ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

KHOA MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG

Môn học : **KỸ THUẬT PHÂN TÍCH MÃ ĐỘC**

BÁO CÁO CUỐI KỲ

ĐỀ TÀI 12: PACKING

Lớp: **NT137.O11.ATCL**

Giảng viên hướng dẫn:

NGUYỄN TẤN CẦM

NGUYỄN CÔNG DANH

Thành viên:

Nguyễn Mạnh Cường 20520421

Nguyễn Trần Đức Anh 20520392

Thành phố Hồ Chí Minh
Ngày 20 tháng 12 năm 2023

MỤC LỤC

A. LÝ THUYẾT 2
1. TỔNG QUAN 2
1.1. Packing 2
1.2. Unpacking 7
2. CHI TIẾT KỸ THUẬT PACKING 11
2.1. Phân cấp kỹ thuật packing 11
2.2. Phân loại packer 14
2.3. Anti-Unpacking 15
3. THỐNG KÊ 17
3.1. Xu hướng packing 17
3.2. Xu hướng các packer 19
3.3. Các phương pháp đề xuất về việc phân tích, phát hiện kỹ thuật packing 22
B. TRIỂN KHAI 26
1. UNPACKING ĐỘNG THỦ CÔNG 26
2. MÔ HÌNH HỌC MÁY PHÂN LOẠI MÃ ĐỘC DỰA TRÊN TẬP DỮ LIỆU VỀ PACKING 34
C. TRÍCH DẪN 37
D. PHỤ LỤC 38

BÁO CÁO CHI TIẾT

A.LÝ THUYẾT

1. TỔNG QUAN

1.1. Packing

Packing là một quá trình biến đổi một tệp thực thi nhị phân thành một cấu trúc khác bằng cách sử dụng nén và/hoặc mã hóa, nhằm bảo vệ/giấu đi nội dung gốc của tệp thực thi. Đầu vào và đầu ra của quá trình packing đều là các tệp thực thi.

Trên hai phương diện tốt và xấu:

- Là phương pháp bảo vệ mã nguồn phổ biến và được sử dụng hợp pháp.
- Cũng là phương pháp làm mờ thông tin phổ biến được sử dụng bởi các tác giả phần mềm độc hại để gây trở ngại cho việc phát hiện và phân tích phần mềm độc hại.

Mục đích hợp pháp:

- Sử dụng để giảm kích thước tệp thực thi và để tránh bị phân tích tĩnh.
- Cho phép phân phối tệp thực thi trên mạng nhanh hơn và an toàn hơn.
- Áp dụng mã hóa, làm cho việc đảo ngược kỹ thuật trở nên khó khăn hơn đối với các hacker cố gắng truy cập vào các tệp của phần mềm và biến đổi chúng theo nhiều cách (ví dụ: sử dụng trình phân tích chương trình, debug, trích xuất bộ nhớ và phân tích thư mục dữ liệu).
- Cung cấp một lớp bảo vệ bổ sung mà không ảnh hưởng đến tính năng ban đầu của một ứng dụng.

Mục đích bất hợp pháp:

- Các tác giả phần mềm độc hại chủ yếu sử dụng các kỹ thuật packing để làm mờ thông tin, nhằm giấu mã độc hại khỏi các chương trình quét phần mềm độc hại.

- Làm tăng thời gian quét của các chương trình diệt virus (AV), vì các tệp thực thi được pack yêu cầu thêm các thao tác từ trình quét, chẳng hạn như kiểm tra mã nguồn và quá trình unpacking chính nó.
- Hầu hết các chương trình phần mềm độc hại đều được pack nhằm tránh sự phát hiện.
- Lợi thế cho các tác giả phần mềm độc hại, vì nó tạo ra nhiều biến thể bằng cách thay đổi quá trình packing dễ dàng.
- Những người viết phần mềm độc hại tận dụng điểm yếu chính của các công cụ phát hiện dựa trên phân tích tĩnh, chúng dựa vào một kho dữ liệu signature và dễ dàng tránh được phát hiện bằng cách packing mã độc hại trong các lớp nén hoặc mã hóa.
- Rất nhiều packer hiện có có thể dễ dàng được sửa đổi, tùy chỉnh và mở rộng bằng cách sử dụng nhiều thuật toán nén khác nhau, bởi vì nhiều thuật toán nén và packer là mã nguồn mở.
- Một phần đáng kể mã độc hại được pack bởi một packer tùy chỉnh, số lượng và độ phức tạp của các packer mới tiếp tục tăng lên.

Hoạt động của packer:

- Packer thực hiện việc lấy một tệp thực thi hoặc thư viện động (dynamic-link library), nén và/hoặc mã hóa nội dung của nó, sau đó packing nó vào một tệp thực thi mới. Tệp đã được pack gồm 2 phần:
    - **1)** The first component is made up of a number of data blocks which form the compressed/encrypted version of the original executable file.
    - **2)** The second part is responsible for dynamic unpacking, during runtime, in order to retrieve the original executable file. (Unpacking stub)

Khi chạy một tệp được pack:

- Khi tệp đã được pack đang chạy, phần unpacking stub được thực hiện để unpacking mã nguồn thực thi gốc vào bộ nhớ và giải quyết tất cả các import của tệp thực thi gốc. Sau đó, unpacking stub chuyển quyền điều khiển cho mã gốc.
- Việc thực thi của tệp gốc không thay đổi và bắt đầu từ điểm ban đầu (Original Entry Point - OEP)
- Trước khi thực thi lệnh đầu tiên của một tệp thực thi, quá trình giải mã và/hoặc giải nén phải diễn ra trong runtime.
- Quá trình viết mã gốc vào bộ nhớ và sau đó thực thi có thể được lặp lại nhiều lần, tức là mã được tạo ra là động và có thể tiếp tục tạo mã mới và thực thi nó. Mỗi lần lặp lại của mã được tạo ra được gọi là một lớp (layer). Nói cách khác, một lớp là một chuỗi các địa chỉ bộ nhớ được thực thi và là kết quả từ mã đã được viết bởi một lớp khác.

Khác nhau giữa tệp được pack và không khi chạy:

- Tệp thực thi thông thường, loader đọc phần header PE từ đĩa và cấp phát không gian trong RAM cho mỗi section của tệp thực thi, dựa trên các cài đặt trong phần header. Sau đó, loader sao chép các section đó vào các không gian đã cấp trong RAM.
- Phần header PE của tệp thực thi đã được pack có thể được định dạng sao cho nó chứa một số phần cho quá trình unpacking, và sau đó tệp đã được unpack tạo ra thêm các phần bổ sung (chương trình gốc). Hoặc, phần header PE có thể chứa nhiều phần để bao gồm tất cả các phần của chương trình cùng với phần unpacking stub. Unpacking stub sau đó unpacking mã nguồn gốc cho mỗi phần và sao chép nó vào khối bộ nhớ đã được cấp phát.

Thay đổi của mã độc sau khi được pack:

- Về cấu trúc và dấu vết của mã độc hại bên trong một tệp thực thi, chúng sẽ thay đổi sau quá trình packing của tệp thực thi.
- Trong trường hợp mà người viết packer sử dụng cả nén và mã hóa, tất cả sự tương tự giữa tệp thực thi gốc và phiên bản đã được pack sẽ bị mất. Tuy nhiên, một số thông tin vẫn được bảo tồn bởi cả nén và các thuật toán mã hóa yếu.
- Các packer hiện có và các thuật toán nén hoặc mã hóa của chúng giữ lại một số tính chất có trong mã gốc (ví dụ: một cặp tệp thực thi có phân phối mã nguồn tương tự vẫn giữ tính tương tự sau quá trình packing nếu việc sắp xếp byte (byte alignment) và hoán vị (permutations) của nó được quản lý).
- Một sự thay đổi quan trọng khác là các packer xóa bảng Import Address Table - IAT của tệp PE nhằm tăng tính phức tạp trong việc phân tích lệnh gọi API.
- Khi một tệp PE độc hại đã được pack được thực thi, phần unpacking stub giải mã mã độc hại thành các trang bộ nhớ và xây dựng lại IAT trước khi tiếp tục thực thi mã gốc từ Original Entry Point – OEP.
- Một tệp PE đã được pack và tệp gốc có cùng chức năng và giữ lại một số đặc tính có trong mã nguồn gốc.
- Một số phương pháp packing biến đổi một số hoặc tất cả các byte mã nguồn gốc thành một chuỗi các byte dữ liệu có dạng ngẫu nhiên. Do đó, một tệp PE đã được pack có cấu trúc byte và signature khác biệt so với tệp gốc.

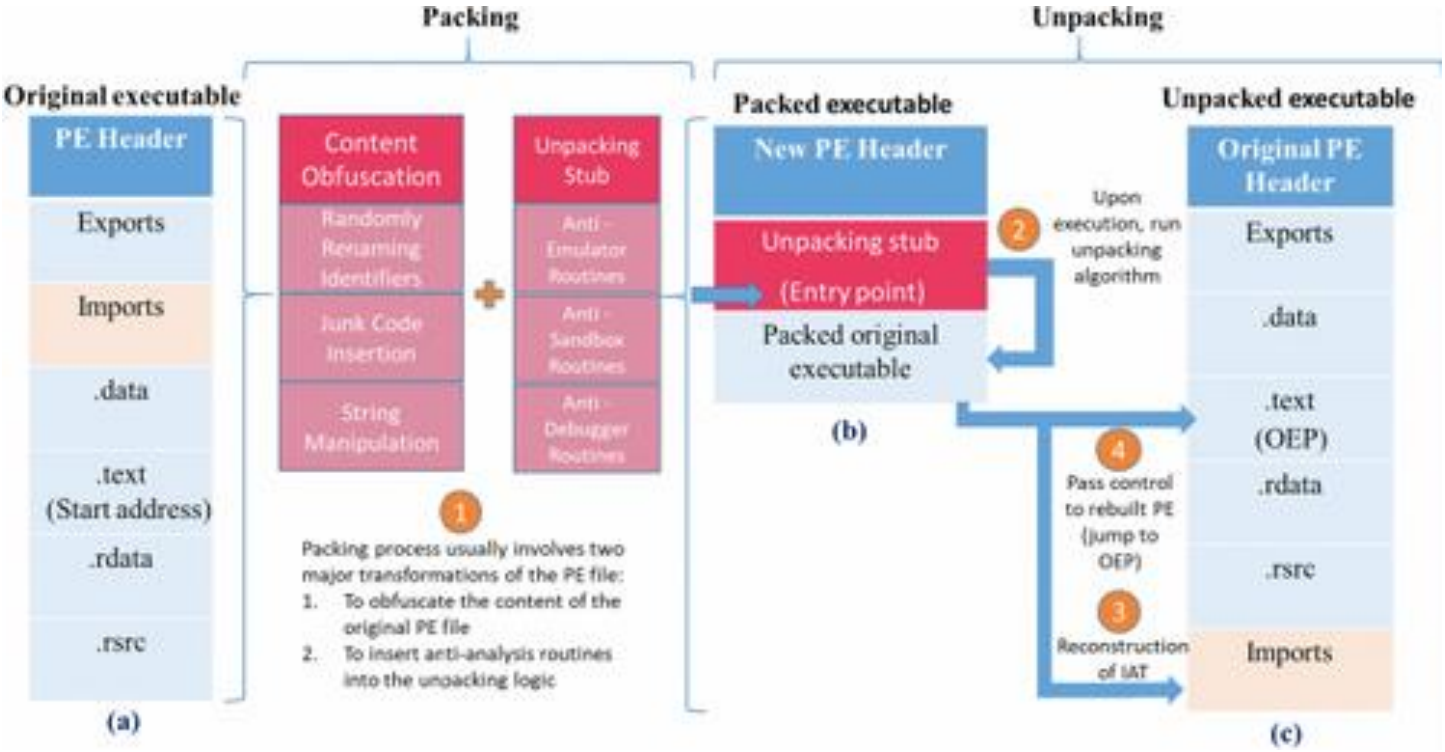
Property	Packed File	Unpacked File
Number of import functions	Smaller number	Higher number
IAT	Reconstructed	Standard
Code visibility	Low	High
Debugger flags	Opening packed file in some debuggers causes a "may be packed" warning to be issued	No such warning is displayed
Section names	Abnormal section names such as "UPX0" or "Dialog"	Standard section names
Section sizes	Abnormal sections sizes such as .text section of size zero	No abnormality seen
Entropy of byte sequences	High entropy	Low entropy

Bảng 1: *So sánh tệp thực thi đã và chưa được pack.*

Quá trình packing:

- Tất cả các phần gốc đều bị nén sau khi packing và một phần unpacking stub được thêm vào đầu tệp thực thi.
- 4 bước packing:
  - 1) Bắt đầu bằng việc làm mờ nội dung của tệp PE gốc (nén và/hoặc mã hóa), tiếp theo là việc thêm các chương trình chống phân tích khác nhau, nhằm làm cho phân tích trở nên khó khăn hơn hoặc ngăn ngừng sự tiết lộ và thực thi.
  - 2) Khi tệp thực thi đã được pack được thực thi, điểm khởi đầu (mới) của tệp thực thi đã được pack thường nằm trong phần unpacking stub (chứa các chương trình chống phân tích đã được đề cập). Nếu các kiểm tra chống phân tích khác nhau được vượt qua, tệp PE gốc sẽ được unpack và được ghi vào bộ nhớ.
  - 3) Sau đó, quá trình giải quyết các tệp thực thi được thực hiện và Import Address Table - IAT của tệp PE gốc được tái xây dựng
  - 4) Quyền điều khiển thực thi được chuyển giao cho Điểm Khởi Đầu Gốc (Original Entry Point - OEP) của tệp thực thi gốc, sau đó tệp này thực thi.

