Trường Đại học Công nghệ Thông tin – ĐHQG Tp. HCM
Khoa Mạng máy tính và truyền thông

BÁO CÁO TỔNG KẾT ĐỒ ÁN MÔN HỌC

Môn học: Cơ chế hoạt động của mã độc

Tên chủ đề: Ransomware Detection and Prevention

Mã nhóm: G05 Mã đề tài: S13

Lớp: NT230.P21.ANTN

1. THÔNG TIN THÀNH VIÊN NHÓM:

(Sinh viên liệt kê tất cả các thành viên trong nhóm)

STT	Họ và tên	MSSV	Email
1	Trần Vỹ Khang	25520628	25520628@gm.uit.edu.vn
2	Hồ Hoàng Diệp	22520249	22520249@gm.uit.edu.vn
3	Nguyễn Đặng Nguyên Khang	22520617	22520617@gm.uit.edu.vn

2. TÓM TẮT NỘI DUNG THỰC HIỆN:[1]

A. Chủ đề nghiên cứu trong lĩnh vực Mã độc: *(chọn nội dung tương ứng bên dưới)*

☐ Dev Track

☑ Research Track

B. Tên đề tài

Ransomware Detection and Prevention

Explainable deep learning-based ransomware detection using dynamic analysis

C. Liên kết lưu trữ mã nguồn của nhóm:

Mã nguồn của đề tài đồ án được lưu tại:
https://github.com/Diephho/Explainable-deep-learning-based-ransomware-detection-using-dynamic-analysis

(Lưu ý: GV phụ trách phải có quyền truy cập nội dung trong Link)

[1] Ghi nội dung tương ứng theo mô tả

D. Tên tài liệu tham khảo chính:

[1] A. A. Aldwairi and N. A. Alsabih, "XRan: Explainable deep learning-based ransomware detection using dynamic analysis," *Computers & Security*, vol. 123, p. 102930, May 2023. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S016740482400004X

E. Tóm tắt nội dung chính:

Dự án của nhóm nhằm phát hiện ransomware dựa trên phân tích động và mô hình học sâu (2L-CNN). Phương pháp bắt đầu bằng việc thực thi ransomware và các mẫu benign/malware trong môi trường Cuckoo Sandbox, từ đó thu thập báo cáo hành vi dưới dạng JSON. Các chuỗi hành vi như các gọi API, DLL và mutex được trích xuất, ghép nối lại thành một vector duy nhất và đưa vào mô hình học sâu CNN một chiều với hai lớp để phân loại nhị phân (ransomware vs benign hoặc malware). Mô hình được huấn luyện với phương pháp sparse categorical cross-entropy, sử dụng validation split để tránh overfitting. Các chỉ số đánh giá bao gồm accuracy, TPR, FPR, và F1-score.

Các thí nghiệm được thực hiện trên một máy với Intel Core i7-11390H, 16GB RAM, chạy máy ảo Windows 7 trong Cuckoo Sandbox. Nghiên cứu tập trung vào hai tác vụ phân loại: Ransomware vs. Benign và Ransomware vs. Malware. Kết quả cho thấy mô hình 2L-CNN đạt độ chính xác cao nhất (99.10%) và TPR (98.04%), không có dương tính giả (FPR = 0.00%) và F-Score là 0.9901, vượt trội so với các mô hình truyền thống như Decision Tree, Random Forest và CNN chuẩn.

Để tăng cường tính minh bạch, nhóm sử dụng LIME để giải thích cục bộ và SHAP để đánh giá sự quan trọng của các đặc trưng toàn cục. Kết quả thực nghiệm cho thấy mô hình 2L-CNN có hiệu quả vượt trội trong việc phát hiện ransomware, đồng thời giải thích rõ ràng các quyết định của mô hình.

F. Tóm tắt các kỹ thuật chính được mô tả sử dụng trong đề tài:

1.Phân tích động (Dynamic Analysis):

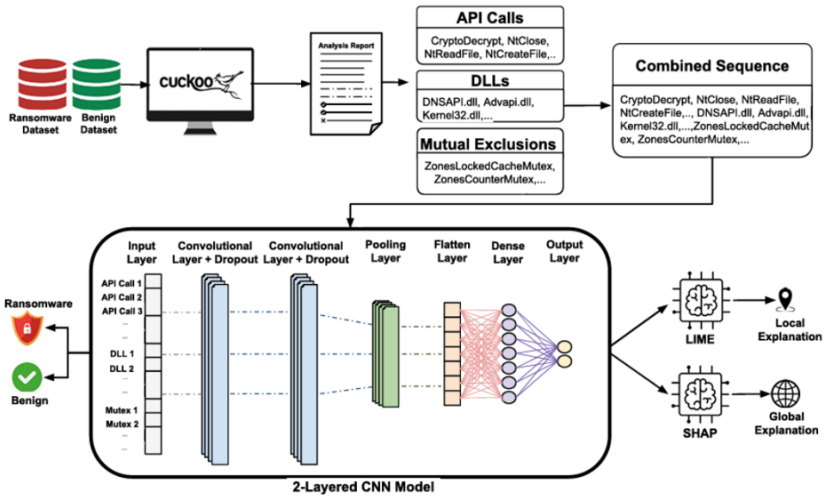
Giám sát hành vi phần mềm trong quá trình thực thi, cung cấp thông tin về các chuỗi gọi API, DLL và mutex—là những chỉ số quan trọng để phát hiện ransomware.

2.Mô hình học sâu CNN hai lớp (2L-CNN):

Phân loại ransomware dựa trên chuỗi đặc trưng (API calls, DLLs, Mutexes) trích xuất từ báo cáo Cuckoo. Mô hình này có thể học được các đặc trưng phức tạp và phân loại nhị phân với độ chính xác cao.

3.Giải thích mô hình với XAI (LIME và SHAP):

LIME và SHAP giúp tăng cường tính minh bạch của mô hình. LIME giải thích cục bộ quyết định của mô hình đối với từng mẫu cụ thể, trong khi SHAP cung cấp cái nhìn tổng thể về sự quan trọng của các đặc trưng trong dự đoán



G. Môi trường thực nghiệm của đề tài:

1.Cấu hình máy tính:
Máy tính sử dụng cấu hình Intel Core i7-11390H, 3.4GHz, 16GB RAM, hỗ trợ tốt cho quá trình thu thập và xử lý dữ liệu trong môi trường máy ảo.

2.Các công cụ hỗ trợ sẵn có:

Cuckoo Sandbox: phân tích động thực thi ransomware và benign/malware trên Windows 7, thu thập các báo cáo JSON về hành vi trong thời gian thực (API calls, DLLs, Mutexes).

LIME và SHAP: Cung cấp giải thích cục bộ và toàn cục cho các quyết định của mô hình học sâu.

Ngôn ngữ lập trình để hiện thực phương pháp:
Python được sử dụng để xây dựng mô hình học sâu 2L-CNN, trích xuất đặc trưng từ báo cáo Cuckoo, và triển khai các thuật toán giải thích mô hình (LIME, SHAP).
Đối tượng nghiên cứu:
Tập dữ liệu bao gồm các mẫu ransomware, benign, và malware được thu thập từ các nguồn như MarauderMap, AnyRun, theZoo và SOREL-20M.

Tiêu chí đánh giá tính hiệu quả của phương pháp:

1.Độ chính xác (Accuracy): Phản ánh khả năng phân loại chính xác giữa ransomware và benign/malware.

2.TPR (True Positive Rate): Đo lường khả năng phát hiện ransomware đúng.

3.FPR (False Positive Rate): Đo lường tỷ lệ nhầm lẫn benign thành ransomware.

4.F1-Score: Cân bằng giữa độ chính xác và độ nhạy của mô hình, đặc biệt trong việc phát hiện ransomware.

H. Kết quả đạt được: Công việc/tính năng/kỹ thuật mà nhóm thực hiện lập trình và triển khai cho demo:

Các công việc nhóm thực hiện:

1.Thu thập và phân tích dữ liệu:
Nhóm đã sử dụng Cuckoo Sandbox để thu thập các báo cáo hành vi từ ransomware, benign và malware, bao gồm các chuỗi gọi API, DLLs và Mutexes. Các dữ liệu này sau đó được trích xuất và kết hợp thành một vector duy nhất.

2.Phát triển mô hình học sâu 2L-CNN:
Nhóm đã triển khai mô hình CNN hai lớp (2L-CNN) để phân loại ransomware dựa trên các chuỗi đặc trưng được thu thập. Kết quả là mô hình đạt được độ chính xác cao và TPR tốt trong việc phân loại ransomware.

3.Giải thích mô hình với XAI (LIME và SHAP):
Nhóm sử dụng LIME giúp giải thích cục bộ các quyết định của mô hình cho từng mẫu riêng biệt, trong khi SHAP cung cấp cái nhìn tổng thể về sự quan trọng của các đặc trưng.

4.Triển khai website demo:
Nhóm phát triển một website để người dùng có thể tải lên các file và kiểm tra xem file có phải ransomware hay không. Kết quả của mô hình được giải thích thông qua LIME trên website demo.

Kết quả thực nghiệm:

Kết quả của nhóm:

Mô hình 2L-CNN đạt độ chính xác 99.10%, TPR 98.04%, không có dương tính giả (FPR = 0.00%), và F-Score 0.9901 khi phân loại ransomware vs benign. Trong tác

vụ phân loại ransomware vs malware, mô hình cũng đạt kết quả cao, với FPR thấp và độ chính xác ổn định.

LIME và SHAP giúp nhóm cung cấp các giải thích minh bạch cho mô hình, từ đó giúp hiểu rõ hơn về các đặc trưng quan trọng và quyết định của mô hình.

So sánh với bài báo tham khảo:
Phương pháp của nhóm cho kết quả tương đương hoặc vượt trội so với các mô hình đã được tham khảo, như Decision Tree, Random Forest, và CNN .

