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: MAB-Malware – Một khung công cụ học tăng cường tấn công vào bộ phân loại phần mềm tĩnh

Mã nhóm: G11 Mã đề tài: S9

Lớp: NT230.O21.ANTT

1. THÔNG TIN THÀNH VIÊN NHÓM:

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

STT	Họ và tên	MSSV	Email
1	Nguyễn Triệu Thiên Bảo	21520155	21520155@gm.uit.edu.vn
2	Nguyễn Lê Thảo Ngọc	21521191	21521191@gm.uit.edu.vn
3	Trần Lê Minh Ngọc	21521195	21521195@gm.uit.edu.vn

2. TÓM TẮT NỘI DUNG THỰC HIỆN:[1]

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

☐ Phát hiện mã độc

☑ Đột biến mã độc

☐ Khác: ……

B. Liên kết lưu trữ mã nguồn của nhóm:
Mã nguồn của đề tài đồ án được lưu tại:
https://drive.google.com/file/d/1ayExwPDV9huxSuDaXVTZb1uL6HQkhy0I/view?usp=sharing

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

C. Tên bài báo tham khảo chính:

Song, W., Li, X., Afroz, S., Garg, D., Kuznetsov, D., & Yin, H. (2020). "Mab-malware: A reinforcement learning framework for attacking static malware classifiers". arXiv preprint arXiv:2003.03100

D. Dịch tên Tiếng Việt cho bài báo:

[1] Ghi nội dung tương ứng theo mô tả

MAB-Malware – Một framework dựa trên học tăng cường để thực hiện tấn công các bộ phân loại mã độc tĩnh

E. Tóm tắt nội dung chính:

Trong thời đại số hóa hiện nay, các hệ thống antivirus hiện đại ngày càng phụ thuộc vào machine learning để đối phó với sự gia tăng nhanh chóng của các phần mềm độc hại mới. Tuy nhiên, nghiên cứu đã chỉ ra rằng các mô hình học máy dễ bị tấn công bởi các mẫu đối kháng (AEs – Adversarial Examples). Để giải quyết vấn đề này, nhóm đề xuất một framework MAB-Malware học tăng cường dựa trên MAB để tạo ra các mẫu đối kháng cho phần mềm độc hại PE Black-box, thực hiện các cuộc tấn công đối kháng vào các mô hình học máy tiên tiến có khả năng phân loại mã độc và các phần mềm chống vi-rút thương mại hàng đầu mà không làm biến đổi chức năng ban đầu của mã độc. Đồng thời cung cấp cái nhìn quan trọng về mô hình có trạng thái so với không có trạng thái, mô hình nhận thức nội dung so với không nhận thức nội dung, và các hành động dư thừa so với cần thiết. Nhóm sẽ thực hiện các thực nghiệm để có thể lập luận rằng một mô hình không có trạng thái, chọn lựa nội dung (content) một cách cẩn thận và một quy trình giảm thiểu hành động (action) là phù hợp hơn cho việc tạo ra phần mềm độc hại PE đối kháng.

MAB-Malware bao gồm hai module chính: Binary Rewriter và Action Minimizer. Binary Rewriter sử dụng phương pháp Thompson sampling để chọn chuỗi hành động và viết lại mẫu mã độc để tạo ra các biến thể mã độc mới. Action Minimizer loại bỏ các hành động không cần thiết và sử dụng các micro-action thay thế các macro-action để tạo ra một mẫu gian lận "tối giản" chỉ thay đổi tối thiểu các đặc điểm để tránh bị phát hiện.

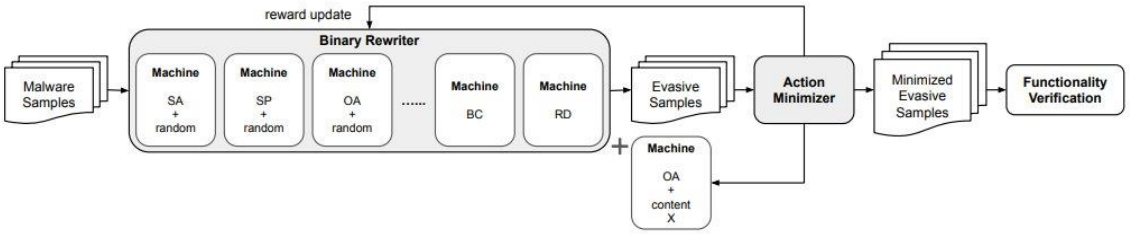
F. Tóm tắt các kỹ thuật chính được mô tả sử dụng trong bài báo:

Loại malware được thảo luận trong bài báo này là các file malware PE, tức là malware dưới dạng các tệp Portable Executable (PE) nhắm vào hệ điều hành Windows. Tài liệu tập trung vào việc né tránh các bộ phân loại malware tĩnh được sử dụng trong các hệ thống antivirus thương mại thông qua các cuộc tấn công đối kháng vào các mẫu malware này.

Mô tả ngắn gọn cách đột biến mã độc được sử dụng trong bài báo:

- X là tập dữ liệu malware.
- f là một bộ phân loại nhị phân cho từng mẫu $x \in$ X.
- Một tập hành động A = {$a1, a2, \ldots an$} được sử dụng để biến đổi các mẫu phần mềm độc hại.
- Một mẫu đối kháng $x' = t(x)$ được tạo ra bằng cách áp dụng một hàm biến đổi t, là một chuỗi hành động được lấy mẫu từ tập A.
- Hàm biến đổi t tuân theo ràng buộc rằng $t(x)$ không thay đổi chức năng của x.

- Loại bỏ các hành động không cần thiết và sử dụng các micro-action để thay thế các macro-action, tạo ra một mẫu gian lận "tối giản" chỉ thay đổi các đặc điểm tối thiểu.



Hình 1. Workflow của mô hình đột biến mã độc

Cụ thể bài báo sử dụng các kỹ thuật chính như sau:

- Kỹ thuật 01: Reward Propagation. Kỹ thuật này nhằm khuyến khích khám phá các loại hành động nhất định.
- Kỹ thuật 02: Binary Rewriter gồm 13 action được phân thành macro-actions và micro-actions. Kỹ thuật này được áp dụng nhằm tạo ra các mẫu mã độc đối kháng từ tập mẫu mã độc gốc bằng cách biến đổi các chuỗi hành động.

Type	Abbr	Name	Description
Macro	OA	Overlay Append	Thêm nội dung lành tính vào cuối file.
	SP	Section Append	Thêm byte ngẫu nhiên vào khoảng trống vào cuối section.
	SA	Section Add	Thêm section mới với nội dung lành tính.
	SR	Section Name	Thay đổi tên section trong file lành tính.
	RC	Remove Certificate	Loại bỏ chữ ký số khỏi file.
	RD	Remove Debug	Loại bỏ thông tin debug.
	BC	Break Checksum	Loại bỏ giá trị checksum khỏi header.
	CR	Code Randomization	Thay đổi các câu lệnh instruction trong mã assembly.
Micro	OA1	Overlay Append 1 Byte	Thêm một byte vào cuối file.
	SP1	Section Append 1 Byte	Thêm một byte vào khoảng trống cuối section.
	SA1	Section Add 1 Byte	Thêm section mới có nội dung một byte.
	SR1	Section Rename 1 Byte	Thay đổi một byte trong section name.
	CP1	Code Section Append 1 Byte	Thêm một byte vào khoảng trống vào cuối code section.