- *Lưu ý:* một tệp PE cũng có thể được pack theo cách lồng ghép bằng cách sử dụng nhiều packer khác nhau, điều này có nghĩa rằng vòng đời packing-unpacking được đề cập ở trên có thể được kết hợp với nhau nhiều lần (dựa trên số lần mà tệp PE được pack lặp đi lặp lại).



Hình 1: *Quá trình packing.*

1.2. Unpacking

Quá trình unpacking là quá trình đảo ngược của quá trình packing.

Hoạt động unpacking:

- Khi quá trình unpacking được hoàn tất, tệp gốc được khôi phục vào bộ nhớ tạm thời. Sau khi tạo lại tệp thực thi gốc, luồng thực thi nhảy từ điểm cuối của mô-đun unpacking stub đến Điểm Khởi Đầu Gốc (OEP) của tệp thực thi gốc, sử dụng lệnh jump.
- Quá trình unpacking được thực hiện khi đạt được OEP.

- Phần unpacking stub là cơ chế chịu trách nhiệm trích xuất mã nguồn gốc từ tệp đã được pack. Hệ điều hành tải phần unpacking stub và sau đó tiếp tục tải chương trình gốc.

Đối với phân tích tĩnh:

- Phần unpacking stub có thể được nhìn thấy bởi trình phân tích phần mềm độc hại, tuy nhiên, để unpacking tệp thực thi, cần phải hiểu rõ các phần khác nhau của stub.
- Khi phân tích tĩnh được thực hiện trên chương trình đã được pack, thì chỉ phần stub (chứ không phải là chương trình gốc) được phân tích.

Các phương pháp unpack cho việc phân tích:

a) *Unpacking tĩnh tự động* là các chương trình giải nén và/hoặc giải mã tệp mà không chạy chúng, các chương trình này chỉ hữu ích cho các packer có một lớp duy nhất.
b) *Unpacking động tự động* chạy chương trình thực thi và cho phép phần unpacking stub tự unpacking mã gốc.
    - Kỹ thuật này dựa vào việc xác định ranh giới giữa cuối phần unpacking stub và đầu chương trình thực thi gốc, điều này có thể khó khăn khi áp dụng các phương pháp che giấu. Quá trình unpacking sẽ thất bại nếu ranh giới này không được xác định đúng cách.
    - Unpacking động tự động được thực hiện trong môi trường cách ly, chẳng hạn như máy ảo hoặc sandbox.
c) *Unpacking động thủ công* là quá trình khôi phục mã gốc, OEP và IAT của nó bằng cách sử dụng các công cụ debug như ImpRec, có thể khôi phục một IAT đã bị hủy bỏ, và LordPE, một công cụ dành cho việc chỉnh sửa các phần header PE. Gồm 2 phương pháp:

- Phương pháp đầu tiên dựa trên việc tìm hiểu thuật toán packing và viết một chương trình sẽ cố gắng đảo ngược thứ tự thực thi của nó để unpacking tệp gốc.
- Phương pháp thứ hai là chạy chương trình đã được pack nhưng để phần unpacking stub tự unpacking chương trình và sau đó dump quá trình ra khỏi bộ nhớ, và thiết lập thủ công phần header PE.

(Hiện nay vẫn thiếu các công cụ unpacking động tự động đáng tin cậy và được công khai; ngược lại, các công cụ unpacking tĩnh tự động được sử dụng rộng rãi)

Technique Name	Description
JIT compilation	This technique involves performing compilation during the execution of the program at run time, rather than before execution.
IAT Hooking	The Import Address Table (IAT) contains information such as DLLs, a list of function names and addresses of these function names from the DLLs that can be called by the binary during execution, etc. Hooking the IAT of a specific target program in the analysis environment allows monitoring of all libraries that are being called and the functions that are being executed.
Memory Dumping	Using memory dumping tools like Scylla [127] to take a snapshot of the memory of the sandbox environment to save the loaded state of the file being analyzed in memory
OEP hunting through debugging	Finding the Original Entry Point through manual debugging using techniques such as setting breakpoints on APIs like LoadLibrary, VirtualAlloc.

Bảng 2: *Các phương pháp unpacking phổ biến.*

So sánh, đánh giá các phương pháp phân tích:

- Các kỹ thuật unpacking tự động nhanh chóng và dễ sử dụng hơn so với các kỹ thuật unpacking thủ công và không đòi hỏi sự can thiệp thủ công từ con người.
- Tuy nhiên các kỹ thuật tự động không luôn hoạt động đúng cách, vì phải xác định trước packer mục tiêu, và không phải lúc nào cũng có một công cụ unpacking tự động phù hợp.
- Unpacking thủ công tốn kém và mất thời gian.
- Unpacking tĩnh các tệp thực thi khó hơn việc unpacking động do cần sử dụng kỹ thuật đảo ngược (reversing) mã nguồn.

- Các công cụ unpacking thông thường dựa trên việc cung cấp một môi trường thực thi động cho các tệp thực thi và do đó cũng bị giới hạn về hiệu suất do việc thực thi mã nguồn.

Hạn chế của các packer:

- Hạn chế lớn của các packer runtime (còn được gọi là các packer tự unpacking, tức là phần mềm tự unpacking trong bộ nhớ khi tệp đã được pack được thực thi) là mã nguồn gốc sẽ phải được thực thi vào một thời điểm nào đó. Hạn chế này có thể bị tận dụng bởi quá trình unpacking thủ công. (đã đề cập)
- Nhiều packer runtime cố gắng chặn các kỹ thuật unpacking này bằng cách sử dụng các quy trình chống debug và làm mờ mã nguồn.
- Suốt một thời gian dài, các nhà nghiên cứu đã cố gắng cải thiện hoặc tùy chỉnh các phương pháp để unpacking các tệp đã được pack.

2. CHI TIẾT KỸ THUẬT PACKING

2.1. Phân cấp kỹ thuật packing

Phân cấp độ phức tạp:

a) *Packing một lớp:* chỉ sử dụng một packer để packing một tệp nhất định, trong đó kích thước, số lượng section và tên của tệp thực thi thay đổi.

b) *Re-packing:* là một kỹ thuật phức tạp hơn, trong đó cùng một packer được sử dụng ít nhất hai lần để packing một tệp. Re-packing sử dụng các kỹ thuật pack tương tự như các packer một lớp.
   - Khi một tệp đã được re-pack được thực thi, mỗi lớp phải được unpack theo thứ tự.

c) *Packing đa lớp:* bao gồm nhiều lớp unpacking, mỗi lớp sẽ được thực thi theo thứ tự để unpacking lần lượt. Loại này khác với kỹ thuật re-packing, vì các lớp được tạo bởi các loại packer khác nhau, hoặc kết hợp của một số packer khác nhau để packing một tệp.
   - Điều này cho phép tạo ra một số lượng lớn các tệp đã được pack từ cùng một đầu vào, do đó có thể dẫn đến việc tạo ra nhiều biến thể phần mềm độc hại từ phần mềm độc hại gốc.
   - Thuật toán này thay đổi kích thước, số lượng section và tên của tệp thực thi đã được pack một lớp.

Đánh giá packing đa lớp:

- Hầu hết packer là đa lớp.
- Packing đa lớp làm cho việc xác định sự kết thúc của quá trình unpacking trở nên khó khăn.
   - Để thực hiện quá trình unpacking, các phương pháp truyền thống liên quan đến việc đi qua từng lớp và unpacking nó để xác định OEP. Quy trình này đòi hỏi phải phân bổ và sử dụng một lượng lớn các địa chỉ bộ nhớ và đòi hỏi tính toán, từ đó làm tăng thời gian chạy.

- Có hai yếu tố chính ảnh hưởng đến sự phức tạp của packing đa lớp:
    - Đầu tiên, tính năng "ghi rồi thực thi" khi unpacking chỉ là một dạng biểu hiện của mã nguồn động được tạo ra sau đó được thực thi. Tuy nhiên, đó không phải là mã nguồn gốc (tức là một tập hợp các địa chỉ bộ nhớ đã được viết bởi một số lớp, sau đó được thực thi bởi lớp tiếp theo).
    - Thứ hai, mã nguồn gốc có thể được tìm thấy ở mọi lớp (tức là mã gốc không nhất thiết phải nằm ở lớp sâu nhất). Đôi khi lớp sâu nhất có thể chứa mã rác chỉ để đánh lừa các hệ thống phân tích hoạt động dựa trên giả định rằng lớp sâu nhất chứa mã độc hại.

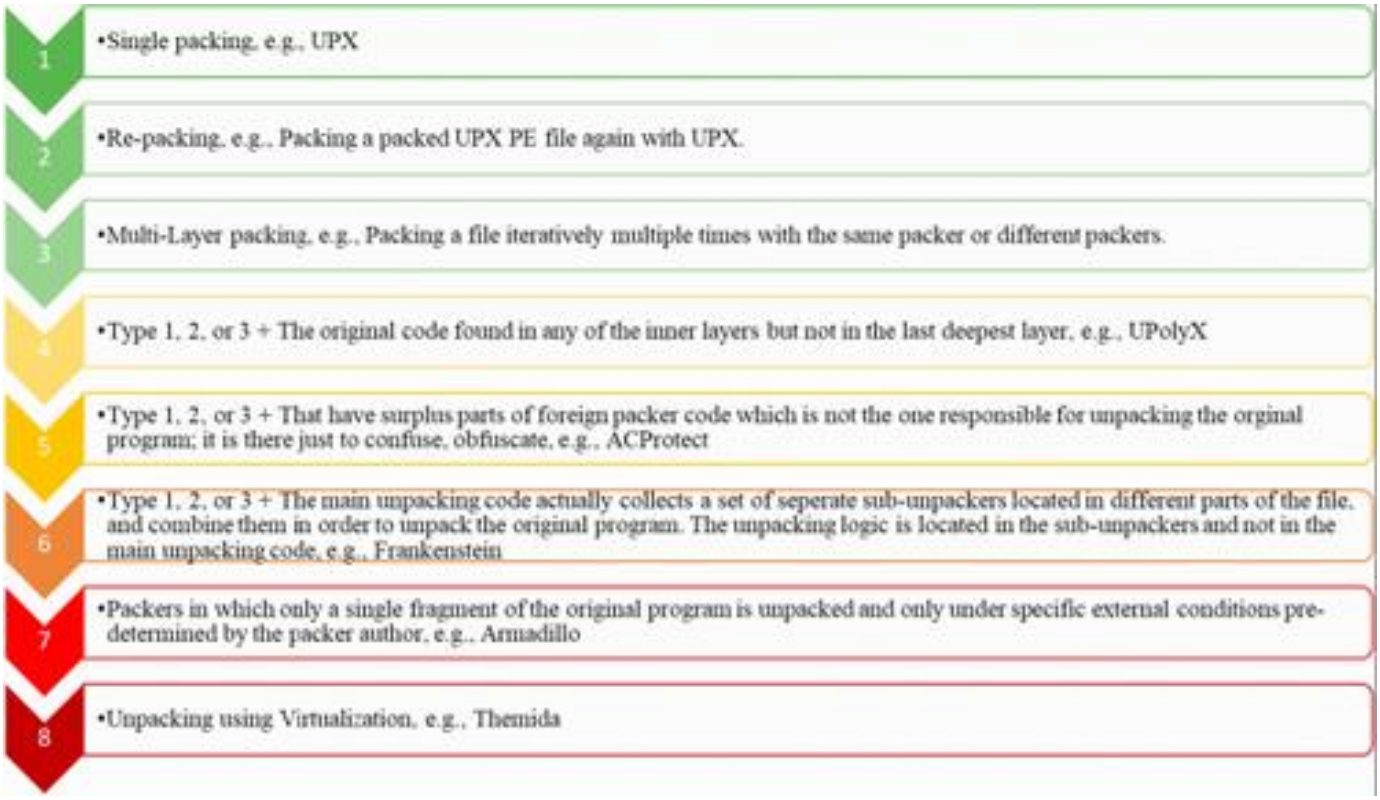
Sự đa lớp:

- Một số packer tạo ra nhiều quy trình nhằm unpacking mã gốc. Các packer đó có dạng các chương trình gieo mã (các tệp có mục đích là cài đặt phần mềm độc hại nghiêm trọng hơn trên hệ thống) và tạo ra một tệp sau đó được thực thi, trong khi các công cụ khác tạo ra một quy trình riêng biệt và sau đó tiêm mã unpacking vào đó.
- Một số trường hợp, các packer thậm chí can thiệp sâu hơn bằng cách unpacking và packing lại mã theo yêu cầu tại các điểm khác nhau trong quá trình thực thi.
- Phần lớn mã độc hại được nhận diện là các biến thể của các mẫu mã độc hại đã được phân tích sau khi được pack. Thể hiện tầm quan trọng của việc phát hiện các biến thể của mã độc hại khi phát hiện các mẫu mã độc hại chưa biết.

