**HỌC VIỆN KỸ THUẬT MẬT MÃ**

KHOA CÔNG NGHỆ THÔNG TIN



**THỰC TẬP CƠ SỞ CHUYÊN NGÀNH**

XÂY DỰNG ỨNG DỤNG PHÁT HIỆN

PHẦM MỀM ĐỘC HẠI ANDROID

|  |  |  |
| --- | --- | --- |
| Giảng viên hướng dẫn: | **Ths. Thái Thị Thanh Vân** | |
| Sinh viên thực hiện: | *Trần Gia Lương* | *CT030433* |
| *Trương Quốc Quân* | *CT030440* |
| *Vũ Thị Thanh Vân* | *CT030358* |

**Hà Nội, 2021-2022**

**LỜI CẢM ƠN**

Trong thời gian làm đề tài thực tập cơ sở chuyên ngành, chúng em đã nhận được nhiều sự giúp đỡ, đóng góp nhiệt tình của thầy cô, gia đình và bạn bè. Đầu tiên, chúng em xin được gửi lời cảm ơn chân thành và sâu sắc tới cô Ths. Thái Thị Thanh Vân, người đã tận tình hướng dẫn, chỉ bảo chúng em trong suốt quá trình thực hiện đề tài thực tập cơ sở chuyên ngành, giúp chúng em có thêm kiến thức chuyên môn cũng như học hỏi được tinh thần trách nhiệm, thái độ làm việc nghiêm túc, hiệu quả từ cô.

Chúng em cũng xin chân thành cảm ơn các thầy cô giáo trong Học viện Kỹ thuật Mật Mã nói chung cùng các thầy cô trong khoa Công nghệ thông tin nói riêng đã tận tình giảng dạy, truyền đạt cho chúng em những kiến thức và kinh nghiệm quý báu trong suốt thời gian học tập, giúp chúng em có được cơ sở lý thuyết vững vàng cũng như sự quan tâm và tạo mọi điều kiện thuận lợi cho chúng em trong quá trình thực hiện đề tài thực tập cơ sở chuyên ngành. Cuối cùng, chúng em xin chân thành cảm ơn gia đình và bạn bè, đã luôn tạo điều kiện, quan tâm, giúp đỡ, động viên chúng em trong suốt quá trình học tập và hoàn thành đề tài thực tập cơ sở chuyên ngành.

Với điều kiện thời gian cũng như kinh nghiệm còn hạn chế nên không thể tránh được những thiếu sót. Chúng em rất mong nhận được sự chỉ bảo, đóng góp ý kiến của các thầy cô để chúng em có điều kiện bổ sung, nâng cao ý thức của mình, phục vụ tốt hơn công tác thực tế sau này.

*Chúng em xin chân thành cảm ơn!*

**LỜI NÓI ĐẦU**

Vào cuối những năm 80, phần mềm chống mã độc (hay phần mềm chống virus) đã ra đời với sự ra đời của mã độc. Công nghệ được sử dụng trong giai đoạn mã độc này tương đối đơn giản, giúp máy tính dễ dàng phát hiện các mã độc đã biết trước. Hầu hết các phần mềm chống virus ban đầu đều áp dụng phương pháp so sánh tính năng và chỉ cần sử dụng chuỗi tính năng để hoàn tất phát hiện. Với sự phát triển của công nghệ mã độc, mã độc bắt đầu bị biến dạng trong quá trình lan truyền để tránh bị giết. Vào thời điểm này, số lượng biến thể của cùng một mã độc đã tăng lên đáng kể và đã thay đổi rất nhiều từ hình thức cấu trúc. Rất khó để phần mềm chống virus trích xuất một đoạn mã làm chữ ký cho mã độc.

Trong tình hình đó, chữ ký phổ rộng được ra đời, chữ ký phổ rộng chia chữ ký thành nhiều đoạn và sử dụng mã hóa cặp byte để phân chia các đoạn cần so sánh và các đoạn không cần so sánh. Tuy nhiên, cho dù đó là quét chữ ký hay chữ ký phổ rộng, cần phải trích xuất các tính năng sau khi lấy các mẫu mã độc, và sau đó mới có thể phát hiện. Điều này làm cho việc phát hiện mã độc có độ trễ nhất định và luôn đi theo mã độc. Để đối phó với các loại virus đã biết và cả chưa biết, quét heuristic đã ra đời. Quét heuristic sử dụng kinh nghiệm và kiến ​​thức đã có để tiến hành phát hiện mã nhị phân không xác định. Kỹ thuật này ghi lại mã độc có các tính năng độc hại mà các tệp nhị phân thông thường không có. Các hành vi, chẳng hạn như các tệp đọc và ghi khác thường, tự động kết thúc, v.v. Trọng tâm và khó khăn của quét heuristic là làm thế nào để trích xuất các đặc trưng hành vi độc hại của mã độc. Quét chữ ký, tìm kiếm các chữ ký phổ rộng và quét heuristic, cả 3 phương pháp tra diệt đều không có tệp nhị phân đang chạy thực tế, có thể được phân loại là phương pháp phát hiện tĩnh mã độc.

Với sự phát triển dần dần của công nghệ chống phần mềm độc hại và công nghệ điện toán đám mây đã được sử dụng ngày càng nhiều bởi các nhà cung cấp bảo mật, nhưng phương pháp phát hiện tĩnh mã độc vẫn là hiệu quả nhất đỡ tốn tài nguyên và thời gian, được sử dụng rộng dãi nhất để phát hiện mã độc.

Sau một thời gian thảo luận, nhóm sinh viên bọn em đã quyết định đăng kí chọn đề tài: “Xây dựng ứng dụng phát hiện phần mềm độc hại Android”

**MỤC LỤC**

CHƯƠNG 1. CƠ SỞ LÝ THUYẾT 1

1.1. Tổng quan đề tài 1

1.1.1. Đặt vấn đề 1

1.1.2. Mục tiêu đề tài 2

1.2. Tổng quan về phần mềm độc hại 3

1.2.1. Mã độc 3

1.2.2. Các dạng mã độc 4

1.2.3. Quan hệ giữa hai loại mã độc và các cơ chế hoạt động 7

1.3. Các cách phát hiện phần mềm độc hại trên Android 8

1.3.1. Phân tích tĩnh 8

1.3.2. Phân tích động 11

1.3.3. Phân tích lai 12

CHƯƠNG 2. MẠNG THÔNG TIN KHÔNG ĐỒNG NHẤT 14

2.1. Giới thiệu về mạng thông tin không đồng nhất 14

2.2. Mạng thông tin không đồng nhất 15

2.3. Lược đồ 16

2.4. Phân loại mạng thông tin không đồng nhất 17

2.5. Siêu đường đi 18

CHƯƠNG 3. XÂY DỰNG HỆ THỐNG 23

3.1. Tổng quan xây dựng mô hình học máy 23

3.1.1. Xây dựng tập API Dataset 23

3.1.2. Trích xuất dữ liệu và xây training mô hình học máy 24

3.2. Chi tiết các bước xây dựng mô hình 26

3.2.1. Thu thập dữ liệu và trích xuất thông tin 26

3.2.2. Xây dựng ma trận độ đo tương đồng 30

3.2.3. Trích xuất vector đặc trưng 33

3.3. Huấn luyện học máy và đánh giá mô hình 34

3.3.1. Phương pháp đánh giá 34

3.3.2. Thực nghiệm và đánh giá 34

3.4. Xây dựng hệ thống 44

3.4.1. Tổng quan vệ hệ thống 44

3.4.2. Các yêu cầu chức năng và phi chức năng của hệ thống 44

3.4.3. Phân tích hệ thống 45

3.4.4. Triển khai xây dựng hệ thống 46

CHƯƠNG 4. KẾT LUẬN 51

4.1. Các vấn đề đã làm được 51

4.2. Các vấn đề còn tồn đọng của hệ thống 51

THAM KHẢO 52

**DANH MỤC BẢNG**

Bảng 1.1: Phân tích động 12

Bảng 2.1: Mô tả ý nghĩa của các siêu đường đi 19

Bảng 3.1: Mười sáu siêu đường đi được sử dụng trong mô hình 32

Bảng 3.2: Train – validate Kneighbors(n\_neighbors = 5) 36

Bảng 3.3: Train – validate Kneighbors(n\_neighbors = 4) 37

Bảng 3.4: Train – validate Kneighbors(n\_neighbors = 6) 38

Bảng 3.5: Train – validate RandomForest(n\_estimator = 100, max\_deep = 10) 39

Bảng 3.6: Train – validate RandomForest(n\_enstimator = 100, max\_deep = 12) 40

Bảng 3.7: Train – validate RandomForest(n\_estimator = 100, max\_deep = 14) 41

Bảng 3.8: Test – Kneighbors(n\_neighbors = 4) 42

Bảng 3.9: Test – Kneighbors(n\_neighbors = 5) 42

Bảng 3.10: Test – Kneighbors(n\_neighbors = 6) 42

Bảng 3.11: Test – RandomForest(n\_esimator = 100, max\_deep = 10) 43

Bảng 3.12: Test – RandomForest(n\_estimator = 100, max\_deep = 12) 43

Bảng 3.13: Test – RandomForest(n\_estimator = 100, max\_deep = 14) 44

**DANH MỤC HÌNH ẢNH**

[Hình 1.1: Mã độc tống tiền 6](#_Toc95589304)

[Hình 1.2: Botnet 7](#_Toc95589305)

[Hình 1.3 Kiến trúc của công cụ ANDRUBIS 12](#_Toc95589306)

[Hình 2.1: Mô hình mạng thông tin thư mục DBLP[3] 15](#_Toc95589307)

[Hình 2.2: Mô hình mạng Twitter 15](#_Toc95589308)

[Hình 2.3: Ví dụ một mạng thông tin không đồng nhất [3] 16](#_Toc95589309)

[Hình 2.4: Minh hoạ các loại mạng thông tin không đồng nhất 17](#_Toc95589310)

[Hình 2.5: Minh họa các siêu đường đi trong mạng DBLP 18](#_Toc95589311)

[Hình 3.1: Code minh hoạ quá trình xây dựng API dataset 23](#_Toc95589312)

[Hình 3.2: Xây dựng các ma trận cơ sở 24](#_Toc95589313)

[Hình 3.3: Xây dựng ma trận độ đo tương đồng 25](#_Toc95589314)

[Hình 3.4: Mô tả cách tổng hợp ra các đặc trưng từ ma trận độ đo tương đồng 25](#_Toc95589315)

[Hình 3.5: Bộ dữ liệu CIC Dataset 26](#_Toc95589316)

[Hình 3.6: Cấu trúc một file apk 26](#_Toc95589317)

[Hình 3.7: Quá trình dịch ngược file apk 27](#_Toc95589318)

[Hình 3.8: Các file smali thu được sau khi dịch ngược 27](#_Toc95589319)

[Hình 3.9: Các lời gọi API trong file smali 28](#_Toc95589320)

[Hình 3.10: Danh sách các lời gọi API 29](#_Toc95589321)

[Hình 3.11: Mạng thông tin không đồng nhất 30](#_Toc95589322)

[Hình 3.12: Tổng theo từng họ ứng dụng 33](#_Toc95589323)

[Hình 3.13: Tổng hợp vector đặc trưng từ ma trận độ đo tương đồng 33](#_Toc95589324)

[Hình 3.14: Stratified Shuffle Split 34](#_Toc95589325)

[Hình 3.15: Mô hình ca sử dụng của hệ thống 45](#_Toc95589326)

[Hình 3.16: Biều đồ tuần tự ca sử dụng đoán nhận ứng dụng 46](#_Toc95589327)

[Hình 3.17: Test API bằng posman 47](#_Toc95589328)

[Hình 3.18: Xây dựng API bằng python flask 47](#_Toc95589329)

[Hình 3.19: Trang chủ Android Studio 48](#_Toc95589330)

[Hình 3.20: Các dependency được sử dụng 48](#_Toc95589331)

[Hình 3.21: Retrofit library 49](#_Toc95589332)

[Hình 3.22: Giao diện chính 50](#_Toc95589333)

[Hình 3.23: Giao diện chọn file 50](#_Toc95589334)

[Hình 3.24: Sau khi chọn file 50](#_Toc95589335)

[Hình 3.25: Kết quả đoán nhận 50](#_Toc95589336)

# CƠ SỞ LÝ THUYẾT

## Tổng quan đề tài

### Đặt vấn đề

Theo các số liệu thống kê mới nhất, Android hiện nay vẫn là hệ điều hành chiếm lĩnh phần lớn thị phần di động trên toàn cầu (87,7% - theo IDC 2018 và tỷ lệ này vẫn giữ nguyên cho đến năm 2021. Vì vậy, các thiết bị sử dụng hệ điều hành Android đã trở thành đối tượng bị tấn công của các ứng dụng độc hại android, gây ra mối đe dọa nghiêm trọng đối với việc rò rỉ dữ liệu cá nhân như vị trí người dùng, thông tin liên lạc, tài khoản, ảnh, vv... Thực trạng trên khiến việc xây dựng ứng dụng phát hiện phần mềm độc hại Android ngày càng trở nên cấp thiết.

Trước đây, hướng tiếp cận để giải quyết bài toán phát hiện phần mềm độc hại chủ yếu dựa vào kỹ thuật đối sánh mẫu truyền thống. Tuy nhiên, với sự phát triển nhanh chóng của các phương pháp học máy và trí tuệ nhân tạo, hướng nghiên cứu về việc phát triển các hệ thống tự động phát hiện phần mềm độc hại bằng cách sử dụng các kỹ thuật khai phá dữ liệu và học máy đang thu hút sự quan tâm của các nhà nghiên cứu trong lĩnh vực an ninh mạng. Vấn đề lớn nhất của bài toán phát hiện phần mềm độc hại sử dụng học máy đó là giải pháp phân tích, biểu diễn dữ liệu và trích chọn đặc trưng. Có hai cách tiếp cận để giải quyết vấn đề này, một là sử dụng kỹ thuật phân tích hành vi và hai là kỹ thuật phân tích chữ ký [1].

Kỹ thuật phân tích chữ ký dựa trên các dấu hiệu số (Digital Footprint) để đoán nhận mã độc. Tất cả các chương trình, dù là lành hay độc, đều có những dấu hiệu số đặc trưng của riêng mình. Các phần mềm diệt virus thường kết hợp với sử dụng một cơ sở dữ liệu lưu trữ các dấu hiệu số của các mã độc. Các phần mềm ấy sẽ đối sánh dấu hiệu số của file đang được quét với các dữ liệu trong cơ sở dữ liệu, Nếu tìm thấy dấu hiệu đó trong cơ sở dữ liệu, file đó sẽ được coi là độc. Khi một loại mã độc mới được phát hiện, dấu hiệu số của nó sẽ được thêm vào cơ sở dữ liệu của công ty viết ra phần mềm virus đó, và sẽ được chia sẻ cho cơ sở dữ liệu từ phía người dùng.

Hướng nghiên cứu phân tích hành vi dựa vào việc phân tích và đánh giá mã nguồn của ứng dụng được nghi ngờ. Trong bài toán đoán nhận mã độc Android, hướng nghiên cứu này dựa vào việc phân tích lời gọi API, quyền truy cập, lời gọi hệ thống. Các thông tin trên được biểu diễn bằng các mô hình mạng thông tin đồng nhất, nghĩa là nếu biểu diễn các dữ liệu này thành đồ thị thì các đỉnh trong đồ thị có chung một kiểu, và các cạnh của đồ thị đều biểu diễn một loại quan hệ. Trên mô hình mạng đã xây dựng, các nhóm sẽ đề xuất các thuật toán trích chọn đặc trưng phù hợp với mô hình học máy sử dụng. Hiệu quả của các hệ thống tự động phát hiện mã độc sử dụng mạng thông tin đồng nhất để biểu diễn dữ liệu và trích chọn đặc trưng cho kết quả dự đoán chính xác từ 85-92%. Nguyên nhân là do dữ liệu trong các hệ thống thực tế - đặc biệt là dữ liệu về mã độc thường phong phú, đa dạng và chứa đựng nhiều ý nghĩa tiềm ẩn khác. Nếu sử dụng mạng đồng nhất để biểu diễn thì có thể sẽ làm mất đi các thông tin ngữ nghĩa quan trọng trong mạng và việc trích chọn đặc trưng trên mô hình mạng đồng nhất đã không tận dụng tối đa các thông tin thu được để phát hiện mã độc, đặc biệt đối với các mã độc thế hệ mới. Vì vậy hướng nghiên cứu ứng dụng các kỹ thuật phân tích dữ liệu phức tạp để giải quyết thách thức trên đang được rất nhiều nhà nghiên cứu quan tâm. Một trong các công cụ thường được sử dụng để phân tích, biểu diễn dữ liệu phức tạp hiện nay là sử dụng mô hình mạng thông tin không đồng nhất (Heterogeneous Information Network- HIN).

