**ĐẠI HỌC QUỐC GIA HÀ NỘI**

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

BÁO CÁO THỰC TẬP LẦN 1

**DỊCH NGƯỢC VÀ UNPACK**

Sinh viên: Đinh Văn Kiệt

MSSV: 14020247

Lớp: K59H

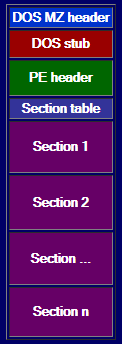
SĐT: 0974659271

Công ty thực tập: Trung tâm an ninh mạng Viettel

Giáo viên hướng dẫn: Ngô Thị Duyên

**LÝ THUYẾT VỀ PE FILE**

PE là viết tắt của cụm từ Protable Executable. Nó là định dạng file thuần túy của Win32. Ý nghĩa của cụm từ “Protable Executable” là định dạng file tiêu chuẩn, rộng rãi trên khắp các nền tảng Win32: PE Loader của mỗi nền tảng Win32 nhận dạng và sử dụng định dạng file này khi hệ điều hành Windows chạy trên các nền tảng chip khác Intel. Điều đó không có nghĩa là file PE của bạn không cần phải điều chỉnh gì để chạy trên các nền tảng khác. Tất cả các file thực thi của win32 (trừ VxDs và 16-bit Dlls) sử dụng định dạng fie PE. Kể cả các nhân driver cũng sử dụng định dạng PE file.



Trên đây là khung cả một PE file. Tất cả các file PE đều phải bắt đầu với trường DOS MZ header. Trong trường hợp file PE được khởi chạy trên môi trường DOS, DOS có thể nhận diện được file này có đúng là file thực thi không nhờ giá trị lưu trong trường DOS MZ header. DOS stub chứa một chuỗi để hiển thị thông báo “This program requires Windows”.

Phía dưới DOS stub là PE header. Chứa struct IMAGE\_NT\_HEADERS. Struct này chứa các trường dữ liệu được sử dụng bởi PE loader. Khi chương trình khởi chạy và hệ điều hành có thể hiểu được PE file thì PE loader sẽ tìm đến offset bắt đầu của PE header, phía sau DOS MZ header.

Nội dung chính của PE file được chia vào trong các block tên là sections. Mỗi section chứa các thuộc tính và thông tin của dữ liệu. Ví như ta coi định dạng PE file như là một ổ đĩa, thì PE header chính là phần boot và các sections chính là các file. Ngay phái sau PE header chính là bảng sections chứa một mảng struct. Mỗi struct bao gồm thông tin về từng sections đó của PE file như thuộc tính, file offset, va,…

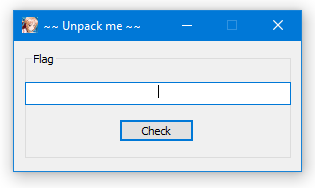
**LÝ THUYÊT VỀ PACK VÀ UNPACK**

**BÀI TOÁN 1**

*[MỨC ĐỘ DỄ]*

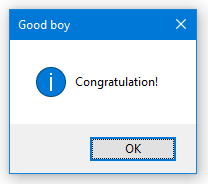
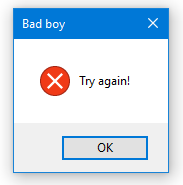
ĐỀ BÀI

Cho một phần mềm A máy tính có giao diện như sau:



Input: một chuỗi nhập từ bàn phím vào khung text

Output: Hiển thị thành công nếu chuỗi nhập vào khớp, hiển thị lỗi nếu không khớp



Nhiệm vụ là ta sẽ đọc code assembly và dịch ngược chương trình xem cách mà chương trình so sánh chuỗi nhập với key vào để hiển thị thông báo. Từ đó ta sẽ tìm được key.

Trước khi thực hiện nhiệm vụ trên, ta cần phải unpack phần mềm này sau đó mới dễ dàng thực hiện dịch ngược.

