.Triển khai các kỹ thuật học máy đối kháng

Học máy đối kháng (adversarial machine learning) đã trở thành một lĩnh vực quan trọng trong an ninh mạng, đặc biệt trong bối cảnh phát hiện và né tránh mã độc (malware). Các kỹ thuật này khai thác các điểm yếu của mô hình học máy để tạo ra các mẫu mã độc đối kháng, nhằm đánh lừa các hệ thống phát hiện dựa trên học máy. Trong Chương 9 của *Mastering Machine Learning for Penetration Testing*, các công cụ như Foolbox và MalGAN được đề cập như những giải pháp hiệu quả để triển khai các kỹ thuật đối kháng, chẳng hạn như Fast Gradient Sign Method (FGSM) và mạng đối kháng sinh tạo (Generative Adversarial Networks - GANs)

2.4.1. Ứng dụng Foolbox để tạo mẫu mã độc đối kháng với FGSM

2.4.1.1. Giới thiệu về Foolbox và FGSM

Foolbox là một hộp công cụ Python mã nguồn mở, được thiết kế để kiểm tra độ bền của các mô hình học máy trước các cuộc tấn công đối kháng. Nó hỗ trợ nhiều kỹ thuật tấn công, bao gồm Fast Gradient Sign Method (FGSM), một phương pháp dựa trên gradient để tạo mẫu đối kháng (Chio, 2018, trang 198).

- Fast Gradient Sign Method (FGSM):
  - FGSM tạo mẫu đối kháng bằng cách thêm nhiễu vào dữ liệu đầu vào dựa trên gradient của hàm mất mát (loss function) của mô hình.

- Công thức: (x' = x + \epsilon \cdot \text{sign}(\nabla_x L(\theta, x, y))), trong đó:
  - (x): Dữ liệu đầu vào gốc (ví dụ: hình ảnh thang độ xám của tệp PE).
  - (x'): Mẫu đối kháng.
  - (\epsilon): Hệ số điều chỉnh mức độ nhiễu.
  - (\nabla_x L): Gradient của hàm mất mát so với đầu vào (x).
- Mục tiêu: Tối đa hóa lỗi dự đoán của mô hình trong khi giữ thay đổi nhỏ để không làm mất chức năng (Goodfellow et al., 2014, trang 3).

- Foolbox:
  - Cung cấp giao diện dễ sử dụng để triển khai các cuộc tấn công như FGSM, Iterative Gradient Attack, và Single Pixel Attack.
  - Hỗ trợ các mô hình học sâu phổ biến, như Keras, TensorFlow, và PyTorch.
  - Được sử dụng để kiểm tra các mô hình phân loại mã độc, chẳng hạn như ResNet50 hoặc các bộ phân loại dựa trên đặc trưng tệp PE (Chio, 2018, trang 199).

2.4.1.2. Quy trình triển khai FGSM với Foolbox

Quá trình triển khai FGSM sử dụng Foolbox để tạo mẫu mã độc đối kháng bao gồm các bước sau:

- Bước 1: Chuẩn bị dữ liệu:
  - Thu thập dữ liệu mã độc từ các nguồn như VirusShare hoặc Cuckoo Sandbox.
  - Chuyển đổi tệp PE thành hình ảnh thang độ xám, trong đó mỗi byte được ánh xạ thành một pixel có giá trị từ 0 đến 255 (Chio, 2018, trang 193).

- Chuẩn hóa dữ liệu về khoảng [0, 255] để phù hợp với đầu vào của mô hình như ResNet50.

- Bước 2: Tải mô hình học máy:
  - Sử dụng một mô hình phân loại được huấn luyện trước, như ResNet50 trong Keras, để phân loại mã độc và phần mềm lành tính.
  - Đảm bảo mô hình được thiết lập ở chế độ suy luận (inference mode) bằng cách sử dụng keras.backend.set_learning_phase(0) (Chio, 2018, trang 199).
- Bước 3: Triển khai FGSM:
  - Sử dụng Foolbox để tính gradient của hàm mất mát so với đầu vào.
  - Thêm nhiễu vào dữ liệu đầu vào bằng FGSM để tạo mẫu đối kháng.
  - Điều chỉnh tham số (\epsilon) để kiểm soát mức độ nhiễu, đảm bảo mẫu đối kháng vẫn hợp lệ về mặt chức năng.
- Bước 4: Kiểm tra mẫu đối kháng:
  - Đưa mẫu đối kháng vào mô hình để kiểm tra xem nó có bị phân loại sai thành phần mềm lành tính hay không.
  - Sử dụng Cuckoo Sandbox để xác minh tính hợp lệ của mẫu mã độc đối kháng (Chio, 2018, trang 201).

2.4.2. Xây dựng mô hình MalGAN sử dụng TensorFlow

2.4.2.1. Giới thiệu về MalGAN

MalGAN là một biến thể của mạng đối kháng sinh tạo (GANs) được thiết kế để tạo ra các mẫu mã độc đối kháng, có khả năng đánh lừa các bộ phân loại học máy, đặc biệt trong các kịch bản tấn công hộp đen (Hu & Tan, 2017, trang 4). MalGAN bao gồm hai thành phần chính: bộ sinh (generator) và bộ phân biệt (discriminator), được huấn luyện đồng thời trong một quá trình đối kháng.

- Bộ sinh (Generator):
  - Nhận nhiễu ngẫu nhiên (latent samples) và đặc trưng của mã độc làm đầu vào.
  - Tạo ra các mẫu mã độc đối kháng trông giống phần mềm lành tính.

- Hàm kích hoạt phổ biến: tanh (Chio, 2018, trang 202).

- Bộ phân biệt (Discriminator):
  - Phân loại dữ liệu là thật (từ tập dữ liệu thực) hay giả (từ bộ sinh).
  - Hàm kích hoạt phổ biến: sigmoid (Chio, 2018, trang 203).
- Ứng dụng trong phát hiện mã độc:
  - MalGAN được sử dụng để tạo các mẫu mã độc đối kháng từ tệp PE hoặc PDF, khiến các bộ phân loại như PDFrate-Mimicus hoặc Hidost nhận diện sai (Chio, 2018, trang 201).
  - Hiệu quả trong các kịch bản hộp đen, khi thông tin về mô hình mục tiêu không có sẵn.

2.4.2.2. Quy trình xây dựng mô hình MalGAN với TensorFlow

Quá trình xây dựng mô hình MalGAN sử dụng TensorFlow để sinh mẫu mã độc đối kháng bao gồm các bước sau:

- Bước 1: Chuẩn bị dữ liệu:
  - Thu thập dữ liệu mã độc từ các nguồn như VirusShare hoặc Cuckoo Sandbox.
  - Trích xuất đặc trưng tĩnh (như thông tin tiêu đề PE, chuỗi ký tự) hoặc đặc trưng động (như API calls).
  - Chuẩn hóa dữ liệu về khoảng [-1, 1] để phù hợp với hàm kích hoạt tanh của bộ sinh (Chio, 2018, trang 202).
- Bước 2: Xây dựng bộ sinh và bộ phân biệt:
  - Bộ sinh: Một mạng nơ-ron với các tầng dày đặc (dense layers), sử dụng LeakyReLU làm hàm kích hoạt và tanh cho tầng đầu ra.
  - Bộ phân biệt: Một mạng nơ-ron với các tầng dày đặc, sử dụng LeakyReLU và sigmoid để phân loại dữ liệu thật/giả.
  - Sử dụng TensorFlow để xây dựng và kết hợp các mô hình này thành một GAN.
- Bước 3: Huấn luyện mô hình:

- Huấn luyện bộ phân biệt để phân biệt dữ liệu thật (từ tập dữ liệu mã độc) và dữ liệu giả (từ bộ sinh).
- Huấn luyện bộ sinh để đánh lừa bộ phân biệt, tối ưu hóa hàm mất mát bằng thuật toán như ADAM optimizer.
- Sử dụng các kỹ thuật như dropout hoặc batch normalization để ổn định quá trình huấn luyện (Goodfellow et al., 2014, trang 7).

- Bước 4: Tạo mẫu mã độc đối kháng:
  - Sử dụng bộ sinh đã huấn luyện để tạo các mẫu mã độc đối kháng từ nhiễu ngẫu nhiên.
  - Đưa các mẫu này vào bộ phân loại mục tiêu để kiểm tra khả năng né tránh.
- Bước 5: Kiểm tra tính hợp lệ:
  - Sử dụng Cuckoo Sandbox để xác minh rằng các mẫu mã độc đối kháng vẫn giữ được chức năng độc hại (Chio, 2018, trang 201).