Mạng thông tin không đồng nhất gồm nhiều loại đối tượng khác nhau và các liên kết (cạnh) trong mạng thông tin không đồng nhất có thể mang nhiều ý nghĩa khác nhau. Bởi vậy, việc khai phá các mạng thông tin không đồng nhất sẽ cho chúng ta biết thêm các tri thức còn tiềm ẩn trong các cấu trúc mạng, từ đó có thể ứng dụng trong nhiều lĩnh vực khác nhau. Cho đến hiện này, HIN đã được ứng dụng trong nhiều lĩnh vực khác nhau, như khai phá dữ liệu văn bản, khai phá dữ liệu sinh học, và đặc biệt ứng dụng trong bài toán phát hiện mã độc. Điển hình như nhóm nghiên cứu của [*Yanfang Ye*](https://www.researchgate.net/scientific-contributions/69831786_Yanfang_Ye)[2] *và đồng nghiệp* đã sử mạng thông tin không đồng nhất để biểu diễn dữ liệu mã độc Android, mạng bao gồm 5 loại đỉnh (ứng dụng, API, IMEI, nhà sản xuất, chữ ký) và 5 loại cạnh (ứng dụng – API, ứng dụng-IMEI, IMEI-Nhà sản xuất, ứng dụng-nhà sản xuất, ứng dụng – chữ ký).

### Mục tiêu đề tài

Mục tiêu của đề tài là xây dựng ra được một hệ thống phát hiện phầm mềm độc hại, đồng thời có thể phân loại được những phần mềm độc hại đã xác định. Mục đích của việc phân loại ra các loại ứng dụng dụng độc hại là để phục vụ cho mục đích nghiên cứu tiếp theo, khi mà chúng ta cần tới một tập dữ liệu mã độc đã được phân loại khổng lồ lên tới hàng trăm nghìn ứng dụng, đổi khi lên đến hàng triệu ứng dụng, việc phân tích động sẽ khó có có thể đáp ứng dược nhu cầu về mặt chi phí với hiệu năng.

Hơn thế nữa, đề tài còn giúp ích cho sinh viên trong việc bổ sung kiến thức liên quan đến mã độc, trong thời kỳ số hiện nay, càng nhiều người có hiểu biết về mã độc, phân mềm độc hại, thì nguy cơ gây hại bở các phần mềm độc hại phần nào được giảm đi rõ rệt.

## Tổng quan về phần mềm độc hại

### Mã độc

Mã độc hay còn gọi là phần mềm độc hại (malware - malicious software) là một chương trình được bí mật chèn vào hệ thống mạng nhằm thực hiện các hành vi phá hoại. Khi xâm nhập thành công, mã độc có thể đánh cắp thông tin, làm gian đoạn hệ thống hoặc gây tổn hại tới tính bí mật, tính toàn vẹn và tính sẵn sàng của máy tính nạn nhân

Phần mềm độc hại là một dạng chương trình hoặc một đoạn chương trình tự cài đặt hoặc được cài đặt trên bất cứ thiết bị nào mà không cần sự cho phép của người dùng và thực hiện các chức năng mà người dùng không hề biết. Động cơ cơ bản của phần mềm độc hại là đánh cắp thông tin bảo mật từ điện thoại thông minh, khóa điện thoại thông minh, gửi tin nhắn SMS/MMS, thực hiện cuộc gọi đến đầu số cước phí đặc biệt, chia sẻ thông tin thông qua định vị GPS.

Tất cả những người dùng thiết bị di động chạy hệ điều hành Android có kết nối internet đều có thể trở thành nạn nhân của phần mềm độc hại. Phần mềm độc hại trên di động chủ yếu lấy cắp các dữ liệu nhạy cảm (như thẻ tín dụng, mật khẩu, nhật ký cuộc gọi, SMS) và theo dõi khách hàng. Ngoài ra, phần mềm độc hại có thể làm giảm tuổi thọ của pin hoặc giảm khả năng xử lý, chiếm quyền điều khiển trình duyệt, gửi tin nhắn trái phép và có thể vô hiệu hóa toàn bộ thiết bị. Một số trường hợp khách hàng mất tiền cước do phần mềm độc hại đăng ký dịch vụ game, thực hiện cuộc gọi quốc tế, …Tốc độ gia tăng nhanh chóng của các ứng dụng được phát triển trên nền tảng hệ điều hành này chính là môi trường thuận lợi cho các tin tặc phát triển các loại phần mềm độc hại mới với chiêu thức hoạt động tinh vi hơn nhằm trục lợi từ thông tin cá nhân người dùng hoặc thậm chí là tống tiền.

### Các dạng mã độc

Mã độc được chia thành nhiều loại tùy theo chức năng và cách thức lây nhiễm:

**Virus:** virus chỉ là một dạng của mã độc nói chung. Điểm khác biệt nằm ở chỗ virus có khả năng lây lan cực nhanh, số lượng virus hiện nay chỉ chiếm chưa đến 10 % tổng số mã độc. Dưới đâu là 3 loại virus thường gặp:

* **Virus Hoax:** Cảnh báo giả về virus, cảnh báo giả này thường núp dưới dạng một yêu cầu khẩn cấp để bảo vệ hệ thống. Mục tiêu của cảnh báo virus giả là cố gắng lỗi kéo mọi người gửi cảnh báo càng nhiều càng tốt qua email. Cảng báo giả này không trực tiếp gây nguy hiểm. Tuy nhiên, các thư cảnh báo có thể chứa các chỉ dẫn về thiết lập lại hệ điều hành hoặc xóa file làm nguy hại tới hệ thống. Kiểu cảnh báo giả này cũng gây tốn thời gian và quấy rối bộ phận hỗ trỡ kỹ thuật khi có quá nhiều người gọi đến và yêu cầu dịch vụ.
* **Scripting Virus:** Scripting virus là loại virus được viết bằng các ngôn ngữ script như VBScript, JavaScript, Batch script. Các loại virus này thường có đặc điểm dễ viết, dễ cài đặt. Chúng thường tự lây lan sang các file script khác, thay đổi nội dung cả các file html để thêm các thông tin quảng cáo, chèn banner… Đây cũng là một loại virus phát triển nhanh chóng nhờ sự phổ biến của Internet.
* **File Virus:** Virus này thường lây vào các file thực thi (ví dụ file có phần mở rộng .com, .exe, .dll) một đoạn mã để khi file được thực thi, đoạn mã virus sẽ được kích hoạt trước và tiếp tục thực hiện các hành vi phá hoại, lây nhiễm. Loại virus này có đặc điểm lây lan nhanh và khó diệt hơn các loại virus khác do phải xử lý cắt bỏ, chỉnh sửa file bị nhiễm. Tuy nhiên, file virus có đặc điểm là chỉ lây vào một số định dạng file nhất định và phụ thuộc vào hệ điều hành.

**Trojan horse:** Tên của loại mã độc này được lấy theo một điển tích cổ. Trong cuộc chiến với người Tơ-roa, các chiến binh Hy Lạp sau nhiều ngày không thể chiếm được thành đã nghĩ ra một kế. Họ giả vờ giảng hòa rồi tặng người dân thành Tơ-roa một con ngựa gỗ khổng lồ. Sau khi ngựa gỗ được đưa vào thành, các chiến binh Hy Lạp từ trong ngựa gỗ chui ra đánh chiếm thành.

Đây cũng chính là cách mà các Trojan áp dụng: các đoạn mã của Trojan được “che giấu” trong các phần mềm máy tính thông thường để bí mật xâm nhập vào máy nạn nhân. Khi tới thời điểm thuận lợi, chúng sẽ đánh cắp thông tin cá nhân và chiếm quyền điều khiển máy tính… Bản chất của Trojan là không tự lây lan mà sử dụng phần mềm khác để phát tán.

Dựa vào cách hoạt động ta có thể phân chia Trojan thành 3 loại chính sau: BackDoor, Adware và Spyware:

* **Backdoor:** Phần mềm Backdoor (cửa sau) là một dạng Trojan. Khi xâm nhập vào máy tính, backdoor sẽ mở ra một cổng dịch vụ cho phép tin tặc điều khiển máy tính nạn nhân. Tin tặc có thể cài phần mềm backdoor lên nhiều máy tính khác nhau thành một mạng lưới các máy bị điều khiển – Bot Net. Từ đó, thực hiện các vụ [tấn công từ chối dịch vụ DDoS](https://securitybox.vn/1353/12-loai-tan-cong-ddos-tan-cong-tu-choi-dich-vu-ddos/).
* **Adware - phần mềm quảng cáo:** Đây là loại Trojan nhằm mục đích quảng cáo. Khi bị nhiễm adware, thiết bị có thể sẽ bị thay đổi trang chủ tìm kiếm, bị làm phiền bởi hàng loạt quảng cáo liên tục…. Adware là phần mềm được thiết kế để hiển thị quảng cáo lên màn hình thiết bị của người dùng, thường ngụy trang dưới dạng một chương trình hợp pháp hoặc ẩn mình trong một chương trình khác để lừa người dùng cài đặt nó. Khi xâm nhập thành công vào thiết bị, adware có thể thực hiện các tác vụ không mong muốn như: tự động mở tab mới, thay đổi trang chủ tìm kiếm, thu thập thông tin người dùng nhằm mục đích quảng cáo, chuyển hướng người dùng đến trang web khác.
* **Spyware - phần mềm gián điệp:** Spyware (phần mềm gián điệp) là phần mềm dùng để đánh cắp thông tin của người dùng. Spyware thường được bí mật cài đặt trong các phần mềm miễn phí và phần mềm chia sẻ từ Internet. Một khi đã xâm nhập thành công, spyware sẽ điều khiển máy chủ và âm thầm chuyển dữ liệu người dùng đến một máy khác. Spyware phần mềm độc hại xâm nhập vào thiết bị di động để ngấm ngầm thu thập thông tin bảo mật của người dùng và chuyển tiếp đến cho bên thứ ba. Spyware thường chạy ngầm trong hệ thống và âm thầm giám sát, thu thập thông tin nhằm phá hoại thiết bị di động cũng như quá trình truy cập Internet bình thường của người dùng.

**Ransomware - Mã độc tống tiền:** Ransomware là một loại mã độc được dùng để ngăn chặn người dùng truy cập dữ liệu và sử dụng máy tính. Để lấy lại dữ liệu và quyền kiểm soát máy tính, nạn nhân cần chuyển tiền cho tin tặc. Đó là lý do ransomware còn được gọi là mã độc tống tiền. Ransomware thường xâm nhập qua email rác hoặc trang web lừa đảo. Trong một số trường hợp, ransomware được cài đặt cùng với Trojan để có thể kiểm soát nhiều hơn trên thiết bị của nạn nhân. Sau khi đã xâm nhập vào máy tính của người dùng, ransomware sẽ rà soát máy tính và thực hiện một số tác vụ như: Mã hóa toàn bộ dữ liệu như dạng đuôi .doc, .xls, .pdf, file email. In một thông báo “tống tiền” ra ngoài màn hình chính hoặc để lại file có nội dung “tống tiền”.

Graphical user interface, text

Description automatically generated

Hình 1.1: Mã độc tống tiền

**Worm - Sâu máy tính:** Worm là loại mã độc phát triển và lây lan mạnh mẽ nhất hiện nay. Worm thường được phát tán qua email. Các email này thường có nội dung giật gân và hấp dẫn để thu hút lượt click của người dùng. Nhờ những email giả mạo đó mà worm có thể lây lan theo cấp số nhân. Nhận thấy khả năng lây lan mạnh mẽ của worm, những kẻ viết phần mềm độc hại này đã đưa thêm vào worm các tính năng khác như phá hoại hệ thống, ăn cắp thông tin… Có thể thấy, sức tàn phá của worm là vô cùng lớn

**Rootkit:** Rootkit lại được coi là một trong những loại mã độc nguy hiểm nhất. Rootkit là một chương trình máy tính được thiết kế để truy cập sâu vào hệ thống máy tính mà vẫn che giấu được sự hiện diện của nó và các phần mềm độc hại khác. Nhờ rootkit, các phần mềm độc hại dường như trở nên “vô hình” trước những công cụ rà quét thông thường, thậm chí trước cả các phần mềm diệt virus. Việc phát hiện mã độc trở nên khó khăn hơn rất nhiều trước sự bảo vệ của rootkit.

**Botnet:** Botnet là những máy tính bị nhiễm virus và bị điều khiển thông qua Trojan, virus… Tin tặc lợi dụng sức mạnh của những máy tính bị nhiễm virus để thực hiện các hành vi phá hoại và ăn cắp thông tin. Thiệt hại do Botnet gây ra thường vô cùng lớn.

Botnet có 2 loại là DNS và IRC:

* DNS Bot: dễ thực hiện và điều khiển đơn giản không quá cầu kỳ, được dùng để chạy Bot trên nền Web. Nhưng vẫn bị hạn chế trong việc trao đổi thông tin giữa Bot master và các Bot
* IRC Bot: Giúp trao đổi thông tin giữa Bot Master và các Bot, điều khiển qua mạng chat IRC. Nhưng lại bị phụ thuộc vào các IRC và người quản trị Server

A picture containing diagram

Description automatically generated

Hình 1.2: Botnet

### Quan hệ giữa hai loại mã độc và các cơ chế hoạt động

Loại phần mềm độc hại thứ nhất và thứ hai của là một điều kiện tiên quyết cho các loại thứ ba, chúng đóng một vai trò cung cấp và duy trì cho các loại thứ ba trên hệ thống đích điều khiển từ xa một cách trái phép, việc thực hiện một mục đích cuối cùng của sự xâm nhập được thực hiện bởi một loại phần mềm độc hại thứ ba. Trong loại phần mềm độc hại đầu tiên, exploit, trojan, worm, bot và virus đều có thể bao gồm phần payload. Do đó, ngoài chức năng có được quyền kiểm soát từ xa đối với hệ thống đích, trojan, worm, bot và virus cũng có thể bao gồm loại phần mềm độc hại thứ hai và loại phần mềm độc hại thứ ba trong payload và phát hành chúng sau khi thực thi.

Một mối đe dọa hiện nay là có loại tấn công mang tên Blended Attck, sử dụng nhiều công nghệ liên quan đến phần mềm độc hại để tối đa hóa hiệu quả tấn công.

* Trong giai đoạn đầu tiên để xâm nhập và giành được quyền điều khiển từ xa của hệ thống đích, các phương tiện xâm nhập như trojan, worm, bot và virus được kết hợp trong một chương trình thực thi cụ thể để có thể kết hợp giữa xâm nhập chủ động và xâm nhập thụ động, từ đó tăng cường đáng kể khả năng xâm nhập vào hệ thống đích.
* Trong giai đoạn duy trì kiểm soát từ xa đối với hệ thống đích, việc sử dụng thành thạo các kỹ thuật liên quan đến rootkit cho phép rootkit và các phần mềm độc hại khác được bảo vệ để tồn tại trong hệ thống đích một cách đáng tin cậy và trong thời gian dài.
* Khi hoàn thành mục đích xấu cụ thể, một loạt các chức năng phần mềm độc hại loại ba được tích hợp, để có thể thu thập thông tin, có thể gây thiệt hại cho hệ thống đích nếu muốn.

