Kỷ yếu Hội nghị Khoa học Quốc gia lần thứ IX "Nghiên cứu cơ bản và ứng dụng Công nghệ thông tin (FAIR'9)"; Cần Thơ, ngày 4-5/8/2016
DOI: 10.15625/vap.2016.00086

SỬ DỤNG PHƯƠNG PHÁP HÌNH THỨC ĐỂ NHẬN DẠNG PACKER

Nguyễn Minh Hải [1], Đỗ Duy Phong[1], Quản Thành Thơ[1]

[1] Khoa Khoa học và Kỹ thuật máy tính, Trường Đại học Bách Khoa Thành phố Hồ Chí Minh

hainmmt@cse.hcmut.edu.vn, doduyphongbktphcm@gmail.com, qttho@cse.hcmut.edu.vn

TÓM TẮT— *Hiện nay, an toàn thông tin là một vấn đề vô cùng quan trọng và thu hút sự quan tâm rất lớn. Những phần mềm độc hại (malware) đã và đang dần trở thành mối đe doạ thực sự đối với mỗi quốc gia. Năm 2006, theo một thống kê bởi Computer Economics 2007 Malware Report, khoảng 13.3 tỉ đô la đã được tiêu tốn để khắc phục những hậu quả của malware và con số này vẫn tiếp tục gia tăng do mức độ độc hai và tinh vi của malware ngày càng lớn. Để phân tích và phát hiện malware, đa phần những phần mềm trong công nghiệp sử dụng kỹ thuật nhận dạng chữ ký. Trong kỹ thuật này, mỗi malware sẽ được biểu diễn dưới dạng một chuỗi bit nhị phân duy nhất đặc trưng cho malware. Tuy nhiên, với việc những malware ngày này sử dụng các phần mềm đóng gói (packer) để tạo ra nhiều biến thể mới đã khiến kỹ thuật nhận dạng chữ ký trở nên không hiệu quả. Hơn thế nữa, khi phân tích malware đã được đóng gói, kỹ thuật dịch ngược vốn được xem là kỹ thuật hiệu quả để phát hiện malware cũng trở nên rất khó khăn và nhiều thách thức bởi packer đã sử dụng rất nhiều kỹ thuật rắc rối hóa, cũng như kỹ thuật chống dịch ngược và chống gỡ lỗi. Xuất phát từ những khó khăn thực tế đó, bài báo đề xuất việc sử dụng phương pháp hình thức nhằm nhận dạng packer bằng cách kết hợp giữa hai công cụ BE-PUM và công cụ NuSMV. Chúng tôi đã phát triển công cụ BE-PUM (Binary Emulator for PU shdown Model generation) với mục tiêu xây dựng mô hình chính xác của một chương trình mã nhị phân và xử lý được những kỹ thuật obfuscation đặc trưng của packer như lệnh nhảy không trực tiếp, code tự thay đổi... Hiện tại, BE-PUM đã xử lý được các kỹ thuật sử dụng trong 28 packer khác nhau. Bằng cách biểu diễn hành vi của các kỹ thuật này dưới dạng công thức CTL và cung cấp như một đầu vào của công cụ kiểm tra mô hình NuSMV, chúng tôi có thể nhận diện được 28 packer này. Chúng tôi đã thí nghiệm hướng tiếp cận này trên 3250 malware khác nhau để nhận diện các packer được sử dụng. Kết quả phân tích đã chỉ ra hướng tiếp cận này là rất khả thi và sinh ra kết quả chính xác hơn so với kỹ thuật nhận diện packer thông qua chữ ký truyền thống.*

Từ khóa— *Phân tích virus, an toàn thông tin, phương pháp hình thức, packer.*

I. GIỚI THIỆU

Malwares là những chương trình máy tính độc hại xâm nhập vào hệ thống một cách trái phép với mục tiêu là ăn cắp thông tin, phá hủy hay làm hư hỏng hệ thống [1]. Những chương trình malware này gây ra những thiệt hại to lớn về mặt tiền bạc đối với các cá nhân, tổ chức và các cơ quan chính phủ. Malware được chia thành nhiều loại khác nhau như: virus, trojan horse, spammer…Có 2 phương pháp chính để phát hiện malware, có thể kể đến phương pháp nhận diện chữ ký (signature recognition) và mô phỏng (emulation) quá trình thực thi chương trình trong môi trường có kiểm soát, thường là hộp cát (sandbox). Signature recognition là kỹ thuật nhận diện malare thông qua những cấu trúc mẫu bit đặc trưng của malware này. Kỹ thuật này được áp dụng rất rộng rãi trong công nghiệp và đạt được những thành công to lớn. Tuy nhiên, hiện nay signature detection gặp phải khó khăn lớn do những virus thế hệ mới áp dụng những kỹ thuật rắc rối hóa (obfuscation technique) để làm thay đổi chữ ký đặc trưng [2]. Ý tưởng chính của emulation là xây dựng một sandbox để khám phá hành vi của malware, bằng cách mô phỏng toàn bộ quá trình thực thi của hệ thống và đặc biệt là hoạt động của APIs [3,15]. Tuy nhiên, đây là một kỹ thuật rất phức tạp, tốn thời gian và đòi hỏi chi phí lớn.