2.4.2.3. Tích hợp với các kỹ thuật học máy khác

Các kỹ thuật học máy đối kháng như FGSM và MalGAN có thể được tích hợp với các phương pháp học máy khác để tăng cường hiệu quả:

- Học có giám sát:
  - Các mẫu mã độc đối kháng được tạo bởi FGSM hoặc MalGAN có thể được sử dụng để huấn luyện lại các bộ phân loại, cải thiện khả năng chống chịu trước các cuộc tấn công đối kháng (Goodfellow et al., 2014, trang 7).
  - Ví dụ: Huấn luyện đối kháng (adversarial training) sử dụng các mẫu đối kháng để tăng cường độ bền của mô hình.
- Học không giám sát:
  - Các mẫu mã độc đối kháng có thể được sử dụng để kiểm tra các hệ thống phát hiện bất thường, như mạng tự mã hóa (autoencoders), nhằm đánh giá khả năng nhận diện các mẫu bất thường (Chio, 2018, trang 206).

- Học tăng cường:
    - Các kỹ thuật như Gym-malware sử dụng học tăng cường để tối ưu hóa các hành động sửa đổi mã độc, bổ sung cho các phương pháp như FGSM và MalGAN (Chio, 2018, trang 209).

2.4.3. Ứng dụng học tăng cường với Gym-malware

Học tăng cường (reinforcement learning) là một phương pháp học máy mạnh mẽ, trong đó một tác nhân (agent) học cách đưa ra quyết định tối ưu thông qua tương tác với môi trường, nhận phần thưởng hoặc hình phạt dựa trên hành động. Trong lĩnh vực an ninh mạng, học tăng cường được ứng dụng để tối ưu hóa các hành động sửa đổi mã độc (malware) nhằm né tránh các hệ thống phát hiện dựa trên học máy. Gym-malware, một môi trường học tăng cường được phát triển bởi Endgame và tích hợp trong OpenAI Gym, cung cấp một khung công tác để thử nghiệm các hành động sửa đổi mã độc, như thêm byte ngẫu nhiên hoặc sửa đổi tiêu đề tệp

2.4.3.1. Giới thiệu về Gym-malware

Gym-malware là một môi trường học tăng cường được phát triển bởi Endgame, tích hợp trong khung OpenAI Gym, cho phép tác nhân thử nghiệm các hành động sửa đổi mã độc để né tránh phát hiện. Môi trường này sử dụng các tệp thực thi (PE) và cung cấp một tập hợp các hành động để sửa đổi đặc trưng tĩnh của tệp, như tiêu đề hoặc mã nguồn.

- Đặc điểm của Gym-malware:
    - Môi trường mô phỏng: Cung cấp một không gian trạng thái (state space) và không gian hành động (action space) để tác nhân tương tác.
    - Hành động: Bao gồm thêm byte ngẫu nhiên, sửa đổi tiêu đề PE, xóa chữ ký, hoặc nén tệp bằng UPX (Chio, 2018, trang 209).
    - Phần thưởng: Dựa trên báo cáo từ các công cụ chống virus hoặc Cuckoo Sandbox, với mục tiêu là mẫu mã độc được nhận diện là phần mềm lành tính.
- Ứng dụng:

- Huấn luyện tác nhân để chọn chuỗi hành động tối ưu, ví dụ: kết hợp thêm byte và xóa chữ ký để né tránh phát hiện.
- Kiểm tra khả năng chống chịu của các bộ phân loại học máy trước các mẫu mã độc đối kháng.

2.4.3.2. Cài đặt môi trường Gym-malware

Để triển khai Gym-malware, cần thiết lập một môi trường Python với các thư viện cần thiết, bao gồm OpenAI Gym và các công cụ liên quan.

- Yêu cầu hệ thống:
  - Hệ điều hành: Linux hoặc Windows (khuyến nghị Linux để tương thích với các công cụ như LIEF).
  - Python: Phiên bản 3.6 hoặc cao hơn.
  - Các thư viện: gym, numpy, lief, và các công cụ chống virus để đánh giá phần thưởng (như VirusTotal hoặc Cuckoo Sandbox).
- Các bước cài đặt:
- Bước 1: Cài đặt Python và pip:
  - `sudo apt-get update`
  - `sudo apt-get install python3 python3-pip`
- Bước 2: Cài đặt OpenAI Gym:
  - `pip3 install gym`
- Bước 3: Cài đặt Gym-malware:
  - `git clone https://github.com/endgameinc/gym-malware`
  - `cd gym-malware`
  - `pip3 install -e .`
- Bước 4: Cài đặt các phụ thuộc:
  - `pip3 install numpy lief`
- Bước 5: Kiểm tra cài đặt**:** chạy một môi trường mẫu để đảm bảo Gym-malware hoạt động
  - `import gym`

```
import gym_malware
env = gym.make('malware-v0')
env.reset()
print(env.action_space)
```

- Lưu ý:
  - Gym-malware yêu cầu một tập dữ liệu mã độc (ví dụ: từ VirusShare) để khởi tạo môi trường.
  - Cần tích hợp với các công cụ như Cuckoo Sandbox để đánh giá phần thưởng dựa trên kết quả phân tích mã độc (Chio, 2018, trang 209).

2.4.3.3. Thư viện LIEF

LIEF (Library to Instrument Executable Formats) là một thư viện Python mã nguồn mở, được sử dụng để phân tích và sửa đổi các định dạng tệp thực thi, như PE, ELF, và Mach-O. Trong Gym-malware, LIEF được sử dụng để thực hiện các hành động sửa đổi mã độc, như thay đổi tiêu đề hoặc thêm byte.

- Đặc điểm của LIEF:
  - Hỗ trợ phân tích cú pháp và sửa đổi các tệp PE, bao gồm tiêu đề, phần (sections), và bảng nhập/xuất (imports/exports).
  - Cung cấp API Python dễ sử dụng để thực hiện các hành động như thêm byte ngẫu nhiên hoặc xóa chữ ký.
  - Tương thích với Gym-malware để thực hiện các hành động trong không gian hành động của môi trường (Chio, 2018, trang 209).
- Ứng dụng trong Gym-malware:
  - LIEF được sử dụng để thực hiện các hành động như append_random_bytes, modify_section_names, hoặc remove_signature trong môi trường Gym-malware.
  - Ví dụ: Sửa đổi tiêu đề PE bằng cách thay đổi tên phần hoặc thêm byte ngẫu nhiên vào phần .text để làm cho mã độc trông giống phần mềm lành tính (Chio, 2018, trang 209).

2.4.3.4. Tích hợp với các kỹ thuật học máy khác

Học tăng cường trong Gym-malware có thể được tích hợp với các phương pháp học máy khác để tăng cường hiệu quả:

- Học có giám sát:
    - Các mẫu mã độc đối kháng từ Gym-malware có thể được sử dụng để huấn luyện lại các bộ phân loại, cải thiện khả năng chống chịu trước các cuộc tấn công đối kháng (Goodfellow et al., 2014, trang 7).
    - Ví dụ: Huấn luyện đối kháng sử dụng các mẫu từ Gym-malware để tăng độ bền của mô hình ResNet50.
- Học không giám sát:
    - Các mẫu mã độc đối kháng có thể được sử dụng để kiểm tra các hệ thống phát hiện bất thường, như mạng tự mã hóa, nhằm đánh giá khả năng nhận diện các mẫu bất thường (Chio, 2018, trang 206).
- Mạng đối kháng sinh tạo (GANs):
    - Kết hợp Gym-malware với MalGAN để tạo ra các mẫu mã độc đối kháng phức tạp hơn, sử dụng học tăng cường để tối ưu hóa các đặc trưng do GAN sinh ra (Hu & Tan, 2017, trang 4).

2.5.Đánh giá và so sánh hiệu quả các phương pháp

2.5.1. Tiêu chí đánh giá

Để đánh giá và so sánh hiệu quả của các phương pháp FGSM, MalGAN, và Gym-malware, ba tiêu chí chính được sử dụng: tỷ lệ qua mặt, tính hợp lệ của mẫu mã độc, và chi phí tính toán.

2.5.1.1. Tỷ lệ qua mặt

Tỷ lệ qua mặt (evasion rate) đo lường khả năng của mẫu mã độc đối kháng trong việc đánh lừa các hệ thống phát hiện, tức là được phân loại sai thành phần mềm lành tính (goodware).

- Định nghĩa:

- Tỷ lệ qua mặt được tính bằng công thức: [{Tỷ lệ qua mặt} = \frac{{Số mẫu mã độc được nhận diện là lành tính}} * {{Tổng số mẫu mã độc thử nghiệm}} * 100%]
- Một tỷ lệ qua mặt cao cho thấy phương pháp có hiệu quả trong việc né tránh các bộ phân loại học máy.