## Các cách phát hiện phần mềm độc hại trên Android

### Phân tích tĩnh

Đối tượng chính của phân tích tĩnh trong phần mềm độc hại Android là kiểm tra các quyền, mã nguồn ứng dụng, bộ phận cấu thành, tài nguyên và mã nhận diện. Tất cả thông tin liên quan đến ứng dụng nằm trong tệp APK. Quyền truy cập, tài nguyên, mã, dịch vụ và tất cả thông tin khác được trích xuất từ tệp APK và được phân tích chính xác. Có nhiều công cụ có sẵn khác nhau cho phân tích tĩnh như apktool, aapt, dex2jar, jd-gui. Đối với việc áp dụng phân tích tĩnh vào ứng dụng Android, ta có thể tiếp cận theo hướng như sau: ứng dụng Android hầu như được xây dựng trên bốn thành phần bao gồm Activity biểu diễn các giao diện có thể tương tác với người dùng, Service thực thi các tác vụ ngầm, Content Provider truy cập cấu trúc dữ liệu cho ứng dụng và Broadcast Receiver xử lý các sự kiện hệ thống hoặc của người dùng. Các thành phần này tương tác với nhau thông qua các phương thức, ví dụ như startActivity () để kích hoạt giao tiếp liên thành phần (Inter-Component Communication – ICC). Phương thức của ICC nhận các tham số Intent chứa dữ liệu thành phần đích mà thành phần gốc muốn giao tiếp cùng. Có hai loại tương tác ICC là ICC rõ ràng, nơi Intent chứa tên thành phần đích và ICC không rõ ràng, nơi Intent chỉ chứa các mục tiêu tiềm năng, có khả năng xử lý, các Intent này cần được khai báo một Intent Filter trong tệp cấu hình AndroidManifest. Các phương pháp phân tích tĩnh có thể kể đến như sau:

* Phân tích tĩnh dựa trên chữ ký/mẫu: Tệp APK chứa chữ ký cụ thể. Mỗi tệp APK có chữ ký khác nhau. Chữ ký chứa thông điệp rút gọn của tệp APK. Nếu có bất kỳ sự thay đổi nào trong tệp APK thì thông điệp rút gọn của chữ ký cũng sẽ thay đổi và người ta có thể nhanh chóng phân tích được ứng dụng là độc hại. Chữ ký của phần mềm độc hại cũng có thể được thu thập để xác định phần mềm độc hại một cách nhanh chóng. Điểm hạn chế của phương pháp này chính là nó chỉ có thể phát hiện được các phần mềm độc hại hiện có, đã được biết tới; trong khi phần mềm độc hại luôn được phát triển và ngày càng tinh vi hơn.
* Phân tích tĩnh dựa trên tài nguyên: Tệp AndroidManifest.xml là một tệp tài nguyên chứa tất cả thông tin về các tài nguyên được sử dụng trong ứng dụng [15]. Tài nguyên của một ứng dụng chứa các module giao diện người dùng như widget, menu, layout, ...Tệp AndroidManifest.xml có trong tệp APK. Phần mềm độc hại chạy ngầm cần sự tương tác của người dùng từ các giao diện người dùng này. Vì vậy, giao diện người dùng cũng một phần quan trọng cần được phân tích chính xác.
* Phân tích tĩnh dựa trên thành phần: Ứng dụng Android được chia thành nhiều thành phần như nhà cung cấp nội dung, dịch vụ, đối tượng, hoạt động và thu nhận tín hiệu. Thông tin của tất cả các thành phần có sẵn trong tệp AndroidManifest.xml. Hầu hết phần mềm độc hại chạy dưới dạng dịch vụ ngầm và thu thập thông tin về đối tượng, hoạt động, đối tượng thu nhận. Vì vậy việc phân tích các thành phần này cũng rất quan trọng trong việc nhận ra các hành vi độc hại.
* Phân tích tĩnh dựa trên quyền truy cập: Android cung cấp mô hình dựa trên quyền truy cập cho việc triển khai bảo mật sẵn có. Tất cả các quyền truy cập được định nghĩa trong tệp AndroidManifest.xml. Các ứng dụng phải yêu cầu quyền truy cập như danh bạ, tin nhắn, internet, GPS, máy ảnh, ...Quyền truy cập đóng vai trò quan trọng trong bất kỳ ứng dụng Android nào. Một ứng dụng chỉnh ảnh đơn giản yêu cầu quyền ‘READ\_SMS’, sau đó thực hiện một số hành vi độc hại. Vì vậy, việc phân tích các quyền cũng rất quan trọng để nhận ra các hành vi độc hại.
* Kỹ thuật phân tích sử dụng công nghệ máy ảo: Kỹ thuật phân tích này kiểm tra hành vi ứng dụng, phân tích quyền kiểm soát và luồng dữ liệu để dò tìm những chức năng nguy hiểm. Nó có thể kiểm tra bytecode của ứng dụng và theo dõi các lời gọi API nhạy cảm. Tuy nhiên với kỹ thuật này, phân tích được thực hiện bởi các lệnh, nó cũng làm tốn pin và dung lượng lưu trữ hơn.

Phương pháp phân tích tĩnh trên Android đôi khi gặp phải những vấn đề cố hữu, mà việc khắc phục rất khó khăn và gần như bất khả thi, bao gồm:

* Mã biên dịch: Như đã biết, ứng dụng Android phần lớn được xây dựng từ ngôn ngữ Java nhưng được biên dịch thành Dex bytecode và thực thi trên máy ảo Dalvik hoặc ART, tuy nhiên các chương trình thu thập dữ liệu phục vụ phân tích tĩnh lại không có khả năng giải quyết Dex bytecode hoặc các định dạng hỗ trợ. Do đó, các chương trình này tuy hoạt động được với mã nguồn Java nhưng lại trở nên vô tác dụng trong hệ sinh thái Android. Ví dụ như phần mềm FindBug 5 được phát triển và hoạt động tốt trong việc phát hiện phần mềm độc hại Java, nhưng lại không thể đưa vào khai thác đối với Android.
* Điểm đầu vào: Không như các chương trình Java hay C, Android không có hàm main (), thay vào đó là các điểm đầu vào được gọi bởi Framework trong quá trình chạy. Đối lập với điều này, các ứng dụng phân tích phần mềm độc hại cần phải xác định điểm vào và sơ đồ hoá các liên kết. Với nhiều điểm đầu vào, việc xác định này rất khó khăn và không thể đảm bảo được sự liên kết giữa chúng.
* Vòng đời ứng dụng: Một vấn đề nữa đối với hệ điều hành Android đó chính là vòng đời của các thành phần ứng dụng. Mỗi một thành phần lại có một vòng đời riêng và đảm nhiệm một tác vụ hệ thống, và được gọi đến với nhiều kịch bản. Ví dụ, một ứng dụng khi không còn được sử dụng mà vẫn chiếm quá nhiều tài nguyên, hệ thống sẽ tự động đóng tiến trình mà không đi qua các hàm xử lý như vòng đời thông thường, các ứng dụng phân tích phần mềm độc hại sẽ gặp rất nhiều khó khăn khi xây dựng kịch bản hoạt động.
* Sự kiện: Sự kiện trên hệ điều hành Android có thể chia làm hai phần là sự kiện hệ thống (Cảnh báo pin, GPS …) và sự kiện người dùng (thao tác với UI). Các sự kiện này có thể xảy ra liên tục, chồng chéo nhau và có thể rơi vào bất kỳ thời điểm nào khi thiết bị hoạt động, khiến việc xây dựng và duy trì mô hình phân tích trở nên bất khả thi.
* Giao tiếp liên thành phần: Android có cơ chế giao tiếp giữa các thành phần, thậm chí còn diễn ra giữa hai ứng dụng khác nhau. Ứng dụng phân tích muốn quản lý các giao tiếp này yêu cầu một phương pháp phỏng đoán tiên tiến, nơi mà các liên kết được kích hoạt bằng các phương thức cụ thể với các tham số đa dạng. Mặt khác, các phương thức này còn được quản lý bởi vòng đời ứng dụng, hệ thống này chịu trách nhiệm trực tiếp trong quá trình khởi chạy.
* Thư viện: Thông thường, một ứng dụng Android sẽ sử dụng một hoặc nhiều thư viện để phục vụ người dùng, như thư viện quảng cáo hoặc mua bán. Những thư viện này có rất nhiều dòng mã, khiến kích thước của ứng dụng bị tăng lên đáng kể. Đối với phân tích tĩnh, việc đi sâu vào mã nguồn ứng dụng với các bộ thư viện này là không cần thiết và tiêu tốn rất nhiều thời gian, thậm chí còn nhiều hơn so với phân tích mã nguồn gốc. Mặt khác, phân tích các thư viện này tiềm ẩn nguy cơ thu thập các mẫu dương tính do cần yêu cầu các tài nguyên mà bị nhầm lẫn không cần thiết cho ứng dụng.

### Phân tích động

Phân tích động bao gồm phân tích hành vi của ứng dụng. Nó kiểm tra sự tương tác của phần mềm độc hại với các tài nguyên di động và các dịch vụ như vị trí, mạng, gói, các hoạt động hệ điều hành. Đặc trưng của phân tích động bao gồm lời gọi hệ thống, lưu lượng mạng, luồng mạng, địa chỉ mạng. Phân tích động giám sát hành vi của hệ thống. Một số framework có sẵn để thực hiện phân tích động có thể kể đến như Ananas, TaintDroid, DroidScope, CopperDroid, Crowdroid. Các kỹ thuật phân tích động phổ biến được trình bày trong bảng sau:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| STT | Kỹ thuật phân tích | Cách thức hoạt động | Ưu điểm | Nhược điểm |
| 1 | Kỹ thuật phân tích dựa trên các hoạt động bất thường | Dựa trên việc quan sát hành vi thiết bị bằng cách theo dõi các tham số khác nhau và trạng thái của các thành phần thiết bị | Theo dõi các tham số khác nhau từ đó có cái nhìn rõ ràng về hệ thống | Số tham số tham gia càng nhiều thì các phép toán được yêu cầu càng nhiều |
| 2 | Kỹ thuật phân tích theo vết dữ liệu | Theo dõi nhiều nguồn dữ liệu nhạy cảm và phát hiện rò rỉ dữ liệu trong các ứng dụng di động | Theo dõi các dữ liệu nhạy cảm một cách hiệu quả | Không thực hiện theo dõi luồng kiểm soát |
| 3 | Kỹ thuật phân tích dựa trên mô phỏng | Phân tích ứng dụng dựa trên kỹ thuật giám sát thời gian chạy của máy ảo cấp hệ thống | Giám sát toàn bộ hệ thống bằng cách ở ngoài môi trường thực thi | Không phát hiện được các phần mềm độc hại mới |

Bảng 1.1: Phân tích động

### Phân tích lai

Phân tích lai là sự kết hợp của cả phân tích tĩnh và phân tích động. Quá trình bắt đầu với phân tích tĩnh. Phân tích tĩnh kiểm tra mã, quyền truy cập và các thành phần của ứng dụng. Sau đó, phân tích động thực hiện phân tích hành vi tổng thể của ứng dụng. Chỉ có một số framework dùng cho phân tích lai như Mobile Sandbox, Andrubis. Hệ thống AASandbox (Android Application Sandbox) đưa ra phân tích hai bước cho các ứng dụng Android. Một ứng dụng di động gửi đến AASandbox, tại đó nó thực hiện phân tích tĩnh và động trong chế độ ẩn (offline). Phân tích tĩnh vô hiệu hóa mã nhị phân ứng dụng cài đặt và sử dụng từng đoạn mã để tìm các mẫu nghi vấn. Phân tích động thực thi mã nhị phân trên bộ mô phỏng Android và ghi lại các lời gọi hệ thống.

Diagram

Description automatically generated

Hình 1.3 Kiến trúc của công cụ ANDRUBIS

Bên cạnh hai phương pháp phổ biến trên cũng có một số kỹ thuật khác cũng rất phổ biến như phân tích quyền ứng dụng, giám sát tuổi thọ pin hay gần đây nhất, với sự bùng nổ của công nghệ học máy và trí tuệ nhân tạo, việc phân tích đã được xây dựng tự động giúp rút ngắn tối đa thời gian cũng như chi phí phân tích và phát hiện phần mềm độc hại trên di động.

Phát hiện phần mềm độc hại là một yếu tố quyết định trong bảo mật hệ điều hành Android. Với sự phát triển nhanh chóng của các phương pháp học máy và trí tuệ nhân tạo, hướng nghiên cứu về việc phát triển các hệ thống tự động phát hiện phần mềm độc hại bằng cách sử dụng các kỹ thuật khai phá dữ liệu và học máy đang thu hút sự quan tâm của các nhà nghiên cứu trong lĩnh vực an ninh mạng. Với đề tài nghiên cứu này, chúng em sử dụng phương pháp phân tích tĩnh trong đó sử dụng thông tin về các lời gọi API và thông tin về mối quan hệ giữa các lời gọi API (các API cùng package, cùng block và cùng kiểu invoke), để xây dựng hệ thống phát hiện phần mềm độc hại dựa trên mạng thô

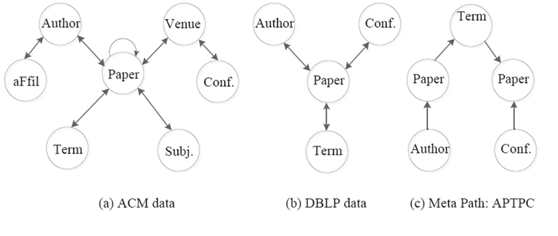
# MẠNG THÔNG TIN KHÔNG ĐỒNG NHẤT

## Giới thiệu về mạng thông tin không đồng nhất

Ngày nay chúng ta đang sống trong một xã hội kết nối, hầu hết các đối tượng đều được liên kết với nhau để tạo thành một mạng kết nối khổng lồ, có thể gọi chung là mạng thông tin (information network). Chẳng hạn như mạng xã hội, mạng world wide web, mạng giao thông giữa các thành phố, mạng sinh học, … Những mạng thông tin phổ biến này tạo thành một thành phần quan trọng của cơ sở hạ tầng thông tin hiện đại.

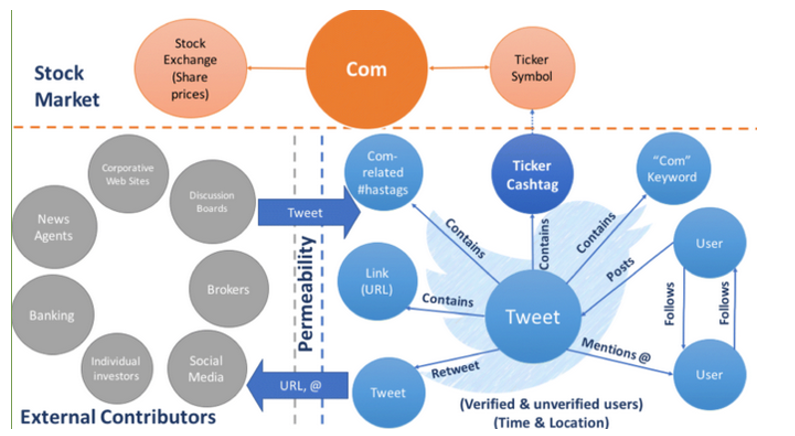
Hầu hết các nghiên cứu hiện nay trong lĩnh vực phân tích mạng thông tin (information network analysis) đều giả thiết các mạng này là *mạng thông tin đồng nhất (homogeneous information network)*, nghĩa là xem các đối tượng (đỉnh) là cùng kiểu và các liên kết giữa các đối tượng trong mạng cũng cùng một kiểu. Ví dụ như mạng liên kết các tác giả trong lĩnh vực nghiên cứu chỉ chứa các đối tượng tác giả cùng mối liên kết đồng tác giả. Các mạng thông tin đồng nhất này được đơn giản hóa từ mạng thông tin trên thực tế bằng cách bỏ qua tính không đồng nhất giữa các đối tượng và liên kết, hoặc chỉ xét một loại liên kết giữa một loại đối tượng. Tuy nhiên, trong thực tế, các mạng thông tin đều chứa các thành phần khác nhau, các tương tác giữa chúng cũng khác nhau và có thể mô hình hóa như một *mạng thông tin không đồng nhất (heterogeneous information network - HIN)*, trong đó các đỉnh có thể là các kiểu đối tượng khác nhau và các cạnh có thể là các liên kết khác nhau.

