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: **File Packing in Malware: Creation and Detection**

Mã nhóm: ***G1****. Mã đề tài:* ***S02***

Lớp: **NT230.N21.ATCL**

1. THÔNG TIN THÀNH VIÊN NHÓM:

STT	Họ và tên	MSSV	Email
1	Nguyễn Mạnh Cường	20520421	20520421@gm.uit.edu.vn
2	Hoàng Văn Anh Đức	20520890	20520890@gm.uit.edu.vn
3	Lê Quang Minh	20520245	20520245@gm.uit.edu.vn

2. TÓM TẮT NỘI DUNG THỰC HIỆN:

A. Chủ đề nghiên cứu trong lĩnh vực Mã độc:

☑ Phát hiện mã độc

B. Liên kết lưu trữ mã nguồn của nhóm:

Mã nguồn của đề tài đồ án được lưu tại:
https://github.com/hiimicebear/Packed_Malware

C. Tên bài báo tham khảo chính:

Aghakhani, H., Gritti, F., Mecca, F., Lindorfer, M., Ortolani, S., Balzarotti, D., ... & Kruegel, C. (2020, January). When malware is packin'heat; limits of machine learning classifiers based on static analysis features. In Network and Distributed Systems Security (NDSS) Symposium 2020.

D. Dịch tên Tiếng Việt cho bài báo:

Khi đóng gói mã độc là xu hướng; hạn chế của các mô hình học máy phân loại dựa trên các thuộc tính phân tích tĩnh.

E. Tóm tắt nội dung chính:

Bài báo nghiên cứu về những hạn chế của các mô hình học máy phân loại dựa trên phân tích tĩnh khi đối mặt với những mẫu mã độc được đóng gói. Bài báo nghiên cứu về những vấn đề, thí nghiệm chứng minh cho vấn đề và đưa ra hướng giải quyết tương đối khái quát, khác với các bài báo đề xuất phương pháp mới cụ thể.

Hiện nay, học máy phân loại dựa trên phân tính tĩnh những tệp thực thi mã độc được ứng dụng phổ biến, nhờ vào sự hiệu quả khi cân bằng giữa hiệu suất và tính bảo mật so với phân tích động. Tuy nhiên, nó cũng gặp phải một số vấn đề, đặc biệt kể đến những mẫu mã độc được đóng gói nhằm tránh né các trình phân loại.

Đóng gói (packing) được thực hiện bởi các trình đóng gói (packer) là phương pháp áp dụng các thuật toán nén và/hoặc mã hóa vào tệp thực thi nhằm ẩn nội dụng và giảm dụng lượng. Việc đóng gói vẫn đảm bảo được chức năng gốc của tệp thực thi đó, nhờ vào các trình giải đóng gói (unpacker) khi tệp được chạy. Do đó, các mô hình phân loại tĩnh khó phát hiện được tính chất độc hại của tệp, khi bản chất mô hình sẽ không chạy thực thi nó.

Tận dụng điều đó, vào khoảng thời gian trước đây, hầu hết các tệp thực thi được đóng gói trong thực tế đều mang nhãn độc hại. Tuy nhiên, thống kê gần đây cho thấy, số lượng phần mềm lành tính áp dụng đóng gói nhằm bảo vệ tài nguyên mã nguồn cho tệp thực thi đã tăng lên đáng kể. Sự thay đổi về tỉ lệ chênh lệch giữa các nhãn nói trên trong thực tế đã đặt ra một số nghi vấn về tính hiệu quả của các giải pháp được cho là có thể chống lại mã độc đóng gói được công bố trước đó.

Vì vậy, tác giả thực hiện một số thí nghiệm kiểm chứng vấn đề trên, và tạo ra 2 datasets nhằm hỗ trợ thí nghiệm và là giải pháp huấn luyện cải thiện các mô hình học máy phân loại để chống lại các mẫu đóng gói.

F. Tóm tắt các kỹ thuật chính được mô tả sử dụng trong bài báo:

Như đã nhắc đến ở trên, tác giả sẽ tạo ra 2 datasets để hổ trợ cho các thí nghiệm chứng minh của mình. Đồng thời, với việc đảm bảo độ chính xác cho các nhãn, trích xuất thuộc tính giá trị và chọn lọc tỉ lệ hợp lý, các datasets sẽ là giải pháp cho việc huấn luyện các mô hình học máy phân loại đạt hiệu quả khi đối mặt với các mẫu được đóng gói. Tất cả các mẫu thực thi trong 2 tập dữ liệu đều thuộc định dạng PE trên hệ điều hành Windows x86.

1. **Wild dataset:** được gọi tên là “hoang dã” vì tập dữ liệu được lấy từ các mẫu tệp thực thi tồn tại trong thực tế.