Hầu hết các malware hiện nay đều được đóng gói bằng các chương trình đóng gói (packer) [4]. Packer là một chương trình chuyển đổi mã nhị phân của chương trình thành một chương trình thực thi khác. Chương trình thực thi mới này vẫn gìn giữ những tính năng nguyên bản nhưng có nội dung hoàn toàn khác nhau khi được lưu trữ. Chính vì điều này đã làm cho kỹ thuật quét signature không thể liên kết giữa 2 phiên bản này. Theo thống kê trên [4], 80% malware được nén bởi rất nhiều loại packers khác nhau. Những packer thông dụng nhất có thể kể đến UPX[1], PECOMPACT[2], TELOCK[3], FSG[4] và YODA's Crypter[5]. Đa phần các chương trình nhận dạng packer hiện nay như Peid[6] đều dựa trên chữ ký để kiểm tra. Tuy nhiên, do các hacker có thể tùy chỉnh (modify) chữ ký của packer, phương pháp này sẽ dễ dàng bị đánh bại khi được áp dụng trong phân tích malware thực.

Trong bài báo này, chúng tôi đề xuất một hướng tiếp cận mới để nhận diện packer. Chúng tôi sử dụng phương pháp hình thức nhằm nhận dạng packed malware bằng cách kết hợp giữa hai công cụ, BE-PUM để xây dựng CFG và công cụ kiểm tra mô hình NUSMV. Để kiểm chứng tính hiệu quả của hướng tiếp cận này, chúng tôi đã tiến hành thí nghiệm trên tập 3250 malware khác nhau. Cấu trúc bài báo được mô tả như sau. Phần 2 giới thiệu ngắn gọn những kiến thức nền tảng về Packer, BE-PUM và công cụ NuSMV. Phần 3 trình bày phương pháp đề xuất để nhận dạng packer. Trong phần 4, chúng tôi giới thiệu về thí nghiệm và phần 5 trình bày kết luận và một số hướng nghiên cứu trong tương lai.

[1]http://upx.sourceforge.net
[2]https://bitsum.com/pecompact
[3]http://www.telock.com-about.com
[4]http://fsg.soft112.com
[5]http://www.yodas-crypter.com-about.com
[6]http://www.secretashell.com/codomain/peid

II. KIẾN THỨC NỀN TẢNG

A. Giới thiệu Packer

Packer [5, 6] là một công cụ phần mềm dùng để đóng gói một tập tin. Packer có 3 mục tiêu cụ thể được mô tả như sau.

- Giảm kích thước của tập tin: đây cũng là mục tiêu mà rất nhiều packer hướng tới, bằng việc áp dụng các kỹ thuật đóng gói và mã hóa. Kích thước của một tập tin được đóng gói qua đó sẽ giảm đi một cách đáng kể. Tuy nhiên, những packer hiện đại ngày nay đã không còn đặt tiêu chí giảm kích thước tập tin lên hàng đầu. Nguyên nhân là vì giảm kích thước đồng nghĩa với việc sử dụng những kỹ thuật bảo mật khác sẽ bị loại bỏ và do đó độ tin cậy của một packer cũng giảm đi. Thậm chí những packer có cơ chế bảo mật tốt hiện nay như Themida hay TELOCK có thể làm tăng kích thước tập tin lên rất nhiều lần.
- Chống dịch ngược: packer được sử dụng như một công cụ chống dịch ngược rất hiệu quả vì những giải thuật đóng gói tập tin phức tạp của packer có thể khiến việc dịch ngược mã nhị phân trở nên vô cùng khó khăn.
- Bảo vệ bản quyền của phần mềm: đây cũng là mục tiêu chính yếu nhất mà các packer ngày nay hướng tới. Bằng cách nâng cao độ phức tạp của các giải thuật đóng gói, đồng thời áp dụng những kỹ thuật chống sửa lỗi, chống dịch ngược, các packer giúp bảo vệ một phần mềm khỏi việc đánh cắp thông tin bất hợp pháp. Malware cũng lợi dụng điểm mạnh này của packer nhằm che dấu quá trình thực thi của mình.

Packer sử dụng rất nhiều kỹ thuật làm rắc rối hóa (obfuscation technique) nhằm mục tiêu làm cho quá trình phân tích trở nên rất khó khăn. Chúng tôi đã phân loại các obfuscation technique [7,13] này thành nhóm các kĩ thuật nén-giải nén (packing-unpacking), ghi đè (overwriting), lệnh nhảy động (indirect jump), bất biến rối rắm (obfuscated constant), chẻ code (code chunking), mượn byte (stolen byte), kiểm tra (checking), bẫy lỗi ngoại lệ (SEH), API đặc biệt (2API), chống sửa lỗi (anti-debugging), code tự thay đổi (dynamic code), hàm lồng nhau (overlapping function), khối lồng nhau (overlapping block), chống ghi đè (anti-rewriting) và kiểm tra thời gian (timing check). Chi tiết cách phân loại, chúng ta đã trình bày trong [13]. Bảng 1 trình bày tóm tắt các kỹ thuật được hỗ trợ trong một số packer. Ví dụ như, trong bảng 1, packer UPX hỗ trợ các kĩ thuật, packing-unpacking, overwrting, indirect jump, obfuscated constants, checksumming, 2API, dynamic code, code layout, overlapping blocks và anti-rewriting.