Sự phát triển về độ phức tạp của kỹ thuật:

- Ban đầu, các packer chỉ sử dụng các cơ chế nén dành cho việc giảm kích thước tệp và tăng tốc độ phân phối qua mạng. Sau đó, các nhà phát triển packer đã thêm mã hóa vào các thuật toán nén cơ bản để ngăn chặn việc đảo ngược mã.
- Các công cụ unpacking thì chậm trong việc thích nghi với sự tiến bộ của các packer, dẫn đến hiệu suất giảm đi của các giải pháp phát hiện mã độc hại mới.

- Gần đây, các kỹ thuật phức tạp hơn như ảo hóa unpacking (cho quá trình packing) và unpacking động (cho quá trình phân tích) đã được tích hợp vào các thuật toán packing.
    - *Ảo hóa unpacking:* khi chạy, chương trình gốc được phiên dịch thành các câu lệnh ảo sau đó được thực thi bởi máy ảo nhúng. Các công cụ unpacking ảo hóa là một hiện tượng đang phát triển được áp dụng trong các packer như HashiCorp, Themida, VMprotect và Obsidium.
    - *Các kỹ thuật unpacking động:* thực thi một chương trình đã được pack trong một môi trường phân tích được cách ly và theo dõi quá trình thực thi của nó. Sau đó, cố gắng xác định khi chương trình đã hoàn tất việc unpacking chính nó, để sau đó có thể xác định mã không được bảo vệ (mã nguồn gốc) trong bộ nhớ.
- Ngoài ra, nếu mã độc đã được pack, re-pack hoặc pack đa lớp, việc phát hiện thông qua một giải pháp chống mã độc dựa trên signature là không khả thi. Điều này đặt ra nhu cầu cho một giải pháp phát hiện dựa trên phân tích tĩnh mà không phải là phương pháp thông thường.
- Việc xác định và phân loại các kỹ thuật packing được áp dụng trên các tệp thực thi có ý nghĩa quan trọng, vì nó giúp phát hiện bản chất thực sự của các tệp thực thi đáng ngờ. Do đó, các giải pháp bảo mật như hệ thống chống mã độc phải có khả năng xử lý (nhận diện hoặc phân loại) một số lượng lớn và đa dạng các packer, đồng thời phải có khả năng thích nghi với các packer hoàn toàn mới, vì các thuật toán packing mới được tạo ra hàng ngày.



Hình 2: *Phân cấp kỹ thuật packing.*

2.2. Phân loại packer

a) Compressors:

Các loại packer này chỉ được xây dựng để làm giảm kích thước của các tệp và thường không chèn các chương trình chống phân tích/unpacking stub vào các tệp mà chúng packing. Một số ví dụ về công cụ nén bao gồm UPack, UPX và ASPack.

b) Crypters:

Các loại packer này có các chương trình mã hóa và làm mờ được tích hợp vào logic của chúng và áp dụng các kỹ thuật chống phân tích này cho nội dung của các tệp mà chúng packing, do đó làm cho các tệp trở nên khó khăn trong việc phân tích tĩnh. Một số ví dụ về các công cụ mã hóa bao gồm Yoda's Crypter và PolyCrypt PE.

c) Protectors:

Các loại packer này kết hợp các phương pháp được sử dụng bởi các công cụ nén và mã hóa. Armadillo và Themida là ví dụ về các công cụ bảo vệ.

d) Bundlers:

Packer này được sử dụng khi cần phải nén và thực thi nhiều tệp dưới dạng một gói. Packer này được sử dụng để kết hợp dữ liệu cần thiết và các tệp thực thi vào một tệp thực thi duy nhất, và khi chạy, nó unpacking và truy cập các tệp trong gói mà không ghi vào đĩa (thực thi trong bộ nhớ). PEBundle và MoleBox là ví dụ về các bundlers.

2.3. Anti-Unpacking

- Các kỹ thuật chống unpacking để làm cho các lệnh unpacking khi thực thi tệp đã được pack trở nên khó đoán (phân tích).
- Các packer hiếm khi tích hợp các kỹ thuật chống unpacking. Tuy nhiên gần đây, việc sử dụng các kỹ thuật chống unpacking và phương pháp làm mờ đã tăng lên và chúng thường được áp dụng cho các packer.
- Xử lý các kỹ thuật chống debug thông qua việc unpacking thủ công là một công việc cực kỳ khó khăn và tốn thời gian.
- Phân loại thuật toán packing là bước đầu tiên trong việc xử lý các kỹ thuật chống debug (vì điều này sẽ cải thiện đáng kể thời gian phân tích).
- Phần mềm độc hại cố gắng phát hiện khi nó đang bị phân tích bởi các hệ thống chống vi-rút, để nó có thể điều chỉnh hành vi của mình để tránh kích hoạt bất kỳ cảnh báo nào.
- Khả năng của nó phát hiện sự hiện diện của bộ debug - phần mềm độc hại có thể đọc một số giá trị từ tệp hoặc sử dụng một số lệnh gọi API để xác định xem nó đang được debug.

- Để theo dõi quá trình tự giải mã của một packer, việc debug thông thường dựa trên các phương pháp phân tích động, chẳng hạn như debug, instrumentation nhị phân động và kỹ thuật API hooking.

Technique Name	Description
Anti-dumping	This technique changes the memory addresses of the running process in order to prevent further analysis of the dumped memory. The change is mainly employed in the following sections: PE header, imports, and entry point codes, which provide valuable information for analysts.
Anti-debugging	This technique is used by malware writers to identify when the code is being investigated by an external debugger (the most convenient tool for tracing code execution) and thwart this. This technique makes it difficult for signature-based solutions (such as AV) to easily use a debugger [48].
Anti-intercepting	Some unpacking mechanisms work by intercepting the execution of newly written pages, in order to guess when the unpacking process is finished and the control shifts to the host. This technique aims to prevent the packer from newly written page instructions [49].
Anti-emulating	This technique is used to attack the software environment, such as an emulator or a virtual machine, which is needed for safe code execution. Such attacks are dummy loops which delay the execution of the main malicious payload or increase error codes caused by invalid parameters which fail to decrypt the data [49].

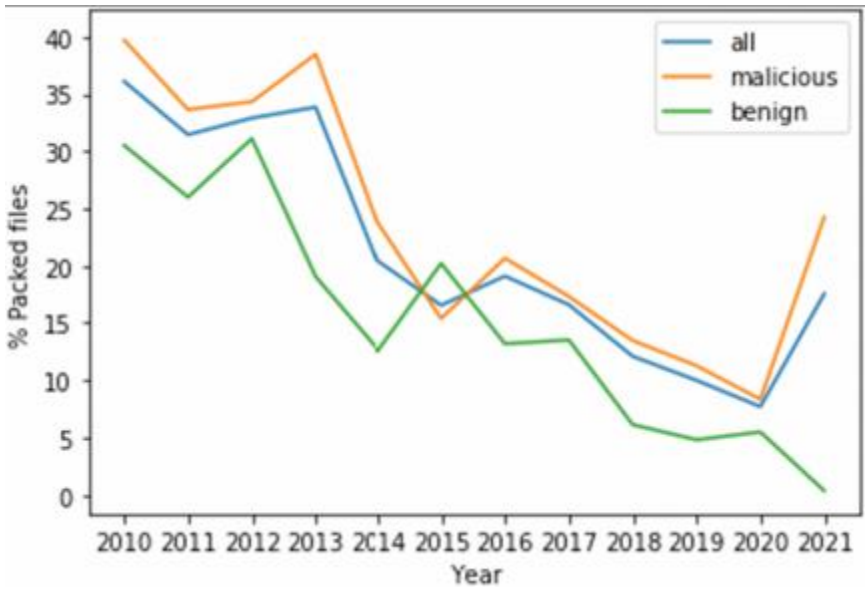
Bảng 3: *Mô tả các kỹ thuật chống unpacking.*

3. THỐNG KÊ

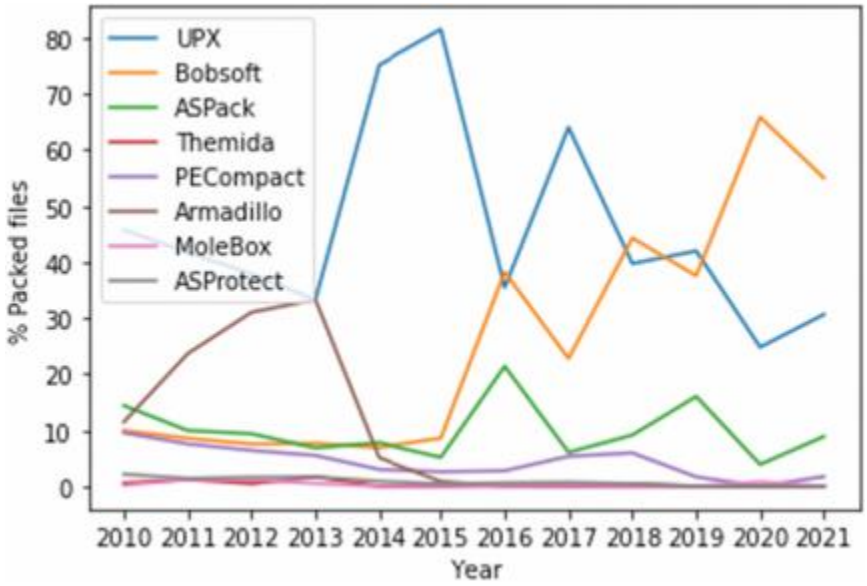
3.1. Xu hướng packing

Việc pack tệp thực thi là chủ yếu được sử dụng trên hệ điều hành Windows, mặc dù trong những năm gần đây nó cũng đang được sử dụng trên các nền tảng khác (ví dụ, gzexe cho Linux, VMProtect cho Mac OS X, UPX đa nền tảng và HASP). Một số nguyên nhân cho điều này là Windows cung cấp một bề mặt tấn công lớn hơn cho các kẻ tấn công và nó cũng là hệ điều hành phổ biến được sử dụng trên máy tính mục tiêu của nạn nhân.

Thống kê ở **Hình 3** mô tả về xu hướng packing, dựa trên việc quét và phân tích các tệp thực thi (PE) trong kho lưu trữ mã độc VirusTotal. Dữ liệu được thu thập thông qua việc chạy một truy vấn thông tin VirusTotal trên 2,000 tệp ngẫu nhiên cho mỗi năm từ 2010 đến 2021 (tổng số 24,000 tệp). Trung bình có 25% tệp lành tính và 75% tệp độc hại.



Hình 3: *Biểu đồ thể hiện xu hướng packing từ 2010 đến 2021 của 24,000 tệp PE trên VirusTotal.*



Hình 4: *Biểu đồ thể hiện xu hướng các packer từ 2010 đến 2021 của 24,000 tệp PE trên VirusTotal.*

Từ 2010 đến 2021, trung bình có 21.35% trên tổng số 24,000 tệp đã được pack. Có một giai đoạn xu hướng có vẻ ít hơn từ 2017 đến 2020 (trung bình 11.6% tệp đã được pack). Điều này có thể là do ảnh hưởng của xu hướng gia tăng trong thời gian đó của việc thực hiện các cuộc tấn công thông qua tệp tài liệu (thay vì thông qua tệp thực thi), như tệp .DOCX, tệp .PDF, hình ảnh .JPEG, và nhiều hơn nữa.