- Cụ thể, những mẫu được lấy từ: (i) các phần mềm anti-malware thương mại, (ii) EMBER dataset, tập dữ liệu được công bố trong một bài báo trước đây và được sử dụng phổ biến trong nhiều nghiên cứu về sau.
- Tiếp theo, tất cả được gán nhãn malware/benign và packed/unpacked từ những nguồn có độ chính xác cao. Cụ thể, nhãn malware/benign được kiểm tra lại bằng những công cụ có độ tin cậy cao trên Virus Total, sau đó chọn lọc những mẫu có sự thống nhất về nhãn giữa kết quả từ Virus Total và nguồn của mẫu. Nhãn packed/unpacked được gán bằng các phương pháp phân tích động, nhờ vào dấu hiệu giải đóng gói khi chạy tệp, điều đó tăng độ chính xác cho nhãn. Trong đó bao gồm: (i) sandbox từ các anti-malware thương mại, (ii) Deep Packer Inspector, (iii) các công cụ khác (Manalyze, Exeinfo PE, yara rules, PEiD, F-Prot).
- Cuối cùng, cho ra 50,724 mẫu, bao gồm 4,396 unpacked benign, 12,647 packed benign và 33,681 packed malicious. (Các mẫu có nhãn packed malicious bị tác giả loại bỏ do số lượng ít và không cần thiết cho thí nghiệm)

Tool	Benign		Malicious	
	packed	unpacked	packed	unpacked
(1) vendor's sandbox	10,463	16,162	26,699	15,095
(2) *dpi*	6,049	20,576	27,995	13,799
(3) *Manalyze*	1,239	17,436	5,376	19,457
(4) *PEiD+F-Prot*	1,189	25,436	2,630	39,164
(5) *yara*	1,524	25,101	3,882	37,912
(6) *Exeinfo PE*	1,088	25,537	5,770	36,024
(1)+(2)+(3)+(4)+(5)+(6)	12,647	4,396	33,681	5,752

2. **Lab dataset:** tập dữ liệu là tất cả những mẫu từ Wild, nhưng được lần lượt đóng gói lại bằng 9 packers cụ thể, sau đó chọn lọc và loại bỏ một số mẫu không thể xử lý.
- 9 packers bao gồm: Obsidium, PELock, Themida, PECompact, Petite, UPX, kkrunchy, MPRESS, tElock.
- Cuối cùng, cho ra 341,444 mẫu đã đóng gói (với 94.56% mẫu có thể bảo toàn được chức năng).

Packer	# Benign Samples	# Malicious Samples	Keeps Rich Header?	# Invalid Opcodes
Obsidium	16,940	31,492	29.82%	0
Themida	15,895	26,908	✓	0
PECompact	5,610	28,346	✓	723
Petite	13,638	25,857	✓	318
UPX	9,938	20,620	✓	0
kkrunchy	6,811	15,494	19.68%	61
MPRESS	11,041	11,494	19.84%	629
tElock	5,235	30,049	✓	8
PELock	6,879	8,474	20.60%	461
AES-Encrypter	17,042	33,681	✗	0

Ngoài ra, cả 2 datasets trên đều được trích xuất đồng bộ chung một bộ thuộc tính, bao gồm 9 nhóm với tổng cộng 56,543 thuộc tính.

PE headers	28	**Byte n-grams**	13,000
PE sections	570	**Opcode n-grams**	2,500
DLL imports	4,305	**Strings**	16,900
API imports	19,168	**File generic**	2
Rich Header	66		

G. Môi trường thực nghiệm của bài báo:

Trong bài báo, tác giả đã thực hiện **12 thí nghiệm** (chia làm **4 vấn đề** khác nhau) để kiểm chứng về hiệu quả của các mô hình học máy phân loại chống lại các mẫu được đóng gói.

Tác giả không đề cập đến các thông số phần cứng và môi trường thí nghiệm, tuy nhiên, dựa theo mã nguồn tác giả công bố, các thí nghiệm được thực hiện trên hệ điều hành Linux với ngôn ngữ Python. Bên cạnh đó, đây là một số thiết lập chung cho các thí nghiệm:

- Datasets sử dụng như đã đề cập ở **F**. Thuật toán phân loại cho tất cả các thí nghiệm được thống nhất là Random Forest.
- Training và testing được chia theo tỉ lệ 70%-30%. Tỉ lệ nhãn benign-malware trong cả tập train và test chia đều 50-50.
- Tác giả lặp lại mỗi thí nghiệm 5 lần. Với mỗi lần, tập train-test được chọn ngẫu nhiên khác nhau. Kết quả là trung bình cộng của 5 lần thực hiện.

- Tác giả dùng toàn bộ 56,543 thuộc tính (đã đề cập ở **F**) cho tất cả các thí nghiệm.

Phần **H** sẽ đề cập tóm lược nội dung từng thí nghiệm và kết quả.

H. Kết quả thực nghiệm của bài báo:

Đây là tóm lược nội dung 12 thí nghiệm thuộc 4 vấn đề và kết quả trong bài báo: *(nếu không có đề cập đặc biệt, hầu hết mô hình phân loại giữa malware/benign)*

1. **Vấn đề 1:** Mô hình phân loại có nhầm lẫn một quá trình đóng gói từ một Packer cụ thể là nhãn Độc hại, khi tỉ lệ dữ liệu để huấn luyện mô hình không tối ưu?

