ĐẠI HỌC QUỐC GIA TP.HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC BÁCH KHOA

KHOA KHOA HỌC & KỸ THUẬT MÁY TÍNH

BÁO CÁO THỰC TẬP TỐT NGHIỆP

Phát triển BE-PUM như hệ thống Anti – Packer Sử dụng Model Checking phát hiện Malware

GVHD: **PGS.TS. QUẢN THÀNH THƠ**

NCS. NGUYỄN MINH HẢI

ThS. LÊ ĐÌNH THUẬN

GVPB: **TS. BÙI HOÀI THẮNG**

SVTH : **ĐỖ DUY PHONG 51102535**

TP. Hồ Chí Minh – Tháng 06/2015

LỜI CAM ĐOAN

Tôi xin cam đoan rằng mọi thông tin và kiến thức được trình bày trong bài báo cáo này ngoài việc tham khảo các nguồn tài liệu khác đã được ghi đầy đủ trong phần phụ lục các tài liệu tham khảo, thì mọi công việc đều do chính tôi thực hiện và bài báo cáo không chứa phần nội dung nào được sao chép từ các nguồn ngoài tham khảo hay các đề tài thực tập tốt nghiệp, luận văn tốt nghiệp của trường này và trường khác. Nếu có bất kì sai phạm hay gian lận nào, tôi xin chịu hoàn toàn trách nhiệm trước Ban Chủ Nhiệm Khoa và Ban Giám Hiệu Nhà Trường.

TP. Hồ Chí Minh, tháng 06 năm 2015

Sinh viên thực hiện

Đỗ Duy Phong

LỜI CẢM ƠN

Trong suốt quá trình hoàn thành đề tài thực tập, tôi xin gửi lời cảm ơn chân thành nhất đến với PGS.TS.Quản Thành Thơ, vì Thầy đã hướng dẫn tận tình, định hướng, giải đáp những thắc mắc về nội dung đề tài, đưa ra những lời góp ý, lời khuyên trong các buổi seminar của tôi để tôi có thể bổ sung thiếu sót, hoàn thành tốt nhất đề tài của mình. Bên cạnh sự giúp đỡ của Thầy, tôi cũng xin cảm ơn sâu sắc đến ThS. Nguyễn Minh Hải đã giúp đỡ tôi tìm hiểu từ những ngày đầu tham gia nhóm, theo sát tình hình nghiên cứu hằng ngày, lắng nghe và giải đáp thắc mắc, đưa ra lời khuyên chân thành, bổ ích để tôi có thêm sự tự tin hoàn thành tốt công việc công mình. Tôi cũng xin cảm ơn đến Ths. Lê Đình Thuận đã giúp đỡ, đưa ra góp ý và sửa chữa cho bài báo cáo của tôi. Hơn hết, tôi xin gửi lời cảm ơn đến PGS.TS.Quản Thành Thơ đã tạo điều kiện và cơ hội để tôi có thể được tham gia thực tập tại JAIST, Nhật Bản, qua chương trình thực tập tôi đã học thêm được rất nhiều kiến thức mới, bổ sung kiến thức thiếu sót và phát triển đề tài của mình.

Tôi xin chân thành biết ơn sự tận tình giảng dạy, giúp đỡ của các Thầy Cô trong khoa Khoa học và Kỹ thuật Máy tính đã truyền đạt những kinh nghiệm quý báu, kiến thức hữu ích thông qua các bài giảng trên lớp để tôi có thể hoàn thành đề tài này.

Cuối cùng, tôi xin chân thành cảm ơn đến gia đình, bạn bè, những người đã quan tâm, động viên, giúp đỡ tôi trong cuộc sống hằng ngày để tôi có thêm nghị lực, sức khỏe, tinh thần tốt hoàn thành đề tài thực tập tốt nghiệp này.

Với lòng biết ơn chân thành nhất, tôi xin gửi lời chúc sức khỏe, lời cảm ơn sâu sắc và lời chúc tốt đẹp nhất tới các Thầy Cô trong khoa Khoa học và Kỹ thuật Máy tính trường Đại Học Bách Khoa TP.HCM.

Trân trọng.

TP. Hồ Chí Minh, tháng 06 năm 2015

Sinh viên thực hiện

Đỗ Duy Phong

TÓM TẮT BÁO CÁO

Mục tiêu của đề tài được trình bày trong báo cáo này là bao gồm những nội dung tôi đã nghiên cứu bao gồm phần lý thuyết và hiện thực để hoàn thành giai đoạn thực tập tốt nghiệp với chủ đề: “Phát triển hệ thống BE – PUM như hệ thống anti – Packer và sử dụng Model Checking phát hiện Malware”. Trong phần một của bài báo cáo tôi sẽ tập trung giới thiệu sơ lược về đề tài, mục tiêu đã đạt được của đề tài. Phần hai của bài báo cáo tôi sẽ trình bày về các vấn đề đặt ra của đề tài và phân tích rõ các vấn đề, đưa ra hướng giải quyết chung nhất cho các vấn đề. Trong phần ba, tôi sẽ nêu rõ các kiến thức nền được áp dụng trong suốt quá trình nghiên cứu và hiện thực đề tài. Phần bốn của đề tài, tôi sẽ trình bày về kiến trúc thiết kế và hiện thực của hệ thống để giải quyết các vấn đề đã được đề ra. Trong phần cuối, tôi sẽ đưa ra các kết quả thí nghiệm dựa trên các hiện thực và cuối cùng sẽ là kế hoạch cụ thể cho giai đoạn luận văn tốt nghiệp sắp tới.

MỤC LỤC

Nội dung **Trang**

DANH MỤC HÌNH

DANH MỤC BẢNG

PHẦN I. GIỚI THIỆU

1. GIỚI THIỆU ĐỀ TÀI

1.1 Phân tích Packer

Packer là những công cụ dùng để đóng gói một chương trình, trong đó mục tiêu chính của một Packer là bảo vệ mã thực thi của chương trình khỏi việc dịch mã ngược, hay nói cách khác các Packer sẽ sử dụng những giải thuật đóng gói phức tạp hoặc những kỹ thuật obfuscation nhằm che dấu Original Entry Point (OEP) của một chương trình thực thi. Chính vì những đặc tính trên, một chương trình được đóng gói bởi Packer sẽ rất khó khăn để có thể tìm được OEP hay ta nói một cách khác là mở gói hoàn toàn.

Hệ thống BE – PUM đã được phát triển trước đó đã có thể thực hiện việc phân tích động mã nhị phân của một chương trình nói chung và Malware nói riêng, đồng thời BE – PUM đã hiện thực và xử lý được các kỹ thuật obfuscation được sử dụng trong Malware bao gồm: Entry Point Obscuring (EPO), Structured Exception Handling (SEH), Indirect Jump, Self – Modifying Code (SMC), Encryption/Decryption. Ngoài ra, khi tiến hành phân tích với các Packer trên công cụ dịch mã ngược OllyDBG, tôi nhận thấy điểm tương đồng giữa Packer và Malware, cụ thể là Packer cũng sử dụng các kỹ thuật mà Malware đã sử dụng. Sau đây, tôi sẽ minh họa một ví dụ cụ thể:

Net – Worm.Win32.Sasser.a sử dụng kỹ thuật Structured Exception Handling:

```
0040284D MOV EAX, 0040980F
00402852 PUSH EAX
00402853 PUSH DWORD PTR FS:[0]
0040285A MOV DWORD PTR FS:[0], ESP
00402861 XOR EAX, EAX
00402863 MOV DWORD PTR DS:[EAX], ECX
```

Packer PETite cũng sử dụng kỹ thuật này nhằm làm rối quá trình phân tích:

```
00404116 PUSH 004022E3
0040411B PUSH DWORD PTR FS:[0]
00404122 MOV DWORD PTR FS:[0], ESP
...
0040421E MOV BYTE PTR DS:[EDI], AL
```

Ngoài điểm tương đồng trên, Packer còn sử dụng các kỹ thuật anti – disassembly nhằm gây khó khăn hơn trong việc phân tích. Ví dụ trong Packer Fastpack, có sử dụng

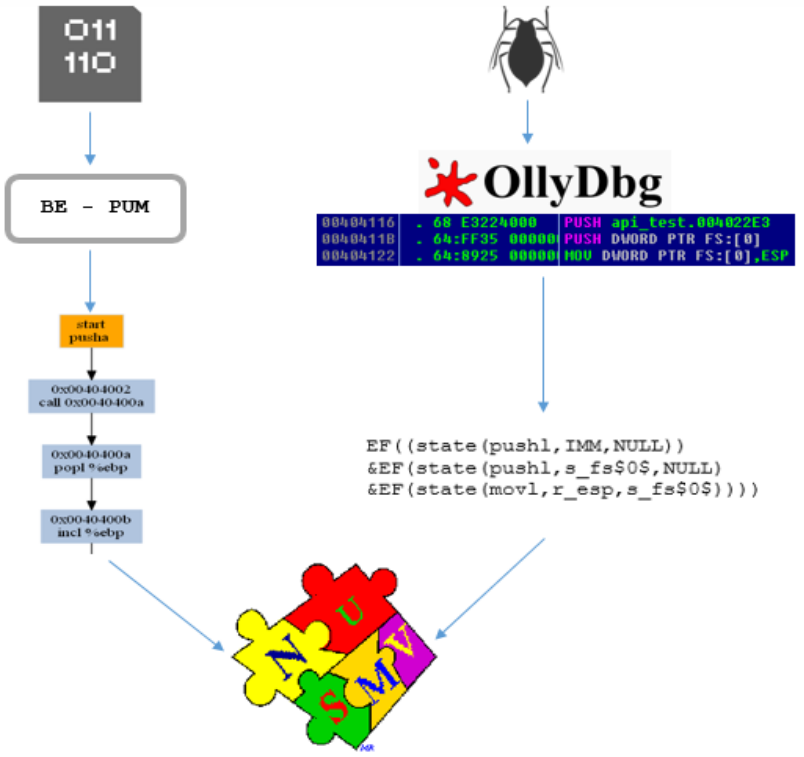
kỹ thuật gọi API để xác định nếu tiến trình được gọi đang chạy dưới ngữ cảnh của một debugger, nếu chương trình đang được phân tích bởi một debugger thì quá trình mở gói sẽ không được tiến hành.

```
004011E5 CALL kernel32.IsDebuggerPresent
```

Dựa trên những đặc điểm trên về Packer, trong đề tài thực tập tốt nghiệp của mình tôi sẽ tập trung vào việc phân tích Packer và các kỹ thuật của Packer một cách thủ công trên công cụ OllyDBG. Sau đó tôi sẽ phân tích các tập tin được đóng gói bởi Packer trong hệ thống BE-PUM, hiện thực các ý tưởng mới nhằm giải quyết những khó khăn và hạn chết của BE – PUM trong việc phân tích Packer, gỡ lỗi phát sinh do quá trình giả lập câu lệnh, môi trường nếu có để đảm bảo BE-PUM có thể mở gói hoàn toàn tập tin đó.

