**Unpack**

1. **unpackme1\_5092D59219567B5DAD7A46D75F1708A0.exe**

**A screenshot of a computer program

Description automatically generated**

**A screenshot of a computer program

Description automatically generated**

**Packer: UPX version 3.08**

**Cho vào x32dbg**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Run thì chương trình nhảy đến entry point bị pack**

**Step over để nhảy qua lệnh pushad**

**A screenshot of a computer

Description automatically generated**

**Chuột phải vào esp, follow in dump**

**A screenshot of a computer

Description automatically generated**

**Esp đang chứa giá trị của các thanh ghi được push lên stack sau khi thực hiện lênh pushad**

**A screenshot of a computer

Description automatically generated**

**Đặt breakpoint hardware, access ở giá trị dword mà esp đang trỏ tới, chương trình sẽ dừng khi có bất kỳ truy cập nào (đọc hoặc ghi) vào vùng nhớ đặt breakpoint**

**A screenshot of a computer

Description automatically generated**

**Ta thấy có 1 đoạn jump ngay dưới lệnh popad, nhảy vào đấy**

**A screenshot of a computer

Description automatically generated**

**Entry point gốc của chương trình**

**A screenshot of a computer

Description automatically generated**

**Tiến hành dump bằng Scylla**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**File sau khi dump khi cho vào CFF Explorer trong Import đã xuất hiện nhiều hàm hơn không giống như khi pack lúc đầu**

**A screenshot of a computer program

Description automatically generated**

**A screenshot of a computer program

Description automatically generated**

**Ném vào DIE và PEid cũng thấy không còn bị pack bởi UPX như trước**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer screen

Description automatically generated**

**Sửa trường DllCharacteristic của file dump, bỏ tick DLL can move để khi chạy file ImageBase load theo ImageBase của file**

**A screenshot of a computer error

Description automatically generated**

**Chạy file thì thấy chạy bình thường, chứng tỏ unpack đã thành công**

1. **unpackme2\_2ED2A8B03B7982F9EA3FD463682D7231.exe**

**A screenshot of a computer program

Description automatically generated**

**File unpack2 bị pack bởi Petite version 2.X**

**A screenshot of a computer

Description automatically generated**

**Các hàm import chỉ có ít**

**A screenshot of a computer

Description automatically generated**

**Cho vào x32dbg và thực hiện giống bài 1, chạy qua lệnh pushad thì dump esp và đặt breakpoint ở vùng nhớ push vào esp**

**A screenshot of a computer

Description automatically generated**

**Chương trình nhảy đến đoạn sau khi thực hiện lệnh popad đầu tiên sau khi pushad. Ta thấy có các tham số được push vào hàm và 1 đoạn jmp bên dưới nhảy vào hàm VirtualFree, vậy ta sẽ nhảy qua hàm này thì sau hàm này chính là OEP**

**A screenshot of a computer

Description automatically generated**

**Dòng call đầu chính là OEP. Ta sẽ dump từ đây ra**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**File sau khi unpack thì không còn dấu hiệu bị pack như trước nhưng khi ta chạy thì lại không chạy được**

**A screen shot of a computer

Description automatically generated**

**Khi debug trên IDA thì hiện thông báo vùng nhớ ở địa chỉ 0x16c094 không thực thi được**

**Ta đặt entry point ở OEP để debug**

**A computer screen shot of a program

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

****

**Chương trình gọi đến hàm ở địa chỉ 0x16c094**

**A screen shot of a computer

Description automatically generated**

**Sau đó push dword 0x7B103AB2 lên stack rồi rol với 0xf1 để lấy địa chỉ của 1 hàm**

****

**Sau khi rol xong thì trên stack chính là địa chỉ của hàm kernel32\_GetProcessHeap**

**Vậy ban đầu chương trình call đến địa chỉ 0x16c094 để lấy địa chỉ hàm GetProcessHeap**

**A screenshot of a computer

Description automatically generated**

**Mà địa chỉ 0x16c094 lại nằm trong section petite, mà section petite lúc đầu là 0xc0000040, tức không có quyền thực thi. Vậy ta phải sửa lại permission cho section petite thành 0xe0000040 thành có quyền thực thi**