Nhận xét về khả năng, ưu và nhược điểm của phương pháp:

Ưu điểm:

Độ chính xác cao: Mô hình 2L-CNN cho kết quả phân loại chính xác cao với tỷ lệ dương tính giả thấp, điều này rất quan trọng trong việc phát hiện ransomware trong môi trường thực tế.

Giải thích minh bạch: Việc áp dụng LIME và SHAP giúp mô hình dễ hiểu hơn và tạo sự tin tưởng với người sử dụng, đặc biệt khi cần phân tích các quyết định của mô hình.

Khả năng áp dụng trong thực tế: Website demo cho phép người dùng dễ dàng kiểm tra các file có phải ransomware hay không, nâng cao tính ứng dụng.

Nhược điểm:

Yêu cầu tài nguyên tính toán cao: Việc sử dụng Cuckoo Sandbox và mô hình CNN đòi hỏi phần cứng mạnh và thời gian xử lý lâu khi phân tích lượng lớn mẫu.

Dữ liệu cần xử lý phức tạp: Quá trình trích xuất và xử lý các chuỗi đặc trưng từ báo cáo Cuckoo có thể gặp khó khăn khi dữ liệu không đầy đủ hoặc bị nhiễu.

I. Các khó khăn, thách thức hiện tại khi thực hiện:

- Dataset malware chứa dữ liệu ransomware: Dữ liệu malware có thể chứa cả ransomware, điều này gây khó khăn trong việc phân loại. Để giải quyết, nhóm đã tìm kiếm và sử dụng các nguồn dữ liệu có **tag family** rõ ràng, giúp phân biệt và xử lý chính xác các mẫu ransomware từ các loại malware khác.

- Nhiều malware có hành vi giống nhau: Các mẫu malware có thể có hành vi tương tự nhau, dẫn đến sự khó khăn trong việc phân loại chính xác. Giải pháp là tăng số lượng mẫu malware để huấn luyện mô hình và lọc ra một tập hợp mẫu đủ lớn và đa dạng, giúp cải thiện khả năng phân loại.

- **Mô hình XAI yêu cầu tài nguyên lớn**: Việc sử dụng các phương pháp giải thích mô hình XAI như LIME và SHAP yêu cầu tài nguyên tính toán lớn và tốn thời gian khi chạy trên máy local. Để khắc phục, nhóm đã tiến hành **debug và thử nghiệm trước** trên nền tảng **Kaggle**, nơi có tài nguyên tính toán mạnh mẽ và hỗ trợ tốt cho các mô hình học sâu.

3. TỰ ĐÁNH GIÁ MỨC ĐỘ HOÀN THÀNH SO VỚI KẾ HOẠCH THỰC HIỆN:

Mức độ hoàn thiện : 100%

4. NHẬT KÝ PHÂN CÔNG NHIỆM VỤ:

STT	Công việc	Phân công nhiệm vụ
1	Triển khai cài đặt Cuckoo	Hoàng Diệp
2	Làm poster, slide , tóm tắt nội dung chính từ bài báo liên quan	Nguyên Khang, Hoàng Diệp
3	Thu thập bộ dữ liệu dataset (ransomware, malware, benign)	Nguyên Khang, Hoàng Diệp
4	Hỗ trợ chạy dữ liệu cho Cuckoo và deep learning.	Hoàng Diệp, Vỹ Khang, Nguyên Khang,
5	Hỗ trợ và hoàn thiện mô hình 2L-CNN	Vỹ Khang, Hoàng Diệp
6	Triển khai pipeline từ kiến trúc mô hình biểu diễn thành website, sửa lỗi xAI (LIME/SHAP) và hiển thị mô hình dưới dạng biểu đồ ảnh	Vỹ Khang
7	Làm Final Report, quay Video Demo	Hoàng Diệp, Vỹ Khang, Nguyên Khang

BÁO CÁO TỔNG KẾT CHI TIẾT

Phần bên dưới của báo cáo này là tài liệu báo cáo tổng kết - chi tiết của nhóm thực hiện cho đề tài này.

Qui định: Mô tả các bước thực hiện/ Phương pháp thực hiện/Nội dung tìm hiểu (Ảnh chụp màn hình, số liệu thống kê trong bảng biểu, có giải thích)

A. Phương pháp thực hiện

- *Thành phần kiến trúc trong bài báo:*

1. Thu thập dữ liệu động (Dynamic Analysis)

- Hệ thống chạy ransomware trong môi trường sandbox để ghi lại các chuỗi hành vi động như gọi API, hoạt động trên file và registry.
- Từ đó, các chuỗi đặc trưng hành vi được trích xuất và tiền xử lý, chuẩn bị làm đầu vào cho mô hình học sâu

2. Mô hình phát hiện — CNN hai lớp

- Trong phần này, bài báo sử dụng mạng Convolutional Neural Network (CNN) để phân loại ransomware. Mặc dù CNN chủ yếu được thiết kế cho nhận dạng hình ảnh, nó cũng cho thấy hiệu quả cao trong xử lý chuỗi có cấu trúc — chẳng hạn như dữ liệu thời gian hoặc chuỗi hệ thống như API/DLL.
- Kiến trúc CNN bao gồm các lớp sau:
- Convolutional Layer: Lớp này sử dụng các kernel (bộ lọc) để trích xuất các mẫu cục bộ trong dữ liệu đầu vào. Việc sử dụng kernel cho phép mạng học được các mô hình hành vi lặp lại đặc trưng trong các chuỗi API, DLL hay Mutex.
- Stride: Giá trị stride xác định khoảng cách dịch chuyển của kernel. Việc điều chỉnh stride giúp kiểm soát mức độ chi tiết của các đặc trưng được trích xuất, giúp giảm nhiễu trong trường hợp chuỗi dài.
- Activation Layer: bài báo sử dụng hàm Sigmoid, phù hợp cho bài toán nhị phân (ransomware vs benign). Hàm này được định nghĩa như sau:

$$S(x) = \frac{1}{1 + e^{-x}}$$

- Pooling Layer: Lớp này giảm chiều dữ liệu bằng cách lấy giá trị đại diện trong từng vùng nhỏ. Bài báo chọn Max Pooling vì nó giúp giữ lại đặc trưng mạnh nhất trong vùng xét, hỗ trợ tốt cho việc học đặc trưng phân cấp.

$$MP(x)_{i,j} = \max_{p,q \in Pool(i,j)} x_{p,q}$$

- Fully-Connected Layer: Sau khi đặc trưng được trích xuất, lớp này hoạt động như một bộ phân loại, kết nối toàn bộ đặc trưng trích xuất với đầu ra. Lớp này đóng vai trò như một mạng nơ-ron chuẩn để ra quyết định dựa trên đặc trưng đã học.

3. Khung XAI giải thích kết quả

- Hệ thống ứng dụng hai kỹ thuật giải thích độc lập mô hình (model-agnostic explanations):
  - ***LIME (Local Interpretable Model-agnostic Explanations):*** cung cấp lời giải thích cục bộ, tập trung vào mỗi dự đoán cụ thể.
  - ***SHAP (SHapley Additive exPlanations)***: đánh giá tầm quan trọng của từng đặc trưng toàn cục và cục bộ

- *Thành phần kiến trúc mà nhóm đã thực hiện:*

1. Phân tích động với Cuckoo Sandbox

- Input: File thực thi (benign, ransomware, malware).
- Xử lý:
- Mẫu được đưa vào môi trường ảo hóa của Cuckoo Sandbox để thực thi.
- Cuckoo ghi lại hành vi của mẫu, sinh ra report dạng JSON (chứa API, DLL, mutex, network, ...).

2. Trích xuất đặc trưng từ report

- Cũng tương tự như bài báo, nhóm đã chọn 3 loại đặc trưng chính sau:
  - **API Calls**: Đây là loại đặc trưng quan trọng nhất vì hầu hết các hành vi của ransomware đều thể hiện thông qua các API. Ví dụ, hành vi mã hóa tập tin, giao tiếp với máy chủ điều khiển (C&C), ghi dữ liệu lên ổ đĩa, hoặc vô hiệu hóa các tiến trình bảo vệ đều cần sử dụng các API hệ thống. Vì thế, chuỗi API cung cấp một góc nhìn gần như hoàn chỉnh về hành vi thời gian chạy của phần mềm.
  - **DLLs (Dynamic Link Libraries)**: DLLs phản ánh các thư viện được tải vào trong quá trình thực thi. Các ransomware có xu hướng sử dụng những thư viện hệ thống cụ thể để thực hiện các hành vi ác ý như mã hóa dữ liệu, thao tác registry hoặc tương tác mạng. Việc truy vết các DLLs được tải giúp bổ sung ngữ cảnh cho các chuỗi API và có thể tiết lộ các hành vi obfuscation hay evasion.

- **Mutual Exclusions (Mutexes)**: Ransomware thường sử dụng mutex như một cách để đảm bảo rằng chỉ một phiên bản đang chạy nhằm tránh mã hóa dữ liệu lặp lại. Do đó, các mutex là đặc trưng mang tính dấu vết độc đáo (signature-like), rất hữu ích để phân biệt ransomware với phần mềm thông thường.

- Các chuỗi đặc trưng API, DLL và Mutex sẽ được trích xuất theo thứ tự thực hiện của các API, DLL và mutex.
- Cắt chuỗi API với số lượng tối đa là 500, DLL số lượng tối đa là 10 và Mutex số lượng tối đa cũng là 10.
- Lưu đặc trưng thành file JSON riêng cho từng mẫu (trong thư mục attributes).

3. Tiền xử lý & Mã hóa đặc trưng

- Chuyển các đặc trưng thành chuỗi số nguyên (ID), padding cho đủ số lượng tối đa(500,10,10). Sau đó concatenate ba chuỗi đặc trưng đó lại có dạng như sau: $API_{500}||DLL_{10}||MUTEX_{10}$.
- Kết quả: Dữ liệu đầu vào cho mô hình học sâu.