1.2 Sử dụng Model Checking xác định Malware

Mục tiêu của hệ thống BE – PUM là phân tích động và xây dựng Control Flow Graph (CFG) để mô hình hóa hoạt động của một chương trình, hay nói riêng là hoạt động của các Malware. Sau đó, bằng việc áp dụng Model Checking với đầu vào là mô hình của chương trình và những mẫu hành vi cơ bản của Malware được biểu diễn dưới dạng hình thức, ta có thể xác định mô hình của chương trình có thỏa mãn tính chất đó hay không, hay nói cách khác chương trình đó có phải là một Malware hay không.



Hình 1: Áp dụng Model Checking phát hiện Malware

Dựa trên kiến thức đã tìm hiểu được về Model Checking và Model Checker NuSMV, tôi sẽ hiện thực chương trình qua đó sẽ chuyển đổi CFG của chương trình dưới dạng SMV Model, cùng với biểu thức logic CTL thích hợp mô tả hành vi cơ bản của Malware để có thể xác định được một chương trình có phải là Malware hay không.

2. MỤC TIÊU ĐỀ TÀI

2.1 Mục tiêu giai đoạn thực tập tốt nghiệp:

Mục tiêu của giai đoạn thực tập tốt nghiệp tôi đã đạt được bao gồm:

- Tìm hiểu được cấu trúc của một PE File.
- Tìm hiểu hệ thống BE-PUM, tham gia hiện thực một số thành phần cơ bản trong BE-PUM gồm: hiện thực câu lệnh x86, API, các kỹ thuật obfuscation.
- Tìm hiểu công cụ debugger OllyDBG trong việc phân tích ngược mã nhị phân, đồng thời qua đó có thể phân tích các kỹ thuật sử dụng của Malware và Packer.
- Tìm hiểu các khái niệm căn bản về Packer, các kỹ thuật obfuscation và kỹ thuật anti – disassembly sử dụng trong Packer.
- Đưa ra hướng giải quyết, hiện thực và xử lý các kỹ thuật của Packer trong hệ thống BE-PUM. Công việc ban đầu đã có thể xử lý và phân tích, mở gói hoàn toàn các packer: ASPack, FSG, NPack, PECompact, PETite, UPX, Yoda.
- Tìm hiểu về Model Checking, hiện thực công cụ sử dụng CTL Model Checking xác định Malware thông qua các mẫu hành vi cơ bản, cụ thể là kỹ thuật Structured Exception Handling.

2.2 Mục tiêu của giai đoạn luận văn tốt nghiệp:

Mục tiêu của giai đoạn luận văn tốt nghiệp tôi đã lên kế hoạch tổng quan bao gồm:

- Tiếp tục tìm hiểu các kỹ thuật được sử dụng trong Packer với các Packer khác: Armadillo, ASProtect, FastPack, MEW, MPESS, PELock, PESpin, Themida, UPack, VMProtect.
- Giải quyết những hạn chế trong việc xử lý Packer của BE-PUM và hoàn tất các Packer trong danh sách các Packer được đề ra trước đó.

- Tiếp tục tìm hiểu về Model Checking và hiện thực công cụ sử dụng NuSMV xác định các hành vi cơ bản khác trong malware như: Indirect Jump, Self – Modifying Code, Encryption/Decryption.

3. CẤU TRÚC BÁO CÁO

Bài báo cáo tôi sẽ trình bày bao gồm các phần như sau:

- Phần I: Giới thiệu sơ lược về đề tài, về mục tiêu đề tài đã đạt được trong giai đoạn thực tập tốt nghiệp và kế hoạch tổng quan cho giai đoạn luận văn tốt nghiệp.

- Phần II: Tập trung giới thiệu các vấn đề liên quan đến Packer, các kỹ thuật obfuscation, kỹ thuật anti – disassembly được sử dụng trong Packer đi kèm với đó là các ví dụ cụ thể cho từng trường hợp. Ngoài ra, tôi cũng sẽ giới thiệu các vấn đề trong việc sử dụng Model Checking xác định Malware thông qua các mẫu hành vi cơ bản.

- Phần III: Tập trung nêu rõ các kiến thức nền được vận dụng khi giải quyết các vấn đề, cụ thể bao gồm các kiến thức về hệ thống BE – PUM, kiến thức về các kỹ thuật obfuscation, anti - disassembly. Và cuối cùng, tôi sẽ trình bày kiến thức tổng quan nhất về Model Checking và vận dụng NuSMV trong công cụ của tôi.

- Phần IV: Tập trung giới thiệu về phần hiện thực các ý tưởng để giải quyết những hạn chế của BE - PUM trong việc phân tích các Packer. Đồng thời tôi cũng sẽ giới thiệu về phần kiến trúc và thiết kế của chương trình sử dụng Model Checker NuSMV phát hiện Malware.

- Phần V: Trình bày các kết quả thí nghiệm đạt được khi phân tích Packer. Ngoài ra, tôi cũng sẽ trình bày kết quả thí nghiệm của công cụ trên tập Malware. Cuối cùng, tôi sẽ trình bày chi tiết các công việc hướng tới cho giai đoạn luận văn tốt nghiệp.

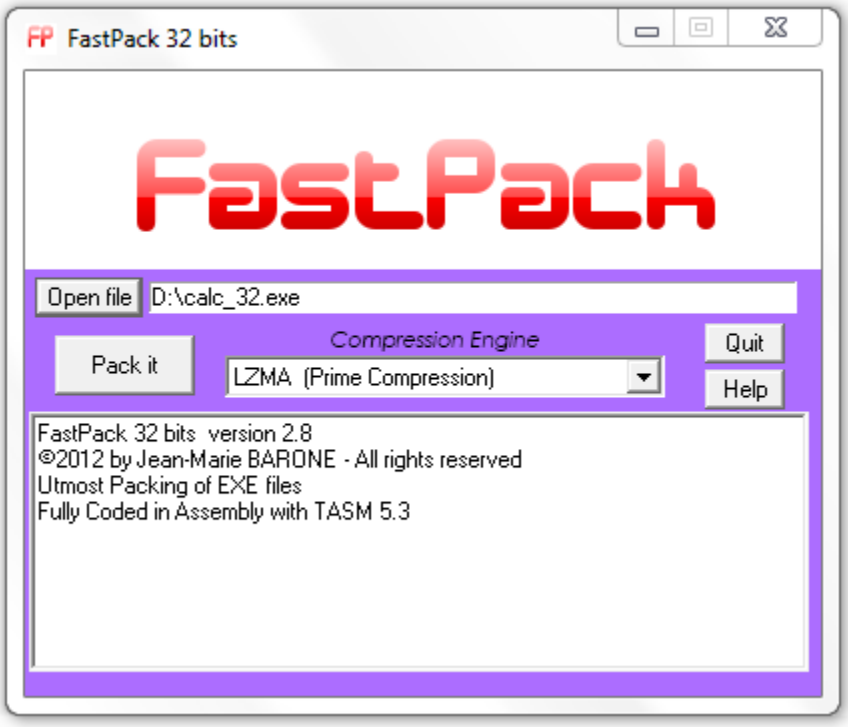
- Phần VI: Liệt kê các tài liệu, các nguồn tham khảo trong đề tài.

PHẦN II. PHÂN TÍCH VẤN ĐỀ

1. VẤN ĐỀ

1.1 Vấn đề phân tích Packer

Khi tiến hành phân tích một Packer cụ thể, mục tiêu cần đạt được cuối cùng là đảm bảo hệ thống BE – PUM có thể mở gói hoàn toàn một tập tin được đóng gói bởi Packer đó. Giả sử ban đầu ta chỉ có công cụ Packer và tập tin mẫu cần đóng gói.

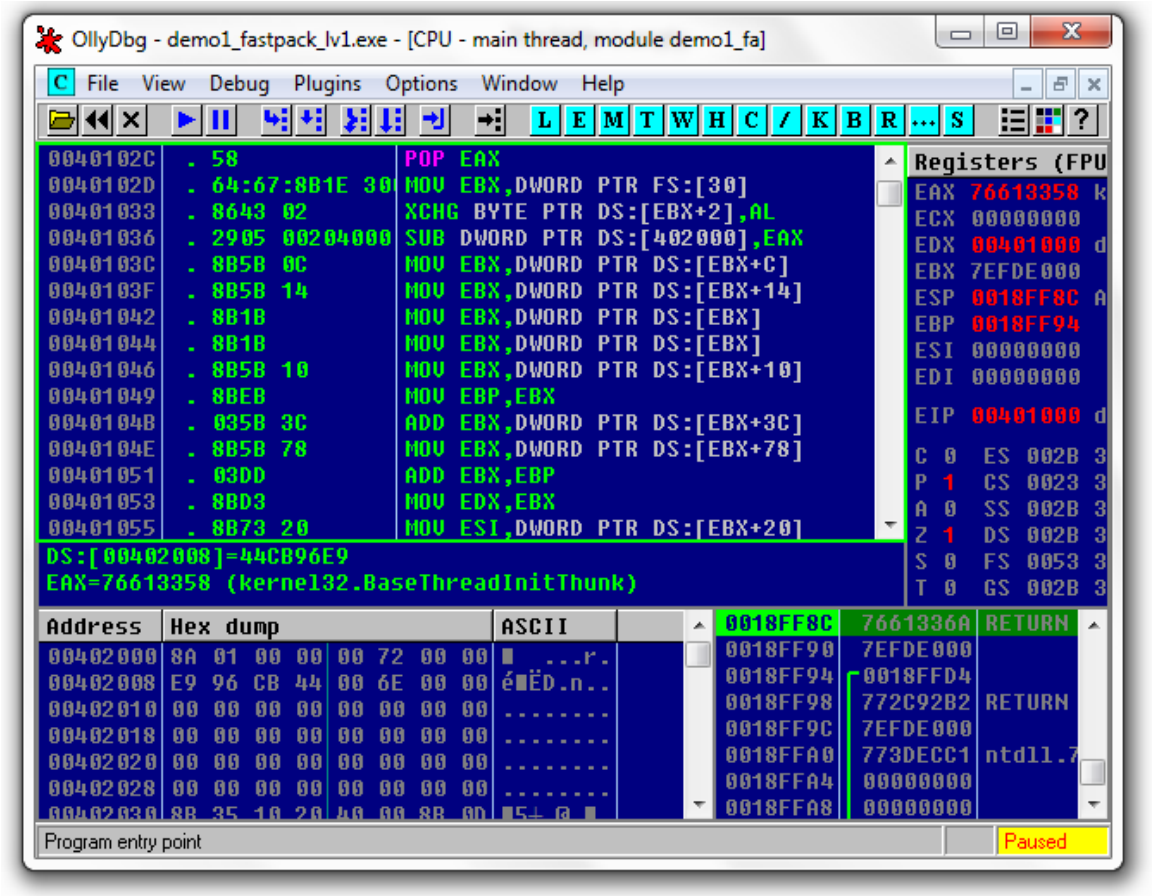


Hình 2: Packer FastPack và tập tin mẫu calc.exe cần đóng gói

Ý tưởng ban đầu là ta cần lựa chọn những giải thuật đóng gói cơ bản nhất thông qua các tùy chọn của chương trình, sau đó sẽ đến các tùy chọn cao hơn tương ứng với các giải thuật đóng gói phức tạp hơn.