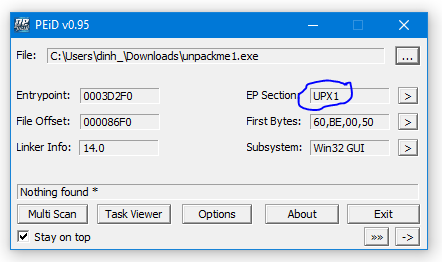
LỜI GIẢI

*CÔNG CỤ SỬ DỤNG:*

* PEID dùng để phát hiện Packer
* OllyDbg 2.01 dùng để debug
* Scylla v0.9.7c (ImpREC 1.7e) dùng để sửa lại Import Table

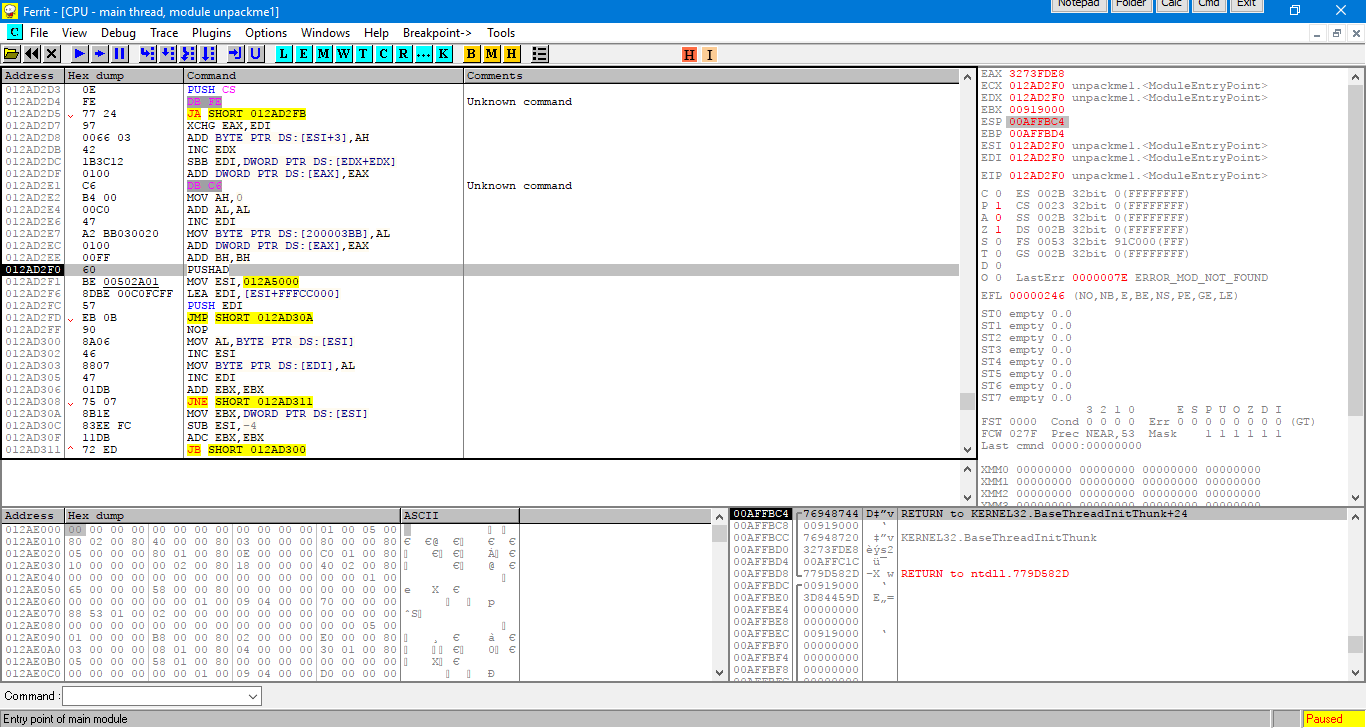
*CÁC BƯỚC:*

Bước 1. Do phần mềm đã được bảo vệ bởi packer, ta tiến hành unpack. Sử dụng PEID xác định packer