Bảng 1. Các obfuscation techniques được hỗ trợ trong packer

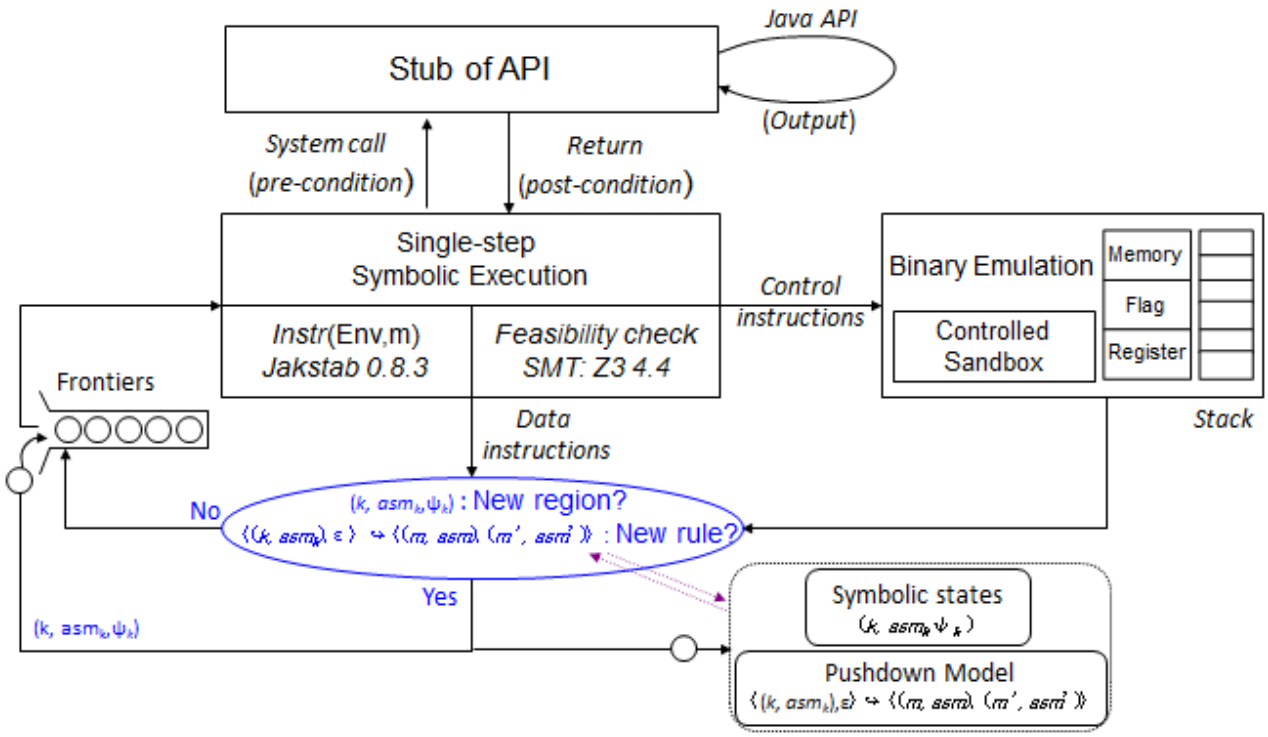
Name	UPX	ASPACK	FSG	NPACK	PECOMPACT	PETITE	YODA
Packing-Unpacking	x	x	x	x	x	x	x
Overwriting	x	x	x	x	x	x	x
Indirect Jump	x	x	x	x	x	x	x
Obfuscated Constants	x	x	x	x	x	x	x
Code Chunking		x	x	x		x	x
Stolen Bytes		x		x	x		
Checksumming	x	x	x	x	x	x	x
SEH					x	x	
2API	x		x	x	x		x
Anti-Debugging							x
Dynamic Code	x	x	x	x	x	x	x
Code Layout	x	x	x	x	x	x	x
Overlapping Function		X	x	x	x	x	x
Overlapping Blocks	x	X	x	x	x	x	x
Anti-Rewriting	x	X	x	x	x	x	x
Timing Check							

B. BE-PUM

1. Giới thiệu BE-PUM

BE-PUM (Binary Emulation for Pushdown Model generation of Malware) [8,9] là một công cụ xây dựng luồng điều khiển của một tập tin thực thi dựa trên kỹ thuật thực thi ký hiệu động [9]. BE-PUM sử dụng thư viện mã nguồn mở Jackstab [10, 11, 14] để dịch ngược mã nhị phân cho từng câu lệnh hợp ngữ tương ứng với từng câu lệnh thực thi của tập tin và chương trình SMT Z3.4.4[7] để giải các điều kiện, từ đó tìm ra đường đi có tính khả thi.

[7]https://z3.codeplex.com/

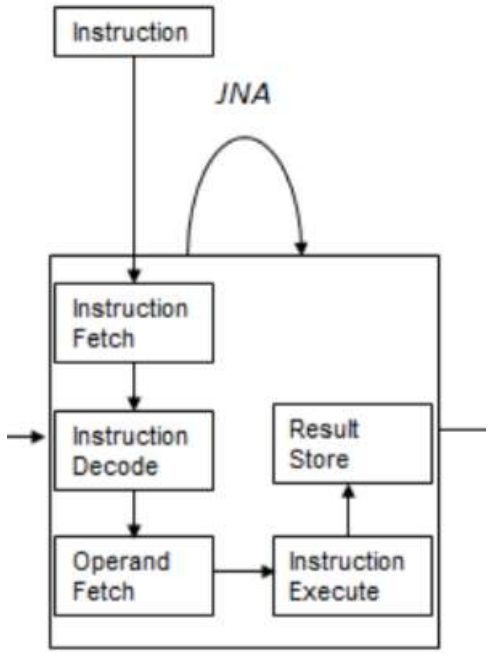


Hình 1. Kiến trúc của BE-PUM

Kiến trúc của hệ thống BE-PUM bao gồm 3 thành phần chính: symbolic execution, binary emulation và CFG storage. Hình 1 mô tả kiến trúc của hệ thống BE-PUM. Trong đó, vai trò của các thành phần này được mô tả như sau.