Sau khi đã có tập tin mẫu được đóng gói bởi Packer, ta tiến hành chạy thử trong hệ thống BE - PUM, khả năng hệ thống BE – PUM vẫn chưa mở gói hoàn toàn cho tập tin là rất lớn bởi Packer có thể sử dụng rất nhiều kỹ thuật mà BE – PUM vẫn chưa hiện thực, xử lý được.

Do đó, vấn đề được đặt ra và cũng là vấn đề quan trọng cần làm trước khi cho mở gói tập tin trên hệ thống BE – PUM, ta sẽ cho tiến hành phân tích các bước thực thi của Packer trên công cụ OllyDBG, quan sát sự thay đổi của Stack, cờ, bộ nhớ qua từng câu lệnh, từ đó ta có thể dễ dàng hiểu rõ các kỹ thuật được sử dụng trong Packer và giải quyết những hạn chế đó trong hệ thống BE – PUM.



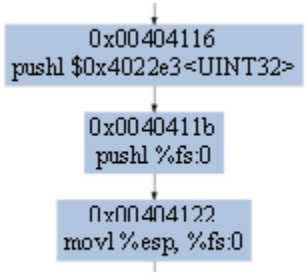
Hình 3: Tiến hành phân tích Packer FastPack trong OllyDBG

1.2 Vấn đề xác định Malware thông qua mẫu hành vi

Việc phân tích chương trình trong hệ thống BE – PUM, sau khi kết thúc quá trình phân tích, ta sẽ có được CFG của chương trình đó, qua đó sẽ mô tả chính xác quá trình thực thi từng câu lệnh tương ứng của chương trình.

Ngoài CFG, cần phải có các mẫu hành vi cơ bản của Malware. Các mẫu này có thể được xác định thông qua việc thực hiện các thí nghiệm trên tập của các Malware có hành vi cơ bản đó. Ví dụ khi ta cần kiểm tra kỹ thuật SEH có được sử dụng trong chương trình đã phân tích hay không, ta cần phải chạy một tập thí nghiệm của các Malware có sử dụng kỹ thuật này để tìm ra được những mẫu hành vi cơ bản.

Vấn đề được đặt ra với bài toán là làm cách nào để ta có thể kiểm tra được chương trình đó có phải thực sự là Malware hay không khi chúng ta chỉ có được mẫu các hành vi và CFG của chương trình đó.




```
push REG/IMM
push FS:[0]
mov FS:[0], ESP
```

Hình 4: CFG và một mẫu hành vi của SEH

2. PHÂN TÍCH

2.1 Phân tích Packer

Qua vấn đề được đặt ra như tôi đã nêu ở phần trên, hướng giải quyết tổng quát cho việc phân tích các Packer sẽ bao gồm hai bước chính:

- Bước một: Phân tích thủ công tập tin được xử lý bởi Packer trên công cụ dịch mã ngược OllyDBG, qua đó có thể nắm được các kỹ thuật cụ thể được sử dụng trong Packer đó. Từ đó ta có thể nắm bắt các kỹ thuật mới, hạn chế, khó khăn mà hệ thống BE – PUM vẫn chưa xử lý được. Bước một kết thúc khi ta tìm thấy được OEP của tập tin ban đầu trước khi chưa được xử lý bởi Packer.

- Bước hai: Phân tích tập tin được xử lý bởi Packer trong hệ thống BE – PUM. Đối với từng kỹ thuật cụ thể đã được thống kê trong bước một, ta sẽ kiểm tra xem tại địa chỉ mà kỹ thuật đó xảy ra, BE - PUM có thể xử lý được từng kỹ thuật đó chưa, nếu chưa ta sẽ tìm lỗi và các hạn chế để giải quyết vấn đề. Bước hai kết thúc khi ta phân tích được giai đoạn đóng gói che dấu OEP của Packer và giai đoạn thực thi của tập tin ban đầu trước khi chưa được xử lý bởi Packer.

Chính vì quá trình đóng gói phức tạp của các Packer, Packer sử dụng rất nhiều giải thuật để mã hóa, giải mã, các kỹ thuật anti – disassembly, do đó vấn đề lớn nhất được đặt ra trong quá trình phân tích các Packer là phân tích được Packer thủ công trong công cụ OllyDBG để có thể tìm được OEP. Hay nói cách khác, vấn đề cần được giải quyết là phát hiện các kỹ thuật obfuscation, anti – disassembly trong Packer và xử lý các vấn đề đó trong cả công cụ OllyDBG và hệ thống BE – PUM.

Các kỹ thuật chính của Packer có thể được chia làm hai nhóm kỹ thuật chính: các kỹ thuật obfuscation và các kỹ thuật anti – disassembly.

Các kỹ thuật obfuscation được Packer sử dụng cũng bao gồm các kỹ thuật đã được sử dụng trong Malware: Entry Point Obscuring, Indirect Jump, Structured Exception Handling, Self – Modifying Code, Encryption/Decryption.

Các kỹ thuật anti – disassembly được Packer sử dụng bao gồm các kỹ thuật như: Softice, RDTSC, PushRet, NTGlobal, Anti – disassembly Yason, các kỹ thuật call API,…

2.2 Phát hiện Malware

Qua vấn đề được đặt ra như tôi đã nêu ở phần trên, vấn đề phát hiện Malware có thể được thực hiện thông qua Model Checker cụ thể là NuSMV, vậy vấn đề phát hiện Malware được hiểu như làm cách nào để chuyển đổi CFG của một chương trình được phân tích dưới dạng mô hình SMV của NuSMV, và biểu diễn mẫu hành vi dưới dạng hình thức của biểu thức logic CTL hoặc LTL. Từ đó ta có thể sử dụng NuSMV để kiểm tra mô hình này của chương trình có thỏa mãn hành vi của một Malware hay không.

Việc phát hiện Malware thông qua các mẫu hành vi cơ bản gồm các bước:

- Bước một: Phân tích chương trình cần kiểm tra trên hệ thống BE – PUM, kết quả của hệ thống BE – PUM sẽ là CFG quá trình thực thi của chương trình đó và cũng là cơ sở đầu vào cho NuSMV.

- Bước hai: Khi cần kiểm tra hành vi nào của Malware, ta sẽ phân tích trên tập thí nghiệm gồm các Malware có hành vi đó. Qua quá trình phân tích thủ công, kết quả sẽ có được là một tập các mẫu hành vi.

- Bước ba: Xây dựng chương trình với đầu vào là CFG của Malware được sinh ra bởi hệ thống BE – PUM và kết quả đầu ra của chương trình là mô hình SMV được chấp nhận bởi NuSMV để kiểm tra hành vi.

- Bước bốn: Chuyển đổi mẫu hành vi dưới dạng hình thức của biểu thức CTL hoặc LTL.

- Bước năm: Kết hợp với NuSMV để kiểm tra mô hình SMV của chương trình có thỏa mãn hành vi Malware đó hay không.

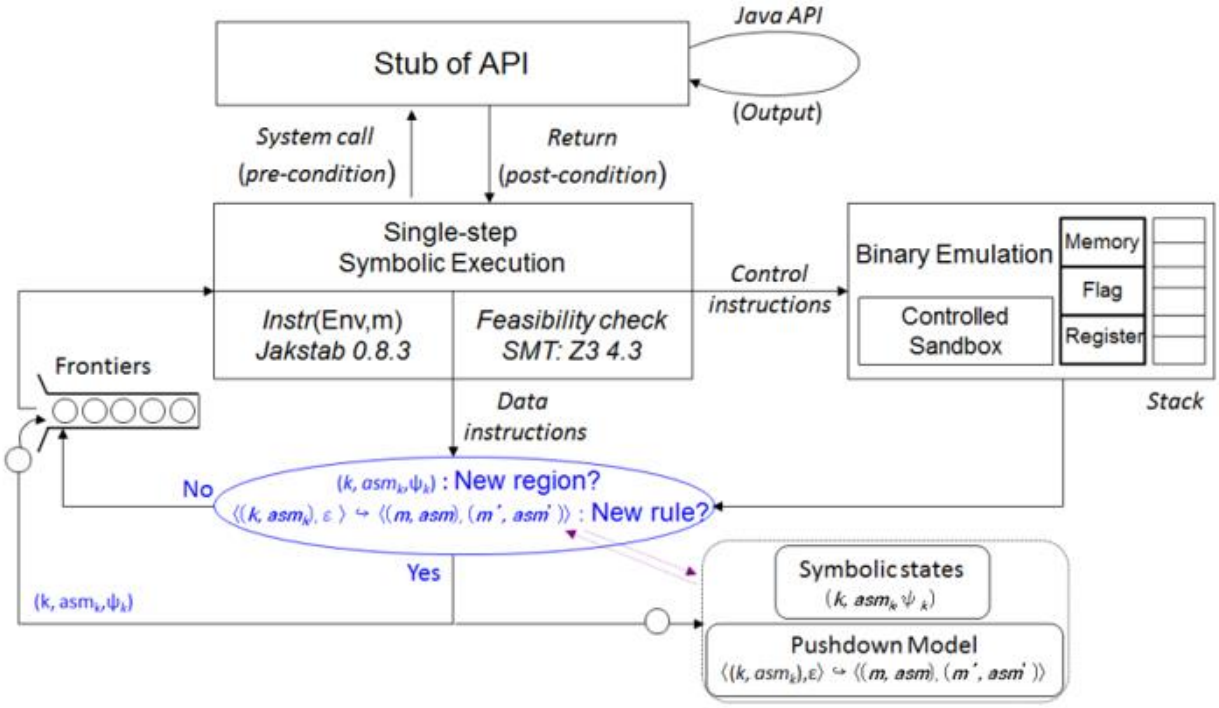
PHẦN III. KIẾN THỨC NỀN

1. HỆ THỐNG BE-PUM

1.1 Tổng quan hệ thống BE – PUM

Binary Emulation for Pushdown Model generation (BE - PUM) là công cụ được xây dựng dựa trên lõi là JakStab, sử dụng giải thuật On – the – fly để phân tích mã binary, từ đó có thể sinh ra CFG như luồng thực thi của tập tin, và CFG đó được xem như mô hình của chương trình, để từ đó ta có thể thực hiện kiểm tra tính chất của mô hình đó.