Hình 2. Tập hành động (Action set)

Algorithm 1 Adversarial Attack

Input: malware sample set X
Output: adversarial example set X_a

```
initialize(M)
X_a ← []
for all x ∈ X do
    list_M ← []
    for all attempt_idx ← 1 to N do
        M ← max(betaSampling(M))
        x' ← apply(x, M)
        list_M.add(M)
        if isEvasive(x') then
            x'_min, list_M_min ← minimize(x, list_M)
            X_a.add(x'_min)
            for all M' ∈ list_M_min do
                incAlpha(M')
                if isGeneric(M') then
                    M'_s ← createSpecificMachine(M')
                    M.add(M'_s)
                else
                    M'_g ← getParentGeneric(M')
                    incAlpha(M'_g)
                end if
            end for
            break
        else
            incBeta(M)
        end if
    end for
end for
return X_a
```

Hình 3. Thuật toán được sử dụng trong Binary Rewriter

- Kỹ thuật 03: Action Minimizer. Đây là kỹ thuật giúp tăng khả năng trốn tránh của mã độc thông qua việc giảm tối đa sự biến đổi trên mã độc mà vẫn giữ nguyên các tính năng ban đầu của nó. Kỹ thuật này được thực hiện bằng cách loại bỏ các hành động dư thừa và thay thế macro-action bằng micro-action, tức là thay thế các đặc điểm mã độc mà các classifiers dùng để nhận biết mã độc.

Algorithm 2 Minimize

Input: malware sample x, applied machines $list\_M$
Output: minimized AE x'_{min}, minimized machines $list\_M_{min}$

```
list_M_min ← list_M
for all M ∈ List_M do
    list_M' ← List_M_min − M
    x' ← apply(x, list_M')
    if isEvasive(x') then
        list_M_min ← List_M'
        x'_min ← x'
    else
        list_micro ← get_micro_actions(M)
        for all M_mic ∈ list_micro do
            list_M' ← List_M_min − M + M_mic
            x' ← apply(x, list_M')
            if isEvasive(x') then
                list_M_min ← List_M'
                x'_min ← x'
                break
            end if
        end for
    end if
end for
return x'_min, list_M_min
```

Hình 4. Thuật toán được sử dụng trong Action Minimizer

G. Môi trường thực nghiệm của bài báo:

Cấu hình máy tính: sử dụng 20 máy ảo trên nền tảng cloud của Microsoft Azure. Mỗi máy ảo gồm 8Ram và 2vcpu.

Các công cụ hỗ trợ sẵn có giúp phân loại và phát hiện mã độc: 3 phần mềm antivirus thương mại được ẩn danh là AV1, AV2 và AV3 (bản free và setting), 2 mô hình học máy có khả năng phát hiện mã độc (EMBER và MalConv). Mỗi phần mềm diệt virus được cài đặt trên một máy ảo Azure với Windows 7 Version 6.1 Build 7601 (Service Pack 1).

Ngôn ngữ lập trình để hiện thực phương pháp là Python 3.6.9.

Sử dụng thư viện pefile và IDA Pro 6.8 trong giai đoạn Binary Rewriter.

Đối tượng nghiên cứu (chương trình phần mềm dùng để kiểm tra tính khả thi của phương pháp/tập dữ liệu – nếu có): sử dụng 5000 mẫu Windows PE binary lấy từ VirusTotal với tỉ lệ hơn 80% được gán nhãn mã độc

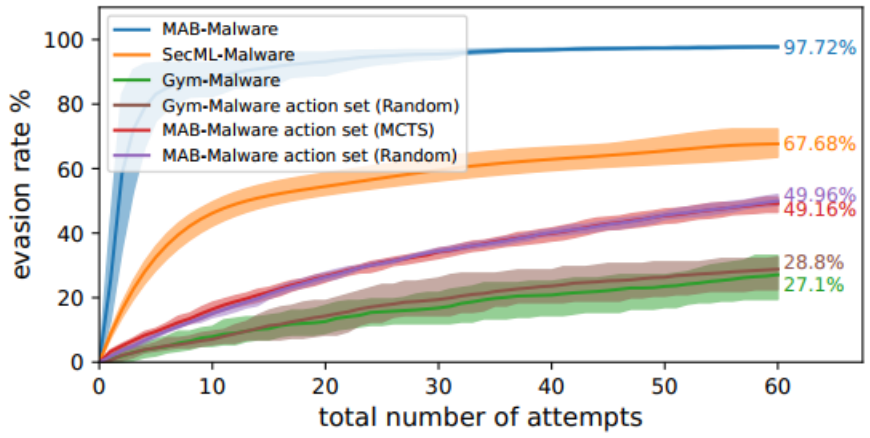
Tiêu chí đánh giá tính hiệu quả của phương pháp:

1. Dựa trên khả năng trốn tránh của mã độc đối kháng.
Mã độc đối kháng do MAB-Malware tạo ra sẽ được so sánh với các mã đối kháng của SecML-Malware và Gym-Malware lần lượt trong 3 môi trường phát hiện mã độc: phần mềm Antivirus chuyên dụng, Ember, MalConv
2. Dựa trên khả năng giữ nguyên bản chức năng mã độc
Sử cuckoo sandbox để phân tích các mã độc đối kháng của Gym-Malware và MAB-Malware với mã độc gốc cùng loại.

H. Kết quả thực nghiệm của bài báo:

1. Kết quả thực nghiệm của bài báo

a) Theo tiêu chí dựa trên khả năng trốn tránh của mã độc đối kháng



Hình 5. Khả năng trốn tránh của mẫu đối kháng

Từ thống kê ở *Hình 5*, chúng ta thấy MAB-Malware có khả năng trốn tránh vượt trội hơn so với các framework khác. Nó có thể tạo ra các mẫu đối kháng 97.72%

mẫu để trốn tránh MalConv, và 74.4% mẫu để trốn tránh EMBER. So sánh với các thuật toán khác, nhóm tác giả quan sát thấy rằng MAB-Malware tạo ra nhiều mẫu đối kháng hơn so với các framework sẵn có khác.