- Thành phần symbolic execution sử dụng Jackstab để dịch ngược và chuyển đổi thành mã nhị phân thành câu lệnh hợp ngữ tương ứng. Trong trường hợp câu lệnh hợp ngữ là câu lệnh chỉ tác động tới môi trường thực thi, cụ thể là các câu lệnh tính toán, câu lệnh ghi hoặc đọc trong bộ nhớ, bao gồm cả stack, môi trường thực thi được thay đổi tương ứng. Ngoài ra, vị trí của câu lệnh tiếp theo được xác định bằng vị trí của câu lệnh hiện tại cộng kích thước câu lệnh. Nếu câu lệnh tiếp theo là câu lệnh điều khiển, cụ thể là các câu lệnh gọi hàm, câu lệnh trả về từ hàm, câu lệnh nhảy và câu lệnh nhảy có điều kiện, thì câu lệnh tiếp theo sẽ được tính toán thông qua quá trình thực thi ký hiệu.
- Thành phần binary emulation là thành phần quan trọng trong tổng thể kiến trúc của BE-PUM. Để có thể xử lý một câu lệnh hợp ngữ, hệ thống BE-PUM sẽ giả lập các thành phần của hệ thống, cụ thể là mô hình bộ nhớ của BE-PUM. Mô hình này bao gồm tập 9 cờ được sử dụng trong hệ thống (AF, CF, DF, IF, OF, PF, SF, TF, và ZF), tập 20 thanh ghi (EAX, EBX, ECX, EDX, ESI, EDI, ESP, EBP, CS, DS, ES, FS, GS, SS, EIP, EFLAGS và 8 thanh ghi debug DRO, DR1, DR2, DR3, DR4, DR5, DR6, DR7, DR8), tập các giá trị bộ nhớ được lưu trữ trong đó một phần của bộ nhớ cho stack được giới hạn bởi thanh ghi esp và thanh ghi ebp. Bộ nhớ được giả lập trong BE-PUM được lưu trữ dưới dạng từng byte. Ngoài việc giả lập câu lệnh hợp ngữ, thì API cũng được giả lập thông qua sử dụng JNA[8] (Java Native Access) cho phép gọi trực tiếp từ các thư viện liên kết động, trong đó giá trị trả về của các API sẽ được lưu trữ trong các thanh ghi tương ứng. Đối với các API đặc biệt tác động trực tiếp đến môi trường thực, một sandbox sẽ được xây dựng cho quá trình xử lý các API này. Hình 2 mô tả các kiến trúc xử lý một câu lệnh thực thi trong hệ thống BE-PUM. Sau khi được nạp vào bộ nhớ, câu lệnh sẽ được giải mã, lấy các toán hạng và tiến hành mô phỏng quá trình thực thi. Kết quả của câu lệnh sẽ được ghi lại vào bộ nhớ.

[8]https://jna.java.net/



Hình 2. Kiến trúc xử lý API trong BE-PUM

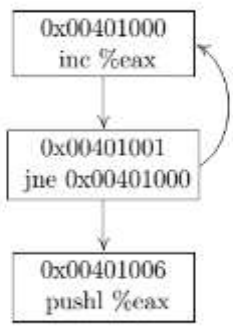
- Thành phần CFG storage được sử dụng để lưu trữ một CFG node và CFG edge sau khi được tính toán chính xác. CFG storage sẽ được sử dụng để xây dựng mô hình quá trình thực thi cuối cùng của tập tin được phân tích.

2. Ví dụ hoạt động của BE-PUM

Kết quả của quá trình phân tích một tập tin thực thi là một mô hình quá trình thực thi. Mô hình được sinh ra bởi hệ thống BE-PUM được biểu diễn dưới dạng Control Flow Graph hay CFG của chương trình. Một CFG là một tập của các đỉnh và cạnh. Trong đó một đỉnh của CFG bao gồm địa chỉ của câu lệnh và câu lệnh hợp ngữ tại địa chỉ đó, một cạnh của CFG là cạnh nối giữa 2 node của CFG. Xem xét đoạn code sau.

```
00401000  pushl 0x00401007
00401005  xor eax, eax
00401007  jne 0x00401005
0040100B  call ss:[esp]
```

Code 1: Code ví dụ của BE-PUM



Hình 3. CFG được xây dựng bởi BE-PUM

Kết quả quá trình phân tích trên hệ thống BE-PUM sẽ sinh ra được CFG của chương trình thực thi và được mô tả trong hình 3.

C. NuSMV

NuSMV [12] là một công cụ kiểm tra mô hình dạng ký hiệu mã nguồn mở. Đây là một dự án kết hợp giữa ITC-IRST, viện nghiên cứu ở phía Đông Trento, Ý và Đại học Carnegie Mellon. Trong bài báo này, chúng tôi đã thay đổi mã nguồn của NuSMV để cung cấp thêm nhiều tính năng và hỗ trợ tốt hơn, nâng cao hiệu suất kiểm tra mô hình trên hệ thống có số trạng thái hữu hạn. Đầu vào của NuSMV là một mô hình cần được kiểm tra được sinh ra bởi BE-PUM và một tính chất được mô tả dưới dạng công thức luận lý có yếu tố thời gian để thực hiện việc kiểm tra mô hình . Mô hình NuSMV được biểu diễn dưới dạng ngôn ngữ SMV. Ngoài ra, NuSMV hỗ trợ cả CTL model checking và LTL model checking.

III. PHƯƠNG PHÁP NHẬN DẠNG PACKER

A. Phát triển hệ thống BE-PUM để xử lý các packer

Để xử lý 14 kỹ thuật của packer [7,15], hệ thống BE-PUM được hiện thực và giả lập môi trường thực thi bao gồm bộ nhớ và các thanh ghi:

- PEB và TIB: Process Environment Block (PEB) dùng để lưu trữ các thông tin của Process và Thread Information Block (TIB) dùng để lưu trữ các thông tin của Thread hiện hành. Mỗi lần thực thi một câu lệnh tác động vào địa chỉ TIB của chương trình thì TIB và PEB sẽ được cập nhật tương ứng.