1.2 Các thành phần của hệ thống BE-PUM



Hình 5: Hệ thống BE – PUM

Trong phạm vi đề tài thực tập tốt nghiệp, tôi tập trung tìm hiểu các thành phần cơ bản trong hệ thống BE – PUM gồm:

- Path Condition Solving: áp dụng symbolic execution để tìm ra điều kiện đường đi cho quá trình thực thi.

- Binary Emulation: thành phần giả lập môi trường trong BE – PUM, các thành phần được giả lập trong BE-PUM là bộ nhớ, thanh ghi, các cờ và stack.

Trong quá trình thực hiện đề tài, tôi cũng tập trung hiện thực và giải quyết các vấn đề của hệ thống BE - PUM trên các câu lệnh x86 và Win32 API được giả lập trong BE – PUM, đồng thời bổ sung cho quá trình xử lý các kỹ thuật obfuscation, đặc biệt là kỹ thuật Structured Exception Handling.

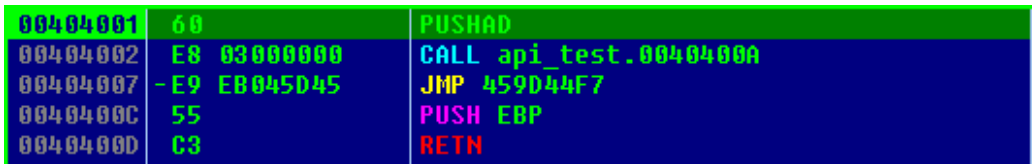
2. CÁC KỸ THUẬT CỦA PACKER

Để có thể phân tích được các Packer, cần nắm rõ các kỹ thuật được sử dụng trong Packer. Các kỹ thuật được sử dụng trong Packer có thể được chia làm hai nhóm kỹ thuật chính: các kỹ thuật obfuscation và các kỹ thuật anti – disassembly. Sau đây tôi sẽ trình bày cụ thể từng nhóm kỹ thuật tương ứng với từng kỹ thuật cụ thể mà tôi đã phân tích được trong các Packer.

2.1 Nhóm các kỹ thuật obfuscation

2.1.1 Kỹ thuật Entry Point Obscuring

Kỹ thuật Entry Point Obscuring (EPO) sẽ làm rối điểm nhập của chương trình. Thông thường, câu lệnh thực thi chính sẽ luôn nằm tại vị trí Entry Point, nhưng trong kỹ thuật EPO, Packer sẽ chèn thêm những câu lệnh giả để có thể gọi đến những câu lệnh thực thi chính này một cách gián tiếp.



Hình 6: Kỹ thuật EPO được sử dụng trong Packer ASPack

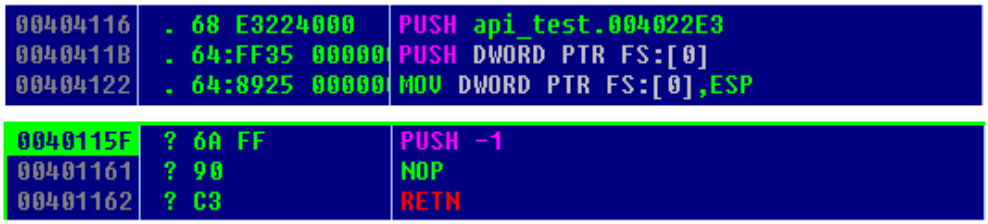
Trong ví dụ trên, ta thấy ASPack sẽ thực thi câu lệnh `CALL 0040400A`, đây là một câu lệnh giả để ASPack có thể nhảy tới điểm nhập thực của chương trình nằm tại vị trí `0040400A`.

2.1.2 Kỹ thuật Structured Exception Handling

Kỹ thuật Structured Exception Handling (SEH) là một kỹ thuật thông dụng được dùng trong cả Malware và các Packer để làm rối quá trình phân tích. Để sử dụng kỹ thuật này, Packer sẽ thực hiện hai bước:

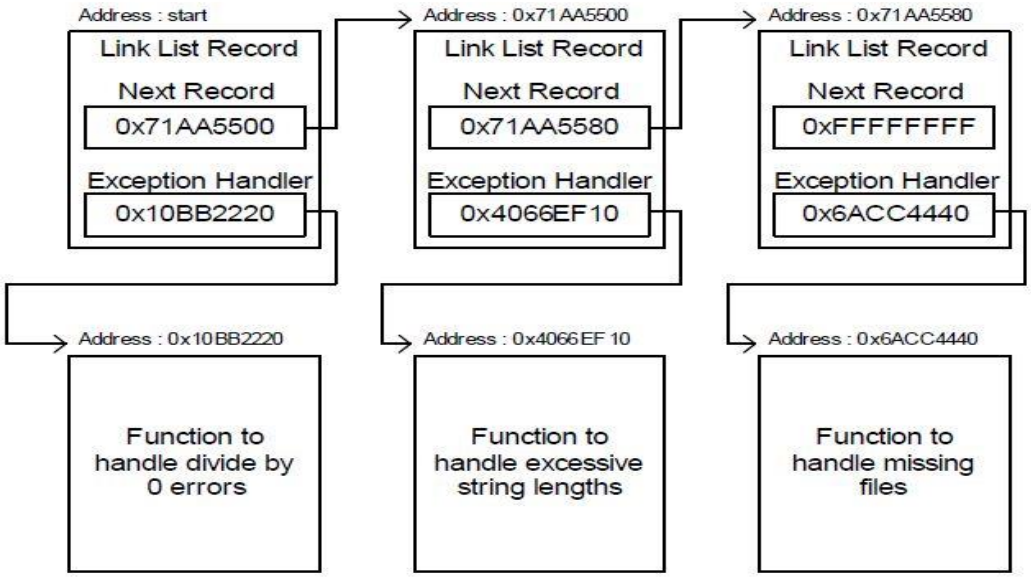
- Ở bước đầu tiên, Packer sẽ thiết lập cài đặt ngoại lệ cho chương trình, để đảm bảo rằng khi một ngoại lệ xảy ra, chương trình sẽ xử lý ngoại lệ tại hàm xử lý ngoại lệ đã được cài đặt trước đó của Packer.

- Với bước thứ hai, Packer sẽ gây ra ngoại lệ cho chương trình, ngoại lệ đó có thể xảy ra do các phép toán chia cho 0, ghi hoặc đọc vào các vùng nhớ không hợp lệ, các câu lệnh ngắt `int`,...



Hình 7: Quá trình thiết lập và xử lý ngoại lệ trong Packer PETite

Trong ví dụ trên, ta thấy quá trình thiết lập cài đặt ngoại lệ của PETite sẽ ghi giá trị địa chỉ của hàm xử lý ngoại lệ vào bộ nhớ tại vị trí `FS:[0]`, vì khi có ngoại lệ xảy ra, chương trình sẽ kiểm tra giá trị địa chỉ tại `FS:[0]` và mặc định là giá trị của hàm xử lý ngoại lệ. Ngoài ngoại lệ được thiết lập bởi Packer, còn có những ngoại lệ được thiết lập bởi hệ thống, những ngoại lệ này được thể hiện dưới dạng danh sách liên kết.



Hình 8: Ví dụ về cấu trúc danh sách liên kết của các ngoại lệ

Do đó để đảm bảo chương trình có thể thiết lập ngoại lệ hoàn toàn, Packer cần gán giá trị địa chỉ của ngoại lệ tiếp theo vào đỉnh của stack, giá trị này được lưu trữ trong thanh ghi `ESP`.

Ngoài ra, trong ví dụ trên, ta có thể thấy Packer gây ra ngoại lệ bằng việc truy cập vào địa chỉ không hợp lệ `0xFFFFFFFF` qua câu lệnh `RETN`. Khi đó, khi ngoại lệ này xảy ra, chương trình sẽ thực thi hàm xử lý ngoại lệ tại vị trí `0x004022E3`.

2.1.3 Kỹ thuật Indirect Jump

Kỹ thuật Indirect Jump cũng là kỹ thuật thường được sử dụng trong Packer và các Malware, kỹ thuật này cũng nhằm làm rối quá trình phân tích,và đặc biệt gây khó khăn cho một quá trình phân tích tĩnh. Thông thường, chương trình sau khi thực thi một câu lệnh sẽ thực thi ngay câu lệnh tiếp theo, nhưng các Packer sẽ thêm các đoạn mã giả để gọi một cách gián tiếp đến câu lệnh đó.

```
004EF03A   83C0 14          ADD EAX,14
004EF03D   894424 08        MOV DWORD PTR SS:[ESP+8],EAX
004EF041   5B               POP EBX
004EF042   58               POP EAX
004EF043   C3               RETN
```

Hình 9: Kỹ thuật Indirect Jump trong Packer Themida

Trong ví dụ trên ta có thể thấy, Themida có thể nhảy trực tiếp đến vị trí địa chỉ được lưu trong thanh ghi `EAX`, tuy nhiên Themida sẽ thay đổi giá trị trong stack tại vị trí `ESP+8`, thực hiện pop giá trị ra khỏi stack 2 lần, sau đó sẽ thực thi câu lệnh `RETN` để nhảy gián tiếp đến vị trí địa chỉ trong thanh ghi `EAX` trước đó.