Vì commercial antivirus là môi trường chỉ có thể thực hiện tấn công blackbox nên tỉ lệ trốn tránh thấp hơn. Tuy nhiên nhìn chung thì MAB-Malware vẫn cho kết quả cao hơn các framework khác.

b) Dựa trên khả năng giữ nguyên bản mã độc

Actions	Functional Rate	
	Gym-Malware Actions	MAB-Malware Actions
(OA) Overlay Append	45/48 (93.75%)	46/48 (95.83%)
(SP) Section Append	11/47 (23.40%)	42/43 (97.67%)
(SA) Section Add	11/47 (23.40%)	39/42 (92.86%)
(SR) Section Rename	11/47 (23.40%)	42/43 (97.67%)
(RC) Remove Certificate	1/3 (33.33%)	3/3 (100.00%)
(RD) Remove Debug	5/13 (38.46%)	13/13 (100.00%)
(BC) Break Checksum	9/48 (18.75%)	32/33 (96.97%)
Average	**93/253 (36.76%)**	**217/225 (96.44%)**

Hình 6. Khả năng bảo tồn chức năng của các Action

Từ *Hình 6*, chúng ta có thể thấy rằng ngoại trừ hành động OA, hầu hết các hành động khác do Gym-Malware thực hiện khiến 63,24% mẫu mã độc bị thay đổi hoặc mất chức năng. Trong khi, chỉ có 4,56% mẫu không còn giữ nguyên chức năng ban đầu khi bị tác động bởi các hành động do MAB-Malware tạo ra. Việc không duy trì được các chức năng gốc của mã độc có thể làm tệp nhị phân bị hỏng nên đây là vấn đề quan trọng đáng quan tâm. Vì vậy, MAB-Malware đã thực hiển rất tốt chức năng này.

2. Ưu và nhược điểm của phương pháp được đề cập trong bài báo

c) Ưu điểm

- Tỷ lệ né tránh cao

Khung này đạt tỷ lệ né tránh rất cao (trên 75%) đối với các mô hình học máy và vượt trội so với các thuật toán tạo mẫu đối kháng Black-box hiện có.

- Mô hình không trạng thái

Mô hình sử dụng cách tiếp cận không trạng thái để tạo AE, coi mỗi hành động là độc lập. Điều này cho phép quá trình học nhanh hơn và tạo AE hiệu quả hơn so với mô hình có trạng thái.

- Mô hình hóa nội dung

Framework này nhấn mạnh tầm quan trọng của nội dung liên quan đến hành động (cặp Action-Content). Nếu một nội dung cụ thể đã chứng minh được hiệu quả trong việc tạo AE thành công, nó sẽ có khả năng hữu ích trong các cuộc tấn công tương lai.

- Tính tổng quát

Framework đề xuất có thể được áp dụng cho bất kỳ bộ phân loại malware nào miễn là nó trả về nhãn cho các mẫu thử nghiệm. Nó đã được thử nghiệm thành công trên nhiều bộ phát hiện khác nhau, thể hiện khả năng áp dụng rộng rãi.

d) Nhược điểm

Một nhược điểm của MAB-Malware framework là nó chỉ tập trung vào việc né tránh tĩnh của các mẫu malware và có thể không hiệu quả đối với các phương pháp phát hiện động. Hạn chế này có thể tạo ra một lỗ hổng cho kẻ tấn công để qua mặt các cơ chế phát hiện động được triển khai trong hệ thống antivirus.

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

Nhóm đã thực hiện lại MAB-Malware framework trên hai mô hình phân loại mã độc tĩnh là EMBER và Malconv dựa trên kiến trúc đã mô tả.

Kết quả cho ra các mẫu đối kháng có khả năng trốn tránh vượt trội so với các framwork hiện có (trên 80% cho cả hai mô hình trên).

Chi tiết sẽ được mô tả ở phần báo cáo chi tiết.

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

Trong quá trình thực hiện lại framework trên, nhóm gặp một số khó khăn như sau:

- Chưa thử nghiệm được sample Malware lớn.

Bài báo thực hiện thử nghiệm trên 1000 sample và nhóm cũng vậy những là dataset khác. Do dataset khác nhau nên có thể dẫn đến sai sót và điểm đánh giá cũng có một chút chênh lệch.

- Chưa thử trên các antiviruss thương mại.

Nhóm chỉ mới thực hiện trên hai mô hình Malware Classifier là EMBER và Malconv, vẫn chưa thể cấu hình và thực hiện thử nghiệm trên các mô hình antiviruss trên thị trường như của bài báo.

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

80% (do chưa thực hiện được trên mô hình antiviruss thương mại)

4. NHẬT KÝ PHÂN CÔNG NHIỆM VỤ:

STT	Công việc	Phân công nhiệm vụ
1	Tìm hiểu bài báo	Thiên Bảo Thảo Ngọc Minh Ngọc
2	Phân tích các phương pháp thực hiện	Thảo Ngọc
3	Triển khai hệ thống	Thiên Bảo Thảo Ngọc Minh Ngọc
4	Đánh giá hệ thống	Thiên Bảo Minh Ngọc

BÁO CÁO TỔNG KẾT CHI TIẾT

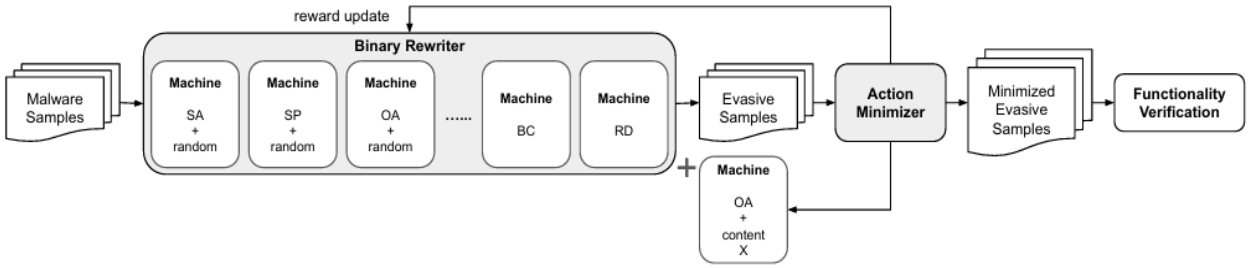
Phần bên dưới của báo cáo này là tài liệu báo cáo tổng kết - chi tiết của nhóm thực hiện cho đề tài này.

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

- Adversarial Goal - Mục tiêu đối kháng: thay đổi mẫu phần mềm độc hại để tránh sự phát hiện của các bộ phân loại PE tĩnh.
- Adversarial Capabilities - Khả năng đối kháng: chỉ được thực hiện những thay đổi "nhỏ" vào mẫu gốc để giữ cho sự can thiệp khó nhận biết mà vẫn đảm bảo tính năng của mẫu gốc.
- Adversarial Knowledge - Kiến thức đối kháng: chỉ có quyền truy cập Black-box, nghĩa là không biết gì về cấu tạo bên trong mã độc.

<Trình bày kiến trúc, thành phần của hệ thống trong bài báo>



Hình 7. Kiến trúc của framework MAB-Malware

Framework MAB-malware gồm 2 module chính: Binary Rewriter và Action Minimizer.

Phương pháp chính xây dựng MAB-Malware được mô tả như phần F. Tóm tắt các kỹ thuật chính được mô tả sử dụng trong bài báo ở trên.

1. Binary Rewriter

Trong framework MAB-Malware, thuật toán MAB sẽ được áp dụng để xây dựng module Binary Rewriter.