- Thí nghiệm 1: Chỉ training với 2 nhãn Packed Malware – Unpacked Benign với Wild dataset.
  Kết quả: *mô hình nhầm lẫn đặc điểm packed là malware, xem tất cả tệp được đóng gói là độc hại.*

- Thí nghiệm 2: Training mô hình phân loại các packers khác nhau với Lab dataset.
  Kết quả: *tỉ lệ đánh giá Accuracy 99.99%, cho thấy việc phân loại các đặc điểm của trình đóng gói là một công việc đơn giản với các mô hình học máy.*

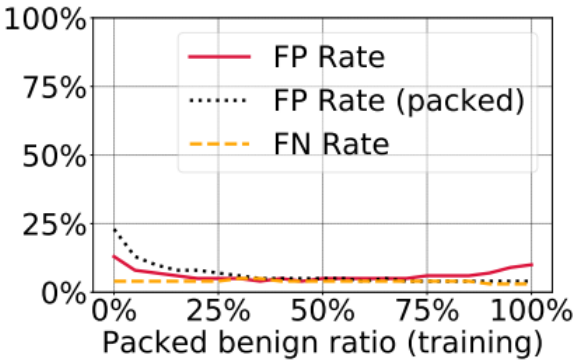
- Thí nghiệm 3: Với Lab dataset, training với chỉ benign từ 4 packers và chỉ malware từ 5 packers còn lại.
  Kết quả: *mô hình nhầm lẫn tất cả các tệp được đóng gói từ 4 packers đều là benign và tất cả các tệp được đóng gói từ 5 packers đều là malware (tập test tỉ lệ đồng đều). Cho thấy mô hình học máy phân loại dễ nhầm lẫn một quá trình đóng gói là benign/malware bất kể tính độc hại thật sự của tệp.*

➔ **Kết luận, giả thuyết từ vấn đề 1 là Chính xác.**

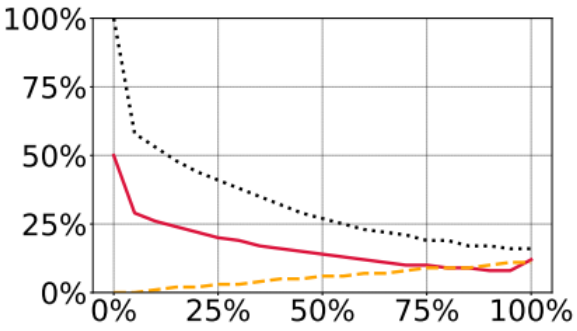
2. **Vấn đề 2:** Vậy mô hình phát hiện mã độc tĩnh có thực sự phân loại được những mẫu đã đóng gói?

- Thí nghiệm 4, 5: Để giải quyết những hạn chế ở vấn đề 1, tác giả thêm nhãn Packed Benign vào tập training với tỉ lệ tăng dần so với tổng các Benign (tức là tăng Packed Benign – giảm Unpacked Benign, vẫn đảm bảo 50-50 cho Benign-Malware) và thực hiện loạt các thí nghiệm với cả 2 datasets. Ở Lab

dataset, vì không có Unpacked Benign trong tập training, tác giả mượn các tệp đó từ tập Wild (tập testing vẫn thuộc tập Lab).
Kết quả: (i) *Wild dataset, duy trì được FP và FN ở mức thấp khi tỉ lệ Packed Benign ~50%,* (ii) *Lab dataset, vì môi trường testing đều là Packed nên FP và FN duy trì được mức thấp khi tỉ lệ Packed Benign ~100% (tức là không cần có Unpacked Benign trong training). Cho thấy, trong môi trường chỉ toàn các tệp được đóng gói, mô hình có thể xác định được tính chất độc hại của tệp. Điều đó rút ra được mặc dù bị đóng gói, các thuộc tính trích xuất tĩnh vẫn hữu ích trong phân loại benign/malware.*



(a) *wild dataset*



(b) *lab dataset*

- Thí nghiệm 6: Đây là thí nghiệm để xác định nhóm thuộc tính nào có giá trị cao trong phân loại, bằng cách thí nghiệm với từng nhóm thuộc tính và từng datasets.
  Kết quả: *nhóm thuộc tính Bytes n-grams và PE sections có giá trị ảnh hưởng cao đến kết quả phân loại, đồng thời nêu ra tầm quan trọng của những nhóm khác là cần thiết trong quá trình phân loại.*

➔ **Kết luận, những thuộc tính trích xuất tĩnh vẫn hữu ích trong quá trình phát hiện tính độc hại của tệp thực thi.**

3. **Vấn đề 3:** Vậy mô hình có thể phân loại hiệu quả với những mẫu được đóng gói từ các packer chưa từng biết, và một số vấn đề khác ở môi trường thực tế không?