Chẳng hạn như mạng thông tin thư mục (Bibliographic information network) [3] DBLP, là một mạng không đồng nhất điển hình, bao gồm ba loại thực thể: bài báo (P), hội nghị (V), tác giả (A) và thuật ngữ (T). Đối với mỗi bài báo , nó sẽ có các mối liên kết dẫn tới một nhóm các tác giả, các hội nghị, một tập hợp các thuật ngữ. Ngoài ra, nó cũng có thể chứa thông tin trích dẫn cho một số bài báo, nghĩa là các liên kết tới một tập các bài báo được trích dẫn và liên kết từ một tập tới các bài báo trích dẫn đó. Lược đồ mạng cho một mạng thông tin thư mục và một thể hiện của nó được hiển thị như hình dưới.



Hình 2.1: Mô hình mạng thông tin thư mục DBLP[3]

Mạng thông tin Twitter: Twitter - một phương tiện truyền thông mạng xã hội cũng có thể được coi là một mạng thông tin, bởi nó chứa các loại thực thể như người dùng, tweet, hashtag và các thuật ngữ, kiểu liên kết như theo dõi (follow) giữa người dùng với người dùng, các bài đăng, tweet giữa các người dùng và các ràng buộc giữa các tweet và hashtag. Hình dưới minh họa một mạng Twitter điển hình.



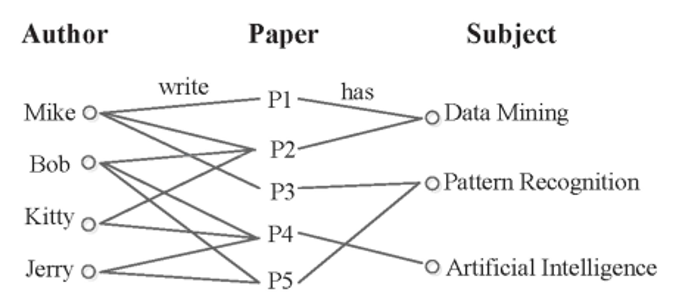
Hình 2.2: Mô hình mạng Twitter

## Mạng thông tin không đồng nhất

Như chúng ta đã biết, phần lớn các hệ thống thực tế thường bao gồm một số lượng lớn các đối tượng với nhiều loại tương tác khác nhau ví dụ như mạng xã hội, mạng máy tính, mạng sinh học, vv… Các đối tượng tương tác với nhau tạo thành một mạng kết nối khổng lồ, gọi chung là mạng thông tin. Mạng thông tin bao gồm các đối tượng và mối liên kết giữa các đối tượng đó. Chúng ta có thể định nghĩa như sau:

* **Định nghĩa 1.1** Mạng thông tin [2]: Mạng thông tin là một đồ thị có hướng với mỗi kiểu thực thể là một ánh xạ và mỗi kiểu liên kết là một ánh xạ . Trong đó, mỗi đỉnh , thuộc về một kiểu đối tượng cụ thể của tập A: và một liên kết thuộc về một kiểu dữ liệu cụ thể của tập R: .
* **Định nghĩa 1.2** Mạng thông tin được gọi là không đồng nhất nếu số lượng phần tử trong hoặc số lượng phần tử trong

Dưới đây là một mạng thông tin không đồng nhất gồm ba loại đỉnh và hai loại liên kết. Ba loại đỉnh gồm Author (tác giả), Paper (bài báo) và Subject (chủ đề). Hai loại liên kết: liên kết *viết* giữa tác giả - bài báo và liên kết *thuộc về* giữa bài báo và chủ đề.



Hình 2.3: Ví dụ một mạng thông tin không đồng nhất [3]

## Lược đồ

Để có thể hiểu rõ hơn các kiểu đối tượng và các kiểu liên kết trong mạng không đồng nhất, người ta đưa ra khái niệm về lược đồ mạng. Lược đồ mạng dùng để mô tả cấu trúc của mạng.

* **Định nghĩa 1.3** Lược đồ mạng [2]: Lược đồ mạng được biểu diễn bằng là một đồ thị có hướng với mỗi đỉnh là một kiểu thực thể trong và mỗi cạnh là một kiểu liên kết trong .

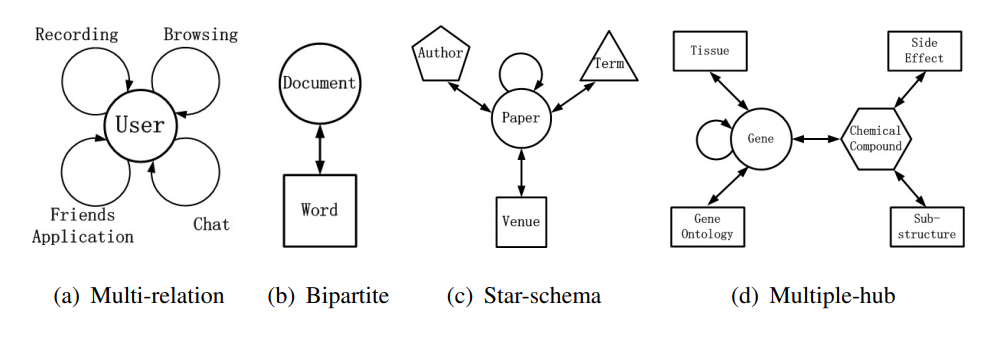
Mỗi kiểu liên kết giữa đối tượng và đối tượng , nghĩa là , trong đó được gọi là tập nguồn, được gọi là tập đích, có thể được ký hiệu là hoặc tương ứng. Quan hệ nghịch đảo được định nghĩa .

## Phân loại mạng thông tin không đồng nhất

Ta có thể phân loại mạng thông tin không đồng nhất dựa trên thù hình mạng (Topology). Dựa vào tiêu chí này, ta phân ra làm những loại sau:

* Mạng đa quan hệ với một kiểu thực thể (multiple relation): Mạng có một kiểu nút nhưng có nhiều mối quan hệ giữa chúng. Mối quan hệ giữa người và người có dạng như vậy, giữa con người với nhau có thể tồn tại các cung bậc cảm xúc, từ yêu mến đến sợ hãi,…
* Mạng lưỡng phân (Bipartite network): gồm hai kiểu thực thể khác nhau và các liên kết giữa chúng. Mạng hai phía được sử dụng để mô phỏng tương tác giữa hai kiểu đối tượng. Một ví dụ điển hình có thể kể đến là quan hệ giữa các kiểu đối tượng, như user – item hay document – word.
* Mạng hình sao (Star - scheme network): là loại mạng phổ biến nhất, gồm một thực thể là thành phần trung tâm và những thực thể khác nằm xung quanh. Ví dụ về mạng thông tin thư mục ở trên thể hiện rõ nhất các khía cạnh của mạng hình sao.
* Mạng đa trung tâm (Multi - hub network): Các mạng hình sao có thể kết hợp với nhau, tạo thành mạng đa trung tâm**~~.~~** Mạng này chứa những cấu trúc phức tạp hơn, liên quan nhiều đến những đối tượng trung tâm. Mạng tương tác thành phần nhân tế bào của Tsuyuzaki và Nikaido chính là một điển hình cho loại mạng này.

Dưới đây là các hình ảnh minh hoạ của các loại mạng trên:



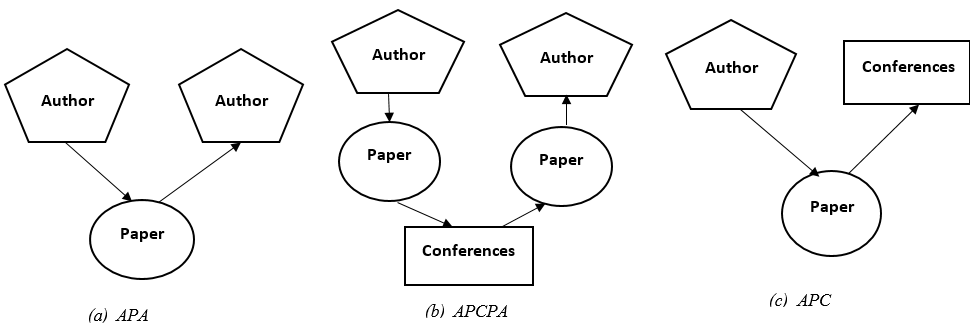
Hình 2.4: Minh hoạ các loại mạng thông tin không đồng nhất

## Siêu đường đi

Khác với mạng đồng nhất, hai đối tượng trong mạng thông tin không đồng nhất có thể kết nối với nhau thông qua các đường đi mang nhiều ngữ nghĩa khác nhau. Các đường này được gọi là siêu đường đi (Meta path) và được định nghĩa như sau.

* **Định nghĩa 1.4** Siêu đường đi [3]: Siêu đường đi (Meta-path) P là một đường đi trên đồ thị của lược đồ mạng và được ký hiệu dưới dạng , xác định mối quan hệ giữa các đối tượng Trong đó, là toán tử biểu thị thành phần quan hệ và l là độ dài của đường đi P.

Trên mạng thông tin không đồng nhất DBLP, ta có thể có các siêu đường đi như sau:



Hình 2.5: Minh họa các siêu đường đi trong mạng DBLP

Các tác giả có thể liên kết với nhau thông qua các siêu đường đi sau “Tác giả - Bài báo – Tác giả” (APA), “Tác giả - Bài báo – Hội nghị - Bài báo – Tác giả” (APCPA), “Tác giả - Bài báo – Hội nghị” (ACP). Cụ thể, bảng sau cho thấy các thể hiện cũng như ngữ nghĩa của các siêu đường đi. Siêu đường đi APA cho thấy mối quan hệ giữa các tác giả cùng cộng tác trong một bài báo, trong khi đường đi APCPA lại biểu diễn cho quan hệ giữa các tác giả công bố bài báo trong cùng một hội nghị.

|  |  |  |
| --- | --- | --- |
| STT | Siêu đường đi | Ý nghĩa |
| 1 | Tác giả - Bài báo – Tác giả (APA) | Biểu diễn mối quan hệ đồng tác giả trong một bài báo. |
| 2 | Tác giả - Bài báo – Hội nghị - Bài báo -Tác giả (APCPA) | Các tác giả công bố bài báo trong cùng một hội nghị. |
| 3 | Tác giả - Bài báo – Hội nghị (APV) | Các tác giả công bố bài báo trong cùng một hội nghị. |

Bảng 2.1: Mô tả ý nghĩa của các siêu đường đi

Thông qua quá trình ta phân tích ví dụ, ta có thể rút ra một số nhận xét về mạng không đồng nhất và siêu đường đi.

* Thứ nhất, siêu đường đi khác nhau sẽ mang ngữ nghĩa khác nhau. Sự phong phú về mặt ngữ nghĩa của siêu đường đi là một đặc tính quan trọng trong mạng thông tin không đồng nhất.
* Thứ hai, chúng ta thấy rằng, giữa A và B có nhiều mối quan hệ, nhưng sẽ có các quan hệ có trọng số lớn hơn những quan hệ khác. Ở ví dụ trên, nếu dựa theo siêu đường đi APA, các tác giả trong một bài báo (đồng tác giả) sẽ được đánh giá là “gần” nhau hơn. Tuy nhiên, nếu theo siêu đường đi APCPA thì các tác giả công bố bài báo trong cùng một hội nghị sẽ được đánh giá là “gần” nhau hơn. Độ tương đồng giữa hai tác giả sẽ được đánh giá khác nhau dựa trên các siêu đường đi khác nhau. Điều này là vì mạng không đồng nhất mang lại rất nhiều sự tương tác giữa các loại đối tượng. Do đó, khả năng dự đoán mối quan hệ giữa chúng cũng không hề nhất quán giữa các siêu đường đi.
* Thứ ba, và cũng là điểm quan trọng nhất, chúng ta sẽ cần có một tiêu chuẩn nhất định để đánh giá trọng số của chúng. Để làm vậy, chúng ta cần có các độ đo tương đồng, được trình bày ở dưới đây.

Độ đo tương đồng

Độ đo tương đồng được sử dụng để đánh giá sự tương đồng giữa các đối tượng. Hiện tại, có hai hướng tiếp cận khác nhau trên lĩnh vực này, là hướng đặc trưng và hướng liên kết.

Hướng đặc trưng hầu hết đều dựa vào các giá trị đặc trưng của các thành phần trên một siêu đường đi, chẳng hạn như độ tương đồng cosine (Cosine similarity score), hệ số Jaccard (Jaccard score) hay khoảng cách Euclid (Euclidean distance). Mặt khác, hướng tiếp cận liên kết lại lợi dụng các cấu trúc trên lược đồ, và đã tạo ra rất nhiều độ đo, như SimRank, đánh giá sự giống nhau của hai đối tượng bởi sự tương đồng giữa các kề cận của chúng; PathSim, được sử dụng để tìm các đối tượng ngang hàng trong mạng, tạo ra kết quả tốt hơn so với việc bước ngẫu nhiên cũng dựa trên phương pháp đo tương đồng; HeteSim để tính điểm phù hợp giữa các đối tượng thuộc các dạng khác nhau.[Laplacian][7].

Vậy nhưng các độ đo hướng liên kết này đều bộc lộ những yếu điểm: SimRank bị ảnh hưởng mạnh bởi nhiễu, và trong một số trường hợp, SimRank có thể bị lệch trọng tâm (Topic Drift). PathSim không những chỉ sử dụng được trên các đường đi đối xứng, mà còn chỉ có thể đo độ tương đồng giữa các đối tượng cùng loại, HeteSim khắc phục được những nhược điểm trên, nhưng vô cùng phức tạp, và hiệu suất thấp, không thể áp dụng cho các hệ thống với bộ dữ liệu lớn.

Để khắc phục những thiếu sót của HeteSim trong tính toán và nhu cầu bộ nhớ cao, Meng cùng các cộng sự đã đề xuất độ đo AverageSim để đánh giá độ đo tương tự thông qua bước đi ngẫu nhiên theo siêu đường đi và đường nghịch đảo tương ứng (P và P-1 ) [3]. Độ đo AvgSim là độ đo đối xứng và có tính tổng quát để đánh giá mức độ liên quan của hai đối tượng cùng kiểu hay khác kiểu. Giá trị AvgSim của hai đối tượng là trung bình của xác suất có thể đến được theo đường đi đã cho và đường ngược lại.

Ứng dụng thực tế của mạng thông tin không đồng nhất trong bài toán phát hiện phần mềm độc hại Android

Mạng thông tin không đồng nhất đang là hướng đi mới trong phát hiện phần mềm độc hại Android, có thể thấy rõ qua các nghiên cứu trong vòng ba năm trở lại đây của các nhóm nghiên cứu trên thế giới.