Module này gồm 13 chuỗi hành động được phân thành Macro-action và Micro-action. Các hành động này đều dùng để biến đổi mã độc nguyên bản thành mã đối kháng bởi vì với các mẫu mã độc, thay đổi một byte có thể phá vỡ định dạng hợp lệ của PE hoặc phá vỡ chức năng độc hại ban đầu. Đây là lý do tại sao những người phát triển phần mềm độc hại thường không trực tiếp sửa đổi byte thô của tệp PE mà thay vào đó, họ xây dựng một tập hợp các hành động.

- Nhóm hành động Macro-action sẽ sử dụng thư viện pefile để viết lại tệp nhị phân nhưng không làm phá huỷ chức năng ban đầu của mã độc cũng như sửa chữa lại những corner case có khả năng này.

- Nhóm hành động micro-action: các action này khá tương tự với macro-action tương ứng nhưng giảm thiểu sự thay đổi lên mã độc hơn bằng cách chỉ tác động lên 1 Byte của đối tượng.

Ví dụ:

Nếu hành động a gây ảnh hưởng tới k đặc điểm $\{f_1, f_2, f_3, \dots, f_k\}$ của mã độc và hành động b tác động tới 1 vài đặc điểm thuộc k. Vậy thì a là macro-action, còn b là micro-action.

Type	Abbr	Name	Description
Macro	OA	Overlay Append	Thêm nội dung lành tính vào cuối file.
	SP	Section Append	Thêm byte ngẫu nhiên vào khoảng trống vào cuối section.
	SA	Section Add	Thêm section mới với nội dung lành tính.
	SR	Section Name	Thay đổi tên section trong file lành tính.
	RC	Remove Certificate	Loại bỏ chữ ký số khỏi file.
	RD	Remove Debug	Loại bỏ thông tin debug.
	BC	Break Checksum	Loại bỏ giá trị checksum khỏi header.
	CR	Code Randomization	Thay đổi các câu lệnh instruction trong mã assembly.
Micro	OA1	Overlay Append 1 Byte	Thêm một byte vào cuối file.
	SP1	Section Append 1 Byte	Thêm một byte vào khoảng trống cuối section.
	SA1	Section Add 1 Byte	Thêm section mới có nội dung một byte.
	SR1	Section Rename 1 Byte	Thay đổi một byte trong section name.
	CP1	Code Section Append 1 Byte	Thêm một byte vào khoảng trống vào cuối code section.

Hình 8. Tập hành động (Action set)

Các đặc trưng bị ảnh hưởng bởi các hành động trên.

		CR	OA	SP	SA	SR	RC	RD	BC	OA1	SP1	SA1	SR1	CP1
Hash-Based Signatures	F_1: File Hash	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	F_2: Section Hash	✓		✓				✓			✓			✓
Rule-based Signatures	F_3: Section Count				✓							✓		
	F_4: Section Name					✓							✓	
	F_5: Section Padding			✓										
	F_6: Debug Info							✓						
	F_7: Checksum								✓					
	F_8: Certificate						✓							
	F_9: Code Sequence	✓												
Data Distribution	F_{10}: Data Distribution		✓		✓									

Hình 9. Các đặc trưng bị ảnh hưởng

a) Luồng làm việc của Binary Rewriter

Với 1 tập mẫu mã độc X được dùng làm đầu vào, module này sẽ tạo ra 1 tập mẫu đối kháng X'_{min} với những thay đổi ít nhất có thể nhưng vẫn trốn tránh được các classifier.

Algorithm 1 Adversarial Example Generation

Input: malware sample x
Output: minimized evasive examples x'_{min}

$\mathcal{M} \leftarrow initial\_actions\_set$
$x' \leftarrow x$
while $max\_attempt > 0$ **do**
  $m \leftarrow$ max(betaSampling($\mathcal{M}$))
  $x' \leftarrow m$.transfer(x)
  if isEvasive(x') == True **then**
    $x'_{eva} = x'$
    x'_{min} = ActionMinimization(x'_{eva})
    $\mathcal{M}_{min} \leftarrow$ getMachines(x'_{min})
    for all $m' \in \mathcal{M}_{min}$ **do**
      if $m' \notin \mathcal{M}$ **then**
        $\mathcal{M}.add((m')$
      else
        $\alpha_{m'} = \alpha_{m'} + 1$
      end if
    end for
  else
    $\beta_m = \beta_m + 1$
  end if
  $max\_attempt = max\_attempt - 1$
end while
return x'_{min}

Hình 10. Thuật toán sử dụng trong Binary Rewriter

Cụ thể cách hoạt động của thuật toán trên được mô tả như sau:

- **Bước 1:** Khởi tạo danh sách các machine M, gán giá trị $\alpha = 1$ và $\beta = 1$ cho mỗi machine trong danh sách. Mỗi machine sẽ thực hiện 1 action tương ứng và khi cần nội dung (để tạo thành một cặp Action-Content), chúng sẽ chọn nội dung ngẫu nhiên.
- **Bước 2:** Áp dụng Thompson Sampling để chọn machine tiếp theo hay action sẽ thực hiện. Chúng ta sẽ lấy mẫu một giá trị từ phân phối β của mỗi machine và chọn machine có giá trị cao nhất. Sau đó chúng ta áp dụng action tương ứng với machine đã chọn cho x.
- **Bước 3:** Lặp đi lặp lại việc áp dụng một loạt action từ các machine cho đến khi nhận được mẫu trốn tránh hoặc vượt quá tổng số lần thử của biến max_attemp.
- **Bước 4:** Khi một mẫu trốn tránh được tạo ra, chúng ta sẽ sử dụng module Action Minimizer để loại bỏ các action dư thừa.

b) Các kỹ thuật áp dụng

Thuật toán trên sử dụng Thompson Sampling - một thuật toán lấy cảm hứng từ Bayesian cho Multi-armed Bandit (MAB) Problem. Nó lựa chọn các hành động giải quyết tình trạng tiến thoái lưỡng nan về thăm dò-khai thác trong bài toán cướp nhiều nhánh (MAB) bằng cách chỉ định một phân phối trước cho xác suất phần thưởng của từng machine (hoặc từng action), sau đó cập nhật các ưu tiên này khi phần thưởng được quan sát.

Trong Mab-Malware, Thompson Sampling sẽ được áp dụng trong module Binary Rewrite để giải quyết vấn đề trì hoãn trong feedback.