- Thí nghiệm 7, 8, 9: Ở những thí nghiệm thuộc vấn đề 2, tác giả chia đều tệp của các packers trong tập training và testing (tức là chỉ test với những quá trình đóng gói từ packers được biết trước trong train). Loạt thí nghiệm này sẽ training mô hình với những mẫu từ một số packers và testing chúng với những mẫu từ các packers khác. Đồng thời, thí nghiệm cũng kiểm chứng

xem các mô hình có phát hiện ra được những đặc điểm đóng gói từ những packer đã đóng gói tệp ở lớp trước đó hay không (tức là một tệp có thể được đóng gói nhiều lần/lớp), bằng cách thay đổi tập train-test từ 2 datasets (tập Lab là tập Wild đã được đóng gói).
Kết quả: *mô hình không thể phân loại đúng được đối với những mẫu đã đóng gói bởi packers chưa từng thấy khi training (kể cả những đặc điểm từ packers ở lớp đóng gói trước/sau).*

- Thí nghiệm 10, 11: Thí nghiệm với những mầu đối kháng (Adversarial Samples) và những mẫu từ những packers mạnh (có những thuật toán mã hóa mạnh). Thí nghiệm ngoài đề tài nên sẽ chỉ đề cập khái quát.
Kết quả: *không hiệu quả với những mẫu đối kháng và và những mẫu từ những packers mạnh.*

➔ **Kết luận, giả thuyết từ vấn đề 3 là Chưa hiệu quả.**

4. **Vấn đề 4:** Những phần mềm phát hiện mã độc thương mại trong thực tế dựa trên học máy phân tích tĩnh có chống lại được các mẫu đóng gói không?

- Thí nghiệm 12: Đưa 6,000 malwares và 6,000 benigns thuộc Lab dataset lên Virus Total để kiểm tra 6 engines phát hiện mã độc (ở đây kiểm nghiệm những engines khác với những engines đã thực hiện gán nhãn cho Wild dataset).
Kết quả: *tỉ lệ FP và FN khá cao.*

Packer	AV1		AV2		AV3		AV4		AV5		AV6	
	FPR	FNR	FPR	FNR	FPR	FNR	FPR	FNR	FPR	FNR	FPR	FNR
PELock	81.48	18.32	72.35	28.26	81.49	19.09	88.34	12.84	89.96	10.29	91.14	8.84
PECompact	81.81	18.36	72.18	28.31	80.53	19.55	88.39	12.06	89.74	10.06	90.88	8.89
Obsidium	79.01	22.24	70.18	30.54	77.03	24.42	83.69	17.42	67.53	32.71	86.55	13.82
Petite	78.73	20.99	68.48	31.07	74.52	25.5	84.79	16.11	72.13	26.92	85.45	14.68
tElock	78.92	20.49	67.49	30.77	74.51	25.06	84.37	14.58	72.53	26.53	84.79	14.17
Themida	79.18	21.58	70.18	30.45	76.95	23.88	84.13	15.95	67.5	33.88	86.23	14.28
MPRESS	82.3	18.58	72.75	28.44	82.04	19.68	88.37	12.73	89.94	9.43	91.58	8.94
kkrunchy	78.79	21.15	69.71	30.32	77.27	23.28	83.11	16.89	67.29	32.07	86.72	13.98
UPX	78.37	22.36	70.04	30.62	76.04	23.72	82.8	17.01	67.34	33.07	85.54	13.98
AES-Encrypter	79.77	21.27	69.14	31.27	75.47	24.4	85.29	15.2	73.25	26.71	85.69	14.19

➔ **Kết luận, một số engines phát hiện mã độc thuộc Virus Total nhầm lẫn đặc điểm đóng gói là độc hại (tương tự vấn đề 1).**

I. Công việc/tính năng/kỹ thuật mà nhóm thực hiện lập trình và triển khai cho demo:

Nhóm thực hiện lại thí nghiệm 1 và thí nghiệm 4 đã đề cập ở **H**. Vì một số hạn chế về phần cứng nên nhóm không thể xử lý Lab dataset, chỉ triển khai được những thí nghiệm chỉ có sử dụng Wild dataset.

Sử dụng Wild dataset tác giả công bố thuộc định dạng .pickle và đọc thành dataframe. Tiền xử lý, chọn lọc tập train-test sao cho đúng tính chất của thí nghiệm gốc. Xây dựng mô hình với loạt các thuật toán có ứng dụng cây quyết định. Đánh giá bằng chi tiết nhiều chỉ số khác nhau.

Kết quả: *2 thí nghiệm đưa ra kết luận giống với bài báo.*

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

Gặp vấn đề khi triển khai lại mã nguồn của tác giả công bố, nhưng đã tự thực hiện lại một số thí nghiệm tương tự bài báo dựa vào datasets của tác giả.

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

99%: tuy không thể triển khai lại mã nguồn tác giả công bố, nhưng đã tự thực hiện lại một số thí nghiệm tương tự bài báo dựa vào datasets của tác giả.