Hình 3 cũng hiển thị tỷ lệ phần trăm của các tệp độc hại và tệp lành tính. Như có thể thấy, tỷ lệ của các tệp độc hại thường cao hơn so với các tệp lành tính, với sự chênh lệch giữa các tỷ lệ đạt tới 22% trong một số năm.

3.2. Xu hướng các packer

Số lượng packer mới được tạo ra để pack mã độc ngày một tăng lên. Những người viết packer tạo ra các packer mới, sửa đổi packer hiện tại, sử dụng các chiến lược tiên tiến như anti-unpacking và các phương pháp né tránh (evasion), làm cho quá trình phân tích tĩnh thông thường của các nhà nghiên cứu bảo mật trở nên phức tạp và tốn thời gian hơn để phát triển các phương pháp nhận diện mã độc.

Các packer phổ biến nhất được sử dụng cho các tệp độc hại trong cùng ngữ cảnh trên được hiển thị trong **Hình 4** và chi tiết ở **Bảng 4**.

Có thể thấy, UPX là packer phổ biến nhất cho đến năm 2018. Điều này có thể được giải thích bởi việc nó miễn phí và dễ sử dụng. Hơn nữa, UPX không bị các nhà cung cấp phần mềm chống virus đưa vào danh sách đen. Tuy nhiên, vào năm 2018, một packer dựa trên Delphi, là BobSoft Mini Delphi, trở thành sự lựa chọn phổ biến nhất cho việc pack các tệp độc hại. Lý do cho điều này là:

- Thứ nhất, bằng khả năng kiểm tra các dấu hiệu khác nhau cho thấy nó đang ở trong môi trường bị phân tích (như chuyển động chuột không tự nhiên hoặc không tồn tại), Bobsoft giúp dễ dàng tránh được phân tích runtime nếu nó được thực thi trong môi trường chạy thử.
- Thứ hai, UPX kém hiệu suất so với một packer dựa trên Delphi như Bobsoft, vì nó chủ yếu hoạt động như một công cụ nén và không có các khả năng trên.

Các protector như ASProtect, Themida và MoleBox dường như ít phổ biến hơn, vì chúng đã bị đưa vào danh sách đen hoặc là công cụ có phí. ASPack, PECompact và Armadillo có vẻ là những lựa chọn tốt để pack.

Packer	Total Packed	Packing Share	Malicious Packed PE Files Out of the 2000 Randomly Sampled Files for each year											
			2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020	2021
Armadillo	534	12.99%	59	119	136	200	17	2	0	0	1	0	0	0
ASPack	416	10.12%	74	50	41	41	26	12	70	17	20	29	5	31
ASProtect	44	1.07%	11	7	7	10	3	1	2	2	1	0	0	0
BobSoft Mini Delphi	847	20.61%	51	43	33	46	23	20	125	64	97	68	85	192
EXECryptor	10	0.24%	1	4	2	3	0	0	0	0	0	0	0	0
eXPressor	1	0.02%	0	1	0	0	0	0	0	0	0	0	0	0
FSG	17	0.41%	6	3	2	6	0	0	0	0	0	0	0	0
KByS	1	0.02%	0	0	0	0	1	0	0	0	0	0	0	0
MEW	10	0.24%	2	0	2	6	0	0	0	0	0	0	0	0
MoleBox	16	0.38%	1	6	5	3	0	0	0	0	0	0	1	0
Morphine	2	0.04%	1	0	0	1	0	0	0	0	0	0	0	0
nPack	2	0.04%	0	0	0	2	0	0	0	0	0	0	0	0
NSPack	39	0.94%	7	4	4	17	2	0	2	2	0	1	0	0
Obsidium	2	0.04%	1	0	0	0	0	1	0	0	0	0	0	0
ORiEN	1	0.02%	1	0	0	0	0	0	0	0	0	0	0	0
Packman	2	0.04%	0	1	0	0	0	0	0	0	0	0	0	1
PECompact	210	5.11%	49	38	28	33	10	6	9	15	13	3	0	6
PENinja	1	0.02%	0	1	0	0	0	0	0	0	0	0	0	0
Petite	24	0.58%	4	1	4	3	0	0	1	0	0	0	0	11
RLPack	10	0.24%	4	1	1	3	0	0	0	0	0	1	0	0
Telock	12	0.29%	0	3	2	2	1	0	1	0	0	2	0	1
Themida	22	0.53%	3	6	2	10	0	0	1	0	0	0	0	0
UPack	36	0.87%	4	5	3	15	0	1	0	1	0	1	6	0
UPX	1850	45.02%	235	210	166	200	251	190	116	180	87	76	32	107

Bảng 4: *Thống kê chi tiết xu hướng các packer từ 2010 đến 2021 của 24,000 tệp PE trên VirusTotal.*

Hầu hết các packer đều miễn phí và dễ sử dụng. UPX và Themida là hai trong những packer phổ biến nhất trên hệ điều hành Windows. Một trong những lý do khiến Themida phổ biến là khả năng thực hiện ảo hóa mã và ngăn chặn các hành vi phân tích tệp đã được pack bằng công cụ này trong quá trình phân tích động. Mặc dù bản chất công cụ này được phát triển với mục tiêu bảo vệ ứng dụng Windows khỏi kẻ tấn công và các cuộc tấn công đánh cắp quyền sở hữu trí tuệ.

Các packer phổ biến khác bao gồm FSG, MEW, MPRESS, NSPack, ACProtect, MoleBox, PECompact, PELock, v.v . Tất cả các packer được đề cập trên đều là cho

các tệp PE và được hỗ trợ trên hệ điều hành Windows (UPX cũng hỗ trợ môi trường Linux). Có số ít packer hỗ trợ chỉ môi trường Linux, như Burneye và Midgetpack.

Theo một thống kê về mã độc trên Linux, trong số 380 tệp thực thi đã được pack, chỉ có 3 trong số đó không phải là biến thể của UPX. Tất cả các biến thể UPX đều được sửa đổi một cách khéo léo, sao cho tệp ELF không có vẻ như được pack bằng UPX, mặc dù tất cả chúng đều sử dụng cùng một thuật toán.

Packer Used	Related Hacking Groups / Malware
Custom Packer	Anchor, APT3, Dyre, Elderwood, Emotet, FatDuke, FinFisher, GreyEnergy, H1N1, IcedID, jRAT, Cobalt Strike, Soft Cell, TrickBot, Uroburos
UPX	APT29, APT39, China Chopper, Dark Caracal, Dark Comet, HotCroissant, OSX_OCEANLOTUS.D, Patchwork, Rocke, SeaDuke, TA505, Trojan.Karagany, yty, Zobrocy, ZeroT
Themida	APT38, Lazarus Group
Enigma	APT38
VMProtect	APT38, Metamorfo
Obsidium	APT38
RPolyCryptor	Astaroth
MPRESS	Dark Comet, Daserf

Bảng 5: *Tổng hợp một số packer được sử dụng bởi các nhóm hacker phổ biến.*

Bảng 5 liệt kê các packer được sử dụng bởi các nhóm hacker hoặc các mã độc phổ biến. Có thể thấy, một số nhóm sử dụng packer được tùy chỉnh để pack mã độc của họ, điều này nhấn mạnh thách thức bổ sung trong lĩnh vực phát hiện và phân loại packer và các kỹ thuật packing. Vì một packer tùy chỉnh có thể hoàn toàn khác biệt và có khả năng rất cao để né tránh các kỹ thuật nhận dạng theo mẫu (signature) ở hiện tại.

3.3. Các phương pháp đề xuất về việc phân tích, phát hiện kỹ thuật packing

Bảng 6 là thống kê về khả năng của các phương pháp đề xuất được nghiên cứu có thể phát hiện và phân tích các tệp thực thi PE (Windows) được pack bởi các packer phổ biến, các nghiên cứu trải dài từ 2012 đến 2020.

- *Phần trăm các nghiên cứu có thể chống lại packer thể hiện ở cột cuối (màu đỏ đối với giá trị dưới 20%, màu vàng đối với giá trị từ 20% đến 50%, và màu xanh lá cây đối với giá trị trên 50%).*
- *Tương tự, ở dòng cuối cùng của bảng, thể hiện phần trăm các packer bị chống lại bởi nghiên cứu được đề cập (biểu thị màu tương tự).*

Thông thường, các phương pháp nghiên cứu áp dụng hướng tiếp cận học máy (Machine Learning) có khả năng xử lý các packer chưa biết (không có trong tập huấn luyện), trong khi các phương pháp dựa trên signature không thể làm điều này. Bảng cũng bao gồm khả năng của các phương pháp thực hiện unpacking tự động tệp thực thi trước khi phân tích nó và khả năng xác định các cấp độ phức tạp của quá trình packing (pack một lớp, re-pack hoặc pack đa lớp).

Các nghiên cứu được xem xét trong bảng hướng tới ít nhất ba packer trong nghiên cứu của họ. Các packer bao gồm một số packer phổ biến nhất được sử dụng để pack mã độc trong môi trường thực. Mặc dù, danh sách này không phải là hầu hết các packer có trong môi trường thực, nhưng nó đã cung cấp cái nhìn bao quát về các packer có trong cộng đồng nghiên cứu.

Phần trăm các nghiên cứu chống lại một packer cụ thể chỉ ra sự phổ biến của packer đó trong các tập dữ liệu mã độc trong môi trường thực. Ngược lại, phần trăm các packer phổ biến mà một nghiên cứu có thể chống lại chỉ ra mức độ phủ sóng đa dạng các packer khác nhau trong các tập dữ liệu được sử dụng trong mỗi nghiên cứu. Như có thể thấy, phương pháp được đề xuất [26] chống lại số lượng lớn nhất các packer

khác nhau, với phủ sóng là 67.4%, trong khi UPX và Themida là những packer mà hầu hết các nghiên cứu (68.75%) có thể chống lại.

Bảng 7 trình bày so sánh giữa các nghiên cứu được thống kê bên trên theo các yêu cầu của mỗi nghiên cứu cần được đáp ứng trước khi phát hiện hoặc phân tích các tệp được pack, và khả năng mà mỗi nghiên cứu đó có để chống lại các kỹ thuật anti-unpacking hoặc né tránh phân tích thường được sử dụng bởi các packer.

Trong bảng, "Y" biểu thị "có", "N" biểu thị "không", "-" biểu thị "không áp dụng" và "M" biểu thị "có thể" (có thể có, nhưng không chắc chắn).

Có thể thấy rằng, hầu hết các phương pháp đều chống lại một số hình thức của né tránh phân tích, trong khi [26] có thể chống lại hầu hết mọi phương thức né tránh và chỉ thiếu về khả năng phân loại packer chưa biết. Yêu cầu mà mỗi phương pháp đề ra là áp lực đối với quá trình xử lý phân tích. Do đó, càng nhiều yêu cầu mà một phương pháp có, thì việc phân tích một tệp mất nhiều thời gian hơn.

Nhiều nghiên cứu thực hiện phân tích động dường như hoạt động dưới giả định rằng tệp đã được pack sẽ không sử các kỹ thuật anti-unpacking hoặc né tránh runtime. Bên cạnh đó, một số phương pháp phân tích tĩnh không xem xét khả năng đối với các tệp đã được mã hóa. Một số nghiên cứu cũng triển khai các giải pháp dễ bị ảnh hưởng về đầu ra, bởi sự chệch lệch kích thước các tệp đầu vào.

(Các đề xuất nghiên cứu được liệt kê trong các bảng chỉ mang tính chất thống kê, không đề cập đến chi tiết từng bài báo trích dẫn.)