- LDR Data: Loader Data được giả lập để có thể lưu trữ các thông tin của các thư viện liên kết động được nạp vào trong quá trình thực thi để gọi các API. Song song với quá trình cập nhật TIB và PEB, với mỗi lần một thư viện liên kết động được nạp vào chương trình thông qua lệnh gọi API LoadLibrary, LDR Data cũng sẽ được cập nhật tương ứng.

- Trap Flag: để có thể xử lý kỹ thuật SEH sử dụng Trap Flag của Packer thì hệ thống BE-PUM được hỗ trợ một lớp để có thể lưu trữ những giá trị địa chỉ của hàm xử lý ngoại lệ, ngoài ra hệ thống BE-PUM còn được hỗ trợ thêm 8 thanh ghi gồm: DRO, DR1, DR2, DR3, DR4, DR5, DR6, DR7.

- Virtual Memory Block: để có thể xử lý kỹ thuật Stolen Bytes trong packer. Một lớp được hiện thực để quản lý nếu quá trình mở gói được thực hiện trên vùng nhớ ảo.

B. Kết hợp hệ thống BE-PUM và NuSMV

Để kết hợp hai công cụ này, chúng tôi đã tiến hành 2 bước, đầu tiên là quá trình xây dựng mô hình NuSMV và thứ hai là mô tả những kỹ thuật của packer thông qua biểu thức CTL.

1. Thiết kế và xây dựng NuSMV Model

Việc thiết kế và xây dựng một mô hình NuSMV để có thể kiểm tra với công cụ NuSMV bao gồm 2 giai đoạn chính, thứ nhất là chuyển đổi mô hình của BE-PUM thành dữ liệu JSON để có thể dễ lưu trữ và sử dụng, và thứ hai là quá trình chuyển đổi dữ liệu JSON thành mô hình NuSMV.

a) Chuyển đổi mô hình BE-PUM qua dữ liệu JSON

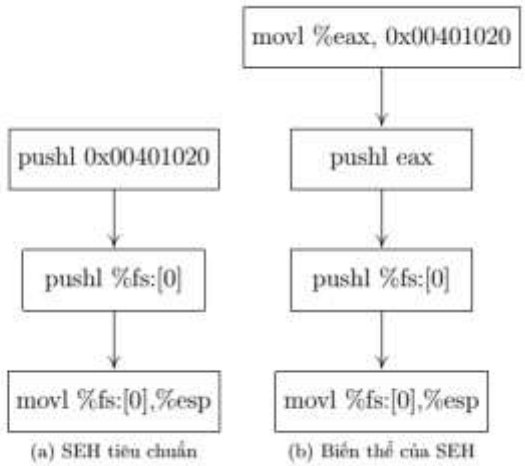
Mục tiêu của quá trình chuyển đổi mô hình của hệ thống BE-PUM sang dữ liệu JSON để đảm bảo không tốn quá nhiều thời gian để hiểu được cấu trúc của một CFG của BE-PUM. Ngoài ra, quá trình này được sử dụng như đầu vào cho bất kì một công cụ kiểm tra mô hình khác. Một dữ liệu JSON mô tả cho mô hình của BE-PUM sẽ bao gồm một tập các node và edge được biểu diễn như mảng JSON. Trong đó mỗi node và edge là những đối tượng JSON được định nghĩa là địa chỉ của câu lệnh tại node đó và nội dung của câu lệnh đó, mỗi edge được định nghĩa bởi source và destination, với source và destination là địa chỉ của các node. Hình 4 mô tả ví dụ một cấu trúc lưu trữ JSON trong BE-PUM

```
{ "model":{
"nodes":[
{"loc":"0x00401000", "inst" : "inc %eax"},
{ "loc" : "0x00401001", "inst" : "jne 0x00401000"},
{ "loc" : "0x00401006", "inst" : "pushl %eax"}
],
"edges":[
{ "src" : "0x00401000", "dest" : "0x00401001"},
{
"src" : "0x00401001",
"dest" : "0x00401006"
}],
```

Hình 4. Ví dụ cấu trúc lưu trữ chữ ký

b) Chuyển đổi dữ liệu JSON qua mô hình NuSMV

Trong thực tế, một packer có rất nhiều phiên bản và sử dụng nhiều biến thể của kĩ thuật rắc rối hóa. Hình 5 mô tả một ví dụ về sử dụng kỹ thuật SEH tiêu chuẩn trong một packer và biến thể của SEH, theo đó thay bằng việc PUSH một giá trị constant vào stack, thì packer sẽ thiết lập giá trị này cho thanh ghi EAX và push trực tiếp thanh ghi EAX vào stack.



Hình 5. Các biến thể của kỹ thuật SEH

Chính vì sự đa dạng đó, mà quá trình chuyển đổi dữ liệu JSON sang mô hình NuSMV đòi hỏi phải thực hiện quá trình phân loại các register, segment register, giá trị immediate và lệnh gọi API.

2. Mô tả các kỹ thuật packer thông qua biểu thức CTL

Những kỹ thuật rắc rối hóa đã được mô tả thông qua biểu thức CTL. Do đó đối với những kỹ thuật này, quá trình nhận dạng thông qua ngữ nghĩa sẽ được kết hợp với hệ thống BE-PUM để có thể tiến hành nhận dạng bằng giải thuật On-the-fly. Các kỹ thuật đã được mô tả qua biểu thức CTL bao gồm: Anti-Debugging, Indirect Jump, Obfuscated Constants, SEH, Stolen Bytes, Timming Check, Two APIs, Checksumming, Code Chunking, Overlapping Block, Overlapping Function, Overwritting, Packing/Unpacking và Hardware Breakpoint. Các kỹ thuật sẽ được nhận dạng qua phương pháp phân tích tuần tự trong quá trình xây dựng mô hình của công cụ BE-PUM. Các công thức CTL dưới đây mô tả một số kỹ thuật obfuscation techniques thông dụng trong packer.