Shifu Hou, Yanfang Ye và đồng nghiệp kết hợp sử dụng các lời gọi API và mối quan hệ giữa chúng để tìm ra những mối quan hệ ẩn giấu bên trong mạng, kết hợp với trích xuất đặc trưng và mô hình học máy MKL. [2]

Một năm sau, Yanfang Ye và đồng nghiệp cũng sử dụng lời gọi API, thế nhưng họ lại phân tích chúng với những mối liên hệ khác trong hệ sinh thái Android để có thể thu thập thêm thông tin về ứng dụng đó. Họ sử dụng mạng thông tin không đồng nhất với 5 loại thực thể (app, API, IMEI, chữ ký và affiliation) và cũng sử dụng khái niệm meta-path. Họ giới thiệu phương pháp trích xuất đỉnh nhúng trong ứng dụng HinLearning~~,~~ và sử dụng mạng neuron học sâu. [4]

Năm 2019, Shifu Hou, Yanfang Ye và đồng nghiệp lại tiếp tục nghiên cứu và phát triển tiếp trên nền AIDroid, với mục đích áp dụng phương thức học biểu diễn (Representation learning). Các ứng dụng trong bộ huấn luyện được nhúng vào lược đồ mạng thông tin không đồng nhất, để từ đó họ có thể trích xuất thông tin từ các nút ngoài bộ huấn luyện mà không cần phải chạy lại hay chỉnh sửa mạng thông tin không đồng nhất. Họ sau đó thiết kế mô hình học sâu, lấy dữ liệu từ mạng để dự đoán phần mềm độc hại ở thời gian thực.[5]

Năm 2020, Hou và Ye lại có một hướng đi mới: sử dụng phương pháp học biểu diễn phân tích đặc trưng (Disentangled Representation Learning) để phân tích các phần mềm độc hại. Lần này, ngoài các đặc trưng của ứng dụng, Hou và Ye còn tập trung vào các yếu tố xã hội giữa ứng dụng, người lập trình và mẫu điện thoại để làm thông tin cho mạng. Họ đề ra phương pháp tích hợp các miền thông tin được tạo ra trước đó dưới các khung nhìn đặc trưng, như nội dung ứng dụng, người sở hữu ứng dụng và quá trình cài đặt để tạo ra một bộ phân tách đặc trưng để trích xuất những thông tin, những hệ số biến đổi ngầm định trong mạng. Đây là lần đầu tiên phương pháp học biểu diễn phân tích đặc trưng được sử dụng trong dự đoán phần mềm độc hại. [6]

Ta thấy rằng, nhóm tác giả trên trung thành với hướng tiếp cận biểu diễn thông tin trên các mạng không đồng nhất, thực hiện trích xuất các thông tin trên mạng đó và sử dụng các mô hình học máy để dò tìm phần mềm độc hại, thế nhưng họ liên tục phát triển mô hình của họ theo các hướng khác nhau.

Điều này có thể thấy rõ trước hết qua số lượng kiểu đối tượng và liên kết họ sử dụng: đầu tiên là chỉ với ứng dụng và API (HINDroid), họ phát triển thành bộ (ứng dụng, IMEI, API, chữ ký và affiliation), và gần đây nhất là tích hợp các đối tượng xã hội để làm phong phú thông tin trong mô hình của họ.

Chúng ta cũng có thể thấy sự phát triển của họ qua các phương pháp trích xuất thông tin. Từ trích xuất bằng meta-path dựa trên MKL, họ tiếp tục chuyển sang phương pháp trích xuất đỉnh nhúng trong AIDroid, rồi mới đây nhất là sử dụng phương thức học biểu diễn và nhúng đồ thị. Bên cạnh đó, các nhóm cũng sử dụng nhiều mô hình học máy khác nhau: từ các mô hình học máy cơ bản, sang các mô hình học sâu.

Là một công cụ tích hợp và biểu diễn dữ liệu lớn hiệu quả, việc nghiên cứu về mạng thông tin không đồng nhất và các thuật toán trích xuất dữ liệu trên mạng này để có thể thu thập được nhiều thông tin ngữ nghĩa còn tiềm ẩn sau cấu trúc mạng hiện đang là một hướng nghiên cứu nổi trội và còn rất nhiều thách thức ở phía trước. Đặc biệt là đối với bài toán phát hiện mã độc nói chung và mã độc Android nói riêng.

Cũng giống như hướng nghiên cứu của nhóm tác giả Shifu Hou và Yanfang Ye, chúng em dựa trên phương pháp phân tích lời gọi API và các mối liên hệ giữa chúng để xây dựng mạng thông tin không đồng nhất. Điểm khác biệt trong hướng nghiên cứu của nhóm là chúng em sử dụng độ đo AvgSim để đo độ tương đồng giữa hai ứng dụng Android bất kỳ, trên cơ sở các thông tin thu được, chúng em đề xuất công thức trích chọn vector đặc trưng để biểu diễn các ứng dụng Android thành các vector rồi đưa vào huấn luyện trong các mô hình học máy. Sau khi thực nghiệm trên các bộ dữ liệu được thu thập từ các nguồn đáng tin cậy và đánh giá kết quả, nhóm lựa chọn ba mô hình tốt nhất để xây dựng hệ thống phát hiện mã độc Android.

# XÂY DỰNG HỆ THỐNG

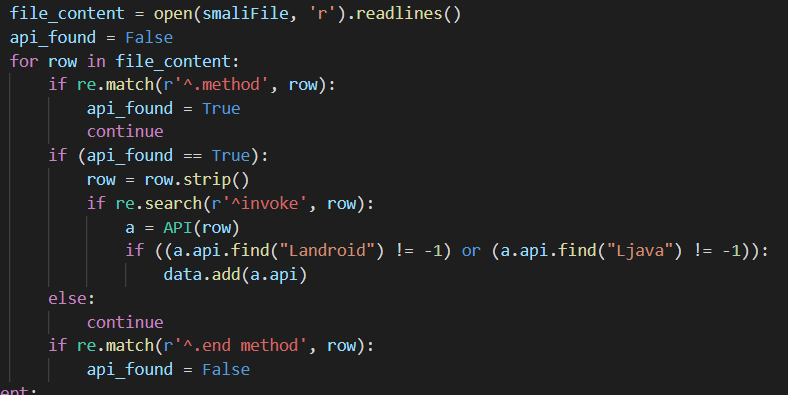
## Tổng quan xây dựng mô hình học máy

### Xây dựng tập API Dataset

Để trích xuất được dữ liệu chứa ở trong các ứng dụng adroid dựa trên các lời gọi API, việc cần làm đầu tiên đó chính là cần có một tập dữ liệu API dataset, tập dữ liệu API dataset này có bao gồm những API có thể được gọi ở trong ứng dụng, khi đi trích xuất dữ liệu, chúng em sẽ đi tìm xem ứng dụng có sử dụng những API ở trong tập dữ liệu không để đưa ra quyết định trích xuất thông tin.

Để xây dựng lên một tập API dataset, chúng em bắt đầu bằng việc từ một bộ bao gồm 600 ứng dụng có cả benign và malware, chúng em tiến hành trích xuất tất cả những lời gọi api mà ứng dụng đấy sử dụng, do số lượng API rất nhiều nên chúng em chỉ lọc và lấy ra những API bắt đầu bằng Ljava hoặc Landoird.

Tiếp theo, dựa trên cơ sở của Jaemin Jung và cộng sự [8], chúng em tiến hành xây dựng bộ api dataset bằng cách lựa chọn những api được sử dụng nhiều nhất ở cả bên benign và malware, số lượng mỗi bên là rơi vào khoảng hơn 250 API, từ đó xây lên được bộ API dataset có hơn 500 API.



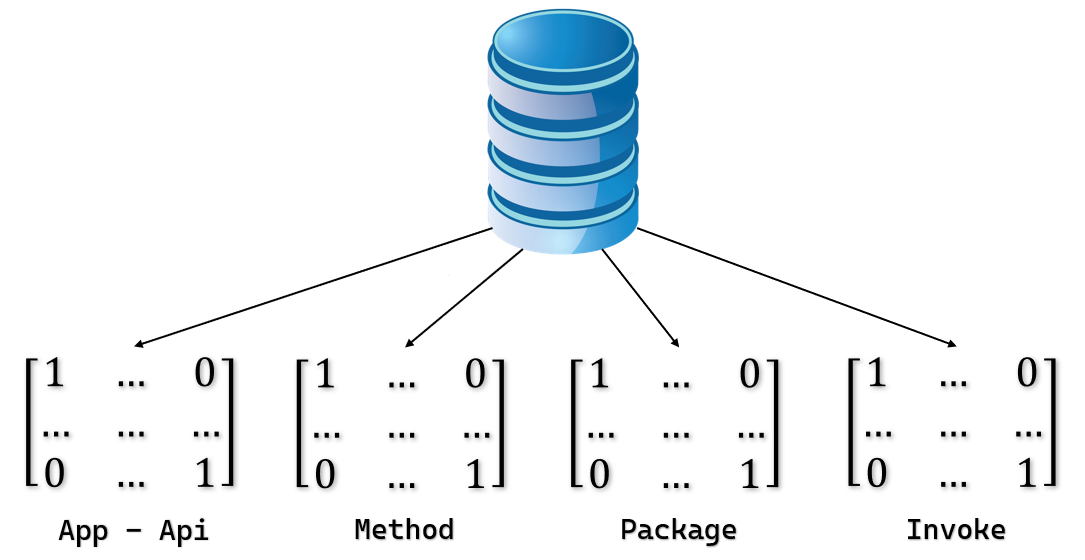
Hình 3.1: Code minh hoạ quá trình xây dựng API dataset

### Trích xuất dữ liệu và xây training mô hình học máy

Nhóm tiến hành tải và sử dụng một tập dữ liệu chứa hơn 12.000 ứng dụng android được chia thành năm họ là: Benign, Adware, Banking, Riskware và SMSmalware để xây dựng nên một mô hình học máy.

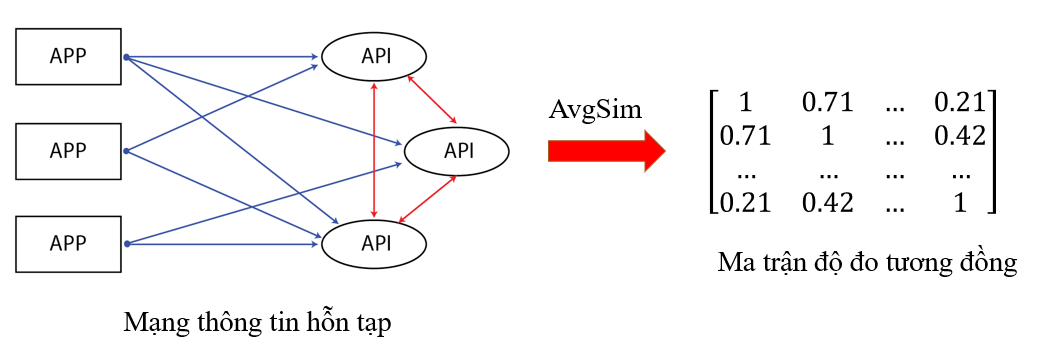
Quy trình triển khai xây dựng mô hình được chia thành ba bước như sau:

* **Bước 1:** Thu thập dữ liệu và xây dựng các ma trận cần thiết. Nhóm đã sử dụng tập dữ liệu CICDataset và cụ thể tập dữ liệu có tên là CICMalDroid 2020 bao gồm hơn 12.000 ứng dụng android, các ứng dụng android được chia thành sáu họ là: Benign (1911 ứng dụng), Adware (1415 ứng dụng), Banking (1900 ứng dụng), Riskware (2262 ứng dụng) và SMSmalware (4722 ứng dụng). Tiếp theo là sử dụng phầm mềm dịch ngược APK Tool để dịch ngược các ứng dụng android, thu được dạng mã smali và tiến hành trích xuất các API của từng ứng dụng. Từ dữ liệu đã được trích xuất, nhóm xây dựng nên các ma trận cần thiết bao gồm ma trận app – api, ma trận invoke, ma trận method và ma trận package



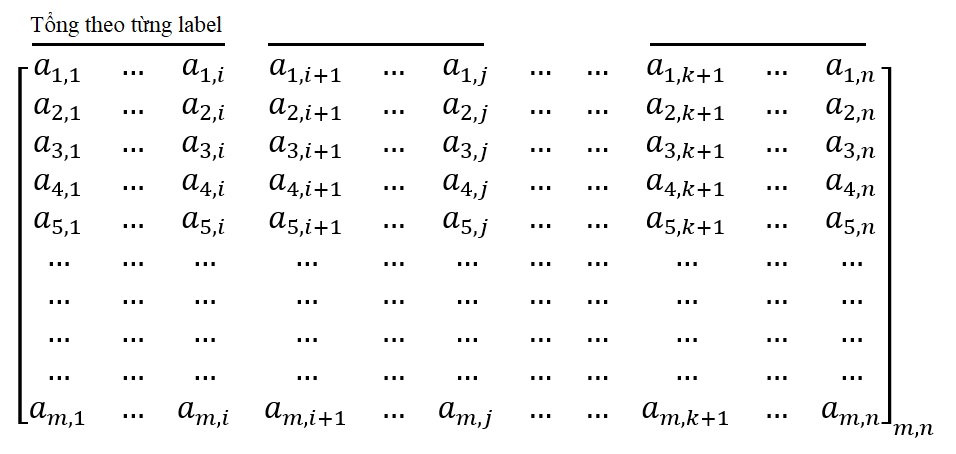
Hình 3.2: Xây dựng các ma trận cơ sở

* **Bước 2:** Xây dựng mạng thông tin không đồng nhất và trích xuất vector đặc trưng. Từ bốn ma trận cơ bản, nhóm xây dựng mười sáu ma trận độ đo tương đồng và trích xuất vector đặc trưng dựa vào các ma trận độ đo trương đồng này.



Hình 3.3: Xây dựng ma trận độ đo tương đồng

* **Bước 3:** Mỗi ma trận độ đo tương đồng sẽ cho ra sáu đặc trưng bằng cách gộp các hàng của ma trận đô đo tương đồng lại theo số lượng từng họ ứng dụng, cùng với mười sáu ma trận độ đo tương đồng kết hợp với sáu họ android ở tập dữ liệu sẽ tạo nên một ma trận vector đặc trưng sẽ có số chiều là sáu mươi. Ma trận vector đặc trưng này sẽ được sử dụng để training model học máy.



Hình 3.4: Mô tả cách tổng hợp ra các đặc trưng từ ma trận độ đo tương đồng

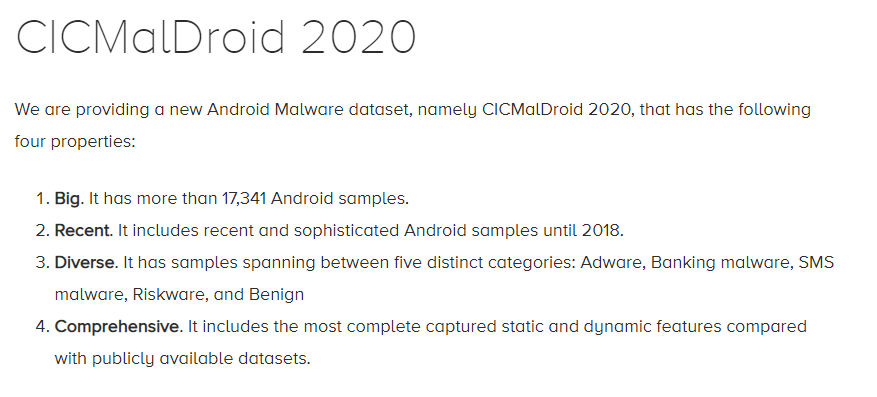
* **Bước 4:** Training mô hình học máy. Từ ma trận feature vector đã trích xuất được, nhóm training các mô hình classificasion khác nhau, với nhiều thông số khác nhau để tìm được mô hình thích hợp nhất, ở đây có 2 thông số quan trọng cần dược để ý đó là thông số recall và thông số persision, nếu persision mà lớn hơn recall thì chứng tỏ tỷ lệ ứng dụng độc bị nhầm thành lành sẽ ít hơn ứng dụng lành thành độc, và từ đó có thể giảm thiểu tác hại của mã độc gây ra

## Chi tiết các bước xây dựng mô hình

### Thu thập dữ liệu và trích xuất thông tin

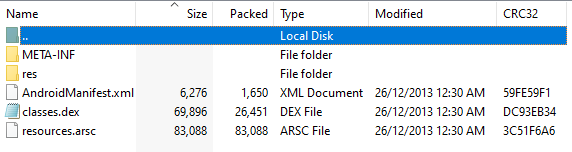
Hiện tại có rất nhiều nguồn dữ liệu dành cho các dự án về mã độc android được công bố trên khắp thế giới như Drebin, CIC Dataset, Android Malware Dataset (AMD), vv…. Trong dự án này, chúng em sử dụng tập dữ liệu của CIC dataset, tổng cộng có 12210 ứng dụng, trong đó bao gồm:

* 1415 ứng dụng độc hại Adware
* 1900 ứng dụng độc hại banking
* 1911ứng dụng lành tính benign
* 2262 ứng dụng độc hại riskware
* 4722 ứng dụng độc hại smsmalware



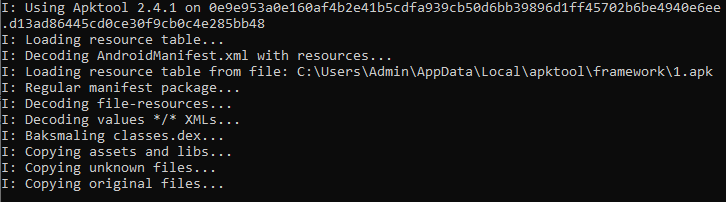
Hình 3.5: Bộ dữ liệu CIC Dataset

File dạng .apk là file cài đặt dưới dạng nén của một phần mềm android, chúng chứa các thông tin như là: file Androidmanifest.xml, file classes.dex, resource.arsc, các folder như là META-INF, res.