4. Huấn luyện mô hình học sâu

- Trong nghiên cứu này, nhóm đề xuất một mô hình phát hiện ransomware dựa trên mạng nơ-ron tích chập một chiều **1D CNN**. Mô hình sử dụng kiến trúc **2-layer 1D CNN**, trong đó có hai lớp tích chập (convolutional layers) liên tiếp nhằm trích xuất đặc trưng tuần tự từ chuỗi hành vi của phần mềm.
- Kiến trúc tổng thể của mô hình gồm bốn lớp chính:
  - **Embedding Layer**:
    - Lớp này ánh xạ các mã số nguyên biểu diễn các hành vi (API call, DLL, mutex) thành các vector đặc trưng có chiều cố định. Việc sử dụng embedding cho phép mô hình học được biểu diễn ngữ nghĩa của từng hành vi trong không gian liên tục.
  - **2-layer Convolutional Block**:
    - Gồm hai lớp Conv1D liên tiếp với số lượng bộ lọc lần lượt là 128 và 64, kết hợp với hàm kích hoạt ReLU. Mỗi lớp được theo sau bởi một lớp MaxPooling1D và Dropout để giảm chiều và ngăn overfitting. Hai lớp Conv1D giúp mô hình học được các đặc trưng cục bộ theo thứ tự thời gian trong chuỗi hành vi, ví dụ như chuỗi hành động mà ransomware thường thực hiện (tạo mutex, gọi hàm mã hóa, ghi log, v.v.). Trong đó mỗi phép tích chập tại vị trí i được tính như sau:

$$c_i = f(w \cdot X_{i:i+k-1} + b)$$

      - Trong đó:

- $X_{i:i+k-1}$: là đoạn con của chuỗi đầu vào (sau khi đã qua embedding), bắt đầu từ vị trí i và kéo dài k bước. Mỗi phần tử trong đoạn này là một vector có d chiều. Như vậy, đoạn con này có kích thước k × d.
- w: là ma trận trọng số của kernel trong lớp tích chập (Conv1D), có cùng kích thước k × d để thực hiện phép nhân với đoạn $X_{i:i+k-1}$.
- b: là hệ số bias (số vô hướng) được cộng vào kết quả tích chập.
- $f()$: là hàm kích hoạt dùng sau phép tích chập. Trong mô hình này, hàm kích hoạt là ReLU, với công thức là $f(x) = max(0, x)$.
- c_i: là đầu ra (feature) tại vị trí i sau khi áp dụng phép tích chập và kích hoạt.

○ MaxPooling Layer: Để giảm chiều và giữ lại các đặc trưng quan trọng nhất, chúng tôi sử dụng max pooling, được định nghĩa như sau:

$$MP(x)_{i,j} = \max_{p,q \in Pool(i,j)} x_{p,q}$$

▪ Trong đó:
- $MP(x)_{i,j}$: là giá trị đầu ra tại vị trí (i, j) sau khi áp dụng phép Max Pooling lên đầu vào x.
- $Pool(i, j)$: là vùng lân cận (window) trong ma trận x được ánh xạ vào vị trí (i, j) của đầu ra. Đây là vùng mà hàm Max Pooling sẽ quét qua để tìm giá trị lớn nhất.
- $x_{p,q}$: là giá trị tại vị trí (p, q) nằm trong vùng Pool(i, j).
- $max\{p,q \in Pool(i,j)\}$ là phép chọn giá trị lớn nhất trong tất cả các phần tử nằm trong vùng $Pool(i, j)$.

- **Fully Connected Layer**:
  - Sau khi đặc trưng được trích xuất và làm phẳng (flatten), một lớp Dense với 64 neuron và hàm kích hoạt ReLU được sử dụng để học các biểu diễn phi tuyến sâu hơn. Một lớp Dropout (tỉ lệ 0.5) cũng được sử dụng nhằm giảm hiện tượng overfitting.
- **Output Layer**:
  - Đây là lớp đầu ra gồm 2 neuron và sử dụng hàm kích hoạt softmax để phân loại đầu vào thành một trong hai lớp: ransomware hoặc benign.

▪ Mô hình được huấn luyện với hàm mất mát sparse_categorical_crossentropy, sử dụng tối ưu hóa Adam và đánh giá bằng độ chính xác (accuracy). Cấu trúc 1D CNN được chọn vì khả năng xử

lý chuỗi hiệu quả với ít tham số hơn so với các kiến trúc RNN, đồng thời tận dụng tốt tính cục bộ và thứ tự thời gian trong dữ liệu hành vi động.

- Xử lý:
  - Chia dữ liệu train/validate/test.
  - Huấn luyện mô hình, lưu trọng số tốt nhất.
  - Đánh giá mô hình trên tập test (accuracy, recall, F1-score, ...).

5. Giải thích mô hình (Explainability)

- Mặc dù CNN đạt hiệu suất cao, nó thường bị xem như một "hộp đen" khiến người dùng khó hiểu được tại sao mô hình đưa ra một quyết định cụ thể. Để tăng tính minh bạch và hỗ trợ kiểm chứng mô hình, nhóm tích hợp hai kỹ thuật XAI (Explainable AI) phổ biến:
  - LIME (Local Interpretable Model-agnostic Explanations): LIME tạo ra một mô hình đơn giản (như tuyến tính) xung quanh một dự đoán cụ thể để giải thích nó. Đây là lựa chọn phù hợp để giải thích những mẫu bất thường hoặc mẫu bị phân loại sai, cho phép các nhà nghiên cứu truy vết lại đặc trưng nào ảnh hưởng nhiều nhất.
  - SHAP (SHapley Additive exPlanations): SHAP cung cấp một cách tiếp cận dựa trên lý thuyết trò chơi để phân bổ đóng góp của từng đặc trưng vào đầu ra của mô hình. SHAP giúp đánh giá mức độ quan trọng trung bình của từng đặc trưng trong toàn bộ dữ liệu, và nhờ đó có thể xác định được các API hoặc DLL mang tính phân loại cao.
- Hoạt động chính:
  - SHAP (Global):
    - Phân tích ảnh hưởng toàn cục của các đặc trưng lên quyết định mô hình.
    - Hiển thị top đặc trưng ransomware/benign/malware.
  - LIME (Local):
    - Giải thích quyết định của mô hình trên từng mẫu cụ thể.
    - Hiển thị top đặc trưng ảnh hưởng đến dự đoán của từng mẫu.
    - Vẽ biểu đồ:
      - Bar chart thể hiện các đặc trưng quan trọng nhất.

6. Triển khai backend phục vụ phân tích

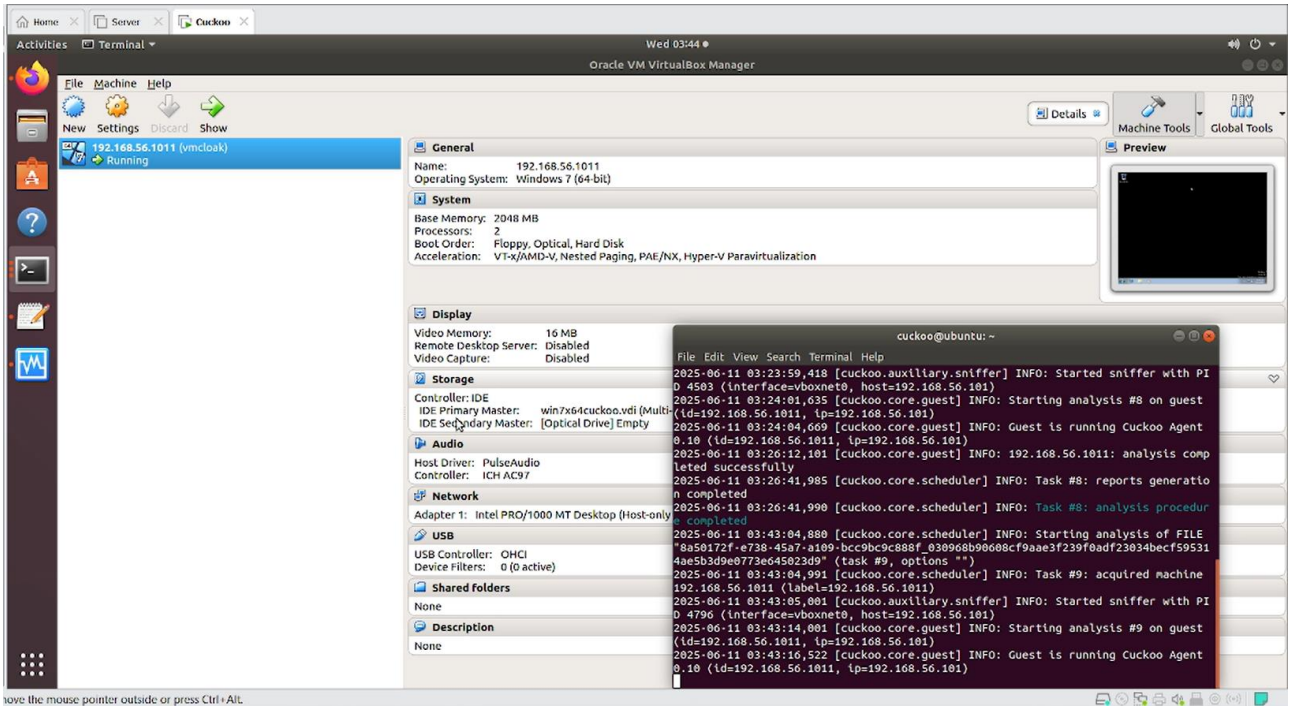
- Xây dựng REST API với Flask.
- Nhận file đầu vào (file thực thi, report, hoặc đặc trưng).
- Thực hiện pipeline phân tích, dự đoán, giải thích.
- Trả về kết quả và biểu đồ cho frontend.