2. Action Minimizer

Action Minimizer là module giúp loại bỏ các action dư thừa và thay thế macro-action bằng micro-action. Nhiệm vụ của module này là giúp tạo ra mẫu đối kháng ít bị thay đổi điểm nhất nhưng vẫn giữ nguyên chức năng gốc của mẫu mã độc.

a) 1.2.1 Luồng hoạt động của Action Minizer

Algorithm 2 Action Minimization

Input: evasive sample x'_{eva}
Output: minimized evasive sample x'_{min}

$x \leftarrow$ getOriginal(x'_{eva})
$action\_seq \leftarrow$ getActionSeq(x'_{eva})
for all $action \in action\_seq$ **do**
  $x' = x$.apply($action\_seq - action$)
  if isEvasive(x') == True **then**
    $action\_seq = action\_seq - action$
    $x'_{min} = x'$
  else
    for all $micro \in$get_micro_actions($action$) **do**
      $x' = x$.apply($action\_seq - action + micro$)
      if isEvasive(x') == True **then**
        $x'_{min} = x'$
        break
      end if
    end for
  end if
end for
return x'_{min}

Hình 11. Thuật toán sử dụng trong Action Minimizer

Từ thuật toán trên, chúng ta thấy luồng hoạt động của Action Minimizer như sau:

- **Bước 1:** Lấy mẫu mã độc gốc x và danh sách action M tương ứng được tạo từ module Binary Rewriter.
- **Bước 2:** Tạm loại bỏ 1 action ra khỏi danh sách M ban đầu, tạo thành danh sách mới M'. Sau đó x được áp dụng các action thuộc M' để trở thành x'

  Nếu x' thực sự là mã trốn tránh thì nhận định action vừa loại bỏ là actoin dư thừa. Chúng ta sẽ thực sự loại bỏ action này ra khỏi danh sách M và tiếp tục thực hiện loại bỏ dần action (quay lại bước 2).

  Nếu ngược lại, thực hiện thay thế các macro-action bằng các micro-action tương ứng sao cho có ít thay đổi nhất trên mẫu.
- **Bước 3:** Sau khi không còn action nào để loại bỏ, chúng ta thu được mã đối kháng tối giản sư thay đổi nhất có thể.

3. Reward Propagation

Phân bổ phần thưởng là một khía cạnh quan trọng trong quá trình tạo ra các ví dụ đối kháng (AE) bằng cách sử dụng các kỹ thuật học tăng cường. Trong khung MAB-Malware, việc gán phần thưởng đóng vai trò then chốt trong việc xác định sự thành công hay thất bại của các hành động thực hiện khi tạo ra các mẫu malware né tránh.

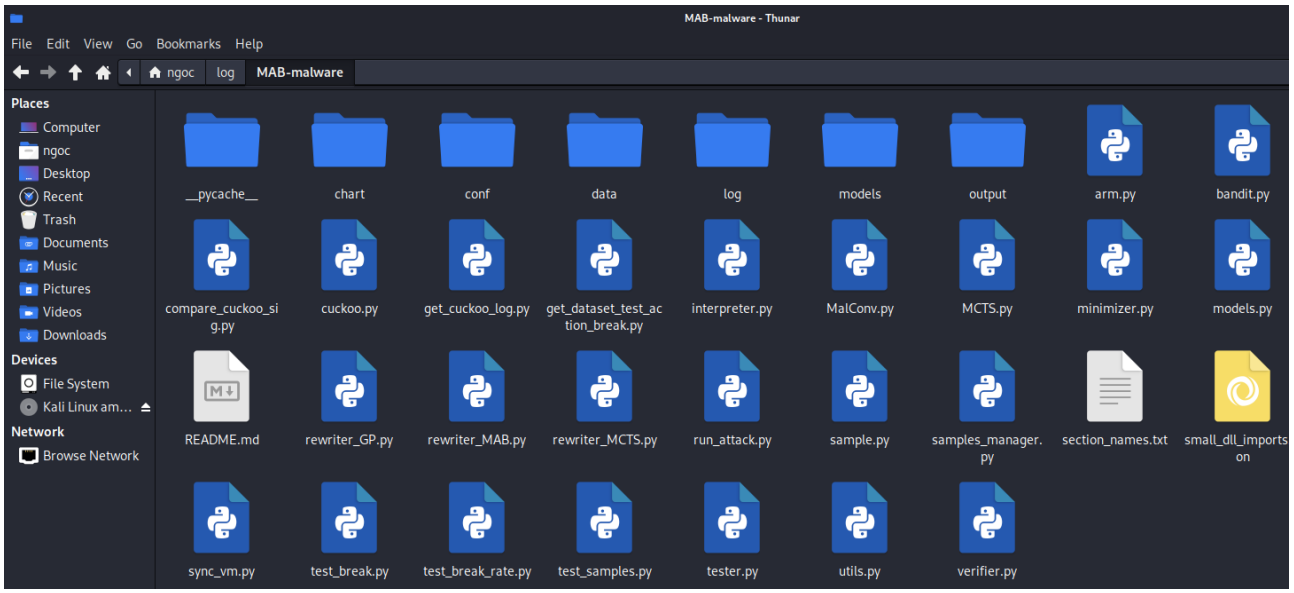
Thay vì gán phần thưởng cho tất cả các hành động tham gia vào quá trình, framework này tập trung vào việc gán phần thưởng chính xác cho những hành động thiết yếu góp phần vào việc né tránh. Cách tiếp cận này giúp mô hình học tăng cường học hỏi hiệu quả hơn bằng cách tập trung vào các hành động liên quan và giảm thiểu ảnh hưởng của các hành động thừa. Bằng cách gán phần thưởng chính xác cho các hành động thiết yếu, khung này tối ưu hóa quá trình học và cải thiện hiệu suất tổng thể của việc tạo ra các mẫu né tránh.

<Trình bày kiến trúc, thành phần đã thực hiện (nội dung mà nhóm đã thực hiện)>

Kiến trúc và các thành phần được thực hiện lại như mô tả lý thuyết ở trên. Dưới đây là chi tiết cách nhóm thực thi framework.

Để thực hiện lại bài báo này, chúng ta sẽ sử dụng github và clone đường link:

https://github.com/weisong-ucr/MAB-malware



Hình 12. Tổng quan code thực thi MAB-Malware

Sau khi clone code về, có thể sử dụng docker để khởi chạy mô hình. Sử dụng lệnh: sudo docker run -ti wsong008/mab-malware bash.

```
┌──(ngoc㉿ngoc)-[~]
└─$ sudo docker run -ti wsong008/mab-malware bash
[sudo] password for ngoc:
root@155d13ed3316:~/MAB-malware# ls
MCTS.py                        get_dataset_test_action_break.py  samples_manager.py
MalConv.py                     interpreter.py                    section_names.txt
README.md                      log                               small_dll_imports.json
__pycache__                    minimizer.py                      submit_samples.py
arm.py                         models                            sync_vm.py
bandit.py                      models.py                         test_break.py
chart                          output                            test_break_rate.py
classifier.py                  process_benign_dataset.py         test_samples.py
compare_cuckoo_sig.py          rewriter_GP.py                    tester.py
conf                           rewriter_MAB.py                   utils.py
cuckoo.py                      rewriter_MCTS.py                  verifier.py
data                           run_attack.py
get_cuckoo_log.py              sample.py
```