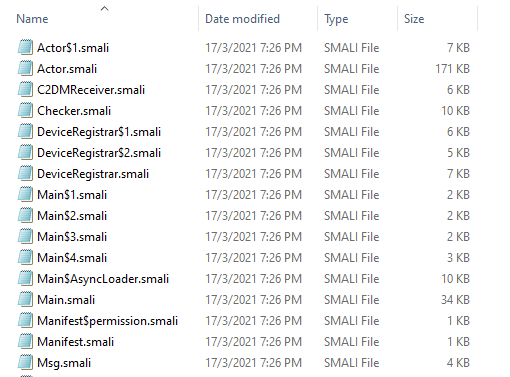


Hình 3.6: Cấu trúc một file apk

Chúng ta không thể xem trực tiếp nội dung file apk mà cần một phần mềm hỗ trợ cho việc này chính là dự án mở Apktool, được phát triển bởi iBotPeaches [11]. Phần mềm Apktool hỗ trợ dịch các file apk thành một hệ thống file smali. File smali có thể được hiểu như là một dạng mã nguồn hợp ngữ của phần mềm android. Khi file apk ở dạng mã smali thì có thể dễ dàng tiến hành đọc thông tin mã nguồn của nó.



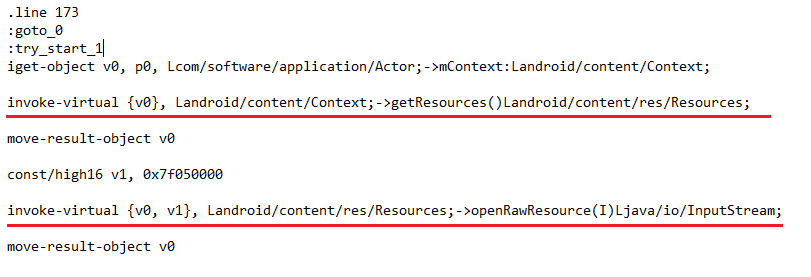
Hình 3.7: Quá trình dịch ngược file apk



Hình 3.8: Các file smali thu được sau khi dịch ngược

Sau khi dịch ngược các file apk, chúng em đi trích xuất các lời gọi API từ các nội dung đã được dịch ngược. Cấu trúc của một lời gọi API trong file smali có dạng bắt đầu bằng từ khóa invoke, tiếp theo đó là thông tin về danh sách tham số, tên lớp, tên lời gọi API, kiểu của danh sách tham số truyền vào và cuối cùng là kiểu trả về của API.

Ví dụ ở hình 3.8 có 2 lời gọi API sẽ được gọi khi ứng dụng thực thi đến đoạn mã này. Hai lời gọi API này có cùng một kiểu invoke là vitrual. Đối với lời gọi thứ nhất, lời gọi này có một tham số là v0, tiếp theo “Landroid/content/Context” chính là tên lớp định nghĩa lời gọi API này, “getReources” là tên của lời gọi API, cuối cùng API này có kiểu trả về là một đối tượng thộc lớp “Landroid/content/res/Resources”. Tương tự, lơi gọi API thứ hai có hai tham số là v0 và v1, có tên lớp là “Landroid/content/res/Resources”, tên lời gọi API là “openRawResource”, kiểu trả về là “Ljava/io/InputStream”.



Hình 3.9: Các lời gọi API trong file smali

Có rất nhiều cách tiếp cận để có thể trích xuất lời gọi API, chẳng hạn như có thể chỉ lọc những hướng dẫn gọi một số chức năng tiêu chuẩn của Android. Tuy nhiên, vì danh sách các lời gọi này có thể khá dài và quan trọng là nó có thể thay đổi so với các phiên bản Android khác nhau nên chúng em chỉ tập trung vào các lời gọi thuộc về không gian tên Android và Java (tức là các lớp bắt đầu bằng Landroid/LJava). Việc trích xuất toàn bộ lời gọi API trên toàn bộ dữ liệu là không khả thi và không có ý nghĩa trong quá trình huấn luyện học máy. Bởi vì, việc trích chọn quá nhiều thông tin để huấn luyện sẽ làm tăng thời gian xử lý, làm sai lệch các kết quả. Hơn nữa, có nhiều ứng dụng lành tính sử dụng các API mà các mã độc không hề sử dụng, lúc đó các API này không mang nhiều ý nghĩa cho việc phân loại. Một số nghiên cứu lại lựa chọn phương án trích xuất và xếp hạng các lời gọi API trên toàn bộ dữ liệu và sau đó lựa chọn theo thứ tự xếp hạng. Phương án này cũng có hạn chế đó là, có thể các phần mềm độc hại sẽ chỉ sử dụng một vài API với tần suất rất thấp, trong khi đó, các API xử lý bình thường như định dạng các chuỗi được sử dụng khá thường xuyên trong mọi ứng dụng nên sẽ có khả năng việc lựa chọn các API theo cách này sẽ bỏ sót các API có trong phần mềm độc hại dẫn tới bỏ sót các phần mềm độc hại khi tiến hành phân loại. Do vậy, giải pháp tốt nhất hiện tại đó chính là lọc các API mà các phần mềm độc hại thường hay sử dụng có trong các mẫu, nghĩa là các API sẽ được trích xuất theo tần suất xuất hiện trong các phần mềm độc hại nhiều nhất. Chúng em giới hạn số lượng API cần trích xuất vào khoảng 500.



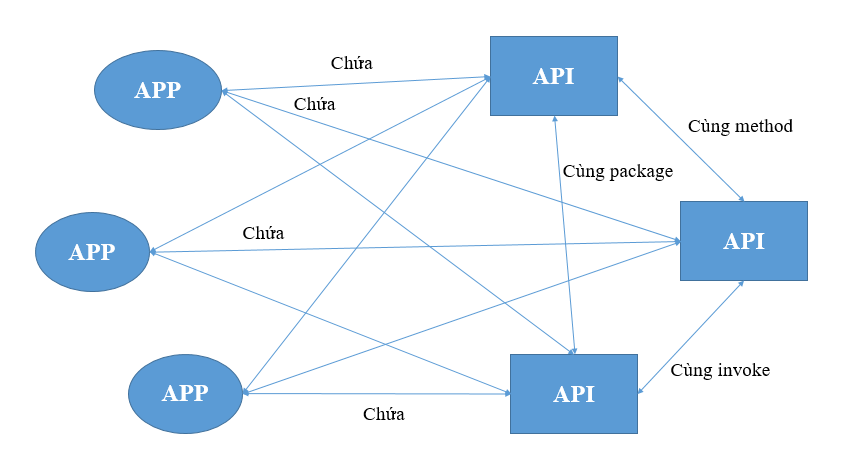
Hình 3.10: Danh sách các lời gọi API

Dựa vào danh sách bao gồm 500 các lời gọi API mà các phần mềm độc hại hay sử dụng nhất, chúng em tiến hành quét trên toàn bộ các file smali của từng ứng dụng. Nếu như ứng dụng nào có sử dụng đến các lời gọi API trong danh sách lời gọi API đã được lọc ra, các thông tin quan trọng như kiểu gọi invoke của API, method nào gọi API đó, và tên package của API sẽ được trích xuất. Những thông tin được trích xuất này là cơ sở để tạo nên bốn ma trận đại diện cho bốn mối quan hệ ở trong mạng thông tin không đồng nhất, đó là:

* Quan hệ giữa APP-API: được biểu diễn bằng ma trận APP-APImn. trong đó m là số lượng app, n là số lượng các API. aij =1 khi Appi có sử dụng lời gọi APIj, ngược lại aij = 0.
* Quan hệ giữa hai API cùng chung lời gọi invoke: được biểu diễn bằng ma trận INVOKEnn, trong đó : aij =1 nếu APIi và APIj có cùng chung phương thức gọi invoke, ngược lại aij =0. Các phương thức invoke bao gồm:
  + Invoke – static: gọi một phương thức tĩnh với các biến số
  + Invoke – virtual: gọi một phương thức ảo với các biến số
  + Invoke – direct: gọi một phương thức với các biến số mà không cần phân giải phương thức ảo
  + Invoke – super: gọi một phương thức ảo của lớp cha ngay liền kề
  + Invoke - interface : gọi một phương thức thuộc giao diện.
* Quan hệ giữa hai API cùng chung method: được biểu diễn bằng ma trận METHODnn, trong đó aij =1 nếu APIi và APIj có cùng chung một method.
* Quan hệ giữa hai API cùng chung package: được biểu diễn bằng ma trận PACKAGEnn, trong đó aij =1 nếu APIi và APIj có cùng lời gọi đến một package.

### Xây dựng ma trận độ đo tương đồng

Từ bốn ma trận thể hiện các mối quan hệ trong mạng thông tin không đồng nhất, chúng em sử dụng công thức tính độ đo tương đồng AvgSim theo các siêu đường đi để đo sự tương tự giữa một Appi đến Appj bất kỳ. Một siêu đường đi trên mạng thông tin không đồng nhất từ một đỉnh s đến một đỉnh t thể hiện mối quan hệ giữa s và t.



Hình 3.11: Mạng thông tin không đồng nhất

Công thức AvgSim dùng cho đỉnh bất kỳ:

Trong đó :

Công thức AvgSim dùng cho ma trận các mối quan hệ

Trong đó : là ma trận chuẩn hóa theo hàng, là ma trận chuẩn hóa theo cột của ma trận , ma trận là ma trận thể hiện mối quan hệ giữa đỉnh và đỉnh trong đồ thị.

Khi thực hiện xử lý dữ liệu, việc tính toán số lượng siêu đường đi từ một App đến một App khác trong mạng thông tin không đồng nhất khá mất thời gian, vì đây là bài toán duyệt có điều kiện trên đồ thị có số đỉnh lớn. Ngoài ra, một số nghiên cứu cho thấy, trong mạng thông tin không đồng nhất thì các siêu đường đi có độ dài lớn thường không đem lại nhiều thông tin để phân biệt mối quan hệ giữa các đỉnh trong mạng. Vì vậy trong nghiên cứu này, chúng em lựa chọn các đường đi đối xứng, độ dài tối đa của một siêu đường đi là 7 và số lượng siêu đường đi là 16.

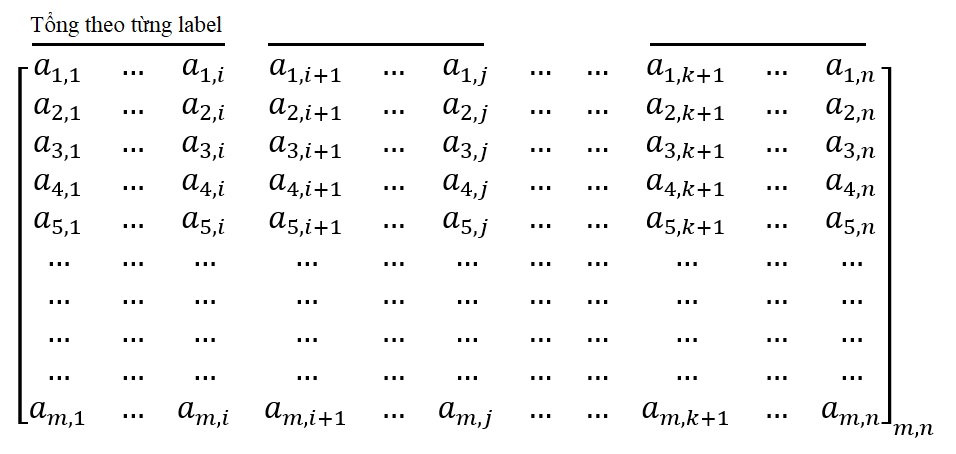
|  |  |  |
| --- | --- | --- |
| STT | Siêu đường | Mô Tả |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Bảng 3.1: Mười sáu siêu đường đi được sử dụng trong mô hình

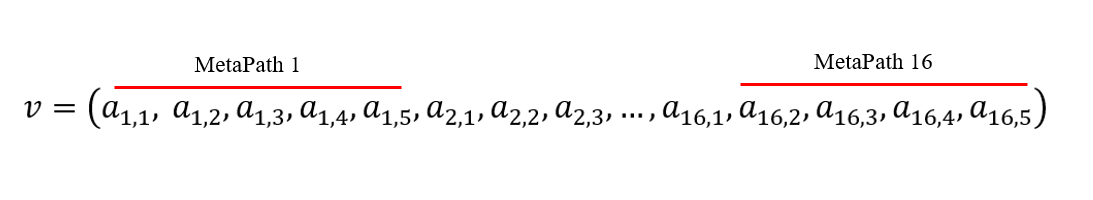
Ứng với 16 siêu đường đi, ta áp dụng công thức AvgSim cho ma trận các mối quan hệ, kết quả thu được 16 ma trận độ đo tương đồng Mk trong đó có các phần tử aij thể hiện sự tương đồng giữa APPi và APPj , như vậy hàng thứ i của ma trận Mk sẽ thể hiện độ tương đồng của ứng APPi so với tất cả các APP còn lại trong tập dữ liệu.

### Trích xuất vector đặc trưng

Sau khi tạo ra được các ma trận độ đo tương đồng, chúng em tiến hành trích xuất các vector đặc trưng để đưa vào mô hình học máy. Có thể thấy, từng hàng ở trong ma trận vector đặc trưng thể hiện sự tương đồng của ứng dụng với tất cả các ứng dụng còn lại, như vậy nếu như ta cộng tổng theo từng họ ứng dụng thì sẽ ra được một đăc trưng thể hiện sự tương đồng với họ ứng dụng đó. ở đây, tập dữ liệu của chúng em có 5 họ bao gồm andware, banking, benign, riskware và smsmalware, vậy từng hàng sẽ được chia ra 5 đặc trưng ứng với từng họ ứng dụng. với 16 ma trận độ đo tương đồng thì mỗi vector sẽ có 80 đặc trưng.



Hình 3.12: Tổng theo từng họ ứng dụng



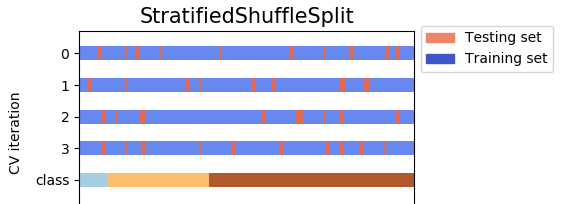
Hình 3.13: Tổng hợp vector đặc trưng từ ma trận độ đo tương đồng

Ma trận vector đặc trưng sẽ được đưa vào mô hình học máy. Ở đây, chúng em sẽ sử dụng các mô hình học máy như mô hình Decision Tree, Random Forest, Support Vector Machines để thực hiện huấn luyện và đánh giá kết quả.