Study / Packer	[93]	[94]	[95]	[96]	[97]	[99]	[100]	[101]	[26]	[21]	[3]	[65]	[22]	[1]	[19]	[6]	% of Studies that counter the packer
ACPROTECT							v		v								12.5%
ALLAPLE													v				6.25%
ALTERNATE_EXE															v	v	12.5%
ARMADILLO							v		v								12.5%
ARMPROTECTOR				v													6.25%
ASPACK	v	v	v			v	v	v	v					v	v	v	62.5%
ASPROTECT		v				v	v		v		v	v		v			43.75%
ENIGMA		v		v			v		v								25%
EXECRYPTOR				v	v												12.5%
EXPRESSOR					v	v	v		v								25%
FISHPACKER									v								6.25%
FSG	v				v	v	v		v		v	v	v	v	v		62.5%
KBYS									v								6.25%
MEW	v					v	v		v	v	v			v	v	v	56.25%
MOLEBOX							v		v								12.5%
MORPHINE				v						v					v		18.75%
MPRESS		v	v											v	v		25%
NPACK				v			v		v					v	v	v	37.5%
NSPACK						v	v		v		v		v	v	v	v	50%
OBSIDIUM		v		v			v		v								25%
ORIEN							v		v								12.5%
PACKMAN						v						v					12.5%
PECOMPACT						v	v		v		v						25%
PELOCK							v		v						v		18.75%
PENINJA				v													6.25%
PETITE			v	v			v		v		v			v			37.5%
PEP									v								6.25%
PESPIN							v		v								12.5%
POLYENE				v									v				12.5%
RLPACK							v		v			v		v	v	v	37.5%
SLV												v					6.25%
SOFTWAREPASSPORT							v		v								12.5%
TELOCK							v		v			v	v	v	v		37.5%
THEMIDA		v		v	v	v	v		v		v	v		v	v	v	68.75%
UPACK			v			v					v						18.75%
UPX	v	v	v		v	v	v	v		v	v	v	v				68.75%
UPX-iT															v		6.25%
UPXN									v						v		12.5%
VMPROTECT		v		v				v						v	v	v	37.5%
WINUPACK							v		v				v				18.75%
YODACRYPTOR				v	v		v		v				v	v	v		43.75%
YODAPROTECTOR							v		v								12.5%
ZPROTECT							v	v	v								18.75%
% of packers covered by the Study	9.3%	18.6%	11.6%	27.9%	13.9%	25.5%	60.4%	9.3%	67.4%	6.9%	20.9%	18.6%	18.6%	30.23%	37.2%	18.6%	

Bảng 6: *Tổng hợp khả năng chống lại các packer của các đề xuất nghiên cứu.*

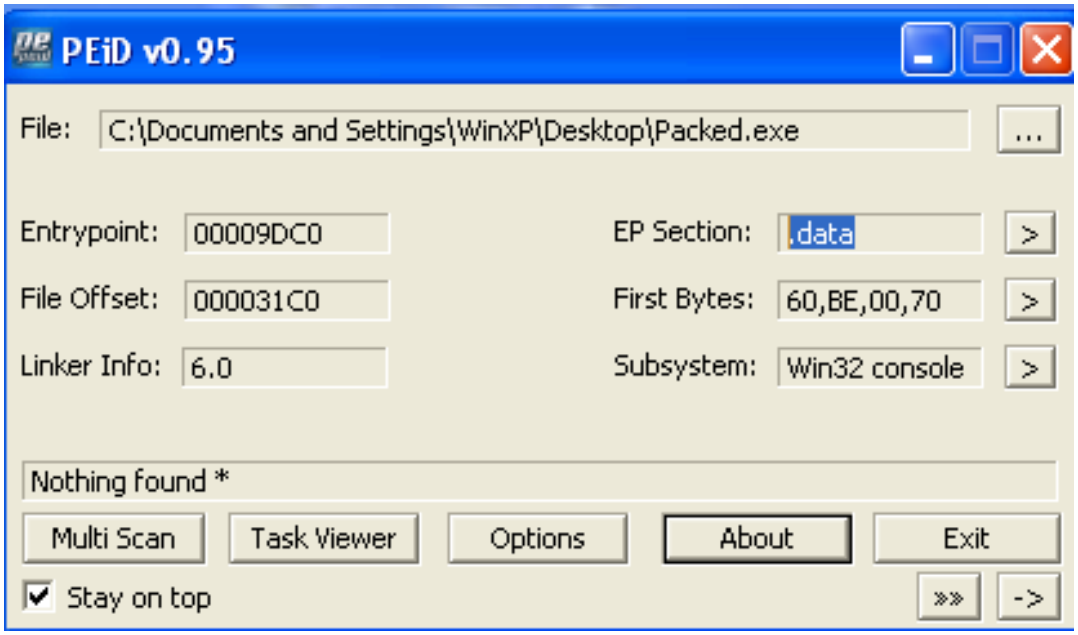
	Requirements									Capabilities						
Study	CFG generation	File disassembly	Signature generation	Bytewise analysis of file	Entropy analysis	Dynamic Analysis	Application-Level Emulation	API Hooking	No Requirement(works as is)	Handling Junk Code insertion	Handling Code Virtualization	OEP Detection	Detection of Polymorphism	Combats API Redirection	Combats DLL Hijacking	Classifies Unknown Packer
[97]	Y	Y	Y	N	N	N	N	N	N	N	N	N	N	N	N	N
[100]	N	N	N	N	N	N	N	N	N	Y	N	N	N	N	N	N
[21]	N	N	N	N	Y	N	N	N	N	N	N	N	N	N	N	N
[22]	N	N	N	N	N	N	N	N	Y	Y	N	N	N	N	N	N
[30]	Y	N	Y	N	Y	Y	Y	N	N	N	N	Y	Y	N	N	N
[26]	N	N	N	N	N	Y	N	Y	N	Y	Y	Y	Y	Y	Y	N
[93]	N	N	N	Y	N	N	N	N	N	M	Y	N	M	-	-	N
[3]	N	N	N	Y	N	N	N	N	N	M	Y	N	M	-	-	N
[94]	N	N	N	Y	N	N	N	N	N	M	N	N	M	-	-	N
[95]	N	N	N	Y	Y	N	N	N	N	N	N	N	N	-	-	N
[96]	N	Y	N	N	N	N	N	N	N	Y	M	N	N	-	-	N
[98]	N	N	N	Y	N	N	N	N	N	Y	N	N	N	-	-	N
[99]	N	N	N	Y	Y	N	N	N	N	Y	N	N	M	-	-	N
[103]	N	Y	N	Y	N	N	N	N	N	N	N	N	N	-	-	N
[10]	N	Y	N	N	Y	N	N	N	N	M	N	N	N	-	-	N
[65]	N	N	N	Y	Y	N	N	N	N	N	N	N	N	-	-	N
[1]	N	N	N	N	Y	Y	N	N	N	M	N	Y	N	N	N	Y
[101]	N	N	N	N	N	Y	N	Y	N	Y	N	N	N	M	M	N
[102]	Y	Y	N	N	N	Y	N	N	N	Y	N	Y	N	M	M	N

Bảng 7: *So sánh về yêu cầu và khả năng của các đề xuất nghiên cứu.*

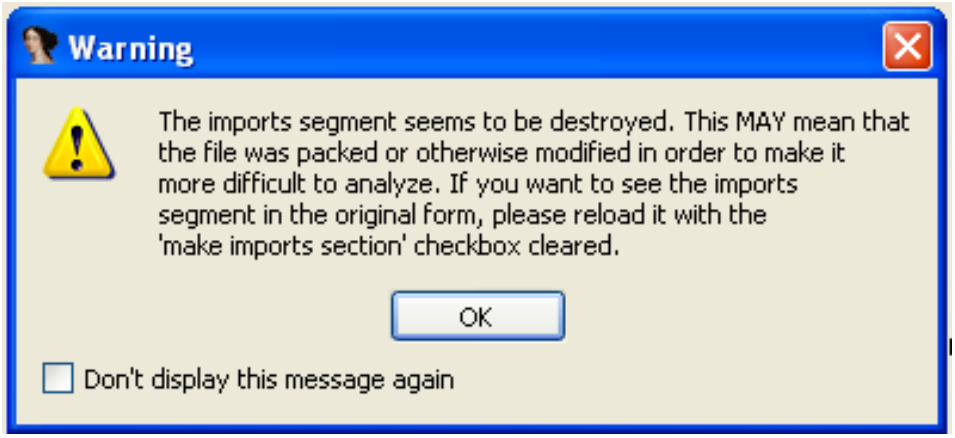
B. TRIỂN KHAI

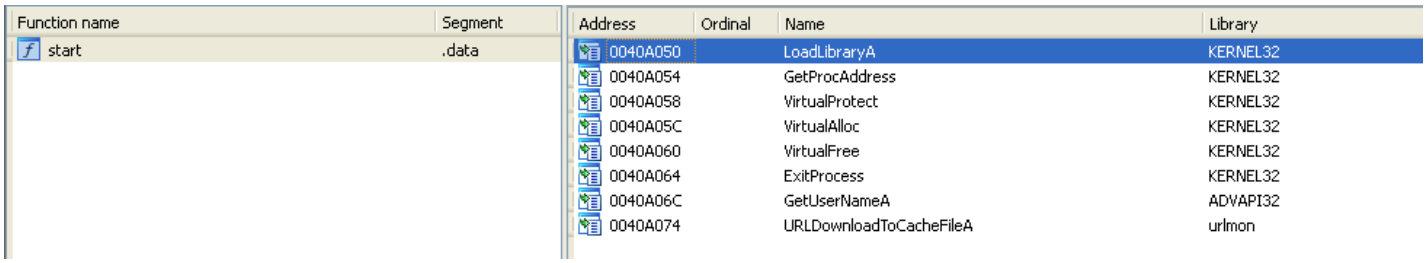
1. UNPACKING ĐỘNG THỦ CÔNG

- Các tools được sử dụng: **DiE, Pestudio, PeiD, IDA pro, x32dbg, plugin Scylla**.
- File ***Packed.exe*** được lấy từ Lab PracticalMalwareAnalysis.
- Xem trên **PEiD** thấy **EP Section là .data** và không phát hiện packer, có vẻ là file bình thường (không được pack) ???



- Xem file bằng **IDA pro**, khi mới mở thấy **warning một số imports segment đã bị hủy**, có thể là đã bị pack. **IDA pro** chỉ hiện hàm start và một số ít hàm được imports.

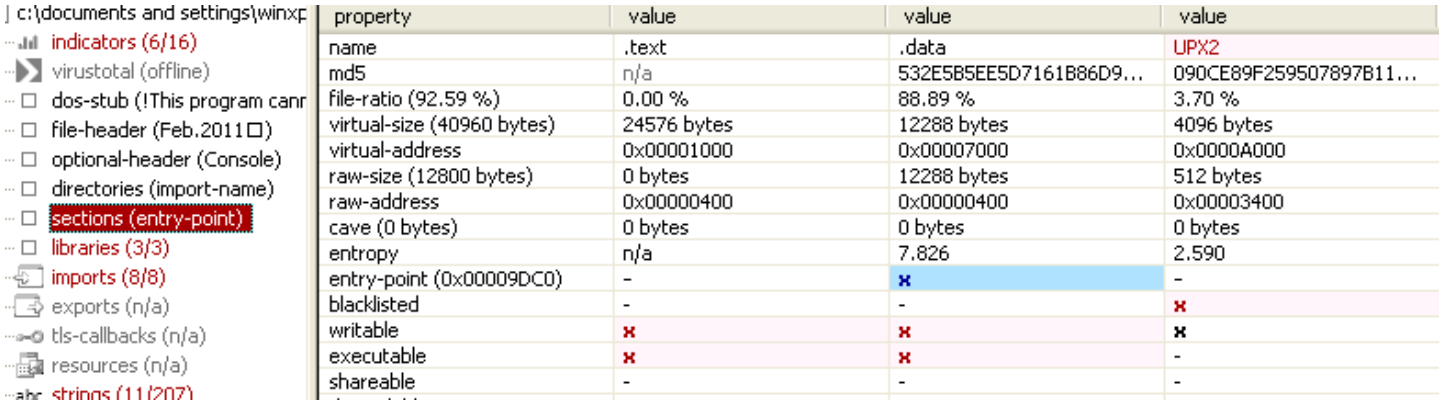




Function name	Segment
start	.data

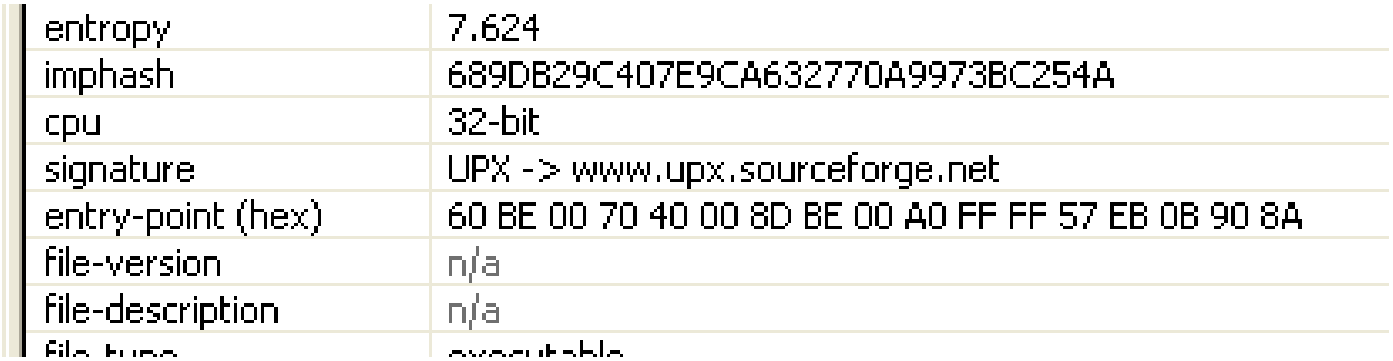
Address	Ordinal	Name	Library
0040A050		LoadLibraryA	KERNEL32
0040A054		GetProcAddress	KERNEL32
0040A058		VirtualProtect	KERNEL32
0040A05C		VirtualAlloc	KERNEL32
0040A060		VirtualFree	KERNEL32
0040A064		ExitProcess	KERNEL32
0040A06C		GetUserNameA	ADVAPI32
0040A074		URLDownloadToCacheFileA	urlmon

- Mở bằng **Pestudio** thấy signature là **UPX**, có section **UPX2**.