4. NHẬT KÝ PHÂN CÔNG NHIỆM VỤ:

STT	Công việc	Phân công nhiệm vụ
1	Đọc bài báo	Tất cả thành viên
2	Nghiên cứu, tóm lược nội dung bài báo	Nguyễn Mạnh Cường
3	Triển khai thực nghiệm	Hoàng Văn Anh Đức
4	Làm slides thuyết trình, thuyết trình báo cáo	Nguyễn Mạnh Cường
5	Viết báo cáo	Hoàng Văn Anh Đức, Nguyễn Mạnh Cường

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.

MỤC LỤC

I. Phương pháp thực hiện

Sử dụng Wild dataset tác giả công bố.

Thực hiện 2 thí nghiệm, trên môi trường Google Colaboratory với Python 3:

- Thí nghiệm 1 (vấn đề 1):
  - Training: chỉ chọn ra nhãn Unpacked Benign và Packed Malware (50:50) thuộc Wild dataset.
  - Testing: tất cả các nhãn thuộc Wild dataset.
- Thí nghiệm 4 (vấn đề 2):
  - Training: đầy đủ nhãn thuộc Wild dataset. (Benign:Malware = 50:50)
  - Testing: tất cả các nhãn thuộc Wild dataset.

Tập test của cả 2 thí nghiệm hoàn toàn giống nhau (Benign:Malware = 50:50).

Từ 56,543 thuộc tính, chọn lọc lại hơn ~6,000 thuộc tính quan trọng nhất cho mô hình.

Thuật toán của mô hình phân loại bao gồm:

- DecisionTree
- RandomForest
- GradientBoosting
- AdaBoost

Chọn ra mô hình có accuracy cao nhất, đánh giá bằng confusion matrix và các chỉ số accuracy, precision, recall, f1-score.

II. Chi tiết cài đặt, hiện thực

1. Lấy dataset:

Lấy từ nguồn tác giả công bố: https://github.com/ucsb-seclab/packware

2. Thí nghiệm 1:

a. Xử lý dataset:

- Đọc file .pickle thành dataframe.

```
df_wild = pd.read_pickle('/content/drive/MyDrive/wild.pickle')
df_wild.head()
```

	sample_sha1	benign	malicious	packed	unp
1	a2aac23e17be570c647fd22080e05b0e58449565	True	False	True	
2	c3aa534bbd20f3a3dd8e7a457f625291af9d0fe4	False	True	True	
3	bea3fb5a68ac91258e8339a8aa037304ae24242b	True	False	True	
10	15cb9eab6b0f0ccd718a9235f07046dff100789f	False	True	True	
12	8792cf08950abada1572b47948e86c9fce26c01a	True	False	True	

5 rows × 56555 columns

```
print(df_wild.shape)
```

```
(50724, 56555)
```

- Với kích thước lớn như thế, môi trường Colab không thể xử lý nổi. Vậy nên cần cắt giảm bớt dòng, chỉ lấy 12,000/50,724 dòng để thí nghiệm.

```
#Chọn ngẫu nhiên 12,000 mẫu
df = df_wild.sample(n=12000, random_state=102)
```

- *Random_state:* khi lấy ngẫu nhiên, nếu cùng dataframe và chỉ số random_state, sẽ cho ra kết quả random luôn giống nhau, hỗ trợ cho việc đồng bộ giữa các thí nghiệm. (Nhóm chọn 102 xuyên suốt 2 thí nghiệm)

- Loại bỏ một số cột không cần thiết (những cột nhãn, thông tin) và một số cột dạng string không thể xử lý. Cột nhãn còn lại là "malicious" và "packed".

```
#Loại bỏ một số cột không cần thiết
df = df.drop(['sample_sha1','benign','unpacked_sample_sha1','unpacked_sample_id'], axis=1)
df = df.drop(['benign_vt','malicious_vt'], axis=1)

#Loại bỏ một số cột dạng string không thể xử lý
string_columns = df.select_dtypes(include=['object']).columns
df = df.drop(string_columns, axis=1)
```

b. Chia tập train-test:

- Tiếp theo, chia ngẫu nhiên 2,000 mẫu cho tập test, còn lại 10,000 để xử lý cho tập train.

```
df_10000 = df.sample(n=10000, random_state=102)
df_2000 = df.drop(df_10000.index)
```

- Ở tập train, loại bỏ các nhãn Packed Benign, kiểm tra lại số lượng.

```
#Vì tập dữ liệu không có Unpacked Malware, chỉ cần loại bỏ Packed Benign
data = df_10000.drop(df_10000[(df_10000['malicious'] == False) & (df_10000['packed'] == True)].index)

#Kiểm tra số lượng mẫu
UB = data[(data['malicious'] == False) & (data['packed'] == False)].shape[0]
PM = data[(data['malicious'] == True) & (data['packed'] == True)].shape[0]
print('Unpacked Benign: %i samples' % UB)
print('Packed Malware: %i samples' % PM)
print('Total TRAIN set: %i samples'% (UB+PM))
```

```
Unpacked Benign: 851 samples
Packed Malware: 6614 samples
Total TRAIN set: 7465 samples
```