## Huấn luyện học máy và đánh giá mô hình

### Phương pháp đánh giá

Để đánh giá hiệu quả của các mô hình phân lớp, chúng em sử dụng kỹ thuật Stratified Shuffle Split để phân chia dữ liệu cho các thực nghiệm. Kỹ thuật Stratified Shuffle Split (SSS) thực hiện việc xáo trộn bộ dữ liệu và sau đó chia thành n phần gọi là n\_splits. Sau đó, SSS chọn một phần để sử dụng làm bộ kiểm tra, n-1 phần còn lại sẽ được chia tiếp để có được n\_1 bộ kiểm tra ngẫu nhiên. Đặc điểm của kỹ thuật này là luôn xáo trộn dữ liệu trước mỗi lần chia tách, việc này đảm bảo cho các bộ kiểm tra giao nhau khác rỗng.



Hình 3.14: Stratified Shuffle Split

Với mỗi lần huấn luyện và kiểm thử một mô hình phân lớp, chúng em tính toán các độ đo sau đây:

* Precision = TP/(TP + FP)
* Recall = TP/(TP + FN)
* Accuracy = (TP + TN)/(TP + TN + FP + FN)
* F1 = 2 \* Precision \* Recall / (Precision + Recall)

Trong đó, TP, TN, FP, FN lần lượt là số lượng ứng dụng độc hại được phân lớp đúng; số lượng ứng dụng lành tính được phân lớp đúng; số lượng ứng dụng lành tính bị phân lớp sai; số lượng ứng dụng độc hại bị phân lớp sai.

### Thực nghiệm và đánh giá

Chúng em thực hiện 2 thực nghiệm dưới những điều kiện sau:

Bộ huấn luyện - kiểm định (train - validation):

* 1415 ứng dụng độc hại Adware
* 1900 ứng dụng độc hại banking
* 1911ứng dụng lành tính benign
* 2262 ứng dụng độc hại riskware
* 4722 ứng dụng độc hại smsmalware

Bộ dữ liệu luyện - kiểm định được chia làm 10 phần, 9 phần dùng để huấn luyện và 1 phần dùng để kiểm định, dựa trên kỹ thuật SSS.

Bộ dữ liệu test, bộ dữ liệu này được sử dụng nằm ngoài tập dữ liệu train – validation, và nó sẽ chứng minh khả năng hoạt động thực tế của mô hình có chính xác không, bộ dữ liệu này chúng em sử dụng 500 ứng dụng và có đủ 5 họ ứng dụng.

Kết quả thử ngiệm được ghi dưới bảng sau:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| KneighborsClassifier | | | | |
| Tham Số | | Model: n\_neightboers = 5 | | |
| Average = “macro” | | |
| Lần Lặp | Accuracy | F1 | Recall | Precision |
| 1 | 0.9197 | 0.9198 | 0.9197 | 0.9202 |
| 2 | 0.9183 | 0.9284 | 0.9183 | 0.9192 |
| 3 | 0.9203 | 0.9233 | 0.9230 | 0.9245 |
| 4 | 0.9227 | 0.9225 | 0.9227 | 0.9229 |
| 5 | 0.9191 | 0.9193 | 0.9191 | 0.9196 |
| 6 | 0.9189 | 0.9188 | 0.9189 | 0.9189 |
| 7 | 0.9202 | 0.9203 | 0.9202 | 0.9209 |
| 8 | 0.9170 | 0.9171 | 0.9170 | 0.9178 |
| 9 | 0.9025 | 0.9206 | 0.9205 | 0.9211 |
| 10 | 0.9164 | 0.9165 | 0.9164 | 0.9169 |

Bảng 3.2: Train – validate Kneighbors(n\_neighbors = 5)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| KneighborsClassifier | | | | |
| Tham Số | | Model: n\_neightboers = 4 | | |
| Average = “macro” | | |
| Lần Lặp | Accuracy | F1 | Recall | Precision |
| 1 | 0.9191 | 0.8995 | 0.9023 | 0.8977 |
| 2 | 0.9208 | 0.9026 | 0.9068 | 0.8996 |
| 3 | 0.9249 | 0.9065 | 0.9100 | 0.9042 |
| 4 | 0.9167 | 0.8968 | 0.9002 | 0.8945 |
| 5 | 0.9191 | 0.9022 | 0.9089 | 0.8973 |
| 6 | 0.9224 | 0.9055 | 0.9086 | 0.9031 |
| 7 | 0.9243 | 0.9076 | 0.9124 | 0.9039 |
| 8 | 0.9159 | 0.8978 | 0.9020 | 0.8946 |
| 9 | 0.9172 | 0.8982 | 0.9005 | 0.8965 |
| 10 | 0.9301 | 0.9142 | 0.9166 | 0.9124 |

Bảng 3.3: Train – validate Kneighbors(n\_neighbors = 4)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| KneighborsClassifier | | | | |
| Tham Số | | Model: n\_neightboers = 6 | | |
| Average = “macro” | | |
| Lần Lặp | Accuracy | F1 | Recall | Precision |
| 1 | 0.9178 | 0.8996 | 0.9014 | 0.8983 |
| 2 | 0.9115 | 0.8927 | 0.8962 | 0.8903 |
| 3 | 0.9189 | 0.9009 | 0.9013 | 0.9011 |
| 4 | 0.9180 | 0.9011 | 0.9039 | 0.8994 |
| 5 | 0.9156 | 0.8979 | 0.9024 | 0.8943 |
| 6 | 0.9131 | 0.8921 | 0.8905 | 0.8899 |
| 7 | 0.9169 | 0.8977 | 0.8998 | 0.8962 |
| 8 | 0.9191 | 0.9007 | 0.9036 | 0.8928 |
| 9 | 0.9161 | 0.8992 | 0.9023 | 0.8969 |
| 10 | 0.9161 | 0.8980 | 0.9011 | 0.8955 |

Bảng 3.4: Train – validate Kneighbors(n\_neighbors = 6)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| RandomForestClassifier | | | | |
| Tham số | | N\_estimator = 100  Max\_deep = 10 | | |
| Average = “macro” | | |
| Lần lặp | Accuracy | F1 | Recall | Precision |
| 1 | 0.9339 | 0.9190 | 0.9223 | 0.9166 |
| 2 | 0.9328 | 0.9164 | 0.9194 | 0.9146 |
| 3 | 0.9322 | 0.9153 | 0.9174 | 0.9138 |
| 4 | 0.9331 | 0.9169 | 0.9194 | 0.9152 |
| 5 | 0.9301 | 0.9135 | 0.9165 | 0.9112 |
| 6 | 0.9287 | 0.9112 | 0.9137 | 0.9092 |
| 7 | 0.9344 | 0.9192 | 0.9235 | 0.9162 |
| 8 | 0.9238 | 0.9048 | 0.9082 | 0.9246 |
| 9 | 0.9282 | 0.9105 | 0.9126 | 0.9091 |
| 10 | 0.9336 | 0.9178 | 0.9207 | 0.9157 |

Bảng 3.5: Train – validate RandomForest(n\_estimator = 100, max\_deep = 10)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| RandomForestClassifier | | | | |
| Tham số | | N\_estimator = 100  Max\_deep = 12 | | |
| Average = “macro” | | |
| Lần Lặp | Accuracy | F1 | Recall | Precision |
| 1 | 0.9363 | 0.9217 | 0.9228 | 0.9213 |
| 2 | 0.9347 | 0.9190 | 0.9203 | 0.9183 |
| 3 | 0.9399 | 0.9254 | 0.9266 | 0.9245 |
| 4 | 0.9451 | 0.9215 | 0.9347 | 0.9290 |
| 5 | 0.9434 | 0.9295 | 0.9308 | 0.9285 |
| 6 | 0.9369 | 0.9212 | 0.9228 | 0.9200 |
| 7 | 0.9328 | 0.9166 | 0.9190 | 0.9147 |
| 8 | 0.9374 | 0.9234 | 0.9249 | 0.9223 |
| 9 | 0.9363 | 0.9221 | 0.9235 | 0.9211 |
| 10 | 0.9303 | 0.9133 | 0.9150 | 0.9123 |

Bảng 3.6: Train – validate RandomForest(n\_enstimator = 100, max\_deep = 12)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| RandomForestClassifier | | | | |
| Tham Số | | N\_estimator = 100  Max\_deep = 14 | | |
| Average = “macro” | | |
| Lần lặp | Accuracy | F1 | Recall | Precision |
| 1 | 0.9327 | 0.9263 | 0.9266 | 0.9264 |
| 2 | 0.9367 | 0.9223 | 0.9236 | 0.9214 |
| 3 | 0.9397 | 0.9260 | 0.9259 | 0.9263 |
| 4 | 0.9358 | 0.9813 | 0.9233 | 0.9199 |
| 5 | 0.9413 | 0.9277 | 0.9296 | 0.9265 |
| 6 | 0.9394 | 0.9252 | 0.9271 | 0.9236 |
| 7 | 0.9350 | 0.9210 | 0.9236 | 0.9194 |
| 8 | 0.9378 | 0.9234 | 0.9257 | 0.9234 |
| 9 | 0.9369 | 0.9218 | 0.9218 | 0.9225 |
| 10 | 0.9372 | 0.9220 | 0.9232 | 0.9211 |

Bảng 3.7: Train – validate RandomForest(n\_estimator = 100, max\_deep = 14)

Như vậy có thể nhận thấy được đối với việc huấn luyện training – validation thì mô hình RandomForest cho kết quả cao hơn dùng Kneighbors.

Tiếp theo chúng xem đi kiểm tra trên thực tế thì độ chính xác có cao không, và bọn em đã thử trên 500 ứng dụng nằm ngoài tập dữ liệu, kết quả được biểu thị dưới các bảng dưới đây.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| KneighborsClassifier | | | | |
| Tham số | | n\_neighbors = 4 | | |
| Average = “macro” | | |
| Accuracy | F1 | | Recall | Precision |
| 0.8320 | 0.8246 | | 0.8320 | 0.8263 |

Bảng 3.8: Test – Kneighbors(n\_neighbors = 4)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| KneighborsClassifier | | | | |
| Tham số | | n\_neighbors = 5 | | |
| Average = “macro” | | |
| Accuracy | F1 | | Recall | Precision |
| 0.8280 | 0.8220 | | 0.8280 | 0.8230 |

Bảng 3.9: Test – Kneighbors(n\_neighbors = 5)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| KneighborsClassifier | | | | |
| Tham số | | n\_neighbors = 6 | | |
| Average = “macro” | | |
| Accuracy | F1 | | Recall | Precision |
| 0.8340 | 0.8279 | | 0.8340 | 0.8289 |

Bảng 3.10: Test – Kneighbors(n\_neighbors = 6)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| RandomForestClassifier | | | | |
| Tham số | | N\_estimator = 100  Max\_deep = 10 | | |
| Average = “macro” | | |
| Accuracy | F1 | | Recall | Precision |
| 0.8020 | 0.7953 | | 0.8020 | 0.7948 |

Bảng 3.11: Test – RandomForest(n\_esimator = 100, max\_deep = 10)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| RandomForestClassifier | | | | |
| Tham số | | N\_estimator = 100  Max\_deep = 12 | | |
| Average = “macro” | | |
| Accuracy | F1 | | Recall | Precision |
| 0.8060 | 0.8043 | | 0.8060 | 0.8045 |

Bảng 3.12: Test – RandomForest(n\_estimator = 100, max\_deep = 12)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| RandomForestClassifier | | | | |
| Tham số | | N\_estimator = 100  Max\_deep = 14 | | |
| Average = “macro” | | |
| Accuracy | F1 | | Recall | Precision |
| 0.8200 | 0.8138 | | 0.8200 | 0.8184 |

Bảng 3.13: Test – RandomForest(n\_estimator = 100, max\_deep = 14)

Tuy vậy, trên thực tế ta thấy mô hình có sự chính xác cao nhất là KneightborsClassifier với tham số n\_neighbores = 6 đạt độ chính xác cao nhất, do đó ta sẽ lựa chọn mô hình này để xây dựng lên hệ thống phát hiện mã độc.

## Xây dựng hệ thống

### Tổng quan vệ hệ thống

Từ những kết quả đã đạt được, chúng em đi xây dựng hệ thống phát hiện phần mềm độc hại thông minh dựa trên mạng thông tin không đồng nhất. Trong dó sử dụng mô hình để đoán nhận đó là KneighborsClassifier. Hệ thống bao gồm 2 thành phần là server và máy client, máy client ở đây là một ứng dụng andoroid giúp cho người dùng có thể dễ dàng thực hiện việc phát hiện phần mềm độc hại.

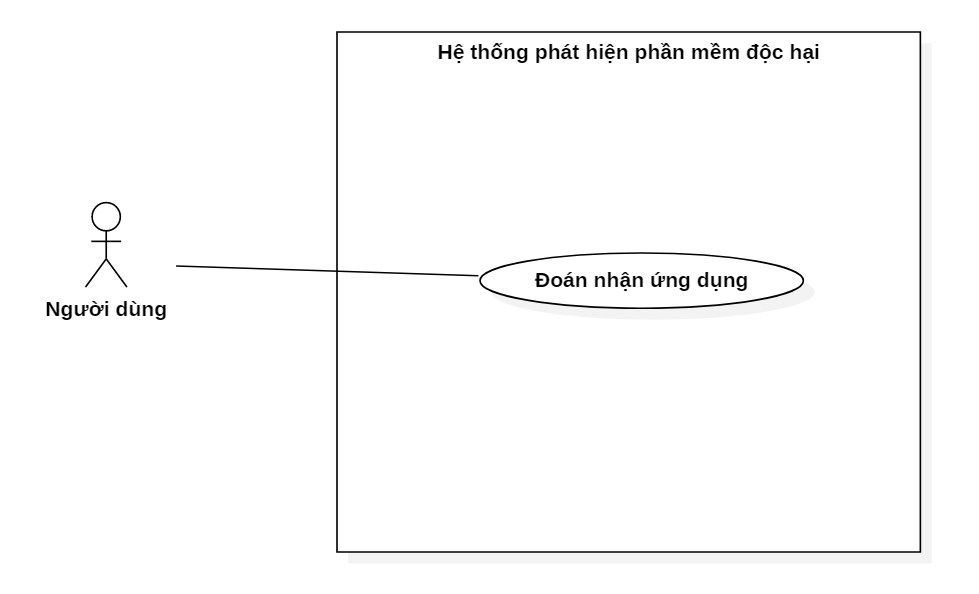
### Các yêu cầu chức năng và phi chức năng của hệ thống

Chúng em đưa ra các yêu cầu thiết kế như sau:

* Yêu cầu chức năng:
  + Đoán nhận ứng dụng mà người dùng gửi lên, xem nó có phải là ứng dụng độc hại hay không.
* Yêu cầu phi chức năng:
  + Giao diện hệ thống đơn giản dễ hiểu, hễ sử dụng.
  + Hệ thống cần phải phản hồi lại kết quả sau trung bình 30 giây.
  + Hệ thống cần phải hoạt động tốt trên các phiên bản andoird khác nhau, cũng nhưng các thiết bị andoird khác nhau.