- Tóm tắt luồng dữ liệu:

- File thực thi → Cuckoo Sandbox → Report JSON → Trích xuất đặc trưng → Tiền xử lý → Huấn luyện mô hình → Giải thích → Triển khai backend phục vụ phân tích thực tế.

B. Chi tiết cài đặt, hiện thực

- *Cấu hình máy:*
  - Cấu hình máy Linux(cuckoo):
    - Hệ điều hành Ubuntu 20.04
    - RAM: 8 GB
    - CPU: Intel Core i7
    - Ổ cứng 100GB
  - Cấu hình máy Window(Toàn bộ kiến trúc):
    - Hệ điều hành Window 11.
    - RAM: 16 GB
    - CPU: Intel Core i7
  - *Cài đặt môi trường lập trình*
    - Cài đặt Python và pip: Dùng python 3.11.9
  - *Cài đặt các thư viện cần thiết:*
    - pip install tensorflow
    - pip install scikit-learn
    - pip install lime
    - pip install shap==0.44.1
    - pip install matplotlib
    - pip install flask
    - pip install streamlit
  - *Chuẩn bị dữ liệu*
    - Phân tích mẫu với Cuckoo Sandbox
      - Đưa file thực thi vào Cuckoo để sinh ra các file report JSON.
      - Lưu các report vào thư mục:
        - benign
        - ransomware
        - malware



- Trích xuất đặc trưng
  - Script python **extract_attribute.py** sẽ đọc các report JSON, trích xuất trường API, DLL, Mutex và lưu dạng **.json** vào thư mục benign, ransomware, malware.
  - Cụ thể thì chương trình sẽ:
    - Trích xuất API từ data*['behavior']['processes']['calls']['api']*.
    - Trích xuất DLL từ *data['behavior']['processes'][`dll_loaded`]*
    - Trích xuất Mutex từ *data['behavior']['processes'][`mutex`]*

```
if 'behavior' in data:
    if 'summary' in data['behavior']:
        summary = data['behavior']['summary']
        if 'mutex' in summary:
            mutexes = summary.get('mutex', [])

        if 'dll_loaded' in summary:
            dlls = summary.get('dll_loaded', [])

    if 'processes' in data['behavior']:
        for process in data['behavior']['processes']:
            if 'calls' in process:
                for call in process['calls']:
                    if 'api' in call:
                        apis.append(call['api'])
```

```
{
    "dlls": [],
    "apis": [
        "NtAllocateVirtualMemory",
        "NtFreeVirtualMemory",
        "NtAllocateVirtualMemory",
        "NtAllocateVirtualMemory",
        "NtAllocateVirtualMemory",
        "NtAllocateVirtualMemory",
        "NtFreeVirtualMemory",
        "NtAllocateVirtualMemory",
        "NtAllocateVirtualMemory",
        "CopyFileA",
        "NtCreateFile",
        "NtAllocateVirtualMemory",
        "GetFileSize",
        "NtAllocateVirtualMemory",
        "GetFileSize",
        "NtReadFile",
        "GetFileSize",
        "SetFilePointer",
        "GetFileSize",
        "SetFilePointer",
        "NtWriteFile",
        "NtWriteFile",
        "NtWriteFile",
        "NtClose",
        "NtFreeVirtualMemory",
        "NtFreeVirtualMemory",
        "NtFreeVirtualMemory",
        "NtFreeVirtualMemory",
        "NtTerminateProcess",
        "NtTerminateProcess",
        "NtTerminateProcess"
    ],
    "mutexes": []
}
```

- Tiền xử lý và tạo dữ liệu huấn luyện
  - Xây dựng từ điển token
  - Mã hóa, padding dữ liệu
  - Sau đó concatenate ba chuỗi đặc trưng.

```
def prepare_sequences(reports, token2id):
    """
    Với mỗi báo cáo chứa 'apis','dlls','mutexes',
    trả về mảng shape=(N, SEQ_LEN).
    """
    sequences = []
    for r in reports:
        api_ids   = encode_and_pad(r['apis'],   token2id, MAX_LEN_API)
        dll_ids   = encode_and_pad(r['dlls'],   token2id, MAX_LEN_DLL)
        mutex_ids = encode_and_pad(r['mutexes'], token2id, MAX_LEN_MUTEX)
        seq = np.concatenate([api_ids, dll_ids, mutex_ids], axis=0)
        sequences.append(seq)
    return np.stack(sequences)
```

- Chia train/validate/test bằng train_test_split

```
X_train, X_test, y_train, y_test, fn_train, fn_test = train_test_split(
    X, y, file_names,
    test_size=0.1,
    stratify=y,
    random_state=RANDOM_STATE
)
```

```
X_train, X_validate, y_train, y_validate, fn_train, fn_validate = train_test_split(
    X, y, file_names,
    test_size=0.1,
    stratify=y,
    random_state=RANDOM_STATE
)
```

- Lưu các file **.npy** phục vụ huấn luyện và giải thích.

```
np.save('X_background.npy', X_background)
np.save('X_test.npy', X_test)
np.save('Y_test.npy', y_test)
np.save('file_names_test.npy', fn_test)
```

```
np.save('X_background_validate.npy', X_background_validate)
np.save('X_validate.npy', X_validate)
np.save('Y_validate.npy', y_validate)
np.save('file_names_validate.npy', fn_validate)
```

- *Huấn luyện mô hình 2-layer CNN:*
  - Mô hình được định nghĩa trong **model.py** với 2 lớp Conv1D liên tiếp.

```
model = Sequential([
    # Lớp 1: Embedding Layer (Input layer)
    Embedding(input_dim=vocab_size,
              output_dim=EMB_DIM,
              input_length=SEQ_LEN),

    # Lớp 2: Convolutional Block - 2 convolutional layers và 2 pooling layers
    # Layer Conv #1
    Conv1D(filters=128, kernel_size=5, activation='relu'),
    Dropout(0.15),
    MaxPooling1D(pool_size=2),
    # Layer Conv #2
    Conv1D(filters=64, kernel_size=3, activation='relu'),
    Dropout(0.15),
    MaxPooling1D(pool_size=2),

    # Lớp 3: Fully Connected Layer
    Flatten(),
    Dense(64, activation='relu'),
    Dropout(0.5),

    # Lớp 4: Output Layer (Sigmoid Classifier)
    Dense(2, activation='softmax')
])
model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
return model
```

  - Khi chạy **train.py** hoặc **train_ransome_mal.py** mô hình sẽ được huấn luyện và lưu trọng số tốt nhất vào file **best_model.weights.h5**.

```
history = cnn_model.fit(
    X_train, y_train,
    validation_split=VALIDATION_SPLIT,
    epochs=EPOCHS,
    batch_size=BATCH_SIZE
)
cnn_model.save_weights('best_model.weights.h5')
```

- *Giải thích:*

- Thực hiện bên trong **explain.py** hoặc **explain_ransome_mal.py**
- Tải mô hình và dữ liệu test

```
# 1. Load token2id và build id2token
token2id = json.load(open("./token2id.json", encoding="utf-8"))
id2token = build_id2token(token2id)

# 2. Khởi tạo và build model để load_weights
cnn_model = build_model(vocab_size=len(token2id)+1)
cnn_model.build((None, SEQ_LEN))
cnn_model.load_weights("./best_model.weights.h5")

# 3. Chuẩn bị dữ liệu SHAP & LIME
X_background = np.load("./X_background.npy")
X_test       = np.load("./X_test.npy")
Y_test       = np.load("./Y_test.npy")
file_names = np.load("./file_names_test.npy", allow_pickle=True)
```

- Sinh các biểu đồ SHAP (global) và LIME (local) để giải thích quyết định của mô hình.

```python
def shap_explain_global_ransome(cnn_model, X_background, X_test, id2token, top_n=10):
    try:
        explainer = shap.GradientExplainer(cnn_model, X_background)
        shap_vals = explainer.shap_values(X_test)
    except Exception:
        f = lambda x: cnn_model.predict(x)
        explainer = shap.KernelExplainer(f, X_background[:50])
        shap_vals = explainer.shap_values(X_test[:20])

    # SHAP values cho lớp ransomware (index 1)
    sv = shap_vals[1]  # shape: (num_samples, seq_len)
    X_display = X_test[:sv.shape[0]]  # cùng số sample

    # Mean SHAP giữ nguyên dấu
    mean_shap = sv.mean(axis=0)

    # Lấy token ID theo từng vị trí trung bình và ánh xạ sang token
    token_ids = X_display[:, 0:sv.shape[1]].mean(axis=0).astype(int)
    feature_names = [id2token.get(i, f"Pos_{idx}") for idx, i in enumerate(token_ids)]

    # Ghép tên + SHAP value
    token_shap_pairs = list(zip(feature_names, mean_shap))

    # Top đặc trưng đẩy về ransomware
    top_pos = sorted(token_shap_pairs, key=lambda x: x[1], reverse=True)[:top_n]

    # Top đặc trưng đẩy về benign
    top_neg = sorted(token_shap_pairs, key=lambda x: x[1])[:top_n]

    # Vẽ tổng quan SHAP
    shap.summary_plot(sv, X_display, feature_names=feature_names)

    # Bar plot top tích cực và tiêu cực (nếu bạn định vẽ luôn)
    plot_top_shap_bar(top_pos + top_neg,'ransomware')  # hoặc tách riêng nếu muốn
```

```python
def lime_explain_instance(cnn_model, sequence, id2token, num_features=20):
    explainer = LimeTextExplainer(class_names=["benign","ransomware"], split_expression=r"\s+")
    def predict_proba(texts):
        seqs = []
        for t in texts:
            tokens = [int(x) for x in t.split()]
            # Padding hoặc cắt để độ dài đúng SEQ_LEN
            if len(tokens) < SEQ_LEN:
                tokens = tokens + [0] * (SEQ_LEN - len(tokens))
            else:
                tokens = tokens[:SEQ_LEN]
            seqs.append(tokens)
        X = np.array(seqs)
        probs = cnn_model.predict(X)
        return probs
    text_input = " ".join(str(int(x)) for x in sequence)
    exp = explainer.explain_instance(text_input, predict_proba, num_features=num_features)
    return [(id2token[int(tok)], weight) for tok, weight in exp.as_list()]
```