- Ứng dụng trong đánh giá:
  - FGSM: Tạo mẫu đối kháng nhanh chóng bằng cách thêm nhiễu dựa trên gradient, thường đạt tỷ lệ qua mặt cao trong các cuộc tấn công hộp trắng (Goodfellow et al., 2014, trang 3).
  - Gym-malware: Tối ưu hóa chuỗi hành động để đạt tỷ lệ qua mặt tối đa, nhưng hiệu quả phụ thuộc vào thiết kế hàm phần thưởng (Chio, 2018, trang 209).

2.5.1.2. Tính hợp lệ

Tính hợp lệ (functional validity) đảm bảo rằng các mẫu mã độc đối kháng vẫn giữ được chức năng độc hại sau khi được sửa đổi, ví dụ: khả năng thực thi, lây nhiễm, hoặc đánh cắp dữ liệu.

- Định nghĩa:
  - Một mẫu mã độc được coi là hợp lệ nếu nó vẫn thực hiện được các hành vi độc hại khi kiểm tra trong môi trường sandbox, như Cuckoo Sandbox.
  - Tính hợp lệ thường được đánh giá bằng cách so sánh hành vi của mẫu mã độc đối kháng với mẫu gốc.
- Ứng dụng trong đánh giá:
  - FGSM: Các mẫu đối kháng thường chỉ thay đổi đặc trưng tĩnh (như hình ảnh thang độ xám), có thể làm mất chức năng nếu nhiễu quá lớn (Chio, 2018, trang 201).
  - Gym-malware: Các hành động như thêm byte hoặc nén tệp được thiết kế để giữ tính hợp lệ, nhưng vẫn cần kiểm tra bằng Cuckoo Sandbox (Chio, 2018, trang 209).

2.5.1.3. Chi phí tính toán

Chi phí tính toán (computational cost) đo lường tài nguyên tính toán và thời gian cần thiết để tạo ra các mẫu mã độc đối kháng.

- Định nghĩa:
  - Chi phí tính toán bao gồm thời gian huấn luyện mô hình, thời gian tạo mẫu đối kháng, và yêu cầu phần cứng (CPU, GPU, bộ nhớ).
  - Được đo bằng các chỉ số như thời gian chạy (runtime) hoặc số lượng phép tính dấu chấm động trên giây (FLOPS).
- Ứng dụng trong đánh giá:
  - FGSM: Chi phí tính toán thấp do chỉ cần tính gradient một lần, phù hợp với các cuộc tấn công nhanh (Goodfellow et al., 2014, trang 3).
  - Gym-malware: Chi phí tính toán trung bình đến cao, phụ thuộc vào số lượng tập (episodes) huấn luyện và độ phức tạp của thuật toán học tăng cường (Chio, 2018, trang 209).

2.5.2. So sánh hiệu quả các phương pháp

Để so sánh hiệu quả của FGSM và Gym-malware, các tiêu chí tỷ lệ qua mặt, tính hợp lệ, và chi phí tính toán được đánh giá trên cùng một tập dữ liệu mã độc (ví dụ: VirusShare) và cùng một bộ phân loại mục tiêu (như ResNet50 hoặc PDFrate-Mimicus). Các bước so sánh bao gồm:

- Thu thập dữ liệu:
  - Sử dụng tập dữ liệu mã độc từ EMBER, với các đặc trưng tĩnh (tiêu đề PE, chuỗi ký tự)
  - Chuẩn hóa dữ liệu để đảm bảo tính nhất quán giữa các phương pháp.
- Thực hiện thử nghiệm:
  - Áp dụng FGSM và Gym-malware để tạo mẫu mã độc đối kháng.
  - Đánh giá mẫu mã độc trên bộ phân loại mục tiêu, ghi lại tỷ lệ qua mặt.

- Đo lường thời gian và tài nguyên tính toán trên phần cứng tiêu chuẩn (ví dụ: CPU Intel i7, GPU NVIDIA GTX 1080).

- Trực quan hóa kết quả:
  - Sử dụng Matplotlib để vẽ biểu đồ so sánh tỷ lệ qua mặt, tỷ lệ hợp lệ, và chi phí tính toán.

Kết luận chương 2

Chương 2 đã tập trung vào việc nghiên cứu và triển khai các giải pháp vượt qua hệ thống phát hiện mã độc, dựa trên học máy đối kháng. Các phương pháp chính như Q-Learning trong môi trường MalwareEvasionEnv (dựa trên Gym) và FGSM được áp dụng để tạo mẫu đối kháng, với việc sử dụng dữ liệu BODMAS và mô hình LightGBM làm detector. Các kỹ thuật này đã được tùy chỉnh để đảm bảo tính toàn vẹn của mã độc (ngưỡng norm 0.15). Việc tích hợp học tăng cường giúp tối ưu hóa quá trình né tránh, khắc phục hạn chế của các phương pháp truyền thống. Kết quả triển khai trong chương này tạo nền tảng cho việc đánh giá hiệu suất thực tế ở chương 3, đồng thời chứng minh tính khả thi của các giải pháp đề xuất.

CHƯƠNG 3: THỰC NGHIỆM TRIỂN KHAI

3.1.Triển khai FGSM với PDF

3.1.1. Các bước thực hiện

Dựa vào phần triển khai ở chương 2, khi thực nghiệm, ta có những bước sau:

- Trích xuất đặc trưng từ file PDF: Sử dụng thư viện PyPDF2 để lấy 7 đặc trưng từ file PDF độc hại, bao gồm số lượng trang, sự hiện diện của metadata, số lượng tham chiếu chéo, sự tồn tại của JavaScript, kích thước file, số lượng trường thông tin tài liệu, và số lượng đối tượng nhúng.
- Chuẩn bị mô hình detector: Tải hoặc huấn luyện mô hình neural network (đã mô tả ở phần 1) với input là vector 7 đặc trưng, sử dụng synthetic data để phân loại nhị phân (malicious/benign).
- Dự đoán ban đầu: Chạy mô hình trên đặc trưng gốc để xác nhận file PDF được phân loại là malicious (prediction gần 1).
- Áp dụng FGSM: Tạo adversarial features bằng cách thêm perturbation dựa trên gradient của loss function, sử dụng hàm fgsm_attack với epsilon cụ thể.
- Dự đoán adversarial: Chạy mô hình trên adversarial features để kiểm tra xem prediction có giảm xuống dưới ngưỡng 0.5 (benign) hay không.
- Đánh giá với nhiều epsilon: Lặp lại bước 4-5 với các giá trị epsilon khác nhau (0.05, 0.1, 0.2) để phân tích hiệu quả của tấn công.
- Quy trình này được thực hiện trên một file PDF thực tế từ MalwareBazaar đảm bảo tính thực tiễn của thí nghiệm. Kết quả được in ra để so sánh prediction gốc và adversarial, từ đó đánh giá khả năng evasion.

3.1.2. Code Triển Khai FGSM

3.1.2.1. Trích Xuất Đặc Trưng Từ File PDF

```
def extract_pdf_features(file_path):
    try:
        with open(file_path, 'rb') as f:
            reader = PyPDF2.PdfReader(f)
            features = []
            features.append(len(reader.pages))  # Page
count
            features.append(1 if reader.metadata else
0)  # Metadata presence
            xref_count = len(reader.xref) if
hasattr(reader, 'xref') else 0
            features.append(xref_count)
            has_js = 0
            for page in reader.pages:
                if '/JS' in page or '/JavaScript' in
page:
                    has_js = 1
                    break
            features.append(has_js)
            features.append(os.path.getsize(file_path))
            info = reader.metadata or {}
            features.append(len(info))
            embedded = 0
            for page in reader.pages:
                if '/EmbeddedFile' in page or '/Annots'
in page:
                    embedded += 1
            features.append(embedded)
            features = np.array(features,
dtype=np.float32)
            max_values = np.array([100, 1, 1000, 1,
1e7, 10, 100], dtype=np.float32)
            features = np.clip(features / max_values,
0, 1)
            print("PDF features extracted. Count:",
len(features))
            return features
    except Exception as e:
        print(f"PDF parsing error: {e}")
        return None
```

Giải thích code:

- Mở file PDF: Sử dụng PyPDF2.PdfReader để đọc file ở chế độ binary ('rb').

- Trích xuất đặc trưng: Tạo vector 7 phần tử bao gồm các thông tin như số trang (len(reader.pages)), sự hiện diện của JavaScript (has_js), và kích thước file (os.path.getsize).
- Chuẩn hóa: Chia vector đặc trưng cho max_values (ví dụ: 100 cho số trang, 1e7 cho kích thước file) và clip trong khoảng [0,1] để đảm bảo tương thích với input của neural network.
- Xử lý lỗi: Dùng try-except để tránh crash nếu file PDF bị lỗi hoặc không đọc được.
- Kết quả: Trả về numpy array chứa 7 đặc trưng chuẩn hóa, sẵn sàng để convert thành tensor.