- Vì tỉ lệ số lượng không đều (50:50), bỏ bớt Packed Malware.

```
#Chia tỉ lệ 50 - 50
data_PM = data[(data['malicious'] == True) & (data['packed'] == True)].sample(n=PM-UB, random_state=102)
data = data.drop(data_PM.index)

#Số lượng tập train
UB = data[(data['malicious'] == False) & (data['packed'] == False)].shape[0]
PM = data[(data['malicious'] == True) & (data['packed'] == True)].shape[0]
print('Unpacked Benign: %i samples' % UB)
print('Packed Malware: %i samples' % PM)
print('Total TRAIN set: %i samples'% (UB+PM))
```

```
Unpacked Benign: 851 samples
Packed Malware: 851 samples
Total TRAIN set: 1702 samples
```

- **Tổng cộng, tập train có 1,702 mẫu (UB = PM = 851).**
- Tách cột để đưa vào mô hình.

```
#Xử lý tập train
y = data['malicious'].values
data = data.drop(['malicious','packed'], axis=1)
```

- Ở tập test, bỏ bớt các Packed Malware để đồng đều tỉ lệ Benign:Malware = 50:50. (UB + PB = PM)

```
UB = df_2000[(df_2000['malicious'] == False) & (df_2000['packed'] == False)].shape[0]
PM = df_2000[(df_2000['malicious'] == True) & (df_2000['packed'] == True)].shape[0]
PB = df_2000[(df_2000['malicious'] == False) & (df_2000['packed'] == True)].shape[0]

df_2000_PM = df_2000[(df_2000['malicious'] == True) & (df_2000['packed'] == True)].sample(n=(PM-(UB+PB)), random_state=102)
df_2000 = df_2000.drop(df_2000_PM.index)

UB = df_2000[(df_2000['malicious'] == False) & (df_2000['packed'] == False)].shape[0]
PM = df_2000[(df_2000['malicious'] == True) & (df_2000['packed'] == True)].shape[0]
PB = df_2000[(df_2000['malicious'] == False) & (df_2000['packed'] == True)].shape[0]

print('Unpacked Benign: %i samples' % UB)
print('Packed Malware: %i samples' % PM)
print('Packed Benign: %i samples' % PB)
print('Total TEST set: %i samples'% (UB+PM+PB))
```

```
Unpacked Benign: 163 samples
Packed Malware: 680 samples
Packed Benign: 517 samples
Total TEST set: 1360 samples
```

- **Tổng cộng, tập test có 1,360 mẫu (UB + PB = PM = 680).**

c. <u>Chọn lọc thuộc tính và hoàn thành tập train-test:</u>

- Tạo mô hình và đưa tập train vào để chọn lọc thuộc tính phù hợp.

```
#Chọn lọc features
print('Researching important feature based on %i total features' % data.shape[1])
fsel = ske.ExtraTreesClassifier().fit(data, y)
```

```
Researching important feature based on 56524 total features
```

- Chọn mô hình và chia các vector cho nó, lọc thuộc tính lại theo những thuộc tính chọn lọc được.

```
#Tạo mô hình và chia tập train - test
model = SelectFromModel(fsel, prefit=True)

X_train = model.transform(data)
X_test = model.transform(df_2000.drop(['malicious','packed'], axis=1))
y_train = y
y_test = df_2000['malicious'].values
```

- Sô lượng thuộc tính quan trọng cho thí nghiệm 1 là 6,048.

```
#In ra số lượng features chọn được
nb_features = X_train.shape[1]
features = []
print('%i features identified as important:' % nb_features)

indices = np.argsort(fsel.feature_importances_)[::-1][:nb_features]
print(indices)
```

```
6048 features identified as important:
[42291 41788 42285 ... 41584 55405 50186]
```

- Thông tin của các thuộc tính chọn lọc ra.

```
#In ra chỉ số quan trọng của từng features
for f in range(nb_features):
    print("%d. feature %s (%f)" % (f + 1, data.columns[indices[f]], fsel.feature_importances_[indices[f]]))

#Gán vào tập features để xử lý mô hình
for f in sorted(np.argsort(fsel.feature_importances_)[::-1][:nb_features]):
    features.append(data.columns[f])
```

```
Kết quả truyền trực tuyến bị cắt bớt đến 5000 dòng cuối.
1049. feature ngram_b'+\xfb\x8a\x047\x88' (0.000149)
1050. feature opcode_push sub (0.000149)
1051. feature ngram_b':\x00\\\x00B\x00' (0.000149)
1052. feature opcode_mov mov sub add (0.000149)
```

3. Thí nghiệm 4:

- Thí nghiệm 4 chỉ khác thí nghiệm 1 ở phần chia tập train-test, còn lại tương tự.

```
#Chọn tương tự để có tập test giống thí nghiệm 1
df_10000 = df.sample(n=10000, random_state=102)
df_2000 = df.drop(df_10000.index)
```

- Ở tập train, vì không còn loại đi nhiều dòng nữa, chỉ lấy lại 5,000 dòng để Colab xử lý được.

```
#Vì tập train không còn lọc đi nhiều nữa
#Chọn lại 5,000 mẫu để có thể chạy trên Colab
data = df_10000.sample(n=5000, random_state=102)
```