- Công thức CTL mô tả kỹ thuật SEH.

EF((state(pushl,IMM,NULL,NULL)|state(pushl,REG,NULL,NULL)) &
EF(state(pushl,s_fs0,NULL,NULL) & EF(state(movl,r_esp,
s_fs0,NULL))))
EF((state(pushl,IMM,NULL,NULL)|state(pushl,REG,NULL,NULL)) &
EF(state(pushl,s_fsREG,NULL,NULL) & EF(state(movl,r_esp,
s_fsREG,NULL))))
EF((state(call,IMM,NULL,NULL)|state(call,REG,NULL,NULL)) & EF(
state(pushl,s_fs0,NULL,NULL) & EF(state(movl,r_esp,s_fs0
,NULL))))
EF((state(call,IMM,NULL,NULL)|state(call,REG,NULL,NULL)) & EF(
state(pushl,s_fsREG,NULL,NULL) & EF(state(movl,r_esp,
s_fsREG,NULL))))

Công thức CTL mô tả kỹ thuật Anti-Debugging.

EF(state(call,api_IsDebuggerPresent,NULL,NULL))

Công thức CTL mô tả kỹ thuật Obfuscated Constants.

EF(state(OPT,IMM,NULL,NULL) | state(OPT,NULL,IMM,NULL) | state(
OPT,NULL,NULL,IMM))

Công thức CTL mô tả kỹ thuật Stolen Byte.

EF(state(call,api_VirtualAlloc,NULL,NULL))

Công thức CTL mô tả kỹ thuật Timing Check.

EF(state(rdtsc,NULL,NULL,NULL) | state(call,api_GetTickCount,
NULL,NULL))

Công thức CTL mô tả kỹ thuật Two APIs

EF(state(call,api_LoadLibrary,NULL,NULL) & EF(state(call,
api_GetProcAddress,NULL,NULL)))

IV. THÍ NGHIỆM

A. Mô tả thí nghiệm

Chúng tôi đã tiến hành thí nghiệm nhận diện packer trên hệ thống WinXP SP3, Intel Core i5 - 2450M 2.5GHz, 2GB RAM, JDK 1.8. Các công cụ được sử dụng để thực hiện kiểm tra mô hình là NuSMV 2.6.0 và BE-PUM 2.0. Chúng tôi đã so sánh phương pháp của chúng tôi và phương pháp nhận diện packer truyền thống sử dụng công cụ CFF Explorer[9] trên 3250 mẫu malware khác nhau. Theo kết quả kiểm tra trên Virus Total Web service[10], trong số này 978 mẫu là downloaders, 2010 là worms, và số còn lại là trojans.

B. Kết quả thí nghiệm

BE-PUM đã nhận diện được 28 packer. Kết quả thí nghiệm phân tích packer trên hệ thống BE-PUM được thể hiện trong bảng 2. Trong bảng 2, cột Packer thể hiện tên những packer cách tiếp cận của chúng tôi đã nhận điện được, số câu lệnh và luồng điều khiển nối giữa các câu lệnh được mô tả trong cột Nodes và Edges tương ứng. Cột Time mô tả thời gian xử lý khi phân tích các packer này.

Bảng 2. Kết quả thí nghiệm

Packer	Nodes	Edges	Time
ASPACK	1047	1112	58969
BJFNT	2061	2126	2810453
EXEPACK	323	353	6719
EXESTEALTH	735	770	238438
EXPRESSOR	1172	1233	76797
FSG	244	268	4000
LAME	235	244	10062
MEW	110	131	3187
MORPHNAH	228	232	3593
MPRESS	459	489	101391
NOODLECRYPT	706	757	33922
NPACK	602	639	9609
PECOMPACT	1127	1178	30891
PEECRYPT	511	517	26750
PETITE	1568	1635	49593
RLPACK	467	501	265937
SCRAMBLEv1.0	266	292	3984
SCRAMBLEv1.2	244	268	3922
TELOCK	2105	2237	6276360
UPACK	443	490	19625
UPX	323	353	6875
WINUPACK	443	490	19734
WWPACK32	329	360	3219
XCOMP	650	724	224734
YODAv1.2	625	659	130500
YODAv1.3	924	960	120516
PELOCK	2032	2033	36903693
PESPIN	431	452	4860547