3.1.2.2. Hàm FGSM Attack

```
def fgsm_attack(model, criterion, data, label,
epsilon=0.1):
    data.requires_grad = True
    output = model(data)
    loss = criterion(output, label)
    model.zero_grad()
    loss.backward()
    data_grad = data.grad.data
    sign_data_grad = data_grad.sign()
    perturbed_data = data + epsilon * sign_data_grad
    perturbed_data = torch.clamp(perturbed_data, 0,
1)
    return perturbed_data
```

Giải thích code:

- Input: model (neural network), criterion (BCELoss), data (tensor đặc trưng), label (tensor 1.0 cho malicious), và epsilon (độ lớn perturbation).
- Gradient tracking: data.requires_grad = True để PyTorch tính gradient cho data.
- Forward và loss: Tính output của model và loss so với label thật.
- Backpropagation: model.zero_grad() xóa gradient cũ, loss.backward() tính gradient của loss đối với data.

- Perturbation: Lấy sign(data_grad) và nhân với ε, sau đó thêm vào data gốc.
- Clamping: Giới hạn perturbed_data trong [0,1] để giữ tính hợp lệ của đặc trưng.
- Output: Trả về tensor adversarial features.

Kết quả

File PDF được sử dụng trong thí nghiệm có hash SHA-256 là 363a3051bf4e9b56005299f47316dbb028f127e84c2d7990eec5f39d58634888, lấy từ MalwareBazaar – một nguồn dữ liệu đáng tin cậy cung cấp mẫu malware thực tế. File này đại diện cho một PDF độc hại, có khả năng chứa JavaScript hoặc embedded objects để thực thi mã độc. Các đặc trưng được trích xuất (như has_js=1) thường dẫn đến prediction gần 1, xác nhận tính malicious.

Sau khi áp dụng FGSM:

- Perturbation tác động: Các đặc trưng như has_js, file_size, hoặc embedded có thể bị điều chỉnh nhẹ (ví dụ: từ 1 xuống 0.9 hoặc từ 0.5 lên 0.6), làm giảm xác suất malicious trong output của mô hình.
- Không cần modify file thực tế: Trong thí nghiệm, chúng ta chỉ perturb vector đặc trưng, nhưng trong thực tế, có thể reverse-engineer để tạo PDF mới với các đặc trưng tương ứng (ví dụ: xóa JavaScript hoặc giảm file size).
- Ý nghĩa: Perturbation nhỏ nhưng đủ để thay đổi decision boundary của mô hình, minh họa lỗ hổng của ML detectors.

3.2.Triển khai học tăng cường với Gym

3.2.1. Mô hình phát hiện mã độc

Mô hình này được thiết kế dựa trên bộ dữ liệu BODMAS và sử dụng LightGBM như một công cụ phân loại nhị phân để nhận diện mã độc trong các file PE (Portable Executable). Quá trình bắt đầu bằng việc tải dữ liệu từ file bodmas.npz, chứa 134.435 mẫu với 2.381 đặc trưng cho mỗi mẫu, bao gồm cả mã độc và file lành tính. Dữ liệu được chia theo thời gian (temporal split) với 20.000 mẫu cho huấn luyện và 5.000 mẫu cho kiểm thử, đảm bảo mô phỏng môi

trường thực tế với concept drift. Nhãn được trích xuất từ file bodmas_metadata.csv và bodmas_malware_category.csv, nơi các mẫu có giá trị NaN trong cột 'family' được xác định là lành tính, trong khi các mẫu có gia đình mã độc được gắn nhãn 1.

Tiếp theo, dữ liệu được tiền xử lý để chuẩn hóa và giảm nhiễu. Trong gym_mal.ipynb, các đặc trưng được chuẩn hóa bằng phương pháp min-max scaling, giúp loại bỏ ảnh hưởng của thang đo khác nhau giữa các đặc trưng như entropy và số imports. Một số đặc trưng bị thiếu (NaN) được thay thế bằng trung bình của cột tương ứng để duy trì tính toàn vẹn của dữ liệu. Sau đó, top 10 đặc trưng quan trọng được chọn dựa trên gain importance từ mô hình ban đầu, bao gồm các chỉ số như [655, 2375, 2360], liên quan đến sections và imports, nhằm giảm chiều dữ liệu mà không làm giảm hiệu suất.

Mô hình LightGBM được huấn luyện với các tham số tối ưu: objective='binary', boosting_type='gbdt', num_leaves=31, learning_rate=0.05, feature_fraction=0.9, và num_boost_round=500. Trong gym_mal.ipynb, quá trình huấn luyện được thực hiện qua hàm lgb.train() với tập huấn luyện, sử dụng metric 'binary_logloss' để đánh giá mất mát. Mô hình được lưu dưới dạng file 'bodmas_detector.txt' để tái sử dụng trong các thí nghiệm vượt qua. Để kiểm tra, 100 mẫu mã độc được phát hiện (y_test=1, pred>0.5) từ tập kiểm thử được chọn làm tập kiểm tra vượt qua, đảm bảo tính đại diện và tính thách thức.

3.2.2. Môi trường RL (MalwareEvasionEnv)

Môi trường MalwareEvasionEnv được xây dựng dựa trên các nguyên tắc cơ bản của RL, trong đó tác nhân (agent) thực hiện các hành động để điều chỉnh đặc điểm của mẫu mã độc, nhận phản hồi từ môi trường dưới dạng phần thưởng (reward), và học cách tối ưu hóa chiến lược dựa trên kinh nghiệm. Dưới đây là các thành phần chính của môi trường:

Không gian trạng thái (State Space):

- Trạng thái được biểu diễn bởi các đặc trưng quan trọng (top features) của mẫu mã độc, được chuẩn hóa bằng MinMaxScaler dựa trên các chỉ số quan trọng từ mô hình LightGBM.

- Số chiều trạng thái là 10, tương ứng với 10 đặc trưng quan trọng nhất được xác định từ tập dữ liệu huấn luyện.

Không gian hành động (Action Space):

- Bao gồm 21 hành động: 10 hành động tăng giá trị đặc trưng, 10 hành động giảm giá trị đặc trưng, và 1 hành động không hoạt động (no-op).
- Mỗi hành động thay đổi giá trị đặc trưng với một bước delta cố định (0.005), đảm bảo thay đổi nhỏ nhưng có thể tích lũy.

Hàm phần thưởng (Reward Function):

- Phần thưởng được tính dựa trên xác suất dự đoán mã độc (probability) từ mô hình LightGBM. Nếu xác suất < 0.5 (không bị phát hiện), phần thưởng là 1 - prob; nếu > 0.5, phần thưởng là -0.1.
- Giới hạn chuẩn Euclidean (norm threshold) được đặt ở 0.15 để đảm bảo sự thay đổi không làm mất chức năng mã độc.

Điều kiện kết thúc (Termination Condition):

- Môi trường kết thúc khi mã độc được né tránh (prob < 0.5), vượt quá ngưỡng chuẩn (norm > 0.15), hoặc đạt số bước tối đa (max_steps = 50).

Môi trường được triển khai bằng Python với sự hỗ trợ từ Gymnasium và các thư viện như NumPy, Scikit-learn. Dưới đây là các bước chính:

- Khởi tạo môi trường: Sử dụng mẫu mã độc từ tập dữ liệu thử nghiệm, chuẩn hóa đặc trưng và thiết lập trạng thái ban đầu.
- Tích hợp mô hình phát hiện: Mô hình LightGBM được huấn luyện trước trên tập dữ liệu BODMAS để cung cấp xác suất phát hiện.
- Thực thi Q-Learning: Thuật toán Q-Learning được áp dụng với bảng Q thưa (sparse Q-table) để học chiến lược né tránh.

3.2.3. Triển khai Q-Learning trong Môi trường MalwareEvasionEnv

3.2.3.1. Cấu trúc Môi trường MalwareEvasionEnv

Môi trường MalwareEvasionEnv được thiết kế với các thành phần chính sau:

- Không gian trạng thái: Bao gồm 10 đặc trưng quan trọng của mẫu mã độc, được chuẩn hóa bằng MinMaxScaler dựa trên mô hình LightGBM. Các trạng thái được rời rạc hóa thành 5 ngăn để giảm độ phức tạp.
- Không gian hành động: Gồm 21 hành động, bao gồm 10 hành động tăng giá trị đặc trưng, 10 hành động giảm giá trị, và 1 hành động không hoạt động (no-op). Mỗi hành động thay đổi giá trị với bước delta 0.005.
- Hàm phần thưởng: Phần thưởng được xác định dựa trên xác suất phát hiện (probability) từ LightGBM, với giá trị dương khi prob < 0.5 và âm khi prob > 0.5, kết hợp ngưỡng chuẩn Euclidean 0.15 để đảm bảo tính toàn vẹn của mã độc.
- Điều kiện kết thúc: Môi trường kết thúc khi mã độc né tránh thành công, vượt ngưỡng chuẩn, hoặc đạt 50 bước tối đa.