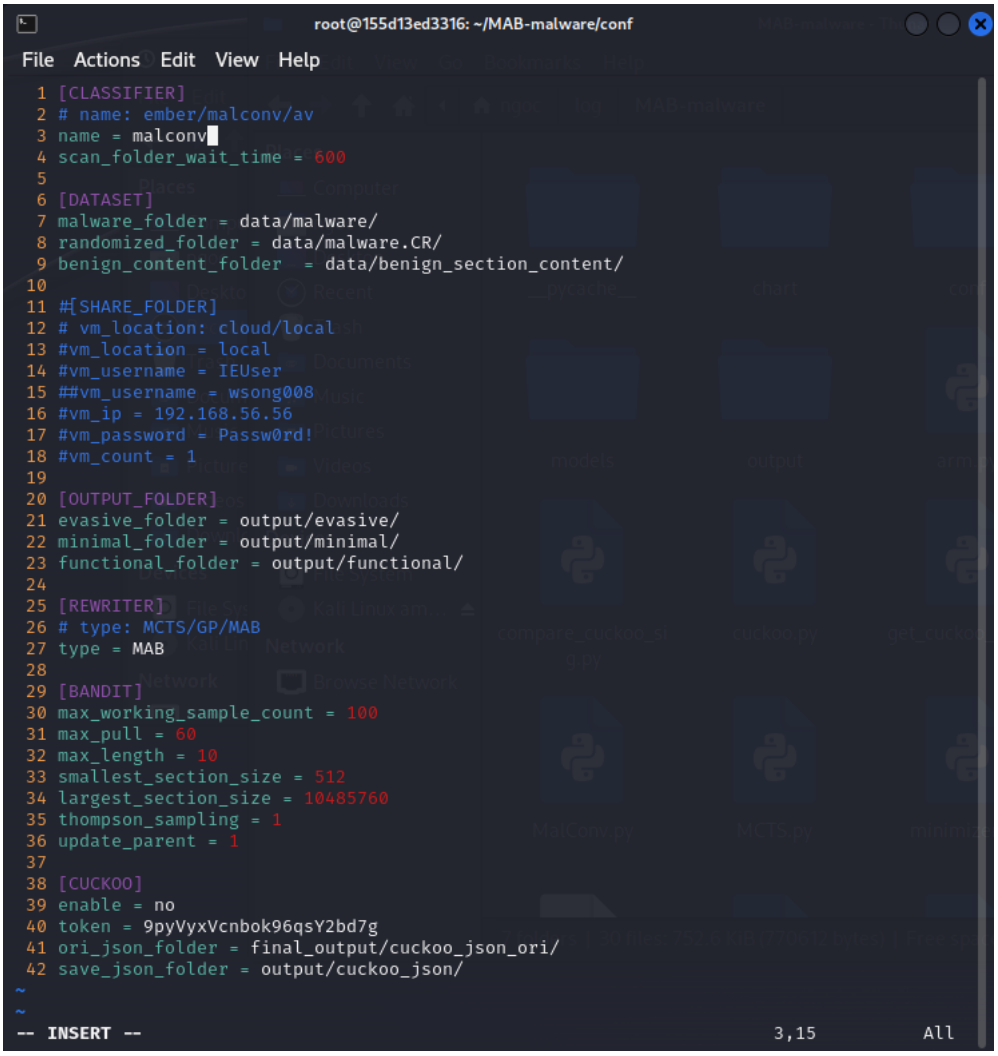
Hình 13. Sử dụng docker

Để lựa chọn mô hình sẽ chạy, chúng ta sẽ thực hiện thay đổi file conf/configure.ini

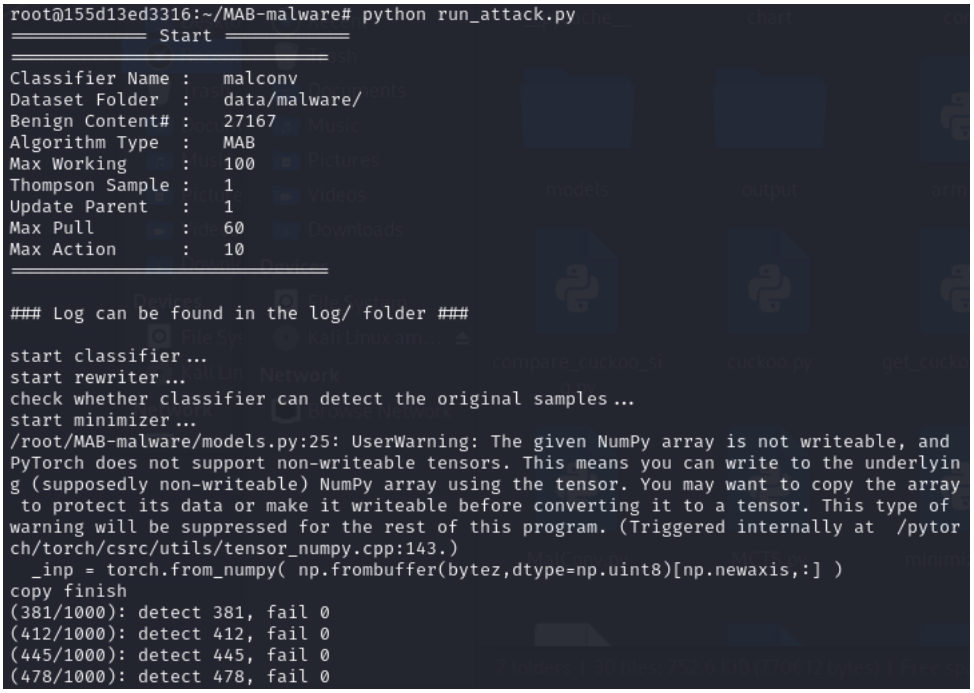
```
root@155d13ed3316:~/MAB-malware# cd conf
root@155d13ed3316:~/MAB-malware/conf# vim configure.ini
root@155d13ed3316:~/MAB-malware/conf#
```

Hình 14. File cấu hình

Mặc định khi vừa clone code về sẽ chạy mô hình phân loại Malconv.



Hình 15. Sử dụng mô hình Malconv



Hình 16. Khởi chạy mô hình Malconv

Nếu muốn chạy mô hình EMBER, ta chỉ cần thay thế tên mô hình Malconv thành EMBER trong trường name.

Hình 17. Sử dụng mô hình EMBER

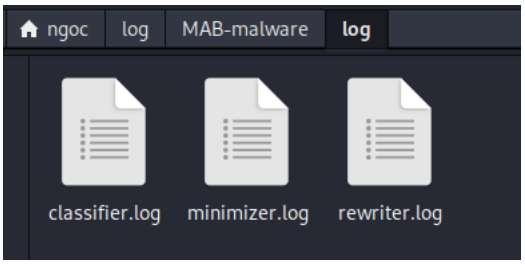
```
root@155d13ed3316:~/MAB-malware# python run_attack.py
========== Start ==========
================================
Classifier Name :   ember
Dataset Folder  :   data/malware/
Benign Content# :   27167
Algorithm Type  :   MAB
Max Working     :   100
Thompson Sample :   1
Update Parent   :   1
Max Pull        :   60
Max Action      :   10
================================

### Log can be found in the log/ folder ###

start classifier ...
start rewriter ...
check whether classifier can detect the original samples ...
start minimizer ...
copy finish
(212/1000): detect 212, fail 0
(295/1000): detect 295, fail 0
(367/1000): detect 367, fail 0
```

Hình 18. Khởi chạy mô hình EMBER

Kết quả sau khi chạy xong các mô hình trên sẽ được lưu trữ trong thư mục Log và sẽ được lưu về để vẽ các biểu đồ đánh giá (trong phần **C. Kết quả thực nghiệm**).