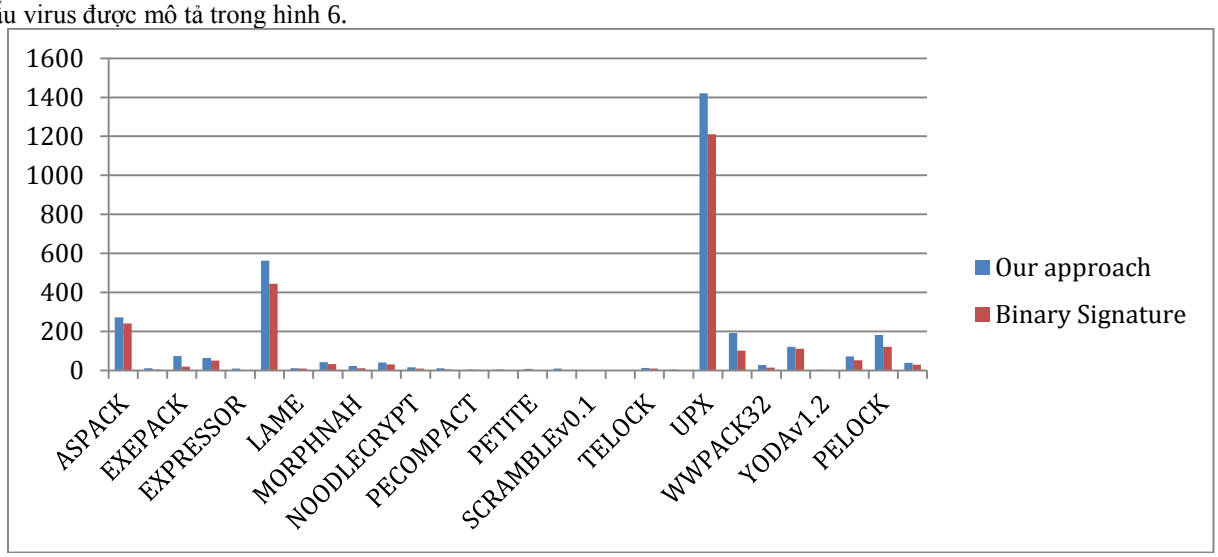
Kết quả thí nghiệm nhận dạng 14 kỹ thuật trên 28 packer qua việc kết hợp giữa hệ thống BE-PUM và công cụ NuSMV được thể hiện trong bảng 3. Những ô kỹ thuật nào được đánh dấu "x" tương ứng với packer có sử dụng kỹ thuật đó. Ví dụ, trong bảng 3, packer ASPACK sử dụng các kĩ thuật antidebugging, checksumming, codechunking, indirect jump, obfuscated constant, overlapping function, overwriting và packing/unpacking.

[9]http://www.ntcore.com/exsuite.php
[10]https://www.virustotal.com/en/

Bảng 3. Kết quả nhận dạng obfuscation technique trên 28 packer

Packer	AntiDebugging	Checksumming	CodeChunking	Indirect Jump	Obfuscated Constants	Overlapping Block	Overlapping Function	Overwritting	Packing/ Unpacking	SEH	Stolen Bytes	Timming Check	Two APIs	Hardware Breakpoints
ASPACK	x	x	x	x	x	x	x	x	x					
BJFNT	x	x	x	x	x	x								
EXEPACK	x	x	x	x	x	x	x							
EXESTEALTH	x	x	x	x	x	x	x	x	x	x	x			
EXPRESSOR	x	x	x	x	x	x	x	x					x	x
FSG	x	x	x	x	x	x	x	x						
LAME	x	x	x											
MEW	x	x	x	x	x	x	x	x						x
MORPHNAH	x	x	x								x			
MPRESS	x	x	x	x	x	x								
NOODLECRYPT	x	x	x	x	x	x	x							
NPACK	x	x	x	x	x	x	x	x	x					
PECOMPACT	x	x	x	x	x	x	x	x	x	x				
PEENCRYPT	x	x	x	x	x	x							x	
PETITE	x	x	x	x	x	x	x	x				x		
RLPACK	x	x	x	x	x	x	x	x	x					
SCRAMBLEv0.1	x	x	x	x	x	x	x	x						x
SCRAMBLEv0.2	x	x	x	x	x	x	x	x						
TELOCK	x	x	x	x	x	x	x	x	x	x	x			
UPACK	x	x	x	x	x	x	x	x						
UPX	x	x	x	x	x	x	x						x	
WINUPACK	x	x	x	x	x	x	x	x						
WWPACK32	x	x	x	x	x	x						x		
XCOMP	x	x	x	x	x	x	x	x	x					
YODAv1.2	x	x	x	x	x	x	x	x	x	x			x	x
YODAv1.3	x	x	x	x	x	x	x	x	x	x	x			
PELOCK	x	x	x	x	x							x		
PESPIN	X	x	x	x	x	x	x	x						x

Để kiểm tra tính chính xác, chúng tôi đã so sánh hướng tiếp cận sử dụng phương pháp hình thức và kỹ thuật nhận diện packer truyền thống sử dụng chữ ký nhị phân sử dụng công cụ CFF Explorer. Kết quả thí nghiệm trên 3250 mẫu virus được mô tả trong hình 6.



Hình 6. Thống kê quá trình nhận dạng packer trên tập 3250 malware

Trong hình 6, trục tung thể hiện số lượng mẫu packer nhận diện tương ứng với phương pháp của chúng tôi (cột màu xanh) và phương pháp truyền thống (cột màu đỏ). Trục hoành thể hiển các loại packer khác nhau. Có thể nhận thấy, cách tiếp cận của chúng tôi cho ra một kết quả chính xác hơn so với cách làm truyền thống. Trong một số trường hợp như packer UPX, sự khác biệt là khá lớn. Nguyên nhân là vì phương pháp tiến cận của chúng tôi không bị ảnh hưởng bởi các kĩ thuật rối rắm hóa làm thay đổi chữ kí của packer. Ví dụ, xem xét mẫu malware "034f9d2dc5627296141bb7d0a22032b1e8c7e47f266ada4a1da7f8dad05668b". Chữ ký của mẫu này là "*60 BE 00 E0 95 00 8D BE 00 30 AA FF C7*", nhưng chữ ký của UPX là "*60 BE ?? ?? ?? 00 8D BE ?? ?? ?? FF 57"* với ' ?' biểu thị một kí tự bất kì. Hai chữ kí này khác nhau ở kí tự cuối *"C7*" và *"57"*. Do đó, cách làm truyền thống sử dụng chữ ký không thể nhận diện được packer này, nhưng cách tiếp cận của chúng tôi dựa trên ngữ nghĩa nên có thể sinh ra kết quả chính xác.