2.1.4 Kỹ thuật Self – Modifying Code

Kỹ thuật Self – Modifying Code (SMC) là kỹ thuật tự thay đổi mã rất thường được sử dụng trong Malware. Theo đó, các đoạn mã thực thi ban đầu sẽ được thay đổi thành đoạn mã thực thi thực sự và các Packer sẽ thực thi trên đoạn mã thực sự này.

```
00407934   FE0F            DEC BYTE PTR DS:[EDI]
00407936   47               INC EDI
00407937   49               DEC ECX
00407938  ^75 FA            JNZ SHORT api_test.00407934
0040793A   B9 22226A49      MOV ECX,496A2222
```

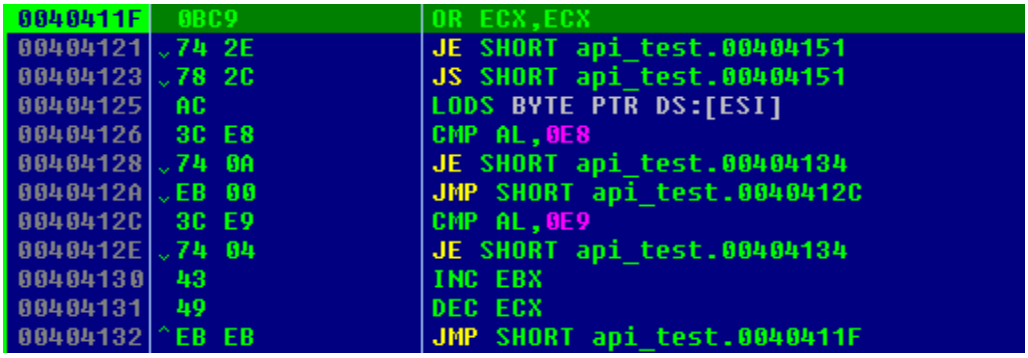
Hình 10: Đoạn mã thực thi ban đầu tại vị trí `0x0040793A`

Hình 11: Đoạn mã thực thi sau khi sử dụng kỹ thuật SMC

Trong ví dụ trên kỹ thuật SMC được sử dụng trong Packer Themida, ta có thể thấy đoạn mã ban đầu tại vị trí `0x0040793A` đã được thay đổi sau khi thực hiện câu lệnh `DEC BYTE PTR DS:[EDI]` với giá trị `EDI = 0x0040793A`, cụ thể giá trị opcode ban đầu tại vị trí này là `B8` tương ứng với đoạn mã `MOV ECX,496A2222` sẽ được tăng thêm 1 byte trở thành `B9` tương ứng với đoạn mã `MOV EAX,496A2222.`

2.1.5 Kỹ thuật Encryption / Decryption

Kỹ thuật Encryption / Decryption được xem như là một kỹ thuật mà trong đó sẽ sử dụng một chuỗi các kỹ thuật SMC một cách liên tiếp.



Hình 12: Giải thuật mã hóa và giải mã trong Packer ASPack

Trong ví dụ trên, ta có thể thấy giá trị địa chỉ tại vị trí `ESI` được thay đổi tương ứng với giải thuật mã hóa.

2.2 Nhóm các kỹ thuật anti – disassembly

Hiện tại trong giới hạn đề tài của mình, cụ thể với các Packer tôi đã phân tích, tôi đã tìm hiểu và xác định được một số kỹ thuật anti – disassembly như: kỹ thuật Softice, kỹ thuật gọi API, kỹ thuật NtGlobal, kỹ thuật Anti – Disassembly Yason, …

2.2.1 Kỹ thuật Softice

Trong kỹ thuật này, Packer sẽ thực thi câu lệnh để lấy giá trị bộ nhớ tại vị trí `FS:[4]` và kiểm tra xem giá trị tại địa chỉ này có bằng với `0x80000004`, nếu bằng với giá trị này tức là tiến trình đang trong ngữ cảnh của một Debugger, khi đó tiến trình sẽ kết thúc và quá trình mở gói sẽ không được tiếp tục.

Hình 13: Kỹ thuật Softice

2.2.2 Kỹ thuật gọi API

Trong kỹ thuật này, Packer sẽ gọi đến các API để xác định tiến trình đang chạy có đang trong ngữ cảnh Debugger, nếu có chương trình sẽ kế thúc và tiến trình mở gói sẽ không được tiến hành. Các API có thể được Packer gọi là:

- kernel32.IsDebuggerPresent: kiểm tra tiến trình được chạy có đang trong ngữ cảnh Debugger hay không.

- kernel32.SetUnhandledExceptionFilter: được sử dụng để thiết lập một hàm xử lý ngoại lệ cho tiến trình, khi có ngoại lệ xảy ra và tiến trình không đang trong ngữ cảnh Debugger. Bên cạnh đó, Packer cũng sẽ gọi API ntdll.ZwQueryInformationThread để truy xuất đến giá trị DebuggerPort xác định tiến trình có đang được phân tích hay không.

Hình 14: Kỹ thuật gọi API được sử dụng trong Packer PETite

2.2.3 Kỹ thuật NTGlobal

Trong kỹ thuật này Packer sẽ truy cập vào bộ nhớ tại Thread Information Block tại địa chỉ `FS:[30]`, cũng chính là ví trị của Process Environment Block (PEB) trong bộ nhớ. Nếu tiến trình đang không chạy dưới ngữ cảnh debugger thì giá trí mặc định tại vị trí offset `0x68` của PEB tức là giá trị `NtGlobalFlag` luôn bằng 0, nhưng nếu tiến trình đang chạy trong ngữ cảnh của debugger, giá trị này sẽ là tổng hợp của 3 cờ:

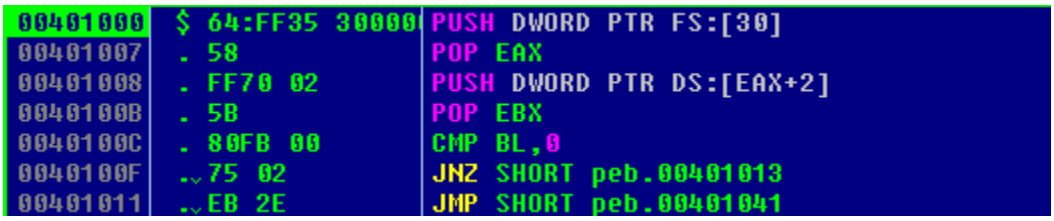
- ✓ FLG_HEAP_ENABLE_TAIL_CHECK = 0x10
- ✓ FLG_HEAP_ENABLE_FREE_CHECK = 0x20
- ✓ FLG_HEAP_VALIDATE_PARAMETERS = 0x40

Vậy ý tưởng chính của kỹ thuật này là sẽ tiến hành kiểm tra giá trị NtGlobalFlag nếu giá trị này bằng 0x70 thì chương trình sẽ kết thúc và quá trình mở gói sẽ không được tiến hành.

Hình 15: Kỹ thuật NtGlobal

2.2.4 Kỹ thuật Anti – disassembly Yason

Kỹ thuật này có ý tưởng chính tương đồng với kỹ thuật gọi API IsDebuggerPresent nhưng Packer sẽ tiến hành lấy giá trị địa chỉ tại FS:[30] sau đó Packer sẽ kiểm tra byte tại vị trí offset 0x2, giá trị này cũng chính bằng giá trị trả về của API IsDebuggerPresent, do đó cũng tương tự nếu tiến trình đang trong ngữ cảnh debugger thì quá trình mở gói sẽ không được tiến hành.



Hình 16: Kỹ thuật Anti – Disassembly Yason

3. MODEL CHECKING

3.1 Tổng quan Model Checking

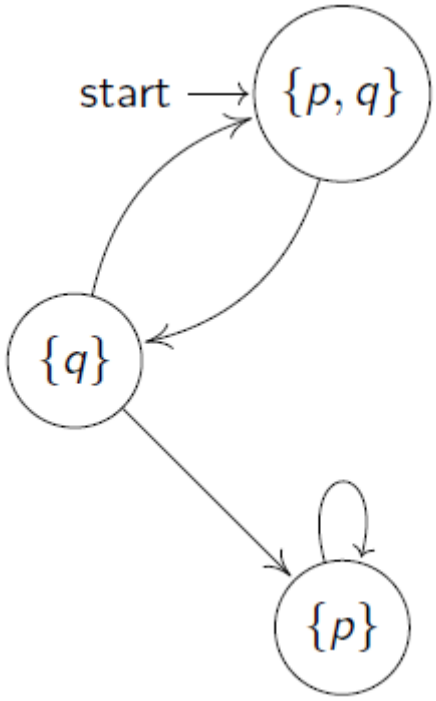
Model Checking là một kỹ thuật tự động theo đó với một mô hình hữu hạn trạng thái của hệ thống $\mathcal{M}$ và một tính chất ϕ được mô tả hình thức, sẽ kiểm tra mô hình $\mathcal{M}$ có thỏa mãn tính chất ϕ hay không.

$$\mathcal{M} \vDash \phi$$

3.2 Cấu trúc Kripke

Vấn đề quan trọng trong Model Checking là xây dựng một mô hình hình thức cho hệ thống. Cấu trúc Kripke là một dạng của mô hình chuyển trạng thái biểu diễn hành vi của hệ thống trong suốt quá trình thời gian. Với AP là một tập của các mệnh đề nguyên tử, một cấu trúc Kripke trên tập AP, $\mathcal{M} = (S, S_0, R, L)$, được định nghĩa như sau:

- S: tập hữu hạn các trạng thái
- $S_0 \subseteq S$: tập của các trạng thái bắt đầu
- $R \subseteq S \times S$: mối quan hệ chuyển trạng thái, theo đó với mọi trạng thái $s \in S$ luôn có trạng thái $s' \in S$ sao cho $R(s, s')$
- $L: S \to 2^{AP}$ là hàm xác định nhãn cho mỗi trạng thái với tập các mệnh đề nguyên tử sao cho các mệnh đề đó là đúng trong trạng thái đó.



Hình 17: Ví dụ về cấu trúc Kripke

Trong ví dụ trên với tập $AP = \{p, q\}$. Cấu trúc Kripke $\mathcal{M} = (S, S_0, R, L)$ được xác định với: $S = \{s_1, s_2, s_3\}$, $S_0 = \{s_1\}$, $R = \{(s_1, s_2),\ (s_2, s_1),\ (s_2, s_3),\ (s_3, s_3)\}$ và $L = \{(s_1, \{p, q\}),\ (s_2, \{q\}),\ (s_3, \{p\})\}$.

3.3 Biểu thức logic CTL

3.3.1 Path quantifiers và Temporal Operators

Path Quantifers trong một biểu thức logic CTL được sử dụng để mô tả cấu trúc nhánh trong một computation tree. Ta có hai path quantifers được sử dụng biểu thức CTL là A, mô tả cho mọi đường tính toán và E, mô tả cho một vài đường tính toán.

Temporal Operators trong một biểu thức logic CTL được sử dụng để mô tả tính chất của đường trong một computation tree. Ta có 4 temporal operators được sử dụng trong biểu thức CTL là X, F, G, U.