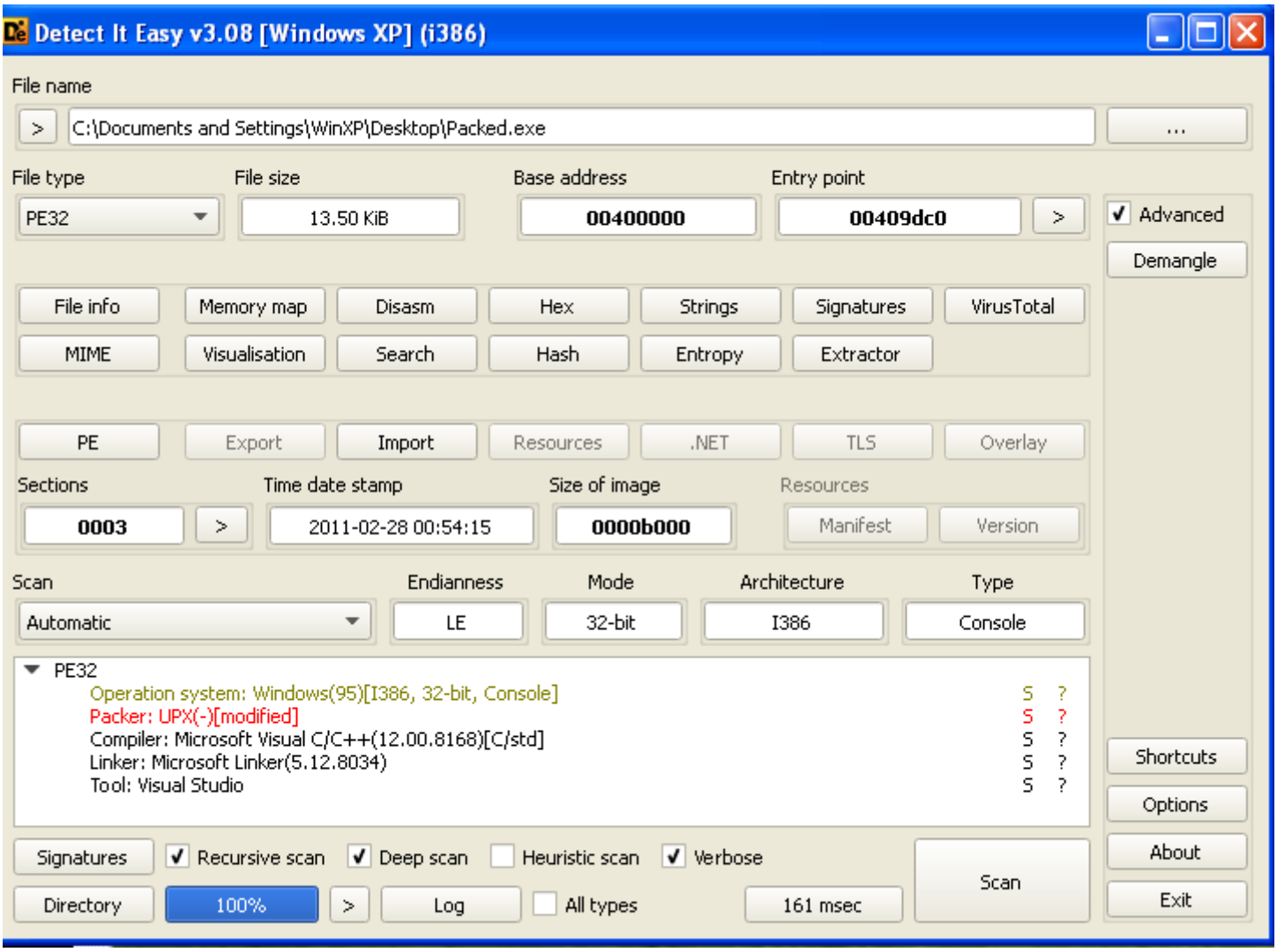
c:\documents and settings\winxp
- indicators (6/16)
- virustotal (offline)
- dos-stub (!This program cann
- file-header (Feb.2011)
- optional-header (Console)
- directories (import-name)
- sections (entry-point)
- libraries (3/3)
- imports (8/8)
- exports (n/a)
- tls-callbacks (n/a)
- resources (n/a)
- strings (11/207)

property	value	value	value
name	.text	.data	UPX2
md5	n/a	532E5B5EE5D7161B86D9...	090CE89F259507897B11...
file-ratio (92.59 %)	0.00 %	88.89 %	3.70 %
virtual-size (40960 bytes)	24576 bytes	12288 bytes	4096 bytes
virtual-address	0x00001000	0x00007000	0x0000A000
raw-size (12800 bytes)	0 bytes	12288 bytes	512 bytes
raw-address	0x00000400	0x00000400	0x00003400
cave (0 bytes)	0 bytes	0 bytes	0 bytes
entropy	n/a	7.826	2.590
entry-point (0x00009DC0)	-	x	-
blacklisted	-	-	x
writable	x	x	x
executable	x	x	-
shareable	-	-	-

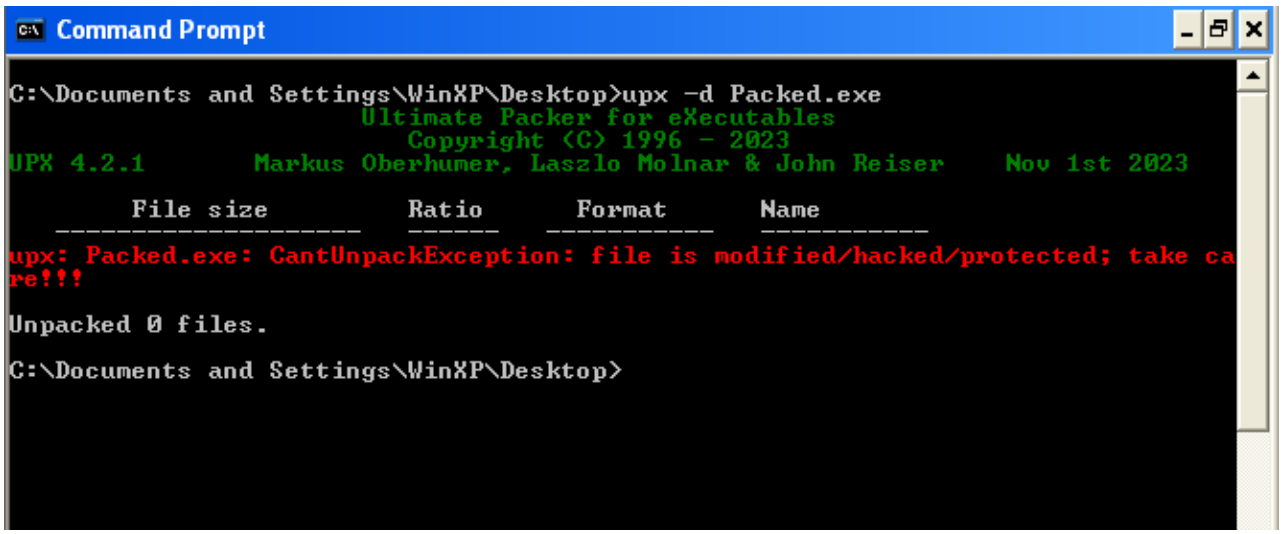


entropy	7.624
imphash	689DB29C407E9CA632770A9973BC254A
cpu	32-bit
signature	UPX -> www.upx.sourceforge.net
entry-point (hex)	60 BE 00 70 40 00 8D BE 00 A0 FF FF 57 EB 0B 90 8A
file-version	n/a
file-description	n/a
file-type	executable

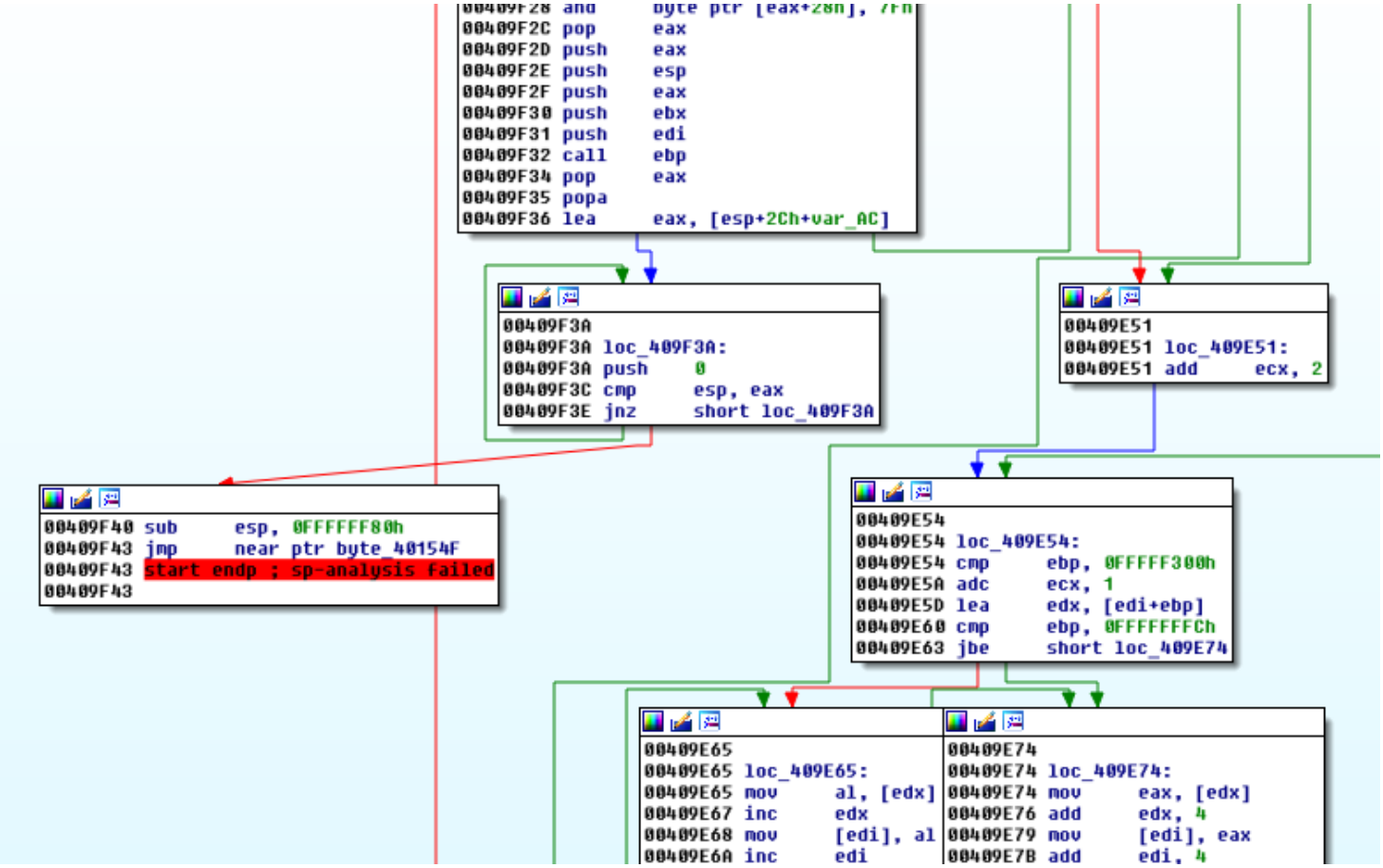
- Xem file bằng **DiE** thấy **có Packer** và đã bị **modified**.