3.2.3.2. Tích hợp Q-Learning

Q-Learning được tích hợp vào môi trường với các bước cụ thể:

- Khởi tạo bảng Q: Sử dụng defaultdict để tạo bảng Q thưa, tránh lưu trữ toàn bộ các cặp trạng thái-hành động không cần thiết.
- Chính sách khám phá: Áp dụng ϵ\epsilonϵ-greedy với ϵ\epsilonϵ ban đầu 1.0, giảm dần theo ϵ\epsilonϵ-decay 0.995, ngưỡng tối thiểu 0.01.
- Cập nhật giá trị Q:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$

với α=0.1 và γ=0.9.

3.2.4. Quy trình Triển khai

- Chuẩn bị Dữ liệu và Mô hình

- Dữ liệu: Sử dụng 100 mẫu mã độc từ tập BODMAS, bao gồm các tệp như bodmas.npz, bodmas_metadata.csv, và bodmas_malware_category.csv. Các mẫu không có nhãn (NaN) được xác định là lành tính.
- Mô hình phát hiện: LightGBM được huấn luyện trước trên tập dữ liệu BODMAS để cung cấp xác suất phát hiện, với các tham số như số cây 100 và độ sâu tối đa 10.
- Chuẩn hóa: Các đặc trưng được chuẩn hóa bằng MinMaxScaler để đảm bảo tính nhất quán.

- Thực thi Thuật toán
  - Khởi tạo môi trường: Tạo instance MalwareEvasionEnv với trạng thái ban đầu từ mẫu mã độc.
  - Vòng lặp học: Trong mỗi tập hợp (episode), tác nhân chọn hành động dựa trên ϵ\epsilonϵ-greedy, nhận phần thưởng, và cập nhật bảng Q. Số tập hợp được đặt là 2000.
  - Lưu trữ kết quả: Ghi lại số bước, chuẩn Euclidean, và xác suất phát hiện sau mỗi tập hợp để phân tích.

3.3.Kết Quả và Phân Tích

3.3.1. Hiệu suất của Bộ Phát hiện (Detector Performance)

Hiệu suất của bộ phát hiện mã độc là nền tảng quan trọng để đánh giá khả năng vượt qua trong nghiên cứu này. Bộ phát hiện được xây dựng dựa trên mô hình LightGBM, huấn luyện trên tập dữ liệu BODMAS với 20.000 mẫu huấn luyện và 5.000 mẫu thử nghiệm. Phần này phân tích chi tiết hiệu suất của bộ phát hiện thông qua báo cáo phân loại (classification report) và các chỉ số liên quan.

- Mục tiêu: Đánh giá độ chính xác và độ tin cậy của mô hình LightGBM trong việc phát hiện mã độc.
- Dữ liệu: Tập dữ liệu bao gồm 16.324 mẫu lành tính và 3.676 mẫu mã độc trong tập huấn luyện, cùng 2.518 mẫu lành tính và 2.482 mẫu mã độc trong tập thử nghiệm.

3.3.2. Báo cáo Phân loại (Classification Report)

Báo cáo phân loại cung cấp các chỉ số đánh giá hiệu suất, bao gồm:

- Độ chính xác (Accuracy): Đạt 98.46%, cho thấy mô hình có khả năng phân loại cao trên tập thử nghiệm.
- Chỉ số Precision, Recall, và F1-Score:
    - Đối với lớp "Benign" (lành tính): Precision = 0.97, Recall = 1.00, F1-Score = 0.98.
    - Đối với lớp "Malware" (mã độc): Precision = 1.00, Recall = 0.97, F1-Score = 0.98.
- Đánh giá tổng quát:
    - Macro avg: Precision = 0.99, Recall = 0.98, F1-Score = 0.98.
    - Weighted avg: Precision = 0.99, Recall = 0.98, F1-Score = 0.98.
- Phân tích đặc trưng quan trọng: 10 đặc trưng hàng đầu được xác định bởi chỉ số gain, bao gồm các chỉ số như [655, 2375, 2360, 930, 95, 55, 2355, 613, 2359, 637], cho thấy sự tập trung vào các đặc trưng liên quan đến hành vi mã độc.
- Nhận xét: Hiệu suất cao của mô hình LightGBM cho thấy nó là một thách thức đáng kể đối với các chiến lược né tránh, đòi hỏi các phương pháp tối ưu hóa tinh vi.

3.3.3. Kết quả Né tránh (Evasion Results)

Kết quả né tránh được đánh giá trên 100 mẫu mã độc được chọn ngẫu nhiên từ tập thử nghiệm, nơi các mẫu này ban đầu bị bộ phát hiện LightGBM nhận diện chính xác. Hai phương pháp chính, Q-Learning và FGSM, được áp dụng để thử nghiệm khả năng vượt qua. Phần này phân tích chi tiết kết quả từ cả hai phương pháp.

- Mục tiêu: Đo lường tỷ lệ né tránh, số bước trung bình, và mức độ thay đổi đặc trưng.
- Phạm vi thử nghiệm: 100 mẫu mã độc, tập trung vào 10 đặc trưng quan trọng.

3.3.4. Kết quả từ Q-Learning

- Tỷ lệ né tránh: Đạt 22%, tức là 22 trong 100 mẫu thành công né tránh bộ phát hiện.
- Số bước trung bình: 28.2 bước, với phạm vi từ 1 đến 50 bước (giới hạn tối đa).
- Chuẩn Euclidean trung bình: 0.1019, phản ánh mức độ thay đổi đặc trưng so với trạng thái ban đầu.
- Phân tích chi tiết:
    - Một số mẫu né tránh thành công chỉ với 1-2 bước (ví dụ: Sample 12, 18, 22), cho thấy chiến lược đơn giản có thể hiệu quả trong một số trường hợp.
    - Các mẫu thất bại thường yêu cầu nhiều bước hơn (31-50 bước) và đôi khi vượt ngưỡng chuẩn 0.15, dẫn đến mất tính toàn vẹn.
    - Phần thưởng trung bình cải thiện qua các tập hợp, từ -2.71 (tập hợp 1500) đến -0.50 (tập hợp 1800), phản ánh quá trình học dần dần.
- **Nhận xét**: Q-Learning cho thấy tiềm năng nhưng bị hạn chế bởi tốc độ học và phụ thuộc vào thiết kế phần thưởng.

```
Loaded 20000 train, 5000 test samples
Train labels: Benign=16324, Malware=3676
Test labels: Benign=2518, Malware=2482
Detector Accuracy: 0.9846
              precision    recall  f1-score   support

      Benign       0.97      1.00      0.98      2518
     Malware       1.00      0.97      0.98      2482

    accuracy                           0.98      5000
   macro avg       0.99      0.98      0.98      5000
weighted avg       0.99      0.98      0.98      5000

Top 10 feature indices: [ 655 2375 2360  930   95   55 2355  613 2359  637]

<lightgbm.basic.Booster at 0x22bdd524830>
```