V. KẾT LUẬN

Bài báo này đã đề xuất hướng tiếp cận mới để nhận diện packer bằng phương pháp hình thức. Phương pháp này kết hợp 2 công cụ BE-PUM để xây dựng mô hình và công cụ kiểm tra mô hình NuSMV. Kết quả thí nghiệm trên tập 3250 malware đã cho thấy hướng tiếp cận này có kết quả chính xác hơn so với hướng tiếp cận truyền thống. Tuy nhiên, mặt hạn chế của phương pháp này là thời gian xử lý khác lâu. Để giải quyết vấn đề này, chúng tôi sẽ áp dụng giải thuật song song hóa để giảm thời gian thực thi. Đây là một hướng nghiên cứu trong tương lai. Ngoài ra, một hướng đi khác là chúng ta sẽ tăng số lượng packer cần nhận dạng để sinh ra một kết quả chính xác hơn.

VI. LỜI CẢM ƠN

Nghiên cứu trong bài báo này được tài trợ bởi Vietnam National Foundation for Science and Technology Development (NAFOSTED), số 102.01 -2015.16.

TÀI LIỆU THAM KHẢO

1. P. Szor, The Art of Computer Virus Research and Defense. Addison-Wesley Professional, 2005.
2. E. Filiol, "Malware pattern scanning schemes secure against black-box analysis," Journal in Computer Virology, vol. 2, pp. 35–50, 2006.
3. A. Mori, T. Izumida, T. Sawada, and T. Inoue, "A tool for analyzing and detecting malicious mobile code," in ICSE 2006, pp. 831–834, 2006.
4. Anti-virus technology whitepaper. Technical report, BitDefender, 2007
5. R. Isawa, M. Kamizono, and D. Inoue. Generic Unpacking Method Based on Detecting Original Entry Point. In NIP, pp.593-600, 2013. LNCS 8226.
6. G. Jeong, E. Choo, J. Lee, M. Bat-Erdene, and H. Lee. Generic Unpacking using Entropy Analysis. In Malware, pp.114-121, 2010
7. Nguyen Minh Hai and Quan Thanh Tho. An Experimental Study on Identifying Obfuscation Techniques in Packer, 5th World Conference on Applied Sciences, Engineering & Technology, 02-04 June 2016, HCMUT, Vietnam, ISBN 978-81-930222-2-1
8. M. H. Nguyen, T. B. Nguyen, T. T. Quan, and M. Ogawa. A hybrid approach for control flow graph construction from binary code. In IEEE APSEC, pp.159-164, 2013
9. M. H. Nguyen, M. Ogawa, and T. T. Quan and. Obfuscation code localization based on CFG generation of malware. In FPS, pp.229-247, 2015. LNCS 9482
10. J. Kinder, F. Zuleger, and H. Veith. An abstract interpretation-based framework for control flow reconstruction from binaries. In VMCAI, pages 214–228, 2009. LNCS 5403.
11. J. Kinder and D. Kravchenko. Alternating control flow reconstruction. In VMCAI, pages 267–282, 2012. LNCS 7148.
12. A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani and A. Tacchella. . NuSMV 2: An OpenSource Tool for Symbolic Model Checking. In Proceeding of International Conference on Computer-Aided Verification (CAV 2002). Copenhagen, Denmark, July 27-31, 2002
13. K. A. Roundy and B. P. Miller. Binary-code obfuscations in prevalent packer tools. In ACM Comput. Surv, 46, pages 4:1–4:32, 2013
14. J. Kinder, F. Zuleger, and H. Veith, "An abstract interpretation-based framework for control flow reconstruction from binaries," in VMCAI 2009, pp. 214–228, 2009. 214–228.
15. T. Izumida, K. Futatsugi, and A. Mori. A generic binary analysis method for malware. In International Workshop on Security, pages 199–216, 2010. LNCS 6434.

APPLYING FORMAL METHOD FOR PACKER DETECTION

Nguyen Minh Hai, Do Duy Phong, Quan Thanh Tho

ABSTRACT— *Nowadays, malware has been becoming a real threat to each nation. In 2006, according to a report by Computer Economics 2007 Malware Report, more than 13.3 billion dollars has been spent on the war against malware and its destruction. To detect and analyze malware, most of industrial anti-virus software tends to focusing on identifying malware via signature based technique. However, since most of the modern popular malwares are either packed or obfuscated, malwares can easily defeat this weak technique. Packed malware also increases the difficulty of the reverse engineering technique which is inherently efficient and available for exploring the malware techniques. The reason is that it often takes a very long time for unpacking or detecting a packed file which causes much trouble for reverse engineering. For counter solution, most of anti-virus software choose the other way is to detect packer signature for verifying the packed malware. However, since hacker can easily modify header signature of packed file, this solution cannot determine precisely whether a malware is packed or not. This paper proposes a formal method approach for packer detection using a combination BE-PUM tool and symbolic model checker NuSMV. BE-PUM (Binary Emulator for PUshdown Model generation) is designed for generating a precise control flow graph (CFG), which handles many of typical obfuscation techniques of malware, e.g., indirect jump, self-modification code, and structured exception handler, which are adopted in most of modern packers. Currently, BE-PUM can cover the patterns for 14 techniques mainly used in supported 28 packers. Applying the CTL, LTL formula for that patterns as properties of proposed model checker tool NuSMV, it can detect completely all the malwares which are packed by these packers. We have performed the experiments on 3250 real-world malwares for checking the accuracy of our approach and the result is very promising.*

.