Hình 19. Kết quả sau khi thực thi

B. Chi tiết cài đặt, hiện thực

<cách cài đặt, lập trình trên máy tính, cấu hình máy tính sử dụng, chuẩn bị dữ liệu, v.v>

Dataset: 1000 mẫu Windows malware PE lấy từ VirussTotal với tỷ lệ hơn 80% được gán nhãn mã độc.

Cấu hình máy tính: sử dụng máy ảo Kali Linux (Debian 11) 2GB Ram, 4 processor cores, 80GB disk.

Ngôn ngữ lập trình để hiện thực phương pháp là Python 3.11.6.

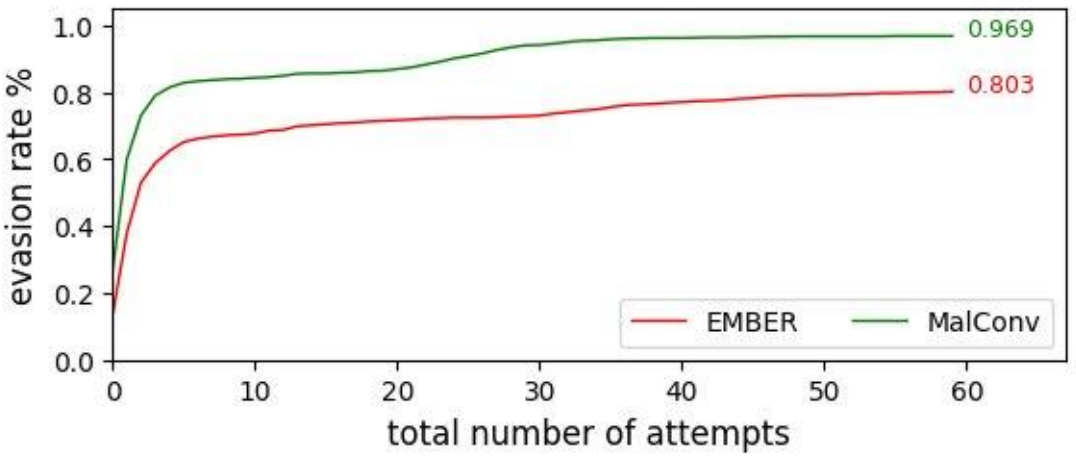
Hai mô hình Malware Classifier sử dụng: Ember và Malconv.

C. Kết quả thực nghiệm

<mô tả hình ảnh về thực nghiệm, bảng biểu số liệu thống kê từ thực nghiệm, nhận xét về kết quả thu được.>

Kết quả đánh giá được vẽ dựa trên code trong file Evaluate.ipynb

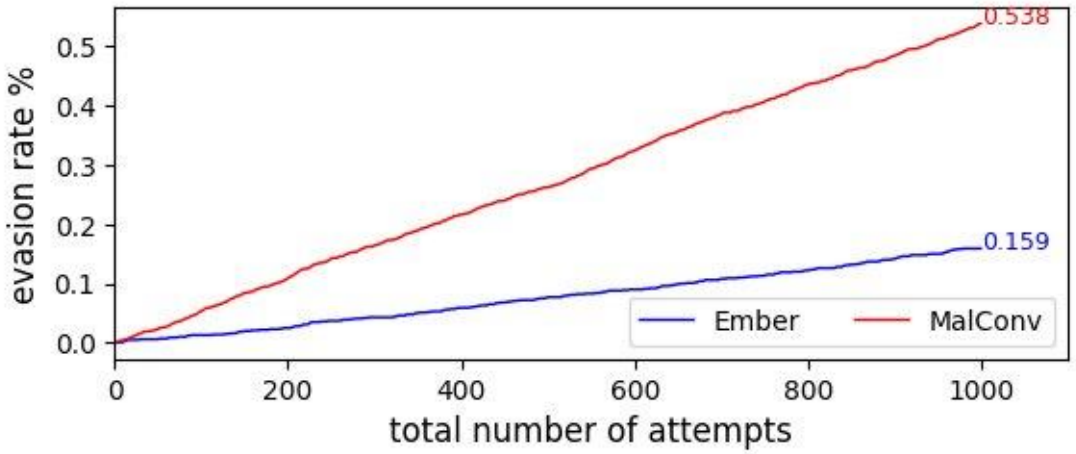
1. Tỉ lệ trốn tránh khi dùng thuật toán MAB



Hình 20. Kết quả tỷ lệ trốn tránh sử dụng MAB

MAB-Malware có khả năng trốn tránh vượt trội hơn so với các framework khác. Nó có thể tạo ra các mẫu AE cho 96.9% mẫu để trốn tránh MalConv, và 80.3% mẫu để trốn tránh EMBER.

2. Tỉ lệ trốn tránh khi dùng thuật toán MCTS

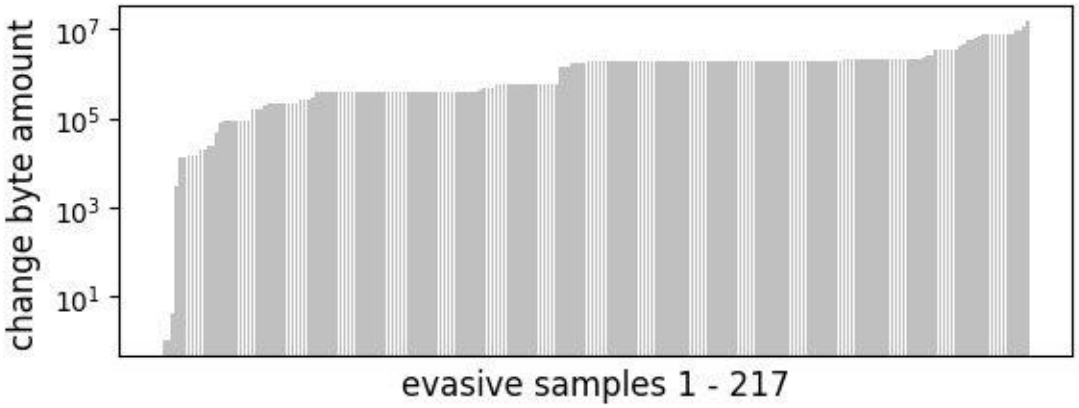


Hình 21. Kết quả tỷ lệ trốn tránh sử dụng thuật toán khác

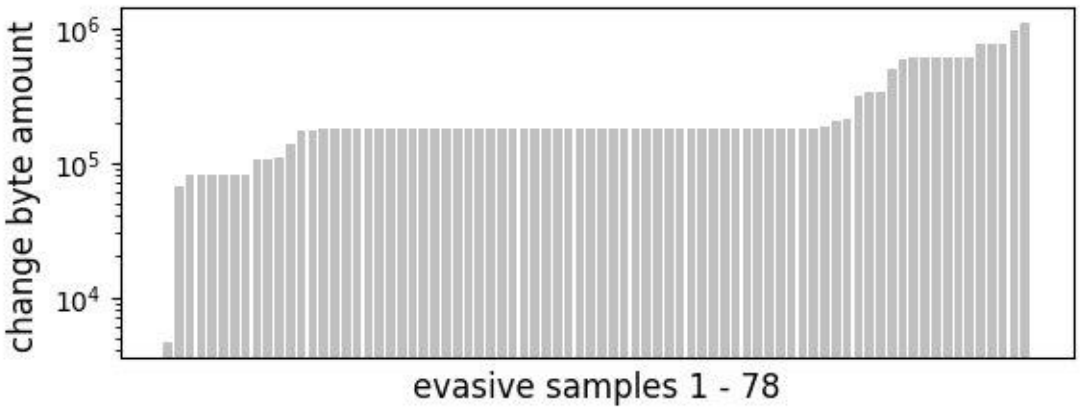
Trong khi MAB-Malware có khả năng trốn tránh vượt trội thì MCTS (Monte Carlo Tree Search) có tỷ lệ trốn tránh khá thấp với 53.8% cho Malconv và 15.9% cho Ember.