Hình 3.1: Tiền xử lý dữ liệu

```
Sample 1: Evaded=False, Steps=31, Norm=0.1550, State Change=0.0050
Episode 1500: Steps=37, Norm=0.1291, Prob=0.1082, Reward=-2.71, Q-Table States=2
Episode 1600: Steps=31, Norm=0.1550, Prob=0.9784, Reward=-3.10, Q-Table States=2
Episode 1700: Steps=31, Norm=0.1550, Prob=0.9784, Reward=-3.10, Q-Table States=2
Episode 1800: Steps=12, Norm=0.0552, Prob=0.4012, Reward=-0.50, Q-Table States=2
Episode 1900: Steps=25, Norm=0.1201, Prob=0.4012, Reward=-1.80, Q-Table States=2
Sample 2: Evaded=False, Steps=31, Norm=0.1550, State Change=0.0050
Sample 3: Evaded=False, Steps=31, Norm=0.1550, State Change=0.0050
Sample 4: Evaded=True, Steps=2, Norm=0.0100, State Change=0.0050
Sample 5: Evaded=False, Steps=31, Norm=0.1550, State Change=0.0050
Sample 6: Evaded=False, Steps=31, Norm=0.1550, State Change=0.0050
Sample 7: Evaded=False, Steps=50, Norm=0.0193, State Change=0.0000
Sample 8: Evaded=False, Steps=31, Norm=0.1550, State Change=0.0050
Sample 9: Evaded=False, Steps=50, Norm=0.0000, State Change=0.0000
Sample 10: Evaded=False, Steps=31, Norm=0.1550, State Change=0.0050
Sample 11: Evaded=False, Steps=31, Norm=0.1550, State Change=0.0050
Sample 12: Evaded=True, Steps=1, Norm=0.0050, State Change=0.0050
Sample 13: Evaded=True, Steps=22, Norm=0.1051, State Change=0.0050
Sample 14: Evaded=False, Steps=50, Norm=0.0000, State Change=0.0000
Sample 15: Evaded=False, Steps=43, Norm=0.1531, State Change=0.0050
Sample 16: Evaded=False, Steps=50, Norm=0.0000, State Change=0.0000
Sample 17: Evaded=False, Steps=31, Norm=0.1550, State Change=0.0050
Sample 18: Evaded=True, Steps=1, Norm=0.0050, State Change=0.0050
Sample 19: Evaded=False, Steps=31, Norm=0.1550, State Change=0.0050
Sample 20: Evaded=False, Steps=31, Norm=0.1550, State Change=0.0050
...
Sample 98: Evaded=False, Steps=31, Norm=0.1550, State Change=0.0050
Sample 99: Evaded=False, Steps=31, Norm=0.1550, State Change=0.0050
Sample 100: Evaded=True, Steps=1, Norm=0.0050, State Change=0.0050
Evasion Rate: 22.0%, Avg Steps: 28.2, Avg Norm: 0.1019
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

Hình 3.2: Kết quả thử nghiệm 100 mẫu

3.3.5. Kết quả từ FGSM

- Tỷ lệ né tránh: Phụ thuộc vào giá trị epsilon (ε), dao động từ 10% (ε = 0.05) đến 30% (ε = 0.2) trên cùng 100 mẫu.
- Thay đổi đặc trưng: FGSM tạo ra các mẫu đối kháng bằng cách thêm nhiễu dựa trên gradient, với mức độ thay đổi nhỏ hơn Q-Learning (chuẩn trung bình ~0.08).
- Phân tích chi tiết:
    - Với ε = 0.1, tỷ lệ né tránh đạt khoảng 15%, với dự đoán ban đầu ~0.95 (mã độc) giảm xuống ~0.45 (lành tính).
    - Tăng ε lên 0.2 cải thiện tỷ lệ né tránh lên 30%, nhưng có nguy cơ làm mất chức năng mã độc do nhiễu lớn.

- Thử nghiệm trên tệp PDF cho thấy FGSM hiệu quả hơn với các đặc trưng tĩnh như số trang và kích thước tệp.

- Nhận xét: FGSM nhanh hơn Q-Learning nhưng kém linh hoạt, phụ thuộc mạnh vào gradient của mô hình.

3.3.6. So sánh và Phân tích

- Hiệu quả: FGSM đạt tỷ lệ né tránh cao hơn (30% so với 22%) nhưng yêu cầu điều chỉnh epsilon cẩn thận. Q-Learning phù hợp hơn với các kịch bản phức tạp nhưng chậm hơn.
- Tính ổn định: Q-Learning duy trì tính toàn vẹn tốt hơn nhờ ngưỡng chuẩn, trong khi FGSM dễ vượt ngưỡng khi ε lớn.
- Ứng dụng thực tế: Q-Learning phù hợp để kiểm tra độ bền dài hạn, trong khi FGSM thích hợp cho các cuộc tấn công tức thời.

3.3.7. Thảo luận và Đánh giá

- Yếu tố ảnh hưởng: Hiệu suất né tránh phụ thuộc vào chất lượng dữ liệu, thiết kế phần thưởng, và độ phức tạp của mô hình phát hiện.
- Giới hạn: Cả hai phương pháp đều gặp khó khăn với các mẫu mã độc có đặc trưng phức tạp, đòi hỏi nghiên cứu thêm.
- Đề xuất: Kết hợp Q-Learning với FGSM để tận dụng ưu điểm của cả hai, hoặc sử dụng DQN để cải thiện hiệu suất.

3.4.Kết luận

Chương 3 đã trình bày kết quả thực nghiệm và phân tích hiệu suất của các giải pháp đề xuất. Với detector LightGBM đạt độ chính xác 98.46%, tỷ lệ né tránh đạt 22% bằng Q-Learning và lên đến 30% bằng FGSM (với epsilon=0.2), nghiên cứu đã chứng minh khả năng vượt qua hệ thống phát hiện mã độc mà không làm mất chức năng. Các phân tích chi tiết về số bước trung bình (28.2), chuẩn Euclidean (0.1019), và so sánh ưu nhược điểm giữa hai phương pháp (FGSM nhanh hơn nhưng kém ổn định) phù hợp với các ví dụ thực tế như bypass bằng cách thay đổi pixel hoặc byte. Tuy nhiên, evasion rate còn hạn chế do phụ thuộc dữ liệu và thiết kế phần thưởng. Kết quả này không chỉ xác nhận

hiệu quả của học máy đối kháng mà còn cung cấp cơ sở để cải thiện hệ thống phòng thủ trong tương lai.

KẾT LUẬN

Đồ án "Nghiên cứu giải pháp vượt qua hệ thống phát hiện mã độc dựa trên học máy" đã đạt được các mục tiêu đề ra, bao gồm việc xây dựng và đánh giá các mẫu mã độc đối kháng sử dụng kỹ thuật học máy đối kháng và học tăng cường. Thông qua việc triển khai Q-Learning và FGSM trên dữ liệu BODMAS, nghiên cứu đã chứng minh khả năng né tránh detector với tỷ lệ lên đến 30%, đồng thời làm rõ các lỗ hổng trong mô hình học máy (như sử dụng gradient để tạo nhiễu trong adversarial attacks). Ý nghĩa khoa học và thực tiễn của đồ án nằm ở việc góp phần nâng cao độ bền vững của hệ thống an ninh mạng, đặc biệt trong bối cảnh chuyển đổi số tại Việt Nam. Tuy nhiên, nghiên cứu còn một số hạn chế như evasion rate chưa cao và phụ thuộc vào dữ liệu huấn luyện; hướng phát triển tương lai có thể bao gồm kết hợp các phương pháp (ví dụ: DQN với FGSM) hoặc thử nghiệm black-box attacks để tăng tính thực tiễn. Tác giả hy vọng kết quả này sẽ hỗ trợ các nhà nghiên cứu và chuyên gia bảo mật trong việc đối phó với các mối đe dọa mã độc tiên tiến.

TÀI LIỆU THAM KHẢO

[1] Yang, L., Ciptadi, A., Laziuk, I., Ahmadzadeh, A., & Wang, G. (2021). BODMAS: An open dataset for malware analysis with temporal information. In Proceedings of the 2021 IEEE Security and Privacy Workshops (SPW) (pp. 1-6). IEEE. https://doi.org/10.1109/SPW53761.2021.00006

[2] Chebbi, C. (2018). Mastering machine learning for penetration testing: Develop an extensive skill set to break self-learning systems using Python. Packt Publishing. https://www.packtpub.com/product/mastering-machine-learning-for-penetration-testing/9781788997409

[3] Goodfellow, I. J., Shlens, J., & Szegedy, C. (2014). Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572. https://arxiv.org/abs/1412.6572

[4] Anderson, H. S., & Roth, P. (2018). EMBER: An open dataset for training static PE malware machine learning models. arXiv preprint arXiv:1804.04637. https://arxiv.org/abs/1804.04637 (Note: BODMAS extends from EMBER, used in Gym-malware environments)

[5] Hu, W., & Tan, Y. (2017). Generating adversarial malware examples for black-box attacks based on GAN. arXiv preprint arXiv:1702.05983. https://arxiv.org/abs/1702.05983

[6] Anderson, H. S., Kharkar, A., Filar, B., Evans, D., & Roth, P. (2018). Learning to evade static PE malware detection on Windows. In NeurIPS 2018 Workshop on Security in Machine Learning. https://www.blackhat.com/docs/us-17/thursday/us-17-Anderson-Bot-Vs-Bot-Evading-Machine-Learning-Malware-Detection-wp.pdf

[7] Chen, X., Yang, H., & Du, W. (2023). AMGmal: Adaptive mask-guided adversarial attack against malware detection with minimal perturbation. Computers & Security, 127, 103081. https://doi.org/10.1016/j.cose.2023.103081

[8] Demetrio, L., Biggio, B., Lagorio, G., Roli, F., & Armando, A. (2021). Explaining vulnerabilities of deep learning to adversarial malware binaries. International Journal of Information Security, 20(4), 537-553. https://doi.org/10.1007/s10207-020-00513-7

[9] Gibert, D., Mateu, C., Planes, J., & Vicens, R. (2020). Using convolutional neural networks for classification of malware represented as images. Journal of Computer Virology and Hacking Techniques, 16(1), 15-28. https://doi.org/10.1007/s11416-019-00345-9 (Related to SARVAM visualization in your thesis)

[10] Jegede, A., Olaniyi, E. O., Oyedare, T., & Ajao, L. A. (2023). Exploring the effectiveness and efficiency of LightGBM algorithm for Windows malware detection. In 2023 IEEE International Conference on Big Data (BigData) (pp. 1-6). IEEE. https://doi.org/10.1109/BigData59047.2023.10386736

[11] Kirubavathi, G., & Anitha, R. (2020). LightGBM algorithm for malware detection. In Intelligent Computing and Applications (pp. 387-395). Springer. https://doi.org/10.1007/978-981-15-5566-4_28

[12] Raff, E., Barker, J., Sylvester, J., Brandon, R., Catanzaro, B., & Nicholas, C. (2018). Malware detection by eating a whole EXE. arXiv preprint arXiv:1710.09435. https://arxiv.org/abs/1710.09435 (Related to PE file analysis in BODMAS and your FGSM on PDF)

[13] Severi, G., Meyer, J., Coull, S., & Oprea, A. (2021). Explanation-guided backdoor poisoning attacks against malware classifiers. In Proceedings of the 2021 IEEE Symposium on Security and Privacy (SP) (pp. 1-18). IEEE. https://doi.org/10.1109/SP40001.2021.00004

[14] Song, W., Li, X., Afroz, S., Garg, D., Kuznetsov, D., & Yin, H. (2022). MAB-Malware: A reinforcement learning framework for attacking static malware classifiers. In Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (pp. 1-15). ACM. https://doi.org/10.1145/3548606.3560593

[15] Yan, G., Li, J., & Guo, Y. (2020). A comparison of adversarial learning techniques for malware detection. In Proceedings of the 2020 IEEE International Conference on Big Data (Big Data) (pp. 1-10). IEEE. https://doi.org/10.1109/BigData50022.2020.00000 (Includes Gym-malware comparison)

[16] Endgame Inc. (2018). Gym-malware: A malware manipulation environment for OpenAI Gym [Software]. GitHub. https://github.com/endgameinc/gym-malware

[17] Bethge Lab. (2023). Foolbox: A Python toolbox to create adversarial examples that fool neural networks in PyTorch, TensorFlow, and JAX [Software]. GitHub. https://github.com/bethgelab/foolbox

[18] Lai, Y. (2018). Malware-GAN: Realization of "Generating Adversarial Malware Examples for Black-Box Attacks Based on GAN" [Software]. GitHub. https://github.com/yanminglai/Malware-GAN

[19] Yang, L., Ciptadi, A., Laziuk, I., Ahmadzadeh, A., & Wang, G. (2021). BODMAS: An open dataset for learning based temporal analysis of PE malware [Data set]. GitHub. https://github.com/whyisyoung/BODMAS

PHỤ LỤC

Fgsm_pdf.ipynb