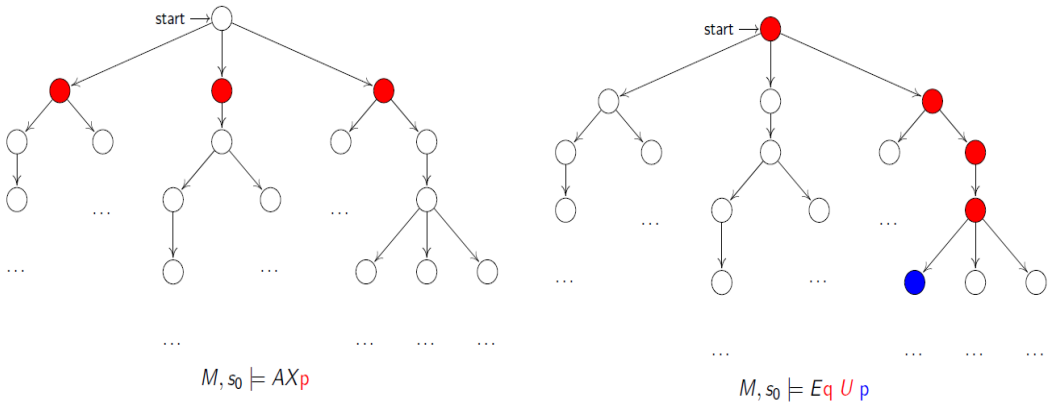
3.3.2 Syntax

Syntax của một biểu thức logic CTL có thể được biểu diễn như sau:

$$\Phi := true \mid p \mid (\neg\Phi) \mid (\Phi_1 \wedge \Phi_2) \mid (A\phi) \mid (E\phi)$$
$$\phi = \mathrm{X}\Phi \mid (\Phi_1\ U\ \Phi_2)$$

3.3.3 Semantics

Một số ví dụ về ngữ nghĩa của một biểu thức logic CTL:



Hình 18: Ví dụ về ngữ nghĩa của biểu thức CTL

4. NUSMV & SMV MODEL

4.1 Model Checker – NuSMV

NuSMV là một chương trình mã nguồn mở và là mở rộng của SMV symbolic model checker, là công cụ model checking đầu tiên dựa trên Binary Decision Diagrams (BDDs).

NuSMV được phát triển như một dự án kết hợp giữa ITC – IRST, Đại học Carnegie Mellon, Đại học Genoa và Đại học Trento.

4.2 SMV Model

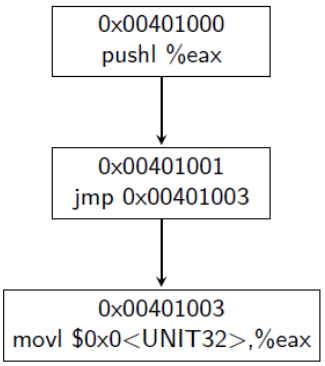
Mô hình SMV cũng bao gồm các trạng thái mà trong đó một trạng s của mô hình SMV là cấu trúc:

- $addr$: địa chỉ của câu lệnh
- $mnem$: mnemonic của câu lệnh, phụ thuộc vào giá trị của địa chỉ. Cụ thể, với một trạng thái trên Control Flow Graph được sinh ra bởi BE – PUM sẽ có địa chỉ và nội dung câu lệnh tại địa chỉ đó, do đó tương ứng với một địa chỉ câu lệnh ta có một giá trị mnemonic tương ứng với câu lệnh đó.
- op: bao gồm $name$ và $type$ trong đó $name$ sẽ phụ thuộc vào địa chỉ và $type$ sẽ phụ thuộc vào $name$.

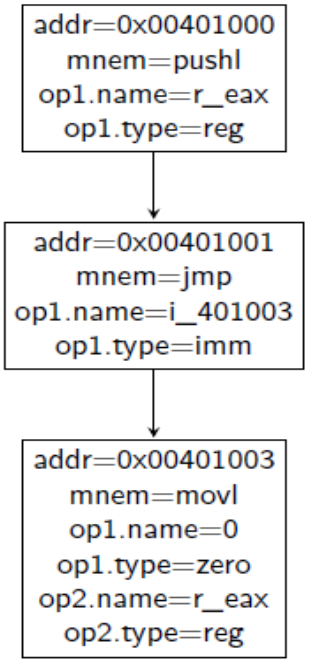
Một ví dụ về mô hình SMV, giả sử ta có đoạn mã thực thi như sau:

```
00401000 push EAX
00401001 jmp 00401003
00401003 mov EAX, 0
```

Khi đó sau khi phân tích trong hệ thống BE – PUM ta sẽ có CFG được sinh ra từ BE – PUM như sau:



Lúc này ta sẽ có mô hình SMV được xây dựng lại như sau:

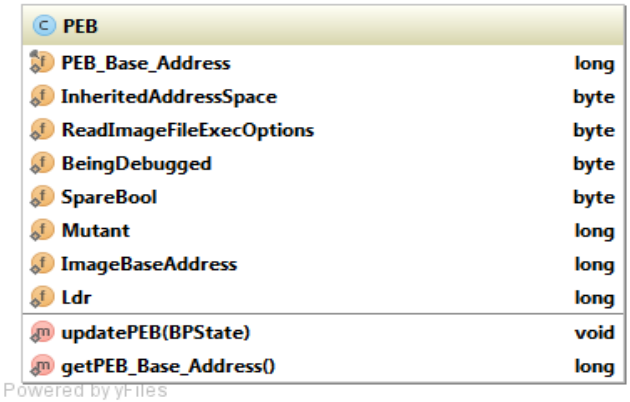


PHẦN IV. KIẾN TRÚC VÀ THIẾT KẾ

1. PHÁT TRIỂN BE – PUM HỖ TRỢ KỸ THUẬT ANTI - DISASSEMBLY

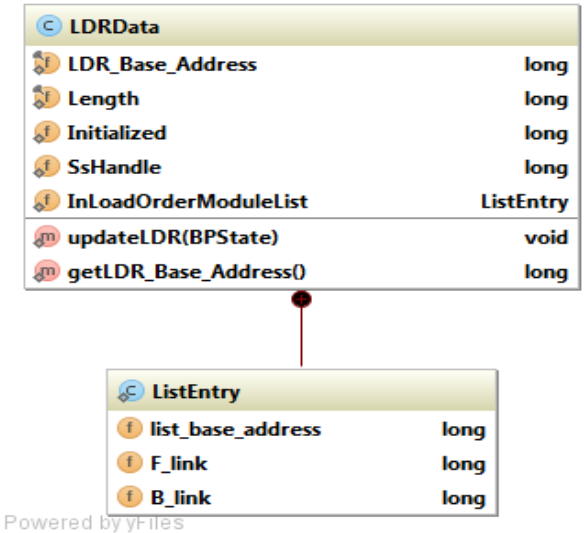
Để có thể giải quyết vấn đề về các kỹ thuật anti – disassembly được sử dụng bởi các Packer, do trước đó BE – PUM vẫn chưa hiện thực Thread Information Block (TIB) và Process Information Block (PEB) nên đối với các Packer sử dụng những kỹ thuật truy xuất vào bộ nhớ tại vị trí của TIB và PEB, giá trị trả về trong BE – PUM sẽ luôn bằng 0. Do đó để giải quyết vấn đề này tôi bổ sung thêm cho BE - PUM bằng cách giả lập TIB và PEB.

1.1 Hiện thực PEB



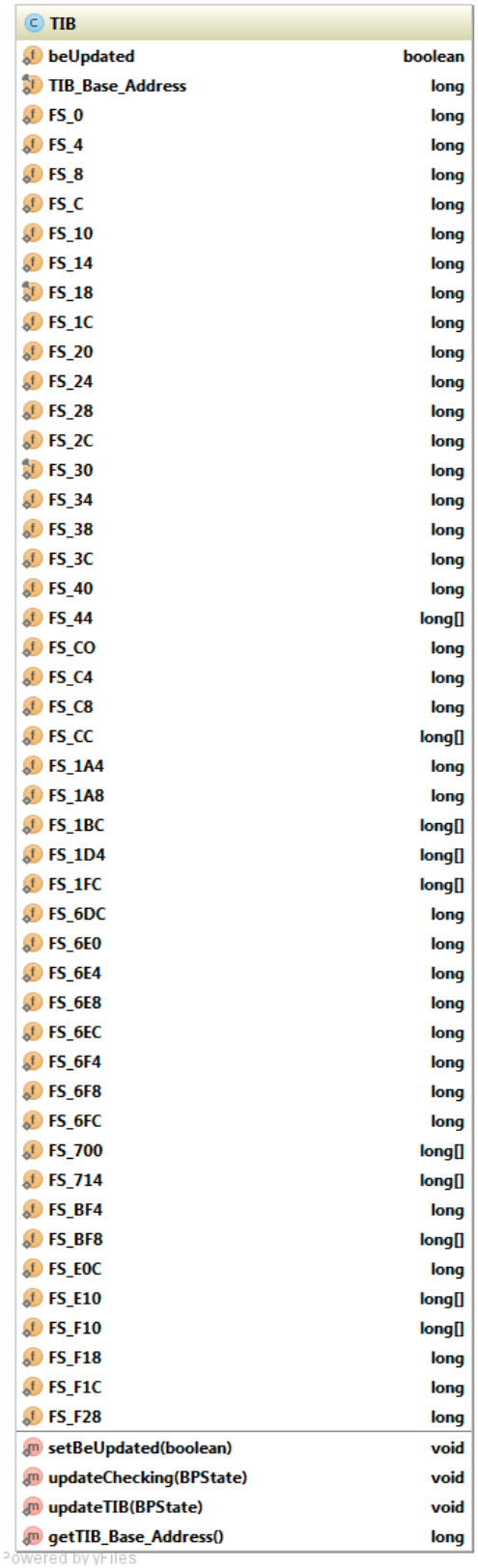
Hình 19: Process Environment Block

1.2 Hiện thực LDRData



Hình 20: LDR Data

1.3 Hiện thực TIB



Hình 21: Thread Information Block

1.4 Cập nhật trong bộ nhớ

Để đảm bảo chương trình có thể lấy đúng các giá trị của TIB, PEB và LDRData, tôi sẽ bổ sung một số thay đổi cho BE – PUM gồm:

- Gán các giá trị ban đầu cho thanh ghi phân đoạn `FS = 0x7EFDD000`.
- Cập nhật hàm `run()` cho lớp OTFModelGeneration, qua đó sẽ gọi hàm `updateTIB()` của lớp TIB, hàm này sẽ ghi tất cả giá trị vào bộ nhớ của BE-PUM và cập nhật trong mỗi lần gọi hàm.