Tại trường EP Section, PEID cho ra thông tin UPX là packer

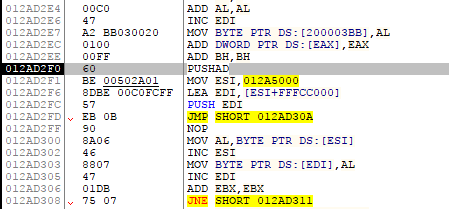
Bước 2. Mở phần mềm A bằng OllyDbg



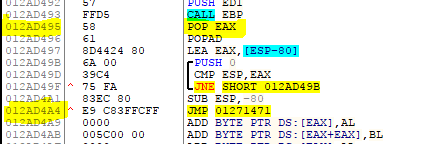
Bước 3. Ta tìm hiểu cơ chế hoạt động của UPX Packer, ta xác định cách thức để unpack như sau

* Do phần mềm A được đóng gói trong 1 file exe của packer, nên khi khởi chạy, packer sẽ tiến hành giải nén phần mềm A
* Do quy định của hệ điều hành, thông số các thanh ghi khi phần mềm khởi chạy phải đc chỉ rõ, nên các packer sẽ phải lưu lại các giá trị của thang ghi bằng lệnh PUSHAD
* Code giải nén của packer được thực hiện
* Sau khi thực hiện xong code giải nén, packer cần phải khôi phục các giá trị của thanh ghi. Nên sẽ sử dụng lệnh POPAD
* Packer hoành thành nhiệm vụ: giải nén và khôi phục thanh ghi. Nên tiếp theo packer sẽ nhảy đến vị trí OEP (Origin Entry Point) của phần mềm A. Đó chính là điểm ta cần tìm
* Packer nhảy đến OEP bằng lệnh JMP (jump không cần điều kiện)
* Nếu có nhiều lệnh JMP thì lệnh JMP nào nhảy đi xa, xa khỏi section hiện tại sẽ là OEP.