- Kiểm tra số lượng mẫu.

```
#Kiểm tra số lượng mẫu
UB = data[(data['malicious'] == False) & (data['packed'] == False)].shape[0]
PM = data[(data['malicious'] == True) & (data['packed'] == True)].shape[0]
PB = data[(data['malicious'] == False) & (data['packed'] == True)].shape[0]
print('Unpacked Benign: %i samples' % UB)
print('Packed Malware: %i samples' % PM)
print('Packed Benign: %i samples' % PB)
print('Total TRAIN set: %i samples'% (UB+PM+PB))
```

```
Unpacked Benign: 417 samples
Packed Malware: 3304 samples
Packed Benign: 1279 samples
Total TRAIN set: 5000 samples
```

- Vì tỉ lệ số lượng không đều Benign:Malware = 50:50 (UB + PB = PM), bỏ bớt Packed Malware.

```
#Chia tỉ lệ 50 - 50
data_PM = data[(data['malicious'] == True) & (data['packed'] == True)].sample(n=(PM-(UB+PB)), random_state=102)
data = data.drop(data_PM.index)

#Số lượng tập train
UB = data[(data['malicious'] == False) & (data['packed'] == False)].shape[0]
PM = data[(data['malicious'] == True) & (data['packed'] == True)].shape[0]
PB = data[(data['malicious'] == False) & (data['packed'] == True)].shape[0]
print('Unpacked Benign: %i samples' % UB)
print('Packed Malware: %i samples' % PM)
print('Packed Benign: %i samples' % PB)
print('Total TRAIN set: %i samples'% (UB+PM+PB))
```

```
Unpacked Benign: 417 samples
Packed Malware: 1696 samples
Packed Benign: 1279 samples
Total TRAIN set: 3392 samples
```

- **Tổng cộng, tập train có 3,392 mẫu (UB + PB = PM = 1,696).**

- Ở tập test, xử lý tương tự và cho ra kết quả tương tự thí nghiệm 1.

```
UB = df_2000[(df_2000['malicious'] == False) & (df_2000['packed'] == False)].shape[0]
PM = df_2000[(df_2000['malicious'] == True) & (df_2000['packed'] == True)].shape[0]
PB = df_2000[(df_2000['malicious'] == False) & (df_2000['packed'] == True)].shape[0]

df_2000_PM = df_2000[(df_2000['malicious'] == True) & (df_2000['packed'] == True)].sample(n=(PM-(UB+PB)), random_state=102)
df_2000 = df_2000.drop(df_2000_PM.index)

UB = df_2000[(df_2000['malicious'] == False) & (df_2000['packed'] == False)].shape[0]
PM = df_2000[(df_2000['malicious'] == True) & (df_2000['packed'] == True)].shape[0]
PB = df_2000[(df_2000['malicious'] == False) & (df_2000['packed'] == True)].shape[0]

print('Unpacked Benign: %i samples' % UB)
print('Packed Malware: %i samples' % PM)
print('Packed Benign: %i samples' % PB)
print('Total TEST set: %i samples'% (UB+PM+PB))
```

```
Unpacked Benign: 163 samples
Packed Malware: 680 samples
Packed Benign: 517 samples
Total TEST set: 1360 samples
```

- **Tổng cộng, tập test có 1,360 mẫu (UB + PB = PM = 680).**
- Ở thí nghiệm 4, số lượng thuộc tính chọn lọc được có khác nhau, 6,388 thuộc tính.

```
nb_features = X_train.shape[1]
features = []
print('%i features identified as important:' % nb_features)

indices = np.argsort(fsel.feature_importances_)[::-1][:nb_features]
print(indices)
```

```
6388 features identified as important:
[41967 41777 41827 ...  3771 26119 12502]
```

III. Kết quả thực nghiệm

1. Thí nghiệm 1 (vấn đề 1):

- Chọn 4 thuật toán cho mô hình cùng một số chỉ số cấu hình.

```
#Chọn 4 thuật toán
algorithms = {
        "DecisionTree": tree.DecisionTreeClassifier(max_depth=20),
        "RandomForest": ske.RandomForestClassifier(n_estimators=100),
        "GradientBoosting": ske.GradientBoostingClassifier(n_estimators=100),
        "AdaBoost": ske.AdaBoostClassifier(n_estimators=200)
    }
results = {}
```

- Xử lý lần lượt với 4 thuật toán. Kết quả test được bằng Accuracy.

```
#Train - test mô hình với lần lượt 4 thuật toán
print("Now testing algorithms")
for algo in algorithms:
    clf = algorithms[algo]
    clf.fit(X_train, y_train)
    score = clf.score(X_test, y_test)
    print("%s : %f %%" % (algo, score*100))
    results[algo] = score
```

```
Now testing algorithms
DecisionTree : 84.264706 %
RandomForest : 87.205882 %
GradientBoosting : 87.205882 %
AdaBoost : 86.617647 %
```