Ngoài ra để đảm bảo các Packer không kiểm tra được tiến trình đang trong ngữ cảnh của debugger tôi sẽ thiết lập giá trị mặc định phù hợp cho các trường tương ứng với các kỹ thuật anti – disassembly của Packer như đã trình bày ở trên.

2. CÔNG CỤ SỬ DỤNG MODEL CHECKING PHÁT HIỆN MALWARE

Để có thể giải quyết vấn đề về sử dụng Model Checking phát hiện Malware khi có CFG của một chương trình thực thi và mẫu hành vi của Malware. Tôi đã hiện thực chương trình với ý tưởng là sẽ đọc vào CFG là kết quả đầu ra của quá trình phân tích của BE – PUM. Chương trình sẽ chuyển đổi CFG của chương trình thành SMV Model được chấp nhận bởi Model Checker NuSMV, đồng thời chương trình cũng sẽ thực hiện việc chuyển đổi mẫu hành vi của Malware mà chương trình cần thực hiện việc kiểm tra dưới dạng biểu thức logic CTL được chấp nhận bởi Model Checker NuSMV.

2.1 Môi trường hiện thực

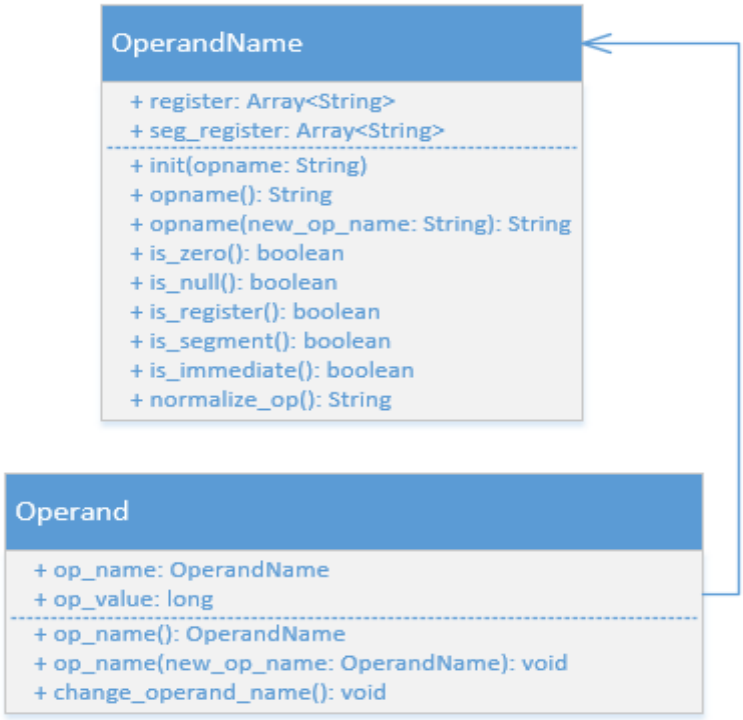
Môi trường để hiện thực chương trình:

- Ngôn ngữ hiện thực: Python 2.7.9
- Model checker: NuSMV 2.5.4
- Cấu hình máy để thực hiện thí nghiệm: Microsoft Windows 7 SP1, Intel Core i5 – 2450M 2.5GHz, 8 GB RAM.

2.2 Cấu trúc của chương trình

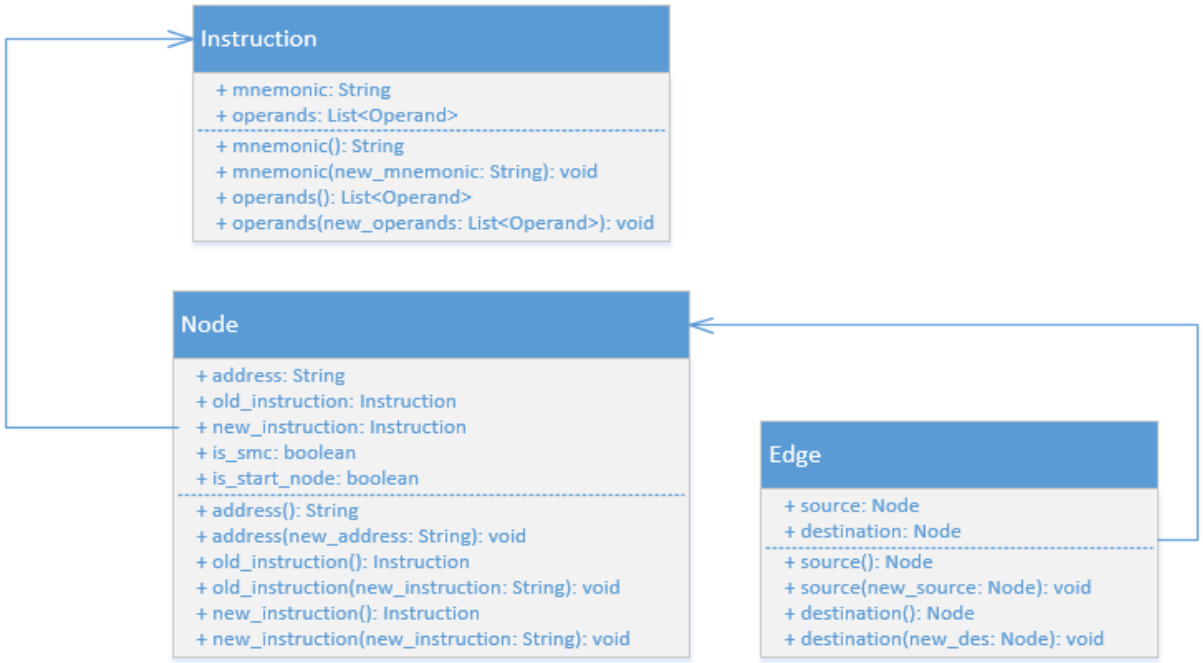
Thiết kế của chương trình bao gồm việc xây dựng các lớp sau:

2.2.1 Operand và OperandName



Hình 22: Hiện thực lớp Operand và OperandName

2.2.2 Node và Edge



Hình 23: Hiện thực lớp Node và Edge

2.2.3 Model Construction

Hình 24: Hiện thực lớp Model Construction

2.2.4 SMV Construction

Hình 25: Hiện thực lớp SMV Construction

2.3 Hạn chế của chương trình

Trong giới hạn của đề tài, chương trình hiện thực của tôi còn một số hạn chế nhất định bao gồm:

- Vấn đề về giá trị thực của thanh ghi: ví dụ nếu ta muốn kiểm tra rằng mô hình có thỏa mãn tính chất thanh ghi `EAX = 0`, trong chương trình hiện thực của tôi, giá trị của thanh ghi không được xét tới do đó việc kiểm tra là hạn chế của chương trình.

```
mov ebx, 0
mov eax, ebx
```

- Truy cập giá trị bộ nhớ gián tiếp: ví dụ nếu ta muốn kiểm tra rằng mô hình có thỏa mãn tính chất là sử dụng câu lệnh nhằm ghi vào giá trị bộ nhớ tại vị trí được lưu trong thanh ghi `EAX`, cũng giống như trên, trong chương trình của tôi giá trị của bộ nhớ là không được xét đến do đó việc kiểm tra cũng là một hạn chế của chương trình.

```
xchg DWORD ptr DS:[EAX], EBX
```

Chính vì những giới hạn như trên mà ta có thể kết luận về kết quả TRUE/FALSE sau khi quá trình kiểm tra mô hình hoàn tất. Theo đó:

- Nếu kết quả trả về là TRUE: ta có thể kết luận rằng chương trình thực thi ban đầu thực sự thỏa mãn tính chất.

- Nếu kết quả trả về là FALSE: ta có thể hiểu kết quả này là UNKNOWN vì ta không thể kết luận được chương trình thực thi ban đầu có thỏa mãn tính chất hay không.

PHẦN V. KẾT QUẢ

1. PHÂN TÍCH CÁC PACKER

Quá trình phân tích Packer tôi sẽ sử dụng tập tin api_test.exe đã được đóng gói bởi các Packer để phân tích trong hệ thống BE – PUM. Để có thể chạy thí nghiệm cho quá trình phân tích Packer tôi sẽ lấy ra 6 tập tin mẫu mà BE – PUM đã phân tích hoàn toàn gồm: api_test.exe, demo1.exe, demo2.exe, helloworld.exe, hostname.exe, ws2_32.exe. Sau đó, tôi sẽ tiến hành đóng gói tất cả tập tin này lần lượt với các Packer mà BE – PUM đã mở gói hoàn toàn với tập tin api_test.exe sau: ASPack, FSG, NPack, PECompact, PETite, UPX, Yoda 1.2 và Yoda 1.3. Kết quả thí nghiệm được trình bày cụ thể trong bảng dưới đây:

Lưu ý:

- *Quá trình thí nghiệm được thực hiện trên môi trường ảo hóa VMWare WinXP SP3, Intel Core i5 – 2450M 2.5GHz, 2 GB RAM, JDK 1.8.*
- *Trong cột trạng thái, những hàng được đánh dấu "x" dùng để xác định những tập tin khi phân tích trên BE – PUM có lỗi và chưa mở gói hoàn toàn.*
- *Trong cột tập tin, những ô được tô xám dùng để xác định những tập tin khi thực hiện đóng gói bằng công cụ phát sinh lỗi, tức công cụ không thể thực hiện đóng gói cho tập tin đó.*

1.1 Thí nghiệm phân tích Packer

Số thứ tự	Thí nghiệm		Kết quả			Trạng thái
	Packer	Tập tin	Nodes	Edges	Time (ms)	
1	ASPack	api_test.exe	1047	1112	122733	
		demo1.exe	14278	14334	1969437	
		demo2.exe	9595	9654	803437	
		helloworld.exe	1032	1103	62781	x
		hostname.exe	889	956	280813	
		ws2_32.exe	882	948	772485	
2	FSG	api_test.exe	244	268	21297	
		demo1.exe	13493	13510	131703	
		demo2.exe	8795	8812	98094	
		helloworld.exe	232	261	8093	x

		hostname.exe	159	178	27688	x
		ws2_32.exe	113	132	56578	x
3	NPack	api_test.exe	602	639	14078	
		demo1.exe	13851	13881	1667500	
		demo2.exe	9153	9183	627657	
		helloworld.exe	590	634	9547	
		hostname.exe	659	711	80968	
		ws2_32.exe	611	659	241828	
4	PECompact	api_test.exe	1127	1178	46328	
		demo1.exe	14376	14419	1575890	
		demo2.exe	9678	9721	744406	
		helloworld.exe	1115	1171	37578	x
		hostname.exe	876	922	175312	
		ws2_32.exe	800	843	351594	
5	PETite	api_test.exe	1568	1635	233328	
		demo1.exe	342	349	15485	x
		demo2.exe	725	732	1641	x
		helloworld.exe				Time out
		hostname.exe	1228	1275	77887	x
		ws2_32.exe				
6	UPX	api_test.exe	323	351	49047	
		demo1.exe	13563	13583	829485	
		demo2.exe	8867	8887	419313	
		helloworld.exe				
		hostname.exe	145	168	50015	x
		ws2_32.exe	139	160	139046	x
7	Yoda 1.2	api_test.exe	677	712	126821	
		demo1.exe	13932	13962	3934051	
		demo2.exe	9235	9265	1553068	
		helloworld.exe	687	728	45671	x
		hostname.exe	453	480	105577	x
		ws2_32.exe	369	389	353402	x
8	Yoda 1.3	api_test.exe	910	946	76827	
		demo1.exe	14158	14189	3665312	
		demo2.exe	9459	9460	1497069	
		helloworld.exe	912	953	50953	x
		hostname.exe	655	682	126765	x
		ws2_32.exe	526	547	236233	x