3. Số lượng byte thay đổi của mẫu đối kháng

Dưới đây là thống kê số lượng byte thay đổi bằng framework MAB-Malware.



Hình 22. Số lượng byte thay đổi trên mô hình EMBER

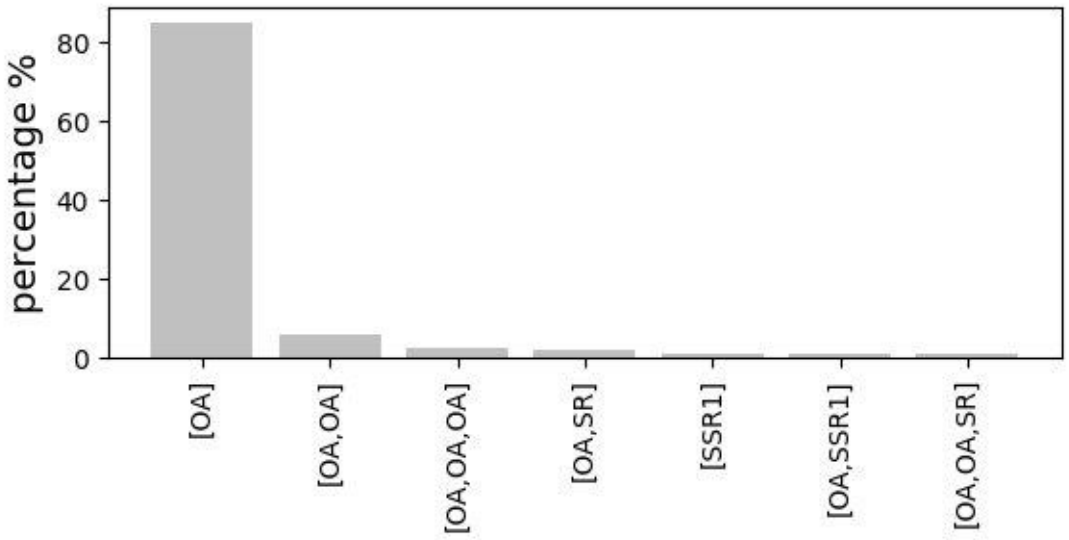


Hình 23. Số lượng byte thay đổi trên mô hình Malconv

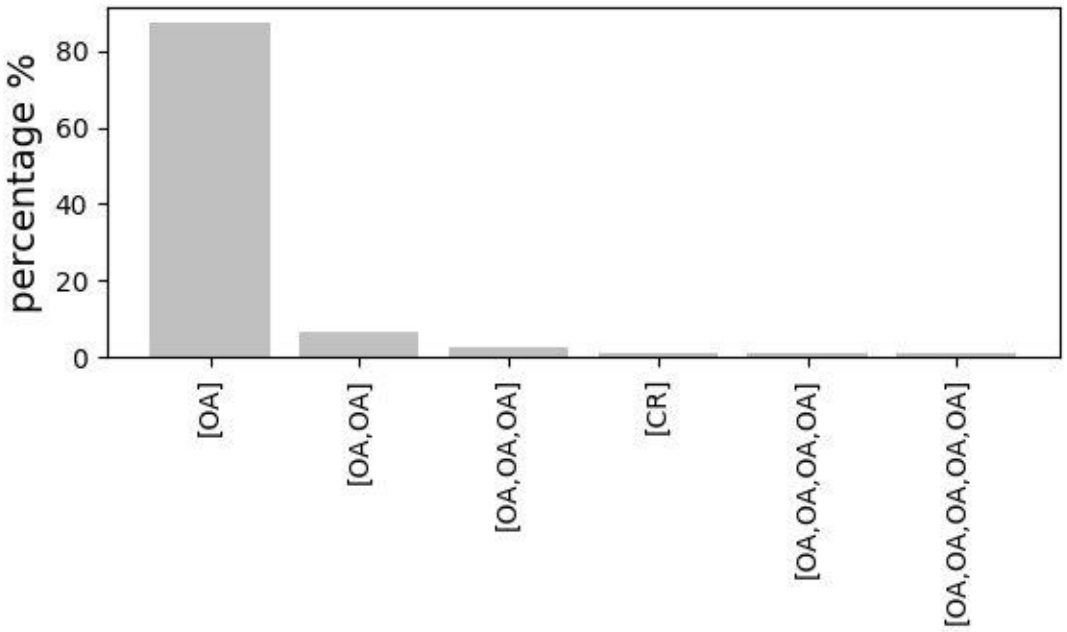
Ember và Malconv đều cần phải thay đổi khá nhiều byte trên từng file mã độc để có thể bypass thành công (khoảng 10^5 đến 10^7 bytes được thay đổi).

4. Chuỗi hành động tạo mẫu đối kháng

Dưới đây là thống kê các chuỗi hành động được sử dụng hiệu quả trong quá trình tạo ra các mẫu đối kháng.



Hình 24. Các chuỗi hành động trong mô hình EMBER



Hình 25. Các chuỗi hành động trong mô hình Malconv

Với 2 mô hình phân loại mã độc MalConv và Ember, chúng ta thấy hành động quan trọng nhất trong chuỗi hành động giúp sinh mẫu đối kháng là Overlay Append (OA).
Những hành động khác chỉ thay đổi 1 vài byte mà hầu như không có tác dụng lắm trong việc trốn tránh, cho thấy sự thay đổi entropy byte là nguyên nhân sâu xa của những cuộc trốn tránh.

D. Hướng phát triển

<Nêu hướng phát triển tiềm năng của đề tài này trong tương lai. Nhận xét về tính ứng dụng của đề tài>.

Hướng phát triển của đồ án này:

- Thực hiện tạo các mẫu đối kháng cho các mô hình antiviruss hiện có trên thị trường.
- Bởi vì bài báo này chủ yếu tập trung vào việc né tránh đối với bộ phân loại mã độc tĩnh nên trong tương lai có thể mở rộng thực hiện tạo các mẫu đối kháng né tránh các bộ phân loại mã độc động.

HẾT