- Sử dụng **upx** để unpack thử thì **không được**.



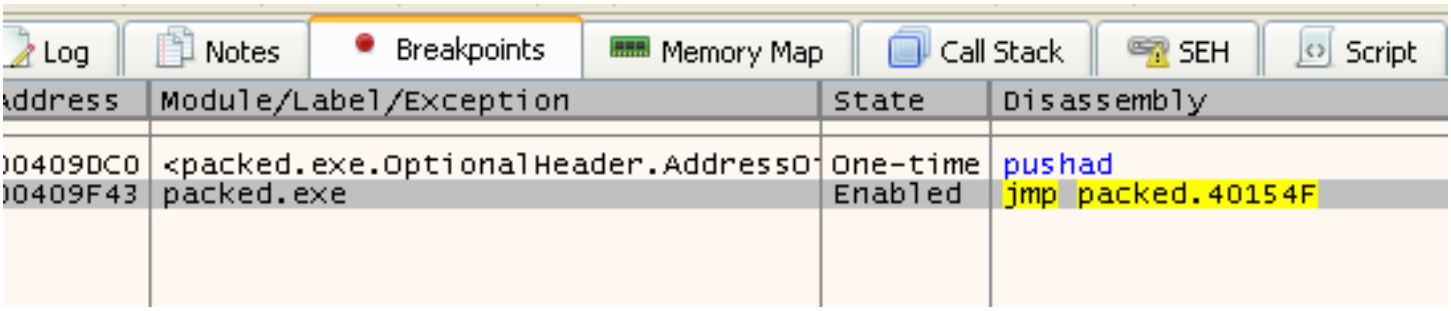
- Xem luồng thực thi của chương trình bằng **IDA pro** thì thấy có đoạn **nhảy đến vị trí khá xa** với vị trí đang thực hiện và **OEP** của chương trình (Tail Jump).



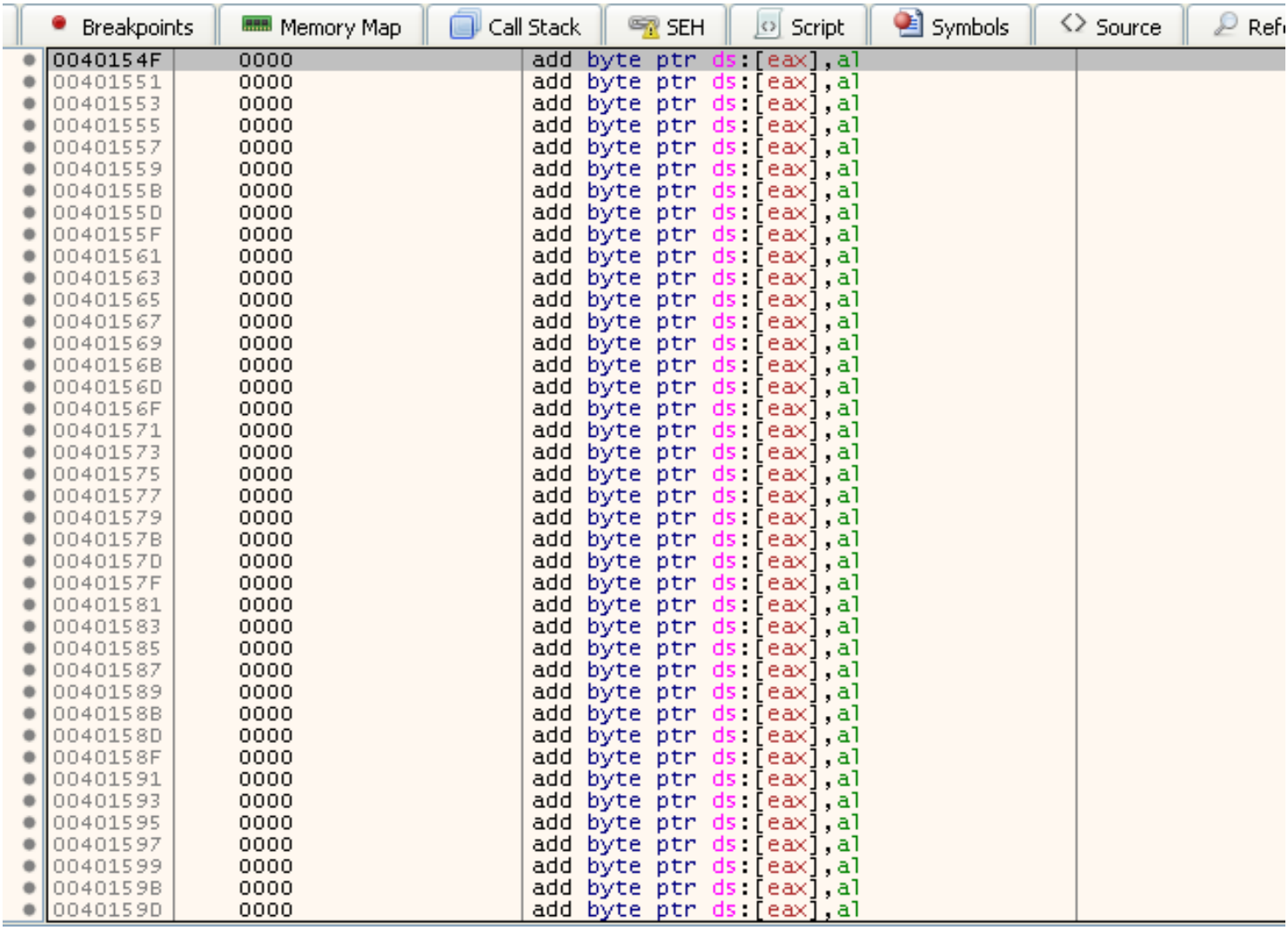
- Tại vị trí nhảy đến có nhiều dấu **"?"**, có thể là phần đã được pack.

```
.text:00401000 ; Segment type: Pure code
.text:00401000 ; Segment permissions: Read/Write/Execute
.text:00401000 _text           segment para public 'CODE' use32
.text:00401000                 assume cs:_text
.text:00401000                 ;org 401000h
.text:00401000                 assume es:nothing, ss:nothing, ds:_data, fs:nothing, gs:nothing
.text:00401000                 dd 153h dup(?)
.text:0040154C                 db 3 dup(?)
.text:0040154F byte_40154F     db ?                    ; CODE XREF: start+183↓j
.text:00401550                 dd 0EDCh dup(?)
.text:004050C0 byte_4050C0     db ?                    ; DATA XREF: .data:00407017↓r
.text:004050C1                 db 3 dup(?)
.text:004050C4                 dd 7CFh dup(?)
.text:004050C4 _text           ends
.text:004050C4
```

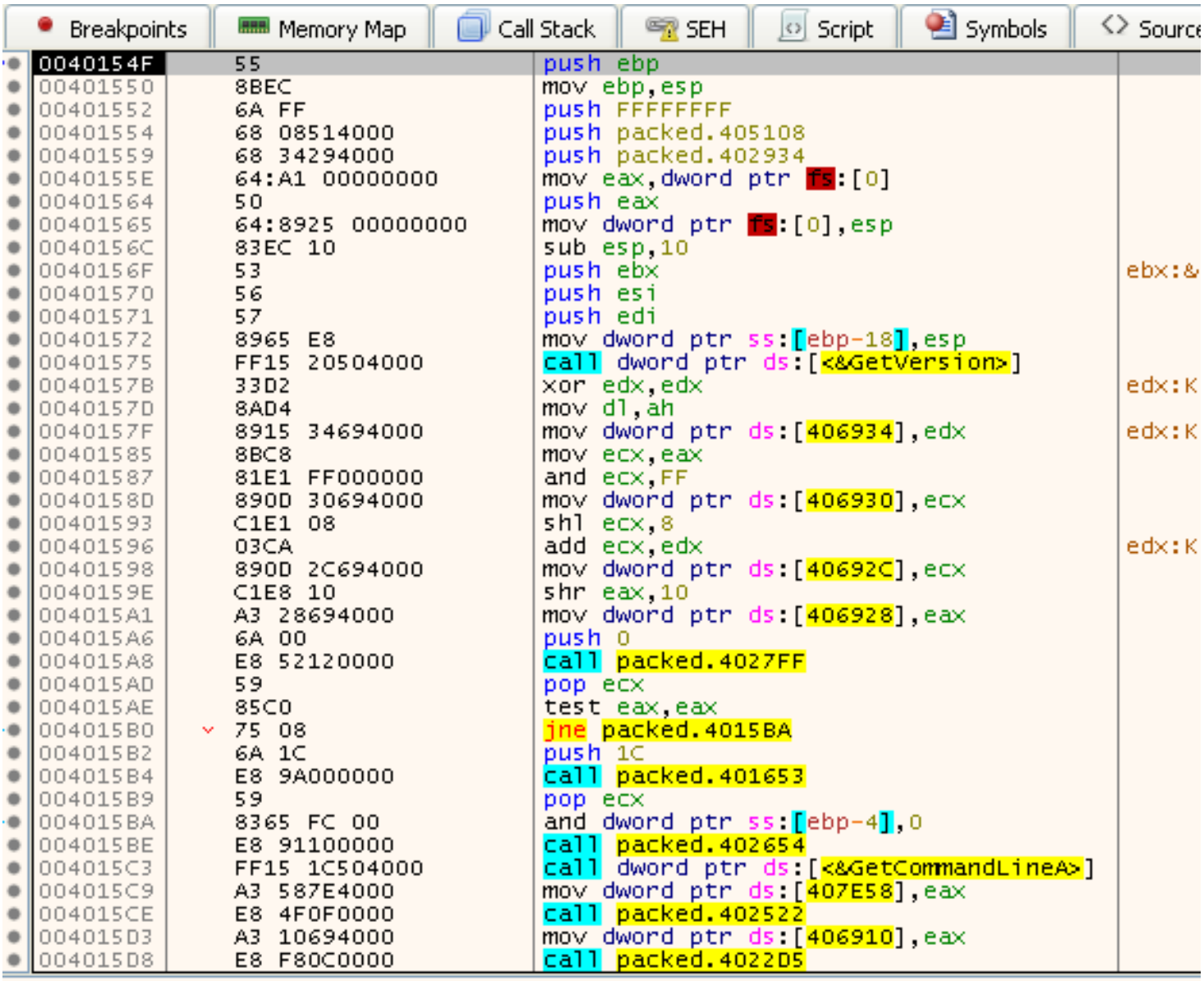
- Sử dụng **x32dbg** để thực hiện debug, đặt **breakpoint tại vị trí sẽ thực hiện lệnh nhảy**.