Bảng 1: Kết quả thí nghiệm phân tích Packer trên BE - PUM

1.2 Kết luận:

Qua kết quả thí nghiệm trên tôi nhận thấy đối với tập tin mẫu api_test.exe được dùng để phân tích, BE – PUM đã mở gói hoàn toàn đối với tất cả các Packer tuy nhiên với một số tập tin khác BE – PUM vẫn chưa thể mở gói hoàn toàn. Trong giai đoạn sắp tới tôi sẽ tiến hành phân tích tiếp tục và tìm hiểu nguyên nhân cụ thể.

2. PHÁT HIỆN MALWARE SỬ DỤNG KỸ THUẬT SEH

Quá trình thí nghiệm của tôi bao gồm 2 giai đoạn chính: giai đoạn phát hiện mẫu hành vi cho kỹ thuật SEH và giai đoạn thí nghiệm chương trình kiểm tra mô hình xác định Malware có sử dụng kỹ thuật SEH.

2.1 Xác định mẫu hành vi kỹ thuật SEH

Trong giai đoạn thí nghiệm này tôi sẽ thực hiện thí nghiệm trên tập gồm 50 Malwares có sử dụng kỹ thuật SEH. Sau quá trình kiểm tra thủ công tôi đã phát hiện 4 dạng của hành vi sử dụng SEH. Qua đó tôi cũng biểu diễn 4 dạng dưới biểu thức logic CTL.

- Dạng 1:

```
push REG/IMM
push FS:[0]
mov FS:[0], ESP
```

Công thức CTL:

```
EF((state(pushl,IMM,NULL,NULL)|state(pushl,REG,NUL
L,NULL))&EF(state(pushl,s_fs$0$,NULL,NULL)&EF(stat
e(movl,r_esp,s_fs$0$,NULL))))
```

- Dạng 2:

```
push REG/IMM
push FS:[REG]
mov FS:[REG], ESP
```

Công thức CTL:

```
EF((state(pushl,IMM,NULL,NULL)|state(pushl,REG,NUL
L,NULL))&EF(state(pushl,s_fs$REG$,NULL,NULL)&EF(st
ate(movl,r_esp,s_fs$REG$,NULL))))
```

- Dạng 3:

```
call REG/IMM
push FS:[0]
mov FS:[0], ESP
```

Công thức CTL:

```
EF((state(call,IMM,NULL,NULL)|state(call,REG,NULL,
NULL))&EF(state(pushl,s_fs$0$,NULL,NULL)&EF(state(
movl,r_esp,s_fs$0$,NULL))))
```

- Dạng 4:

```
call REG/IMM
push FS:[REG]
mov FS:[REG], ESP
```

Công thức CTL:

```
EF((state(call,IMM,NULL,NULL)|state(call,REG,NULL,
NULL))&EF(state(pushl,s_fs$REG$,NULL,NULL)&EF(stat
e(movl,r_esp,s_fs$REG$,NULL))))
```

2.2 Thí nghiệm chương trình xác định Malware

Để chuẩn bị cho thí nghiệm, tôi đã chuẩn bị 2 tập thí nghiệm, một tập tạm gọi là SEH_SET gồm 10 Malwares có sử dụng kỹ thuật SEH và một tập tạm gọi là NONSEH_SET gồm 10 Malwares không sử dụng kỹ thuật SEH.

Kết quả cho hai tập thí nghiệm trên như sau:

- Đối với tập SEH_SET:

Tên	BE – PUM		Thời gian kiểm tra (s)	Kết quả
	Nodes	Edges		
Email-Worm.Win32.Avron.e	1646	1644	9.89441	TRUE
Email-Worm.Win32.Badtrans.b	254	253	0.33388	TRUE
Worm.Win32.Viking.e	94	94	0.14136	TRUE
Virus.Win32.Ditto.1539	106	106	0.14607	FALSE
Virus.Win32.Benny.3219.a	217	217	0.26555	TRUE
Virus.Win32.Bacros.a	329	328	0.56463	FALSE
Virus.Win32.Delf.n	441	440	0.62098	TRUE

Virus.Win32.HLLW.Walker	361	360	0.44177	FALSE
Virus.Win32.Zakk.a	94	93	0.13973	TRUE
Virus.Win32.Spinder.61440	380	379	0.53451	FALSE

Bảng 2: Kết quả thí nghiệm 10 Malwares tập SEH

- Đối với tập NONSEH_SET:

Tên	BE – PUM		Thời gian kiểm tra (s)	Kết quả
	Nodes	Edges		
Worm.Win32.ZwQQ.a	353	353	0.48685	FALSE
Virus.Win32.Younga.4434	108	108	0.15252	FALSE
Virus.Win32.Tufik.c	438	435	0.66547	TRUE
Virus.Win32.Small.2280	316	305	0.39218	FALSE
Virus.Win32.Satan.1529	223	223	0.25606	FALSE
Virus.Win32.Redemption.a	381	377	0.53234	FALSE
Virus.Win32.Partriot.a	362	357	0.50441	FALSE
Virus.Win32.Magic.7045.f	177	177	0.20048	FALSE
Virus.Win32.JuNy.b	370	368	0.49269	FALSE
Virus.Win32.HLLW.Timese.c	370	368	0.51568	FALSE

Bảng 3: Kết quả thí nghiệm 10 Malwares tập NONSEH

2.3 Kết luận:

Qua thí nghiệm của tập SEH_SET, ta có thể thấy mặc dù đều là những Malware có sử dụng kỹ thuật SEH, tuy nhiên kết quả kiểm tra vẫn là FALSE, điều này có thể xảy ra do 2 trường hợp cụ thể sau:

- Những Malware này có sử dụng những kỹ thuật SEH mà kỹ thuật này có mẫu hành vi không thuộc 4 dạng trên.
- Quá trình sinh CFG cho Malware có lỗi hoặc quá trình chuyển đổi từ CFG sang SMV Model là chưa chính xác.

Qua thí nghiệm của tập NONSEH_SET, ta có thể thấy mặc dù tập này bao gồm những Malwares không sử dụng kỹ thuật SEH, tuy nhiên kết quả kiểm tra vẫn có những giá trị là TRUE, điều này có thể xảy ra do mô tả biểu thức CTL sai.

3. KẾ HOẠCH GIAI ĐOẠN LUẬN VĂN TỐT NGHIỆP:

Số thự tự	Công việc cần hoàn thành	Thời gian
1	Tìm lỗi trong quá trình mở gói trong BE-PUM của các Packer đã phân tích	16/06 – 21/06
2	Tiếp tục phân tích thủ công và giải quyết các lỗi, vấn đề trong BE – PUM với các Packer còn lại	22/06 – 19/08
3	Phát triển tiếp tục công cụ kiểm tra mô hình Malware với các mẫu hành vi cơ bản của các kỹ thuật: Entry Point Obscuring, Self – Modifying Code, Indirect Jump, Encryption/Decryption	20/08 – 23/10
4	Tiến hành thí nghiệm trên các mẫu Malware với các kỹ thuật như trên, tiến hành sửa lỗi nếu có	24/10 – 06/11
5	Giải quyết các hạn chế trong chương trình kiểm tra mô hình, thực hiện chạy với các test-case cơ bản và sửa lỗi nếu có	07/11 – 20/11
6	Tìm hiểu và củng cố các kiến thức lý thuyết liên quan	21/11 – 1/12
7	Viết báo cáo luận văn tốt nghiệp	02/12 – 15/12
8	Chỉnh sửa báo cáo	16/12 – 30/12

Bảng 4: Kế hoạch thực hiện luận văn tốt nghiệp

PHẦN VI. PHỤ LỤC

Tài liệu tham khảo

1. Nguyen Minh Hai, Mizuhito Ogawa and Quan Thanh Tho, *"Pushdown Model Generation of Malware"*, June 24th 2006

2. Quan Thanh Tho, with Nguyen Minh Hai, in Collaboration with Mizuhito Ogawa, *"BE – PUM: Binary Emulation for Pushdown Model Generation, a tool for under approximated model generation"*, March 2015

3. Nguyen Minh Hai, with Quan Thanh Tho, in Collaboration with Mizuhito Ogawa, *"Pushdown Model Generation for Malware Deobfuscation"*, March 2015

4. Mizuhito Ogawa, with Nguyen Minh Hai, Quan Thanh Tho, *"Pushdown model generation for binary code"*

5. Mizuhito Ogawa, *"Software Model Checking"*, June 6th 2006

6. Edmund M.Clarke, Orna Grumberg and Doron Peled, *"Model Checking"*

7. S. Bannr, F. Kordon, L.Petrucci, *"CTL Properties"*

8. Win32 Thread Information Block, *http://en.wikipedia.org/wiki/Win32_Thread_ Information_Block*

9. PEB Structure, *https://msdn.microsoft.com/en-us/library/windows/desktop/ aa813706(v=vs.85).aspx*

10. NuSMV, *http://en.wikipedia.org/wiki/NuSMV, http://nusmv.fbk.eu/NuSMV/ tutorial/index.html*

11. Tất cả thông tin về Win32 API Windows API Index, *https://msdn.microsoft.com /en-us/library/windows/desktop/ff818516%28v=vs.85%29.aspx*

12. Tất cả thông tin về các câu lệnh x86, *http://x86.renejeschke.de/html*

13. Hướng dẫn unpack các Packer thủ công, *http://comcrazy.net76.net/REA/*

14. Jeremy Gordon, *"Win32 Exception handling for assembler programmers".*