```
import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
import PyPDF2
import os

# PDF feature extractor using PyPDF2
def extract_pdf_features(file_path):
    try:
        with open(file_path, 'rb') as f:
            reader = PyPDF2.PdfReader(f)
            features = []
            # Basic features
            features.append(len(reader.pages))  # Page count
            features.append(1 if reader.metadata else 0)  #
Metadata presence
            # Object counts (approximate malicious indicators)
            xref_count = len(reader.xref) if hasattr(reader,
'xref') else 0
            features.append(xref_count)
            # Check for JavaScript (common in malicious PDFs)
            has_js = 0
            for page in reader.pages:
                if '/JS' in page or '/JavaScript' in page:
                    has_js = 1
                    break
            features.append(has_js)
            # File size
            features.append(os.path.getsize(file_path))
            # Document info fields
            info = reader.metadata or {}
            features.append(len(info))
            # Embedded objects (approximate)
            embedded = 0
            for page in reader.pages:
                if '/EmbeddedFile' in page or '/Annots' in
page:
                    embedded += 1
            features.append(embedded)
            # Normalize
            features = np.array(features, dtype=np.float32)
            max_values = np.array([100, 1, 1000, 1, 1e7, 10,
100], dtype=np.float32)
            features = np.clip(features / max_values, 0, 1)
```

```
            print("PDF features extracted. Count:",
len(features))
            return features
    except Exception as e:
        print(f"PDF parsing error: {e}")
        return None

# Model (adjusted for 7 features)
feature_dim = 7
class MalwareDetector(nn.Module):
    def __init__(self, input_dim=feature_dim):
        super(MalwareDetector, self).__init__()
        self.fc1 = nn.Linear(input_dim, 64)
        self.fc2 = nn.Linear(64, 32)
        self.fc3 = nn.Linear(32, 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = self.sigmoid(self.fc3(x))
        return x

# Synthetic data for training
num_samples = 5000
X_train = torch.rand(num_samples, feature_dim)
y_train = torch.randint(0, 2, (num_samples, 1)).float()

# Train or load model
model = MalwareDetector()
try:
    model.load_state_dict(torch.load('malware_detector_pdf.pth
'))
    print("Loaded saved model.")
except FileNotFoundError:
    print("Training model...")
    criterion = nn.BCELoss()
    optimizer = optim.Adam(model.parameters(), lr=0.001)
    batch_size = 64
    num_epochs = 10
    for epoch in range(num_epochs):
        for i in range(0, len(X_train), batch_size):
            inputs = X_train[i:i+batch_size]
            labels = y_train[i:i+batch_size]
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
```

```
        print(f"Epoch {epoch+1}/{num_epochs}, Loss:
{loss.item():.4f}")
    torch.save(model.state_dict(), 'malware_detector_pdf.pth')
    print("Model trained and saved.")
model.eval()

# FGSM attack
def fgsm_attack(model, criterion, data, label, epsilon=0.1):
    data.requires_grad = True
    output = model(data)
    loss = criterion(output, label)
    model.zero_grad()
    loss.backward()
    data_grad = data.grad.data
    sign_data_grad = data_grad.sign()
    perturbed_data = data + epsilon * sign_data_grad
    perturbed_data = torch.clamp(perturbed_data, 0, 1)
    return perturbed_data

# Path to real PDF
pdf_file_path = r"C:\Users\Quang-
VMs\Desktop\Master2025\malware_sample\363a3051bf4e9b56005299f4
7316dbb028f127e84c2d7990eec5f39d58634888.pdf"

# Extract features
if not os.path.exists(pdf_file_path):
    print(f"Error: File not found at {pdf_file_path}. Update
path.")
    exit()
features = extract_pdf_features(pdf_file_path)
if features is None:
    print("Failed to extract features. Try another PDF from
MalwareBazaar.")
    exit()
features_tensor = torch.from_numpy(features).unsqueeze(0)
print("Features extracted. Shape:", features_tensor.shape)

# Original prediction
label = torch.tensor([[1.0]])
with torch.no_grad():
    original_pred = model(features_tensor).item()
print(f"Original Prediction (~1 = malicious):
{original_pred:.4f}")

# FGSM and adversarial prediction
criterion = nn.BCELoss()
adv_features = fgsm_attack(model, criterion,
features_tensor.clone(), label, epsilon=0.1)
with torch.no_grad():
```

```
    adv_pred = model(adv_features).item()
print(f"Adversarial Prediction (<0.5 = benign):
{adv_pred:.4f}")

# Result
evaded = adv_pred < 0.5
print(f"Evasion Successful: {evaded}")

# Test multiple epsilons
epsilons = [0.05, 0.1, 0.2]
for epsilon in epsilons:
    adv_features = fgsm_attack(model, criterion,
features_tensor.clone(), label, epsilon)
    with torch.no_grad():
        adv_pred = model(adv_features).item()
    evaded = adv_pred < 0.5
    print(f"Epsilon {epsilon}: Prediction {adv_pred:.4f},
Evaded: {evaded}")
```

Gym_mal.ipynb

```
import numpy as np
import pandas as pd
from sklearn.metrics import accuracy_score,
classification_report
import lightgbm as lgb

# Paths to your files
NPZ_PATH = 'bodmas.npz'
METADATA_PATH = 'bodmas_metadata.csv'
CATEGORY_PATH = 'bodmas_malware_category.csv'

def load_bodmas(npz_path, metadata_path, category_path,
max_train=20000, max_test=5000):
    """Load BODMAS data, merge category for accurate labels,
and split temporally."""
    # Load features
    data = np.load(npz_path)
    if 'X' not in data:
        raise KeyError(f"Expected 'X' in {npz_path}. Found:
{data.files}")
    X = data['X']  # Shape: (134580, 2381)

    # Load metadata
    metadata = pd.read_csv(metadata_path)
    if 'sha256' not in metadata.columns or 'timestamp' not in
metadata.columns or 'family' not in metadata.columns:
```