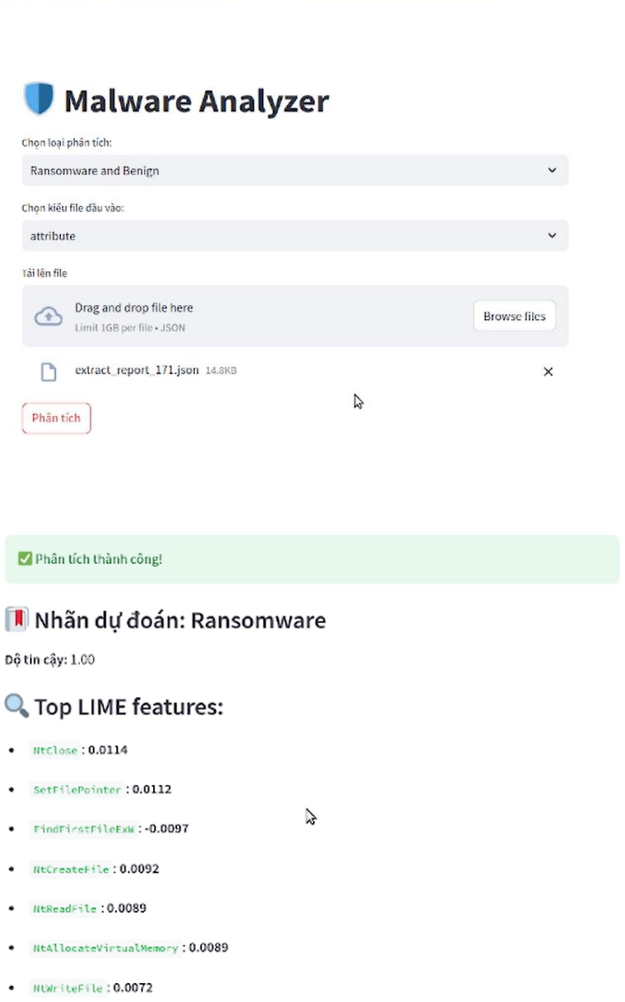
- Và sinh biểu đồ tương ứng với từng explain.

```python
def plot_top_shap_bar(top_tokens, type):
    features, weights = zip(*top_tokens)
    colors = ['red' if w > 0 else 'blue' for w in weights]

    plt.figure(figsize=(10, 6))
    bars = plt.barh(features, weights, color=colors, edgecolor='black')

    # Vẽ đường x = 0
    plt.axvline(x=0, color='gray', linestyle='--', linewidth=1)

    # Chú giải
    red_patch = plt.Line2D([0], [0], color='red', lw=4, label='Pushes to Ransomware')
    blue_patch = plt.Line2D([0], [0], color='blue', lw=4, label='Pushes to Benign')
    plt.legend(handles=[red_patch, blue_patch], loc='lower right')

    # Căn chỉnh
    plt.xlabel('Mean SHAP Value', fontsize=12)
    plt.title(f'Global SHAP Values for {type} Prediction', fontsize=14)
    plt.gca().invert_yaxis()  # Đảo ngược trục y để giá trị lớn nằm trên
    plt.grid(axis='x', linestyle='--', alpha=0.5)
    plt.tight_layout()
    plt.show()
```

```python
def plot_lime_top_5_5(local_lime, count, pred_label, pred_proba,filename=None):
    # 1. Tách 2 nhóm
    pos = [(f, w) for f, w in local_lime if w > 0]
    neg = [(f, w) for f, w in local_lime if w < 0]

    # 2. Sort theo |weight| giảm dần và lấy top 5 mỗi nhóm
    pos = sorted(pos, key=lambda x: abs(x[1]), reverse=True)[:5]
    neg = sorted(neg, key=lambda x: abs(x[1]), reverse=True)[:5]

    # 3. Xen kẽ: bắt đầu bằng pos[0], rồi neg[0], v.v.
    interleaved = []
    for i in range(max(len(pos), len(neg))):
        if i < len(pos):
            interleaved.append(pos[i])
        if i < len(neg):
            interleaved.append(neg[i])

    # 4. Tách lại để vẽ
    features, weights = zip(*interleaved)
    colors = ['green' if w > 0 else 'red' for w in weights]

    # 5. Vẽ
    plt.figure(figsize=(8, 4))
    plt.barh(features[::-1], weights[::-1], color=colors[::-1])  # Đảo để mạnh nhất nằm trên
    plt.axvline(x=0, color='black', linewidth=0.8)
    plt.xlabel('LIME Weight')
    plt.title(f'Sample {count}({filename}): {pred_label} (prob={pred_proba[1]:.5f})')
    plt.tight_layout()
    plt.show()
```

- *Triển khai website phục vụ phân tích*
  - Nhận file đầu vào, thực hiện pipeline phân tích, trả về kết quả và biểu đồ cho frontend.
  - Cụ thể pipeline như sau:
    - Nếu là file execute thì sẽ được đẩy sang cuckoo để trích xuất report.
    - Report sẽ được mang đi trích xuất cách thành phần API, DLL và Mutex.
    - Sau đó, các thành phần API, DLL, Mutex được mã hóa (encode) thành chuỗi số nguyên dựa trên từ điển token đã xây dựng.
    - Chuỗi số nguyên này được **padding** về cùng độ dài (SEQ_LEN) để phù hợp với đầu vào của mô hình.
    - Dữ liệu đã chuẩn hóa sẽ được đưa vào **mô hình 2-layer CNN** để dự đoán nhãn (Ransomware/Benign hoặc Ransomware/Malware).
    - Kết quả dự đoán cùng với **giải thích mô hình** (LIME) sẽ được sinh ra.

```
    if input_type == 'execute':
        task_id = module.submit_sample(filepath)
        module.wait_for_report(task_id)

        report_path = os.path.join(REPORT_FOLDER, f"report_{filename}.json")
        module.download_report(task_id, report_path)

        attr_path = os.path.join(REPORT_FOLDER, f"attributes_{filename}.json")
        module.extract_fields(report_path, attr_path)

        # Hàm này trả về (label, confidence, lime_result) và vẽ plot
        label, confidence, lime_result = module.check_and_explain(attr_path)

    elif input_type == 'report':
        attr_path = os.path.join(REPORT_FOLDER, f"attributes_{filename}.json")
        module.extract_fields(filepath, attr_path)
        label, confidence, lime_result = module.check_and_explain(attr_path)

    elif input_type == 'attribute':
        label, confidence, lime_result = module.check_and_explain(filepath)

    else:
        return jsonify({"error": "Kiểu đầu vào không hợp lệ"}), 400

    # Vẽ biểu đồ LIME rồi lưu ra file plot dưới dạng PNG (giả sử module trả về plot object)
    plot_filename = f"plot_{uuid.uuid4()}.png"
    plot_path = os.path.join(PLOT_FOLDER, plot_filename)
    module.plot_lime_top_5_5_to_file(lime_result, filename, label, confidence, save_path=plot_path)

    # Trả về JSON kết quả + đường dẫn plot để frontend tải
    return jsonify({
        "label": label,
        "confidence": confidence,
        "lime_features": lime_result,
        "plot_url": f"/plot/{plot_filename}"
    })
```



C. Kết quả thực nghiệm

- **Kết quả đánh giá mô hình:**
  - Đối với Ransomware/Benign

```
Test Accuracy : 0.9910
Test Recall (TPR): 0.9804
Test FPR      : 0.0000
Test F1-Score : 0.9901
```

- **Test Accuracy:** 0.9910

  → Mô hình phân loại chính xác 99.10% tổng số mẫu.

- **Recall (TPR):** 0.9804

  → Trong tất cả các mẫu ransomware thực sự, mô hình phát hiện đúng 98.04%.

- **FPR (False Positive Rate):** 0.0000

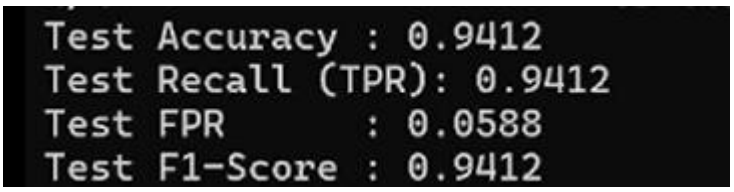
  → Không có mẫu benign nào bị nhận nhầm là ransomware (tức là FPR = 0).

- **F1-Score:** 0.9901

  → Sự cân bằng giữa Precision và Recall rất cao, cho thấy mô hình ổn định.

- **Kết luận:** Mô hình phân biệt ransomware với benign rất tốt, gần như không có lỗi phân loại sai. Đây là mức độ hiệu quả rất cao, phù hợp dùng cho mục đích phát hiện ransomware trong môi trường thực tế.

o Đối với Ransomware/Malware



- **Test Accuracy**: 0.9412
  → Mô hình phân loại đúng 94.12% mẫu.
- **Recall (TPR)**: 0.9412
  → Trong các mẫu ransomware thực sự, mô hình nhận diện đúng 94.12%.
- **FPR**: 0.0588
  → Có 5.88% mẫu malware bị nhầm là ransomware.
- **F1-Score**: 0.9412
  → Vẫn khá tốt, mặc dù thấp hơn so với trường hợp Ransomware/Benign.
- **Kết luận**: Mô hình gặp khó khăn hơn khi phân biệt giữa ransomware và các loại malware khác. Tuy nhiên, độ chính xác vẫn ở mức chấp nhận được (>94%), cho thấy đặc trưng của ransomware vẫn khá nổi bật so với các malware khác.