Bước 4. Đầu tiên ta tìm kiếm lệnh POPAD trong danh sách các câu lệnh



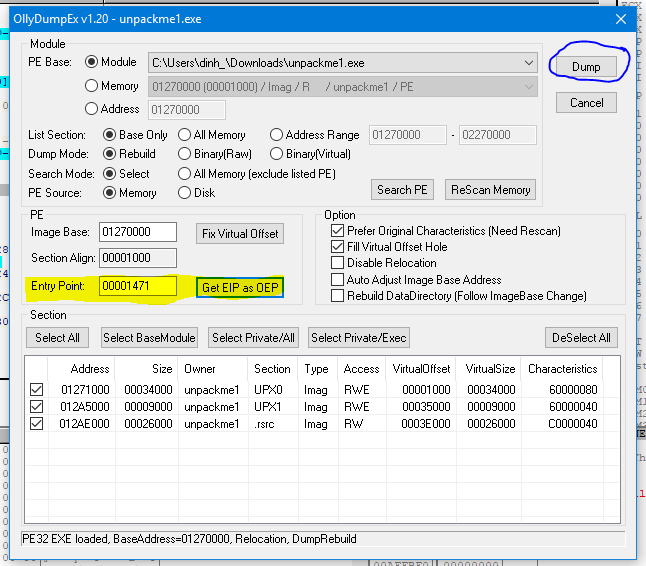
*EP của Packer ngay tại PUSHAD*



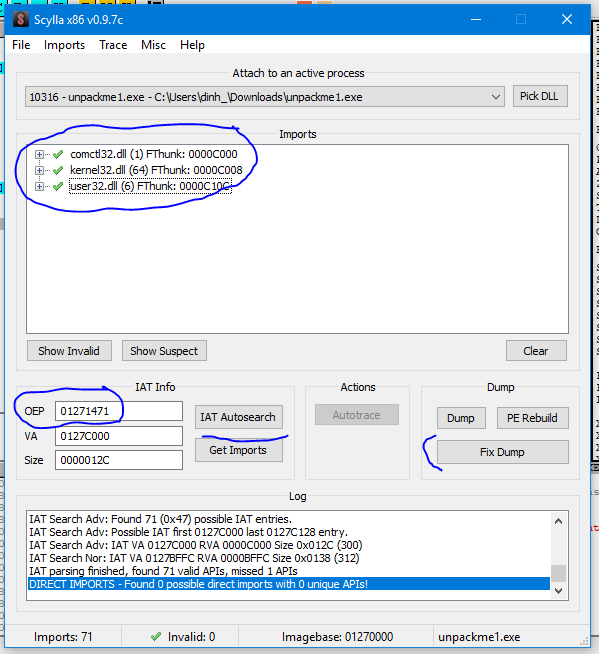
*Tìm kiếm lệnh POPAD*

Vì sau khi khôi phục thanh ghi, packer sẽ nhảy đến OEP thật nên ta chỉ cần tìm lệnh JMP XXX gần nhất sau POPAD. XXX chính là địa chỉ OEP thật. Với bài này OEP cần tìm là **01271471**

Bước 5. Tiến hành jump đến đía chỉ OEP tìm được. Tại đây ta sẽ dump file phần mềm này với OEP vừa tìm được



Bước 6. Sau khi dump, ta cần sửa lại bảng Import Table. Tiến hành mở Scylla, nhập OEP tìm được, Get Import và Fix Dump



Bước 7. Ta thu được phần mềm A đã unpack có kích thước lớp hơn file đã được pack ban đầu (do đã được nén lại)

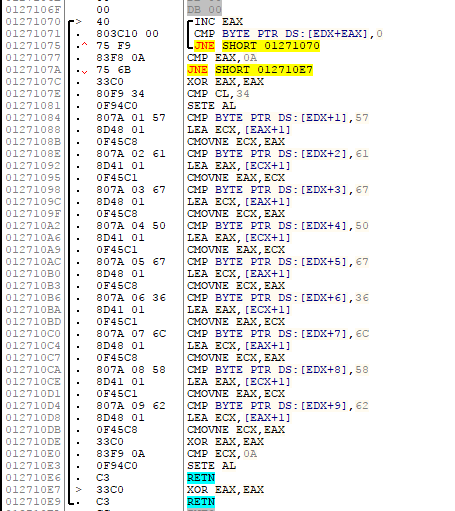
Bước 8. Tiến hành load phần mềm A đã unpack vào trong OllyDbg để tìm key

Bước 9. Dự đoán phần mềm ở dạng Win32 Application, ta sẽ tìm các hàm mà chương trình nào cũng sử dụng (thông tin các hàm có thể tra trên msdn của Microsoft)

* Để tạo cửa sổ, hàm sử dụng DialogBoxParam(…)
* Để hiển thị Popup thường là dùng hàm MessageBox(…)
* Lấy dữ liệu từ Edit Text bằng GetDlgItemText(…)

Đặt các breakpoint tại đây ta sẽ theo dõi được luồng chạy của phần mềm.

Bước 10. Ta theo dõi dữ liệu đi qua hàm GetDlgItemText(…) nhận thấy đoạn code so sánh text ta nhập vào



Tại đây ta nhận thấy phần mềm A so sánh từng kí tự mà ta nhập vào với mã hex của các kí tự khóa, ta ghi lại các mã hex này rồi chuyển qua ASCII sẽ thu được key

34 -> 4

57 -> W

61 -> a

…

Cứ như vậy ta sẽ thu được chuỗi key như sau **4WagPg6lXb**

Bước 11. Thử key này vào và thành công

**BÀI TOÁN 2**

*[MỨC ĐỘ DỄ]*

ĐỀ BÀI

Cho phần mềm B có

LỜI GIẢI