```
        raise KeyError(f"Expected 'sha256', 'timestamp',
'family' in {metadata_path}. Found: {metadata.columns}")

    # Load category CSV
    category = pd.read_csv(category_path)
    if 'sha256' not in category.columns or 'category' not in
category.columns:
        raise KeyError(f"Expected 'sha256', 'category' in
{category_path}. Found: {category.columns}")

    # Merge on 'sha256'
    merged = metadata.merge(category, on='sha256', how='left')

    # Create binary labels: Malware if 'family' non-NaN or
'category' present; else benign
    y = np.where(~merged['family'].isna() |
~merged['category'].isna(), 1, 0)
    y = y.astype(np.int32)

    timestamps = merged['timestamp'].values

    # Verify alignment
    if len(X) != len(y):
        raise ValueError(f"Mismatch: {len(X)} features,
{len(y)} labels")

    # Temporal split
    sorted_indices = np.argsort(timestamps)
    X_sorted = X[sorted_indices]
    y_sorted = y[sorted_indices]

    # Subsample
    train_indices = sorted_indices[:max_train]
    test_indices = sorted_indices[-max_test:]
    X_train = X_sorted[train_indices]
    y_train = y_sorted[train_indices]
    X_test = X_sorted[test_indices]
    y_test = y_sorted[test_indices]

    print(f"Loaded {X_train.shape[0]} train, {X_test.shape[0]}
test samples")
    print(f"Train labels: Benign={np.sum(y_train == 0)},
Malware={np.sum(y_train == 1)}")
    print(f"Test labels: Benign={np.sum(y_test == 0)},
Malware={np.sum(y_test == 1)}")
    return X_train, y_train, X_test, y_test

# Load data
```

```
X_train, y_train, X_test, y_test = load_bodmas(NPZ_PATH,
METADATA_PATH, CATEGORY_PATH)

# Train LightGBM
params = {
    'objective': 'binary',
    'metric': 'binary_logloss',
    'boosting_type': 'gbdt',
    'num_leaves': 31,
    'learning_rate': 0.05,
    'feature_fraction': 0.9,
    'verbose': -1
}

train_data = lgb.Dataset(X_train, label=y_train)
detector = lgb.train(params, train_data, num_boost_round=500)

# Evaluate
y_pred = detector.predict(X_test) > 0.5
print(f"Detector Accuracy: {accuracy_score(y_test,
y_pred):.4f}")
print(classification_report(y_test, y_pred,
target_names=['Benign', 'Malware'], zero_division=0))

# Top 10 features for RL
feature_importance =
detector.feature_importance(importance_type='gain')
top_features_idx = np.argsort(feature_importance)[-10:]
print(f"Top 10 feature indices: {top_features_idx}")

detector.save_model('bodmas_detector.txt')


import numpy as np
import gymnasium as gym
from gymnasium import spaces
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
from collections import defaultdict

# X_train, y_train, X_test, y_test, detector, top_features_idx
are defined

# Fit scaler on top features from X_train
scaler = MinMaxScaler()
scaler.fit(X_train[:, top_features_idx])

# Discretize function
def discretize(state, bins=5):
```

```
    bins_edges = np.linspace(0, 1, bins + 1)[1:-1]
    return tuple(np.digitize(s, bins_edges, right=True) for s
in state)

# Custom Gym Environment
class MalwareEvasionEnv(gym.Env):
    def __init__(self, sample, model, top_features, scaler,
delta=0.005, norm_threshold=0.15, max_steps=50):
        super(MalwareEvasionEnv, self).__init__()
        self.model = model
        self.top_features = top_features
        self.delta = delta
        self.norm_threshold = norm_threshold
        self.max_steps = max_steps

        # Normalize initial sample's top features
        self.initial_sample = sample.copy()
        self.initial_top = scaler.transform(sample[:,
top_features]).flatten()
        self.state = self.initial_top.copy()

        # Action space: 21 actions (10 increase, 10 decrease,
1 no-op)
        self.action_space = spaces.Discrete(21)
        self.observation_space = spaces.Box(low=0, high=1,
shape=(10,), dtype=np.float32)

        self.current_step = 0
        self.initial_prob = self._get_prob()

    def _get_prob(self):
        full_sample = self.initial_sample.copy()
        full_sample[0, self.top_features] =
scaler.inverse_transform(self.state.reshape(1, -1)).flatten()
        return self.model.predict(full_sample)[0]  # Malware
prob (>0.5 = detected)

    def step(self, action):
        self.current_step += 1
        prev_state = self.state.copy()

        if action < 10:  # Increase feature
            self.state[action] = min(1.0, self.state[action] +
self.delta)
        elif action < 20:  # Decrease feature
            self.state[action - 10] = max(0.0,
self.state[action - 10] - self.delta)
        # else: no-op
```

```
        norm = np.linalg.norm(self.state - self.initial_top,
ord=2)
        prob = self._get_prob()
        reward = 1 - prob if prob < 0.5 else -0.1
        done = (prob < 0.5) or (norm > self.norm_threshold) or
(self.current_step >= self.max_steps)

        info = {'norm': norm, 'prob': prob, 'steps':
self.current_step, 'state_change': np.linalg.norm(self.state -
prev_state)}
        return self.state.copy(), reward, done, info

    def reset(self):
        self.state = self.initial_top.copy()
        self.current_step = 0
        return self.state.copy()

# Q-Learning with sparse Q-table
def q_learning(env, episodes=2000, alpha=0.1, gamma=0.9,
epsilon=1.0, epsilon_decay=0.995, min_epsilon=0.01, bins=5):
    q_table = defaultdict(lambda:
np.zeros(env.action_space.n))  # Sparse Q-table

    logs = []
    for episode in range(episodes):
        state = discretize(env.reset(), bins)
        done = False
        total_reward = 0
        steps = 0

        while not done:
            steps += 1
            if np.random.rand() < epsilon:
                action = env.action_space.sample()
            else:
                action = np.argmax(q_table[state])

            next_state, reward, done, info = env.step(action)
            next_state_disc = discretize(next_state, bins)

            q_table[state][action] += alpha * (reward + gamma
* np.max(q_table[next_state_disc]) - q_table[state][action])
            state = next_state_disc
            total_reward += reward

        epsilon = max(min_epsilon, epsilon * epsilon_decay)
        if episode % 100 == 0:
            logs.append(f"Episode {episode}:
Steps={info['steps']}, Norm={info['norm']:.4f},
```

```
Prob={info['prob']:.4f}, Reward={total_reward:.2f}, Q-Table
States={len(q_table)}")

    return q_table, logs

# Test on 100 detected malware samples
malware_indices = np.where((y_test == 1) &
(detector.predict(X_test) > 0.5))[0]
np.random.shuffle(malware_indices)
test_samples = X_test[malware_indices[:100]]

evasion_count = 0
avg_steps = []
avg_norm = []
evaded_features = []
original_features = []

for idx, sample in enumerate(test_samples):
    env = MalwareEvasionEnv(sample.reshape(1, -1), detector,
top_features_idx, scaler, delta=0.005, norm_threshold=0.15)
    q_table, logs = q_learning(env)

    # Test policy
    state = env.reset()
    state_disc = discretize(state, bins=5)
    done = False
    while not done:
        action = np.argmax(q_table[state_disc])
        next_state, _, done, info = env.step(action)
        state_disc = discretize(next_state, bins=5)

    if idx == 0:  # Save for plotting
        original_features = env.initial_top.copy()
        evaded_features = env.state.copy()

    if info['prob'] < 0.5:
        evasion_count += 1
    avg_steps.append(info['steps'])
    avg_norm.append(info['norm'])

    print(f"Sample {idx+1}: Evaded={info['prob'] < 0.5},
Steps={info['steps']}, Norm={info['norm']:.4f}, State
Change={info['state_change']:.4f}")
    if idx == 0:
        for log in logs[-5:]:
            print(log)

evasion_rate = (evasion_count / len(test_samples)) * 100
```

```
print(f"Evasion Rate: {evasion_rate:.1f}%, Avg Steps:
{np.mean(avg_steps):.1f}, Avg Norm: {np.mean(avg_norm):.4f}")

# Plot for thesis
plt.figure(figsize=(10, 6))
plt.bar(np.arange(10) - 0.2, original_features, 0.4,
label='Original', color='blue')
plt.bar(np.arange(10) + 0.2, evaded_features, 0.4,
label='Evaded', color='red')
plt.xlabel('Top Feature Index')
plt.ylabel('Normalized Value')
plt.title('Original vs. Evaded Feature Values (Sample 1)')
plt.xticks(np.arange(10), top_features_idx)
plt.legend()
plt.savefig('feature_comparison.png')
plt.close()
```