- Ngoài ra nhóm cũng đã so sánh khả năng nhận diện của 2L-CNN với một số mô hình khác như CNN, Decision Tree, Random Forest và lập được bảng so sánh dựa trên các tiêu chí *Accuracy, TPR, FPR và F1-Score*. Và được đánh giá trên 2 trường hợp là Ransomware/Benign và Ransomware/Malware

Table 1
Comparison with existing methods (ransomware/benign)

Model	Accuracy	TPR	FPR	F-Score
Decision Tree	0.9820	0.9608	0.0000	0.9800
Random Forest	0.9900	0.9783	0.0000	0.9890
CNN	0.9820	0.9804	0.0167	0.9804
2L-CNN	0.9910	0.9804	0.0000	0.9901

Table 2
Comparison with existing methods (ransomware/malware)

Model	Accuracy	TPR	FPR	F-Score
Decision Tree	0.9118	0.9412	0.1176	0.9143
Random Forest	0.9412	0.9608	0.0784	0.9423
CNN	0.9216	0.9412	0.0980	0.9231
2L-CNN	0.9412	0.9412	0.0588	0.9412

- **Phân tích tham số:**
  Dựa trên giới thiệu ở Phương pháp thực hiện, ba chuỗi đặc trưng (apis, dlls, mutexes) sẽ được nối thành một chuỗi tổng hợp A || D || M
  Trong đó:
  - d là độ dài chuỗi DLLs
  - a là độ dài chuỗi API calls
  - m là độ dài chuỗi mutexes

Để chọn giá trị cho d, a, m, nhóm đã tính trung bình độ dài thực tế của mỗi tập đặc trưng trên toàn bộ dữ liệu (**compute_stats.py**):

```
--- Thống kê cho 'benign' ---
Avg. DLLs    : 36.10
Avg. APIs    : 101776.58
Avg. Mutexes: 0.51

--- Thống kê cho 'ransomware' ---
Avg. DLLs    : 16.46
Avg. APIs    : 39517.08
Avg. Mutexes: 1.93

--- Thống kê cho 'attributes_full (tổng)' ---
Avg. DLLs    : 26.36
Avg. APIs    : 70894.38
Avg. Mutexes: 1.21
```

Với API calls, độ dài gốc rất lớn (~70000+) vượt quá khả năng xử lý của mô hình, nên nhóm giới hạn thử nghiệm trong khoảng 50 (ngưỡng dưới) đến 1000 (ngưỡng trên)

Từ kết quả trên, nhóm tiến hành khảo sát:

- DLL và mutex trong khoảng [5, 15]
- API trong dài [50, 100, 500, 1000], với 1000 là ngưỡng trên (phủ hầu hết các samples, tránh padding quá nhiều) và 50 là ngưỡng dưới (để phát hiện xem model bắt đầu mất thông tin từ bao nhiêu calls đầu)

Phép thử tham số được thực hiện kiểm tra cho các giá trị Accuracy, Loss, TPR (True Positive Rate) và FPR (False Positive Rate) được thực hiện với file **plot_acc_loss.py** và **plot_tpr_fpr.py** và cho ra kết quả như hình bên dưới.

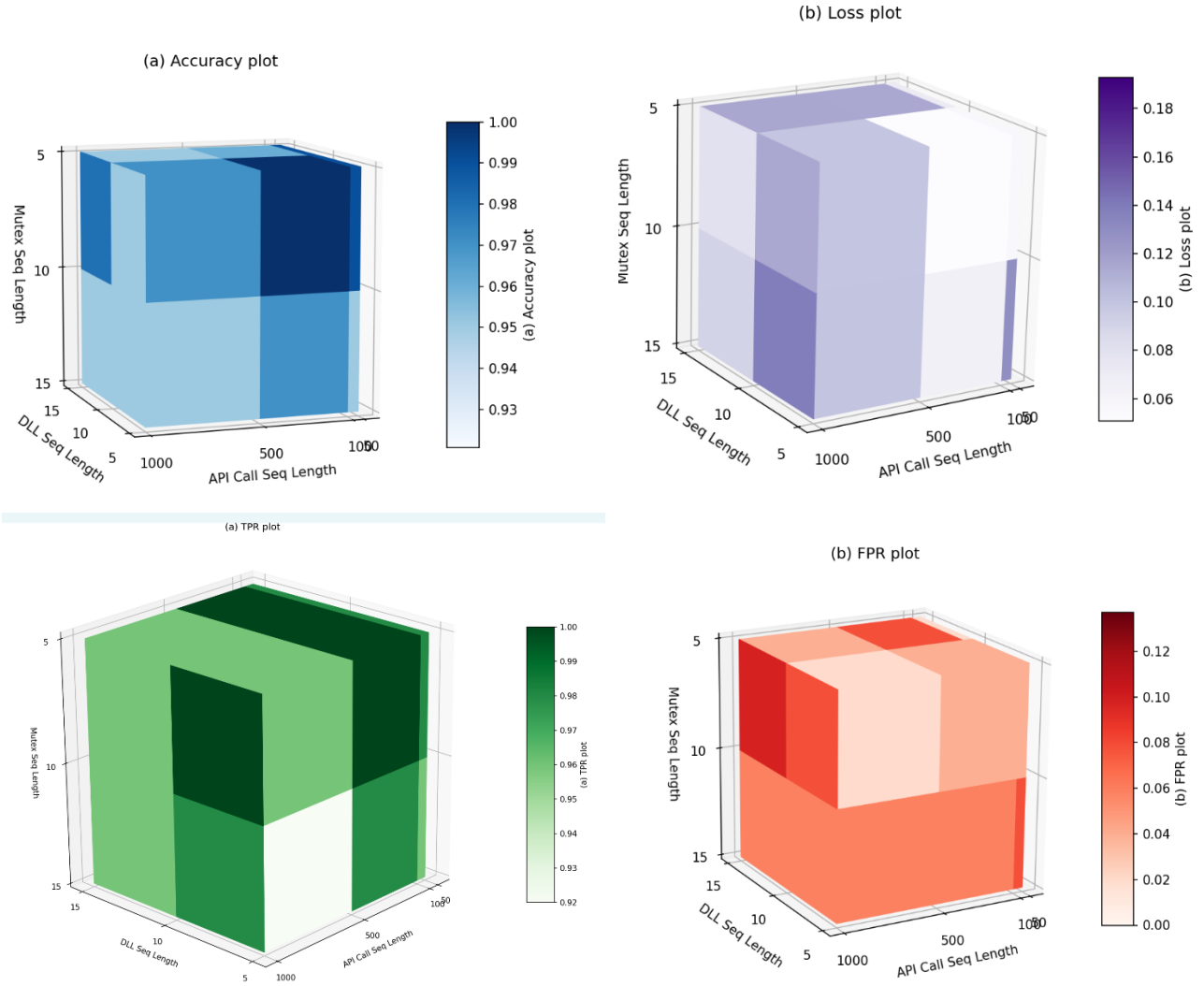
Với kết quả cho thấy:

- Ảnh a và b đầu cho thấy Accuracy đạt được giá trị cao và đồng thời Loss đạt giá trị thấp ở cấu hình a > 500, d > 10, m > 10. Tức là chỉ cần tối thiểu 500 API calls đầu, 10 DLLs và 10 Mutexes là có thể phát hiện tốt ransomware

- Với kết quả từ TPR plot và FPR plot cho thấy:

  - Với TPR: tỉ lệ phát hiện đúng ransomware cao nhất nằm ở khoảng dưới 500 API calls, khoảng dưới 10 DLLs và khoảng dưới 10 Mutexes (a = 500-, d = 10- và m = 10-
  - Với FPR: tỉ lệ phát hiện sai thấp nhất nằm ở đoạn 500-1000 API calls, 5-10 DLLs và 5-10 Mutexes

Kết hợp các yếu tố trên, việc chọn số lượng API calls, DLLs, Mutexes là **[500, 10, 10]** là phù hợp cho tập dữ liệu trên, đảm bảo đủ để phát hiện tối đa ransomware.



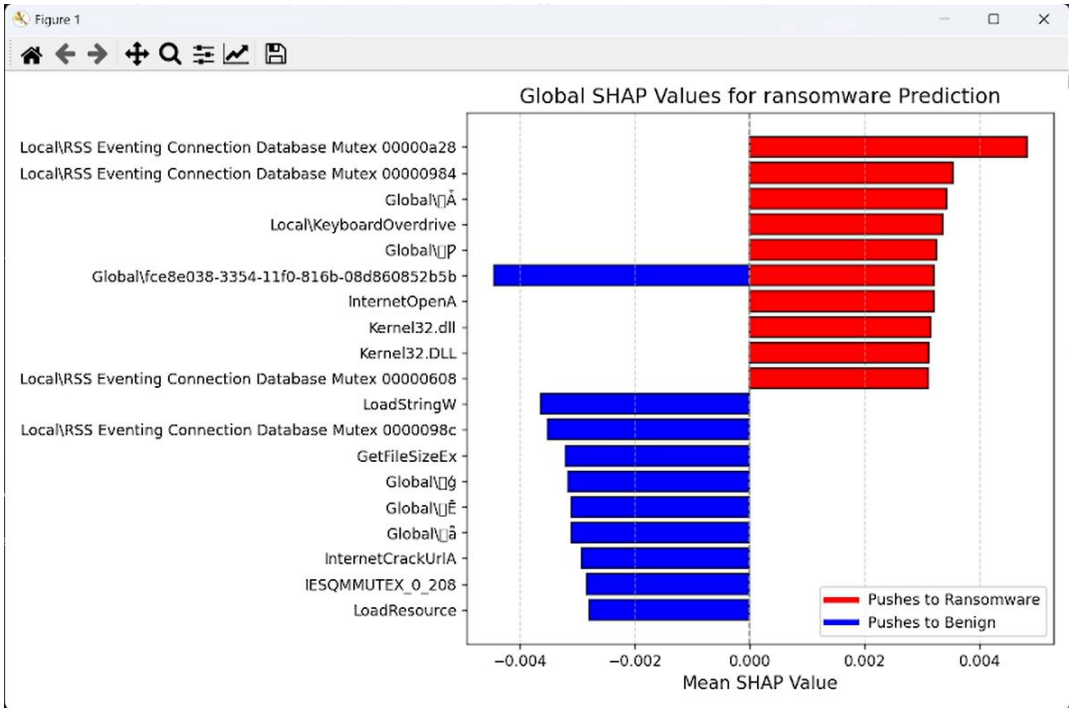
- **Kết quả phân tích SHAP:**
  - Ý nghĩa biểu đồ SHAP:
    - Biểu đồ SHAP toàn cục thể hiện mức độ ảnh hưởng trung bình của từng đặc trưng (API, Mutex, DLL) đến việc mô hình dự đoán một nhãn cụ thể: Malware, Benign, hoặc Ransomware.
    - Thanh đỏ: đặc trưng làm củng cố khả năng dự đoán nhãn đang xét.
    - Thanh xanh: đặc trưng làm giảm khả năng dự đoán nhãn đó.
  - *Đối với Ransomware*