- Random Forest có kết quả đánh giá cao nhất.

```
#Chọn mô hình có Acc lớn nhất
winner = max(results, key=results.get)
print('Algorithm with highest accuracy on train/test is %s with a %f %% success' % (winner, results[winner]*100))
```

```
Algorithm with highest accuracy on train/test is RandomForest with a 87.205882 % success
```

- Các thông số đánh giá khác của Random Forest.

```
#In ra các chỉ số đánh giá khác của mô hình
clf = algorithms[winner]
res = clf.predict(X_test)
mt = confusion_matrix(y_test, res)

print("True positive : %i " % (mt[1][1]))
print("True negative : %i " % (mt[0][0]))

print("False positive : %i " % (mt[0][1]))
print("False negative : %i " % (mt[1][0]))
```

```
True positive : 646
True negative : 540
False positive : 140
False negative : 34
```

```
report = classification_report(y_test, res)
print("Classification Report:\n", report)
```

```
Classification Report:
              precision    recall  f1-score   support

       False       0.94      0.79      0.86       680
        True       0.82      0.95      0.88       680

    accuracy                           0.87      1360
   macro avg       0.88      0.87      0.87      1360
weighted avg       0.88      0.87      0.87      1360
```

- *Nhận xét:*
  - Tuy kết quả độ chính xác khá cao, nhưng tỉ lệ FP và FN vẫn chưa được tối ưu.
  - FP ~17.81% (bài báo FP ~23.40%).
  - Mô hình nhầm lẫn đặc điểm packed là malware, xem hầu hết tệp được đóng gói là độc hại.

2. Thí nghiệm 4 (vấn đề 2):

- Vẫn sử dụng 4 thuật toán như trên và cùng chỉ số cấu hình.

```
algorithms = {
        "DecisionTree": tree.DecisionTreeClassifier(max_depth=20),
        "RandomForest": ske.RandomForestClassifier(n_estimators=100),
        "GradientBoosting": ske.GradientBoostingClassifier(n_estimators=100),
        "AdaBoost": ske.AdaBoostClassifier(n_estimators=200)
    }

results = {}
```

- Tiến hành xử lý.

```
print("Now testing algorithms")
for algo in algorithms:
    clf = algorithms[algo]
    clf.fit(X_train, y_train)
    score = clf.score(X_test, y_test)
    print("%s : %f %%" % (algo, score*100))
    results[algo] = score
```

```
Now testing algorithms
DecisionTree : 92.205882 %
RandomForest : 94.411765 %
GradientBoosting : 94.411765 %
AdaBoost : 94.926471 %
```

- Đưa ra kết quả của thuật toán Random Forest để so sánh công bằng với thí nghiệm 1.

```
clf = algorithms['RandomForest']
res = clf.predict(X_test)
mt = confusion_matrix(y_test, res)

print("True positive : %i " % (mt[1][1]))
print("True negative : %i " % (mt[0][0]))

print("False positive : %i " % (mt[0][1]))
print("False negative : %i " % (mt[1][0]))
```

```
True positive : 646
True negative : 638
False positive : 42
False negative : 34
```

```
report = classification_report(y_test, res)
print("Classification Report:\n", report)
```

```
Classification Report:
              precision    recall  f1-score   support

       False       0.95      0.94      0.94       680
        True       0.94      0.95      0.94       680

    accuracy                           0.94      1360
   macro avg       0.94      0.94      0.94      1360
weighted avg       0.94      0.94      0.94      1360
```

- *Nhận xét:*
  - So với thí nghiệm 1, kết quả độ chính xác đã tối ưu đáng kể.
  - FP và FN duy trì được mức thấp (<10%).
  - Những thuộc tính trích xuất tĩnh vẫn hữu ích trong quá trình phát hiện tính độc hại của tệp thực thi.

IV. Hướng phát triển

- Bài báo tập trung hơn vào việc chứng minh các vấn đề đang tồn tại và đưa ra một hướng giải quyết mở đối với các mô hình học máy phân tích tĩnh khi đối mặt với mẫu đóng gói.
- Với đề xuất của tác giả, những mô hình học máy cần chú trọng hơn vào việc training mô hình phân loại, cũng như chọn dataset có độ chính xác cao. Hơn nữa, đóng gói là yếu tố cần phải được cân nhắc và đề cập đến khi đưa ra những nghiên cứu cho các mô hình phân loại mã độc.

- Tuy với kết luận từ vấn đề 3 cho thấy hạn chế khó giải quyết của các mô hình, nhưng sự tối ưu hóa ở vấn đề 2 là cần thiết và khả thi nhất trong thời điểm hiện tại khi có thể giải quyết vấn đề 1.
- Hướng giải quyết khả thi cho vấn đề 3 trong tương lai là cập nhật liên tục những mẫu từ packers mới cho mô hình phân loại. Ngoài ra, có thể đưa ra giải pháp để thống kê được những tính chất độc hại còn sót lại sau khi đóng gói của các packers, từ đó có thể áp dụng và cải thiện độ chính xác khi đối mặt với các mẫu từ packes mới.

HẾT