### Phân tích hệ thống

Từ những yêu cầu đã phân tích, chúng em đi xây dựng mô hình ca sử dụng:

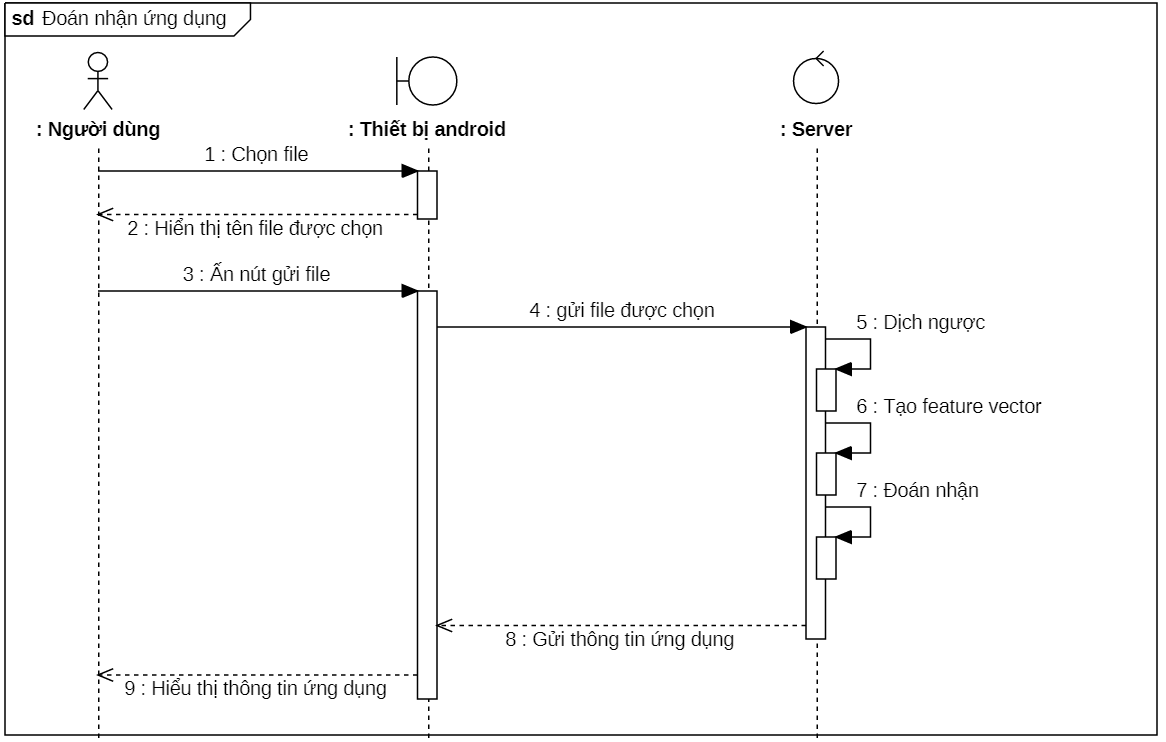


Hình 3.15: Mô hình ca sử dụng của hệ thống

Đặc tả ca sử dụng Đoán nhận ứng dụng:

|  |  |
| --- | --- |
| Ca sử dụng | Đoán nhận ứng dụng |
| Tác nhân | Người dùng |
| Mô tả | Hệ thống đoán nhận ứng dụng |
| Luồng cơ bản | 1. Người dùng ấn vào nút chọn file ở trên giao diện thiết bị andorid. 2. Người dùng tìm vào chọn file muốn đoán nhận. 3. Thiết bị người dùng gửi file muốn đoán nhận lên server. 4. Hệ thống nhận file. 5. Hệ thống tiến hành dịch ngược file cần đoán nhận. 6. Hệ thống trích xuất vector của ứng dụng cần được đoán nhận. 7. Hệ thống đoán nhận ứng dụng. 8. Hệ thống trả về kết quả đoán nhận cho thiết bị andoird 9. Thiết bị android của người dùng hiển thị thông thi được trả về |
| Luồng phụ |  |

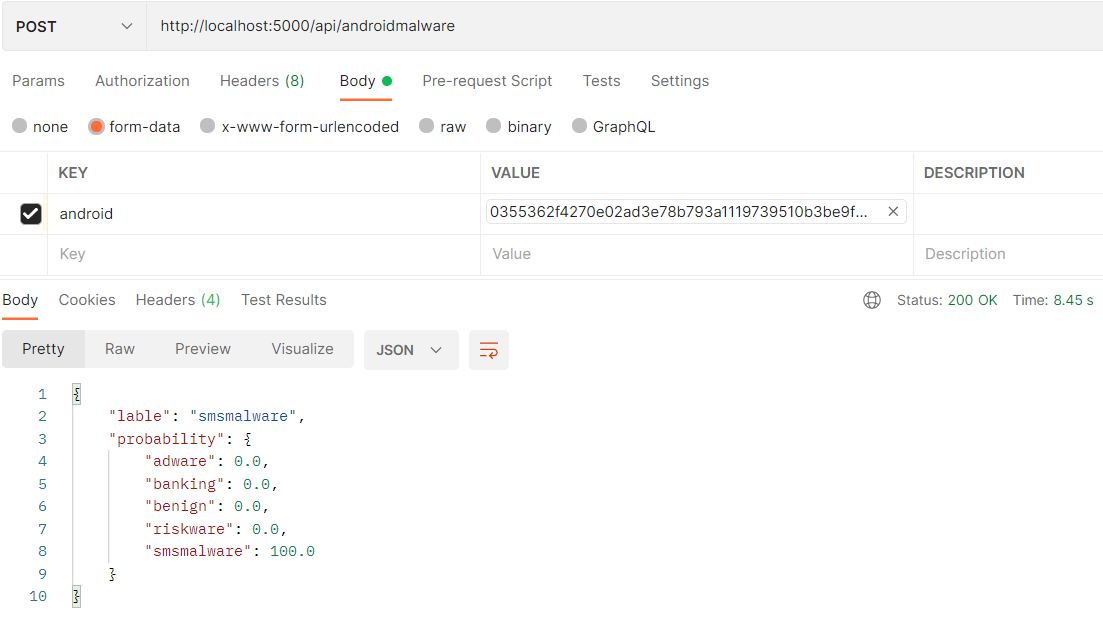
Phân tích ca sử dụng Đoán nhận ứng dụng



Hình 3.16: Biều đồ tuần tự ca sử dụng đoán nhận ứng dụng

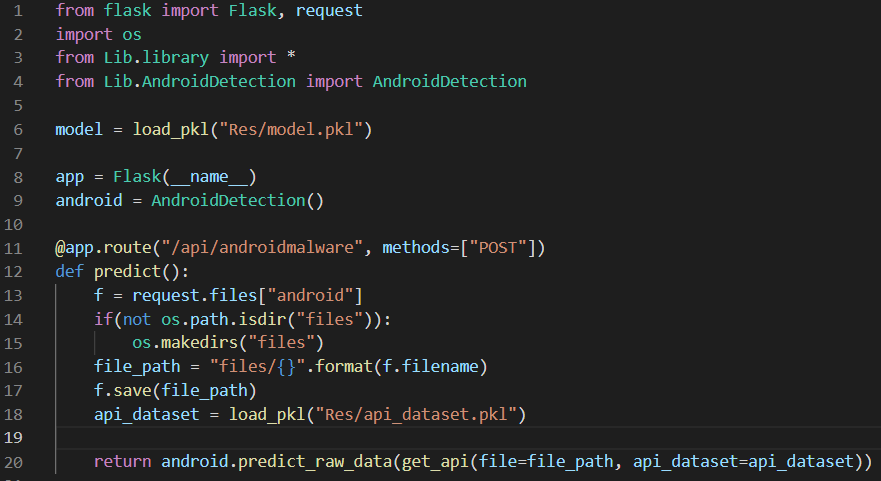
### Triển khai xây dựng hệ thống

Hệ thống server được lựa chọn xây dựng trên nền tảng flask python, nó có ưu điểm là dễ triển khai, cộng theo việc nó đồng bộ với hệ thống đoán nhận ứng dụng, server được xây dựng có một API giúp cho các ứng dụng có thể giao tiếp đoán nhận ứng dụng andorid, api có dạng form data với key là “andorid” và value là 1 file ứng dụng andoid. Dạng trả về của API bao gồm loại ứng dụng là gì, kèm theo xác xuất rơi vào ừng lớp của ứng dụng.



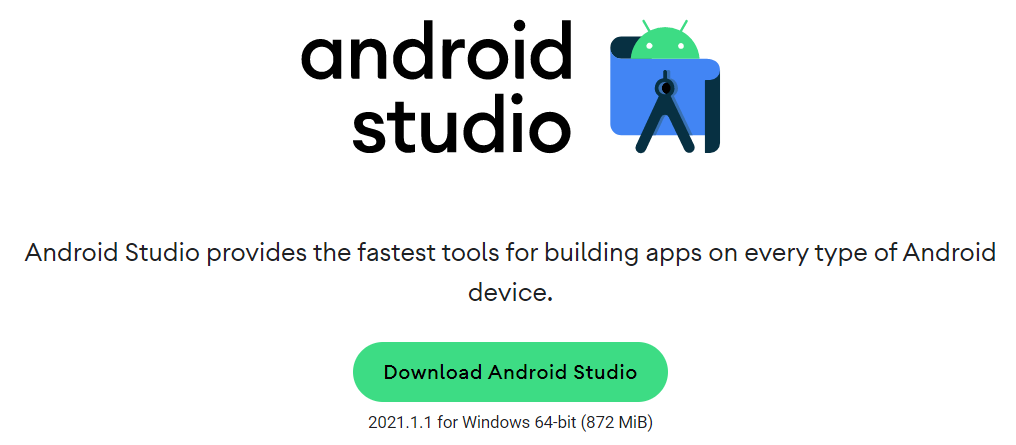
Hình 3.17: Test API bằng posman

Để tạo được API này thì python flask hỗ trợ các hàm kèm theo các anotation để thiết lập các API.



Hình 3.18: Xây dựng API bằng python flask

Ứng dụng andoird client được xây dựng đựa trên AndoirdSDK, sử dụng một IDE của google là Andoid Studio, IDE này hỗ trợ rất nhiều trong việc phát triển phần mềm android như là Máy ảo, Code Editer, các Extension, …



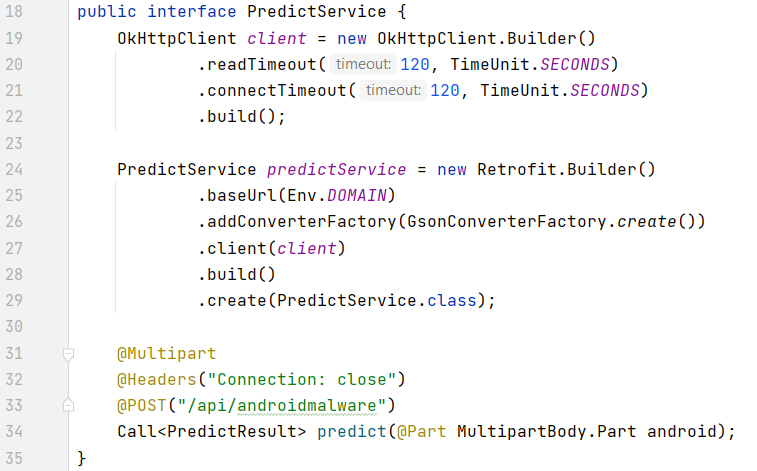
Hình 3.19: Trang chủ Android Studio

Ngôn ngữ được lựa chọn để xây dựng ứng dụng là java, kèm theo đó là việc sử dụng các thư viện đi kèm hỗ trợ cho việc xây dựng ứng dụng đơn giản hơn, có thể kể đến như là gson hỗ trợ convert các object dạng json sang java object, thư viện retrofit hỗ trợ ứng dụng thao tác với các API



Hình 3.20: Các dependency được sử dụng

Để có thể call API tới server, ta sử dụng Retrofit, thư viện này hỗ trợ rất nhiều tiện ích để thao tác với API, ví dụ như là gán header, tuỳ chỉnh các thông số.



Hình 3.21: Retrofit library

Giao diện của ứng dụng bao gồm một nút chọn file và một nút gửi file, người dùng sử dụng ứng dụng bằng cách ấn vào nút choose file và chọn file cần đoán nhận, sau đó ấn gửi file và đợ kết quả trả về

|  |  |
| --- | --- |
| Hình 3.22: Giao diện chính | Hình 3.23: Giao diện chọn file |
| Hình 3.24: Sau khi chọn file | Hình 3.25: Kết quả đoán nhận |

## Chạy thử ứng dụng và hệ thống

Chúng em tiến hành chạy thử hai mươi lăm ứng dụng trên thiết bị giả lập, trong đó bao gồm năm ứng dụng andware, năm ứng dụng banking, năm ứng dụng benign, năm ứng dụng riskware và năm ứng dụng smsmalware.

Kết quả thu được như sau:

* Chạy đoán nhận năm ứng dụng adware thì có bốn ứng dụng đoán đúng, một ứng dụng bị đoán thành banking
* Chạy đoán nhận năm ứng dụng banking thì có bốn ứng dụng đoán đúng, một ứng dụng bị đoán sai thành riskware
* Chạy đoán nhận năm ứng dụng benign thì có năm ứng dụng đoán đúng
* Chạy đoán nhận năm ứng dụng riskware thì có bốn ứng dụng đoán đúng, một ứng dụng bị đoán nhầm thành adware
* Chạy đoán nhận năm ứng dụng smsmalware thì có năm ứng dụng đoán đúng

Kết luận vậy chạy tổng số hai mươi lăm ứng dụng thì có ba ứng dụng bị đoán sai, còn lại là đoán đúng, trong đó không có trường hợp ứng dụng độc hại nào bị nhầm thành lành tính, sự sai sót liên quan đến việc nhầm lẫn giữa các họ mã độc với nhau. Điều này được đánh giá là sẽ giảm thiểu thiệt hại của ứng dụng độc hại với người dùng thiết bị android.

# KẾT LUẬN

## Các vấn đề đã làm được

* Hiểu được cấu trúc của .apk và sử dụng công cụ để hỗ trợ giải nén và dịch ngược các file apk thành hệ thống file smali.
* Trích xuất thông tin từ file smail và xây dựng nên các ma trận kề, biểu thị các mối quan hệ giữa App – API, API – API.
* Xây dựng mô hình mạng thông tin không đồng nhất và tính độ đo tương đồng giữa các App dựa trên công thức AvgSim.
* Xây dựng công thức trích xuất đặc trưng để biểu diễn các ứng dụng thành các vector đặc trưng.
* Huấn luyện học máy và đánh giá các mô hình. Trên cơ sở đó lựa chọn các mô hình hiệu quả để xây dựng hệ thống phát hiện phần mềm độc hại Android.
* Xây dựng hệ thống client server có client là ứng dụng andoird.

## Các vấn đề còn tồn đọng của hệ thống

* Thời gian đoán nhận một ứng dụng còn mất nhiều thời gian.
* Độ chính xác chưa đạt tuyệt đối

THAM KHẢO

[1]. M. Nilay, P. Nitin, *“Review of Behavior Malware Analysis for Android” International Journal of Engineering and Innovative Technology”*, vol 2, pp.230 – 235, 2013.

[2]. Shifu Hou, Yanfang Ye, Yangqiu Song, Melih Abdulhayoglu, *“HinDroid: An Intelligent Android Malware Detection System Based on Structured Heterogeneous Information Network”.*

[3]. Ding Xiao, X. Meng, *“AVGSIM: Relevance measurement on massive data in heterogeneous networks”.*

[4]. Shifu Hou, Yanfang Ye et al., *“AiDroid: When heterogeneous information network marries deep neural network for real-time Android malware detection”.*

[5]. Shifu Hou, Yanfang Ye et al., *“Disentangled Representation Learning in Heterogeneous Information Network for Large-scale Android Malware Detection in the COVID-19 Era and Beyond”.*

[6]. Shifu Hou, Yanfang Ye et al., *“Out-of-sample Node Representation Learning for Heterogeneous Graph in Real-time Android Malware Detection”.*

[7]. Rongye Liu, Ning Yang, Xiangqian Ding, Lintao Ma, *“An Unsupervised Feature Selection Algorithm: Laplacian Score Combined with Distance-Based Entropy Measure”.*

[8]. Jaemin Jung, Hyunjin Kim, Dongjin Shin, Myeonggeon Lee, Hyunjae Lee, Seong-je Cho, Kyoungwon Suh. *“Android Malware Detection base on Usefull API Calls and Machine Learning”, 2018.*