Đặc trưng củng cố khả năng là Ransomware:

Loại	Đặc trưng	Nhận xét chuyên sâu
Mutex	`Local\RSS Eventing Mutex 00000a28` , `00000984` , `0000098C`	Các mutex dạng ngẫu nhiên và "RSS Eventing" cho thấy dấu hiệu điển hình của mã độc sử dụng kỹ thuật đa luồng hoặc kiểm soát tiến trình.
Mutex	`Local\KeyboardOverdrive`	Mutex có tên gợi ý thao tác tự động trên bàn phím – một hành vi nguy hiểm.
API	`InternetOpenA` , `InternetCrackUrlA`	Các hàm liên quan đến kết nối mạng – thường được sử dụng để giao tiếp với máy chủ C2 hoặc tải payload.
DLL	`Kernel32.dll`	Dù là DLL hợp pháp, nhưng thường bị ransomware dùng để truy cập các API hệ thống quan trọng như tạo file, cấp phát bộ nhớ.
API	`GetFileSizeEx` , `LoadStringW`	Gợi ý hành vi thao tác và phân tích tệp – thường xảy ra trước khi mã hóa.

Đặc trưng phản đối khả năng là Ransomware:

Loại	Đặc trưng	Nhận xét
Mutex	`IESQMMUTEX_0_208`	Mutex hợp lệ từ Internet Explorer – thường không liên quan đến ransomware.
API	`LoadResource`	API hợp lệ – không đặc trưng riêng cho ransomware.
DLL	`Kernel32.dll` (trong một số ngữ cảnh)	DLL hợp pháp – nếu dùng không đặc biệt, không đủ tạo nghi ngờ.



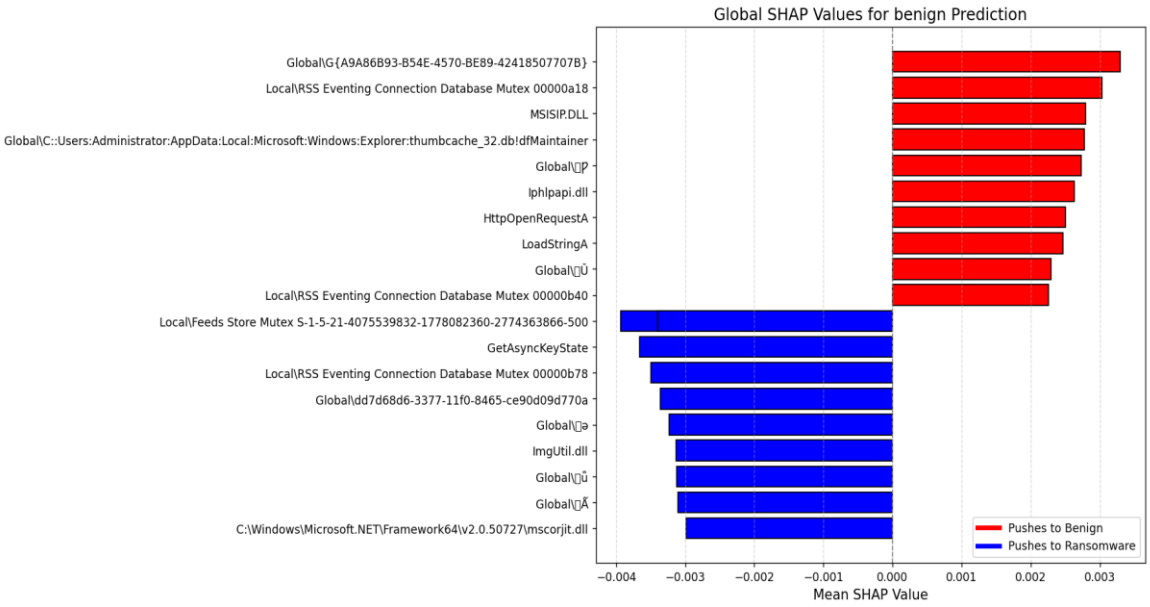
- *Đối với Benign:*

Đặc trưng cũng cố khả năng là Benign:

Loại	Đặc trưng	Nhận xét chuyên sâu
Mutex	`Global\{A9A86B93-...}`	Mutex theo định dạng GUID – thường thấy ở phần mềm hợp pháp sử dụng COM hoặc .NET.
DLL	`MSISIP.DLL` , `Iphlapi.dll` , `ImgUtil.dll`	Các thư viện hợp pháp trong hệ điều hành Windows – gắn với chức năng chuẩn như mạng, hình ảnh, bảo mật file.
API	`HttpOpenRequestA` , `LoadStringA`	Các hàm phổ biến trong ứng dụng GUI và web – không mang tính chất độc hại.
Khác	`Explorer\thumbcache_32.db`	Tệp hệ thống được Explorer truy cập thường xuyên – gợi ý hành vi bình thường của người dùng.

Đặc trưng phản đối khả năng là Benign:

Loại	Đặc trưng	Nhận xét
API	`GetAsyncKeyState`	API thường bị lợi dụng để giám sát bàn phím – có thể liên quan đến spyware hoặc ransomware.
Mutex	`Local\RSS Mutex 0000b78` , `0000b40`	Các mutex dạng ngẫu nhiên, bất thường – không phải hành vi của phần mềm hợp pháp.
DLL	`Global\Q*.dll`	Tên DLL không rõ ràng – có thể là DLL được tiêm động hoặc đi kèm gói malware.



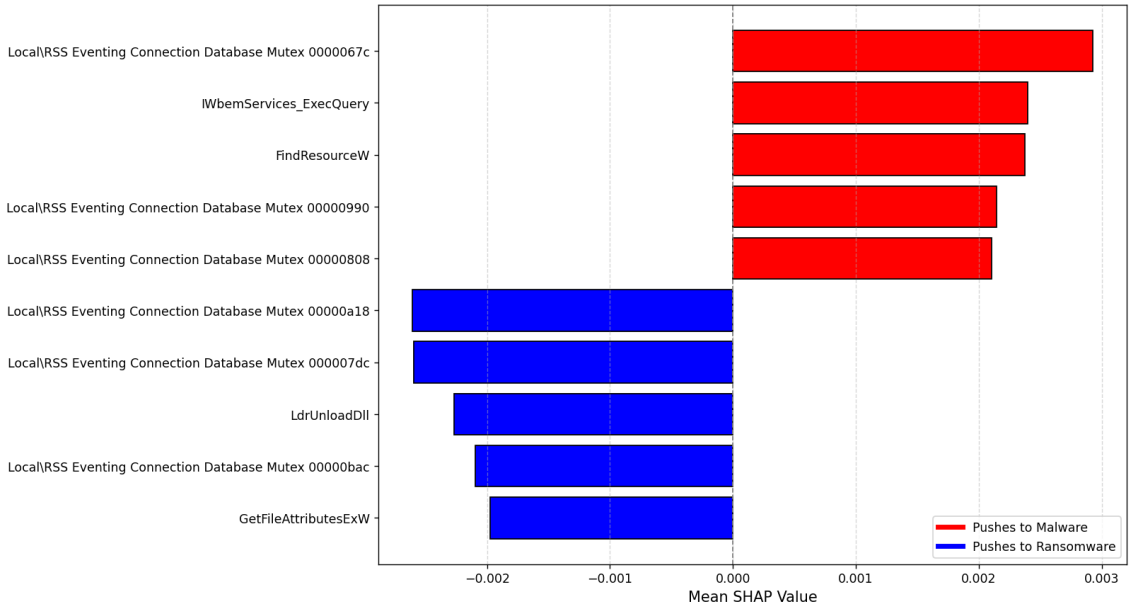
- *Đối với Malware:*

Đặc trưng cũng có khả năng là Malware:

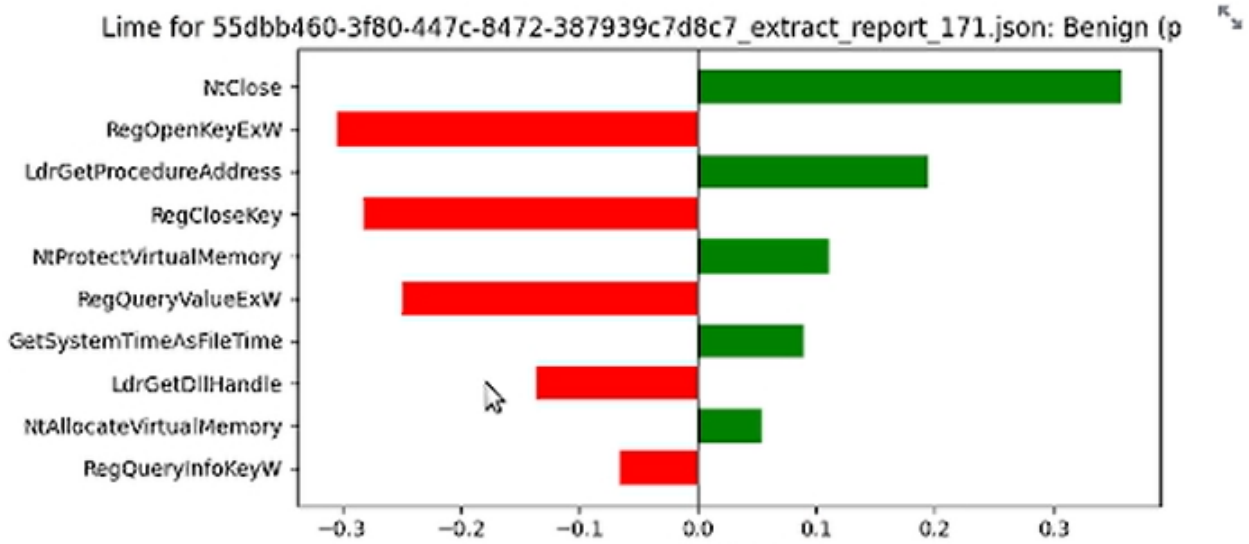
Loại	Đặc trưng	Nhận xét chuyên sâu
Mutex	`Local\RSS Eventing Mutex 0000067c`	Mutex này không phổ biến trong phần mềm hợp pháp. Việc xuất hiện nhiều biến thể gần giống nhau cho thấy khả năng cao đây là mẫu malware được pack hoặc nhân bản tự động.
API	`IWbemServices_ExecQuery`	API WMI thường dùng để thu thập thông tin hệ thống – kỹ thuật trinh sát phổ biến trong malware.
API	`FindResourceW`	Dù là API hệ thống hợp lệ, nhưng nếu được dùng bất thường hoặc lặp lại, có thể là dấu hiệu malware đang unpack tài nguyên từ chính nó.
Mutex	`Local\RSS Eventing Mutex 00000990` , `00000808`	Việc có nhiều mutex ngẫu nhiên dạng này cho thấy phần mềm có hành vi tự động hóa/phân nhánh tiến trình – dấu hiệu tiềm tàng của malware.

Đặc trưng phản đối khả năng là Malware:

Loại	Đặc trưng	Nhận xét
Mutex	00000a18 , 000007dc , 00000bac	Xuất hiện nhiều trong ransomware – làm mô hình nghiêng về Ransomware thay vì Malware.
API	LdrUnloadDll , GetFileAttributesExW	Là các API phổ biến trong phần mềm hợp pháp, hoặc có mặt trong cả benign lẫn ransomware, không đặc trưng cho malware thông thường.



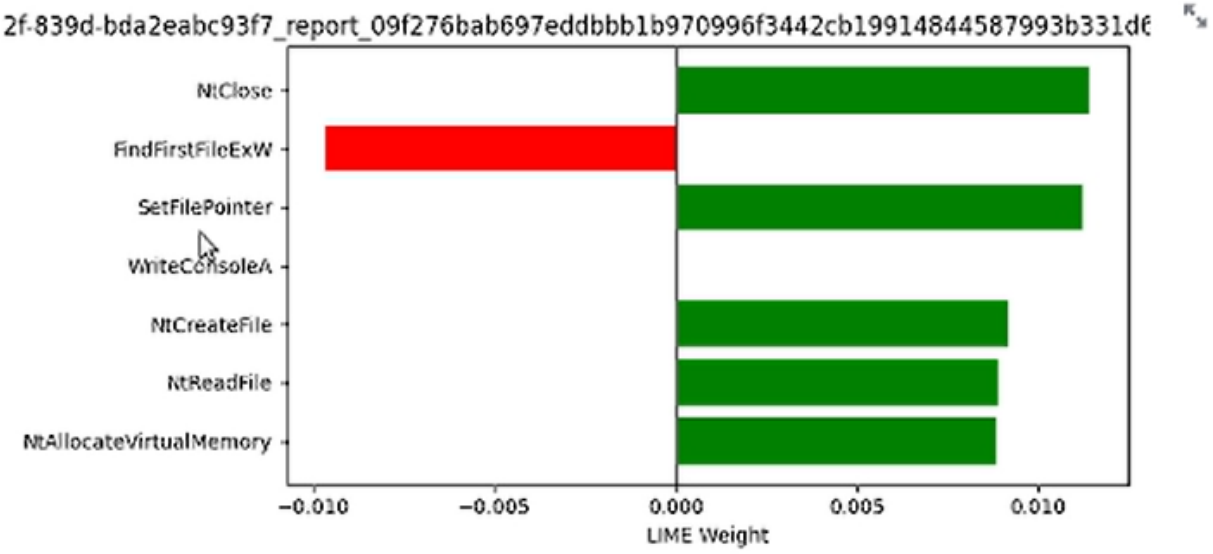
- **Kết quả phân tích LIME của một số đối tượng:**



- API củng cố quyết định mẫu là Benign:
  - **NtClose**: trong mẫu này API này là hành vi dọn dẹp hợp lệ.
  - **LdrGetProcedureAddress**: load hàm động, thường gặp ở benign apps.

- **NtProtectVirtualMemory, GetSystemTimeAsFileTime**: không đặc trưng độc hại.
- **NtAllocateVirtualMemory, RegQueryInfoKeyW**: trong mẫu này 2 API này không đặc trưng độc hại. (Do chuỗi API được gọi không đáng nghi).

- API phản đối quyết định mẫu là Benign:
  - **RegOpenKeyExW, RegCloseKey, RegQueryValueExW**: can thiệp registry (hành vi phổ biến của malware).
  - **LdrGetDllHandle**: thường dùng để load DLL, có thể xuất hiện trong cả malware.
  - Tuy nhiên, các API "nghi ngờ" này không đủ mạnh để lấn át phần "benign", nên tổng thể mô hình vẫn phân loại mẫu là Benign.
- **Kết luận:**
  - Mẫu này có một vài API đáng nghi, chủ yếu là thao tác registry (Reg*), nhưng:
    - Các hành vi hợp lệ như NtClose, NtProtectVirtualMemory, Ldr*, GetSystemTimeAsFileTime xuất hiện nhiều hơn và có ảnh hưởng lớn hơn.
    - Ngoài ra mẫu này không có những đặc trưng quan trọng của ransomware là các thao tác với file.
    - Do đó, mô hình quyết định rằng mẫu này không phải ransomware, mà là benign.



Biểu đồ LIME

- API củng cố quyết định mẫu này là Ransomware:
  - **NtClose**: đóng handle, hành vi phổ biến khi mã độc xử lý nhiều file liên tục.
  - **SetFilePointer**: dịch chuyển con trỏ file, thường dùng trong quá trình ghi mã hóa nội dung.

- **WriteConsoleA**: ghi thông tin ra console, có thể được dùng để hiển thị yêu cầu tiền chuộc.
- **NtCreateFile**: tạo hoặc mở file, một bước cần thiết trước khi mã hóa.
- **NtReadFile**: đọc nội dung file, có thể dùng để phân tích nội dung trước khi mã hóa.
- **NtAllocateVirtualMemory**: cấp phát vùng nhớ mới, thường xuất hiện trong hành vi unpacking hoặc mã tự sửa chính nó.

- API phản đối quyết định mẫu là ransomware:
  - **FindFirstFileExW** — tìm kiếm file hoặc thư mục trên hệ thống. Thường được sử dụng bởi phần mềm hợp pháp để liệt kê file.
- **Kết luận:**
  - Mẫu này có một số hành vi được xem là benign (ví dụ: FindFirstFileExW), nhưng:
  - Các đặc trưng liên quan đến ghi/đọc/điều chỉnh file và bộ nhớ có trọng số cao và mang tính đặc trưng cho ransomware.

➔ Recheck bằng report cuckoo:


```
{
    "category": "file",
    "status": 1,
    "stacktrace": [],
    "api": "NtReadFile",
    "return_value": 0,
    "arguments": {
        "file_handle": "0x0000004c",
        "buffer": "MZ\u0090\u0000\u0003\u
        "length": 28672,
        "offset": 0
    },
    "time": 1747455583.2185,
    "tid": 2288,
    "flags": {}
},
```


- Khi dò trong report của cuckoo ta sẽ thấy hàm **EncryptDumpFile** được ẩn bên trong buffer(Khả năng là hàm này được được sử dụng API resolution để che dấu). Ta có thể nhận định đây là 1 file ransomware.

D. Hướng phát triển

- Kết hợp phân tích tĩnh và động
  - Bổ sung thêm kỹ thuật phân tích tĩnh (static analysis) để tăng khả năng phát hiện sớm, ngay cả trước khi ransomware được thực thi.
  - Tích hợp các đặc trưng như PE header, mã opcode, hoặc graph control flow để mô hình toàn diện hơn.
- Tối ưu mô hình phát hiện thời gian thực
  - Rút gọn mô hình CNN hoặc thay thế bằng kiến trúc nhẹ hơn để phát hiện ransomware theo thời gian thực (real-time detection), phù hợp với hệ thống endpoint hoặc gateway.
- Tăng cường cơ chế giải thích
  - Phát triển bảng điều khiển trực quan (dashboard) với kết quả từ SHAP/LIME dễ hiểu cho người không chuyên.
  - Tích hợp thêm causal inference để tìm mối quan hệ nguyên nhân-hậu quả giữa hành vi và kết quả phát hiện.
- Tự động phản hồi và phục hồi hệ thống
  - Tích hợp hệ thống phản ứng: cách ly tiến trình độc hại, khôi phục file từ backup, hoặc gửi cảnh báo đến SOC.
  - Hướng đến một hệ thống tự phục hồi và chống ransomware tự động (resilient & autonomous defense).

HẾT