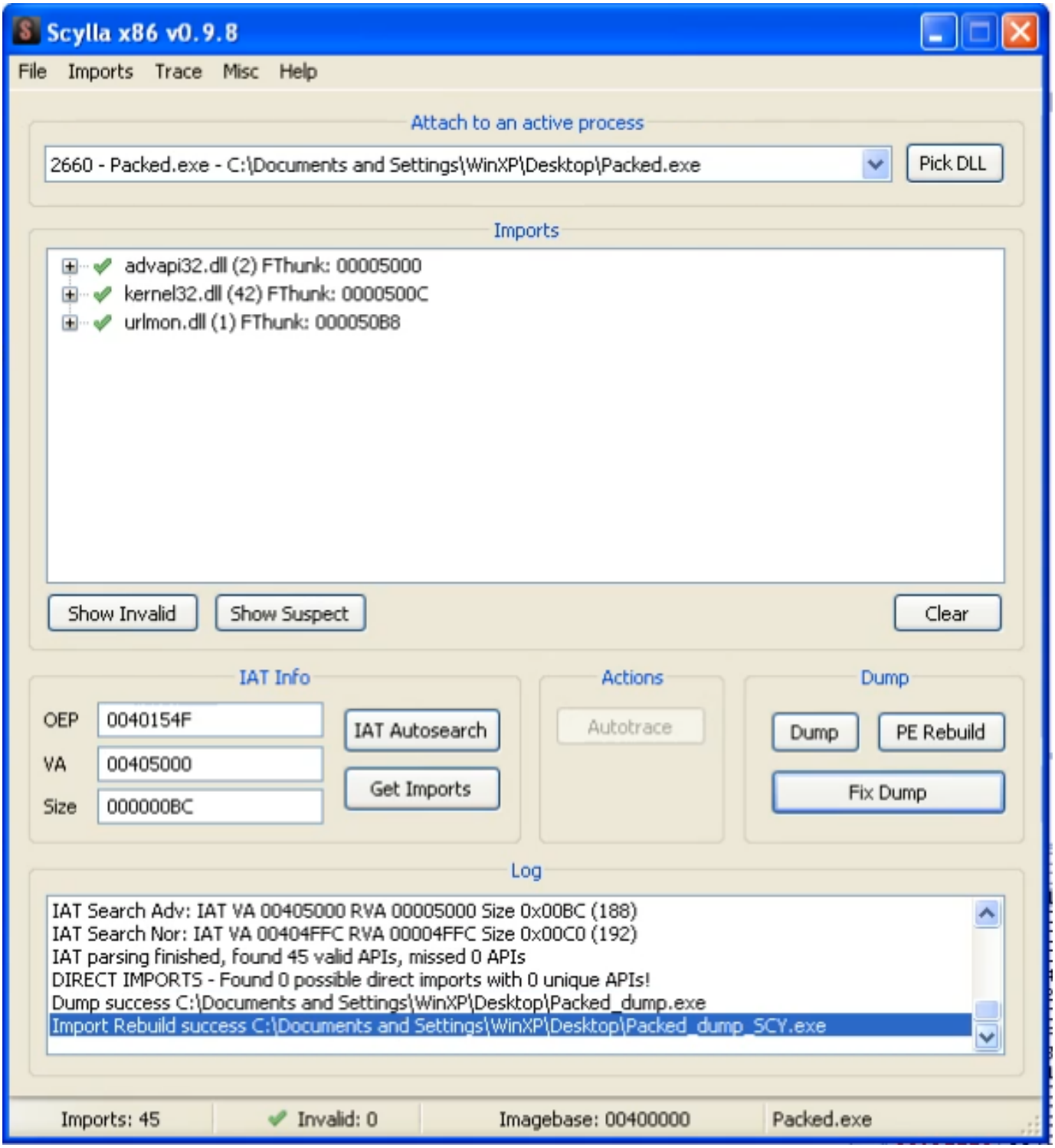
- Tại vị trí sẽ nhảy trước khi thực thi chương trình chỉ là **những bytes rỗng**.



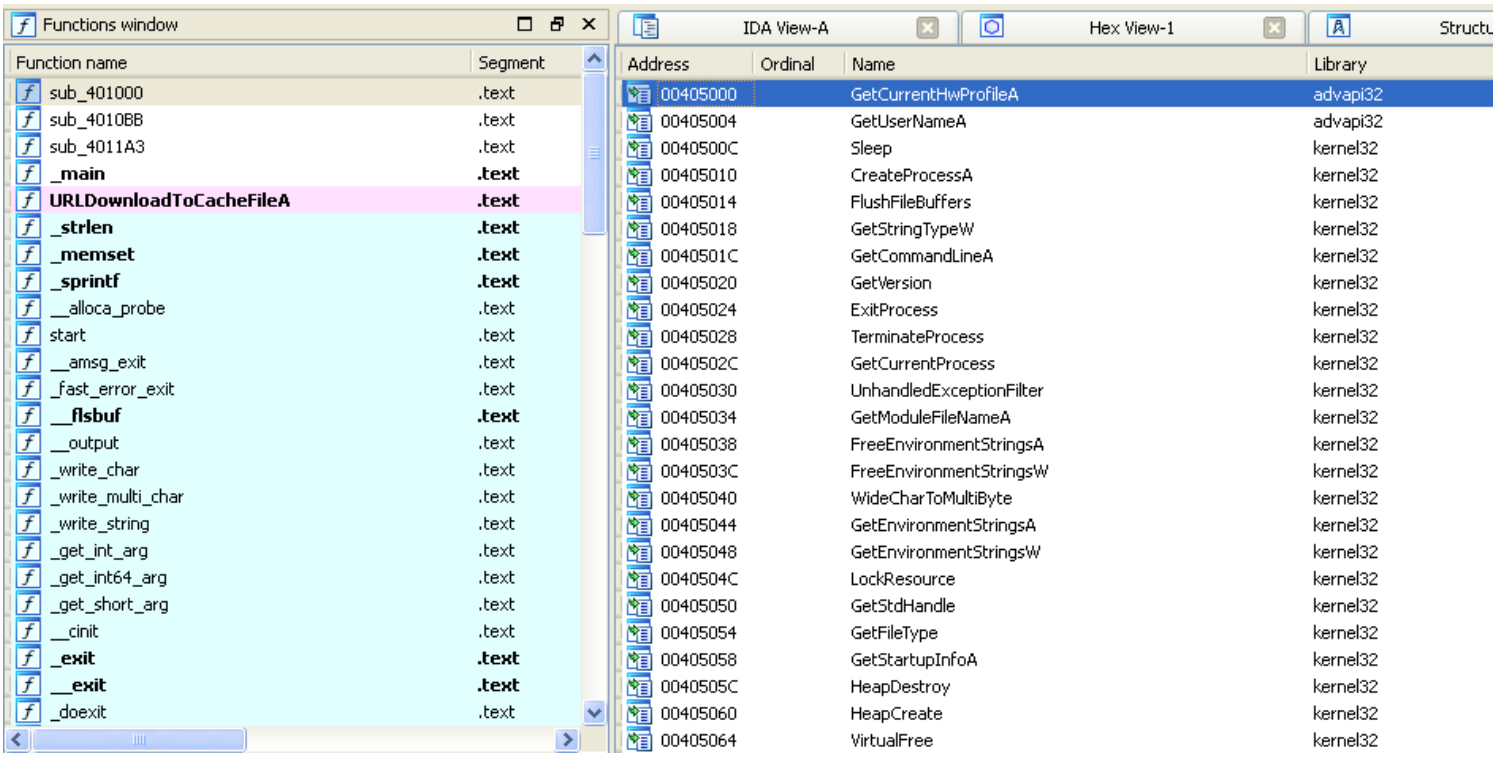
- Sau khi thực hiện đến breakpoint, vị trí nhảy đến **có các lệnh**.



- Sử dụng **plugin Scylla** để thực hiện **kết xuất đoạn mã sau khi được unpack vào chương trình ban đầu** và **thay đổi OEP mới đến nơi thực hiện đoạn code của mã độc**. *(Dump để thêm code vào chương trình mới, Fix Dump để thực hiện liên kết các lời gọi hàm, thư viện)*



- Sau khi thực hiện xong, mở file ***Packed_dump_SCY.exe*** vừa được tạo ra bằng **IDA pro**. Lúc này các hàm, các imports đã được thêm vào.



- Các bước thực hiện unpacking động thủ công đã thực hiện:
  1. Xác định Original Entry Point (OEP) của chương trình gốc.
  2. Thực hiện đặt breakpoint tại vị trí sẽ nhảy tới OEP, sau đó thực thi chương trình đến breakpoint đó. Mục đích là để phần unpacking stub được thực thi và unpack mã độc.
  3. Thực hiện kết xuất đoạn mã được unpack vào chương trình, đặt OEP về vị trí chương trình trước khi pack.
  4. Thực hiện sửa lại các Import Address Table (IAT) từ file sau khi kết xuất.

2. MÔ HÌNH HỌC MÁY PHÂN LOẠI MÃ ĐỘC DỰA TRÊN TẬP DỮ LIỆU VỀ PACKING

Wild dataset: được gọi tên là "hoang dã" vì tập dữ liệu được lấy từ các mẫu tệp thực thi PE (Windows x86) 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 unpack 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 unpacked malicious bị tác giả loại bỏ do số lượng ít và không cần thiết cho thí nghiệm trong bài báo. Trong thực nghiệm được triển khai, điều đó không ảnh hưởng quá lớn đến kết quả.)*

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

- Tác giả đã sử dụng thư viện pefile để trích xuất thuộc tính từ ba nguồn khác nhau: cấu trúc tệp PE, mã hợp ngữ của chương trình và các raw byte của tệp thực thi nhị phân. Các thuộc tính trích xuất được 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		

Mô hình học máy phân loại mã độc được xây dựng với 4 thuật toán, bao gồm: **DecisionTree, RandomForest, GradientBoosting, AdaBoost**. Thực nghiệm triển khai trên **Google Colaboratory.** *(chi tiết quá trình thực nghiệm được chú thích trong Demo2.ipynb được đính kèm)*

- Từ **Wild dataset** với 50,724 mẫu, chỉ lấy 7,000 mẫu để tiến hành sàn lọc và chia tỉ lệ cho 2 tập train và test (Colab có hạn về RAM).
- Tỉ lệ tập train – test được chia gồm: **3402 mẫu cho tập train – 1338 mẫu cho tập test**. Với mỗi tập, tỉ lệ nhãn **Benign - Malware là 50:50**.
- Áp dụng thuật toán **ExtraTreesClassifier** để từ 56,543 thuộc tính gốc, chọn lọc ra **6,544 thuộc tính quan trọng nhất** để tiến hành xử lý.
- Kết quả Accuracy như sau:
  - DecisionTree: 92.526158 %
  - RandomForest: 94.544096 %
  - GradientBoosting: 95.216741 %
  - AdaBoost: 94.917788 %

- Kết quả chi tiết cho **GradientBoosting** có Accuracy cao nhất:
  - True positive: 639
  - True negative: 635
  - False positive: 34
  - False negative: 30

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

       False       0.95      0.95      0.95       669
        True       0.95      0.96      0.95       669

    accuracy                           0.95      1338
   macro avg       0.95      0.95      0.95      1338
weighted avg       0.95      0.95      0.95      1338
```

Mặc dù không thể bao quát toàn bộ Wild dataset, nhưng thực nghiệm cũng cho thấy được tập dữ liệu đã được trích xuất, xử lý nhãn kỹ càng đối với các tệp thực thi PE đã được pack. Kết quả của mô hình phân loại cho thấy tập dữ liệu có thể được áp dụng hiệu quả trong việc phát hiện mã độc. Đồng thời, với việc hướng tới nghiên cứu khả năng xử lý các tệp được pack trong tập train, điều đó hỗ trợ các mô hình phân loại mã độc nâng cao chất lượng của chúng và giúp các trình phát hiện mã độc bao quát được nhiều ngữ cảnh, nhiều các biến thể khác nhau của mã độc.

C.TRÍCH DẪN

- *Nội dung lý thuyết, thống kê:* Trivikram Muralidharan, Aviad Cohen, Noa Gerson, and Nir Nissim. 2022. File Packing from the Malware Perspective: Techniques, Analysis Approaches, and Directions for Enhancements. ACM Comput. Surv. 55, 5, Article 108 (May 2023), 45 pages. https://doi.org/10.1145/3530810

- *Đề xuất nghiên cứu [26] trong thống kê:* Binlin Cheng, Jiang Ming, Jianmin Fu, Guojun Peng, Ting Chen, Xiaosong Zhang, and Jean-Yves Marion. 2018. Towards Paving the Way for Large-Scale Windows Malware Analysis: Generic Binary Unpacking with Orders-of-Magnitude Performance Boost. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18). Association for Computing Machinery, New York, NY, USA, 395–411. https://doi.org/10.1145/3243734.3243771

- *Bài báo của Wild dataset:* Aghakhani, Hojjat, Gritti, Fabio, Mecca, Francesco, Lindorfer, Martina, Ortolani, Stefano, Balzarotti, Davide, Vigna, Giovanni, & Kruegel, Christopher. When Malware is Packin' Heat; Limits of Machine Learning Classifiers Based on Static Analysis Features. Network and Distributed Systems Security (NDSS) Symposium 2020, (). Retrieved from https://par.nsf.gov/biblio/10155112. https://doi.org/10.14722/ndss.2020.24310

D.PHỤ LỤC

- Tài nguyên báo cáo đính kèm: Liên kết Google Drive
  - Báo cáo chi tiết **(.docx và .pdf)**.
  - Slides thuyết trình **(.pptx và .pdf)**.
  - Video ***Demo1*** về phần triển khai Unpacking động thủ công **(.mp4)**.
  - Tệp ***Demo2*** về phần triển khai Mô hình phân loại mã độc **(.ipynb)**.
- Đóng góp, phân công công việc:

	Công việc	Mạnh Cường	Đức Anh
1	Tổng hợp lý thuyết	x	
2	Tổng hợp thống kê		x
3	Triển khai 1 (Unpacking)		x
4	Triển khai 2 (Học máy)	x	
	TỔNG KẾT	**50%**	**50%**