**A screenshot of a computer error

Description automatically generated**

**Sau khi sửa quyền, chạy thử lại file unpack, kết quả chạy thành công.**

1. **unpackme3\_1F9D59F2DA2E82EB1C7659BFA63CB589.exe**

**A screenshot of a computer program

Description automatically generated**

**File bị pack bởi ASPack**

**A screenshot of a computer

Description automatically generated**

**Load vào x32dbg và làm theo các bước như các bài trên**

**A screenshot of a computer

Description automatically generated**

**Chương trình sẽ nhảy đến đoạn popad đầu tiên**

**Ta thấy chương trình push 1 địa chỉ của 1 vùng nhớ hay 1 hàm nào đó, sau đó ret**

**A screenshot of a computer

Description automatically generated**

**Follow in disassembler để xem đoạn push vào là gì, thì ta thấy có call đến 1 hàm nào đó. Mà như các bài trên đã phân tích, thì có vẻ đây chính là OEP. Ta sẽ thử dump thử**

**A screenshot of a computer

Description automatically generated**

**Khi dump từ OEP là từ địa chỉ trên, IAT Autosearch thì ta thấy các dll được tìm thấy có số lượng hàm lớn, nên có vẻ ta đã lấy OEP chuẩn, dump thử**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**Cho vào CFF Explorer thì thấy có các section .text, .rdata, .data và phần Import các hàm cũng nhiều nên ta đã dump chuẩn**

**A screenshot of a computer program

Description automatically generated**

**Cho vào DIE thì lớp packer đã bị mất**

**Tiến hành run file thì không chạy được, có thể file không chạy được khi load image base ngoài khoảng image base mặc định khi check trong CFF. Vậy ta thử bỏ check flag DLL can move trong DllCharacteristics**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer error

Description automatically generated**

**File chạy thành công**

1. **unpackme4\_FCEC8D94D5FC3FA6300F5570AB651F3A.exe**

**A screenshot of a computer program

Description automatically generated**

**File bị pack bởi FSG(2.0)**

**A screenshot of a computer

Description automatically generated**

**Load vào x32dbg, ta thấy ở ngay gần entry point của file pack đã có lệnh popad**

**Sau khi popad, chương trình sẽ thực hiện 1 loạt lệnh call dword ptr ds: [ebx], ta sẽ thử step into vào để xem**

**A screen shot of a computer

Description automatically generated**

**Hàm này thực hiện những lệnh khá khó hiểu, bình thường nếu có jump thì ở trước đó phải là test, hoặc cmp, ở đây lại không có, nên hàm được gọi có thể là junk code nhằm để anti analysis**

**A screenshot of a computer

Description automatically generated**

**Sau khi thoát ra ngoài lệnh call, ta thấy ở dưới cũng là lặp lại các lệnh call dword ptr ds: [ebx] và jae với logic khá khó hiểu, có khả năng đây cũng là junk code để anti analysis**

**A screenshot of a computer

Description automatically generated**

**Ở phía dưới có 1 lệnh jmp đến lệnh mov eax, ebp ở sau lệnh jmp đó 1 vài lệnh**

**A screen shot of a computer

Description automatically generated**

**Trace tiếp xuống phía dưới lại có 1 lệnh jmp nữa nhảy đến lệnh inc ecx ở sau lệnh jmp đó 1 vài lệnh**

**A screenshot of a computer

Description automatically generated**

**Từ lệnh jmp thứ 2 nhảy đến, ta trace tiếp xuống dưới thì có thêm 1 lệnh jmp nữa**

**A screenshot of a computer

Description automatically generated**

**Lệnh này lại nhảy đến đoạn call ebx ban đầu đã phân tích**

**Như vậy, 3 lệnh jmp này nhằm mục đích để anti analysis chứ không nhảy đến OEP ban đầu**

**A screenshot of a computer

Description automatically generated**

**Lệnh jmp thứ 4 ta nhìn trên ảnh thì có vẻ nó nhảy đến 1 vùng nào đó ở xa vì không hiện mũi tên trỏ lên trên hoặc xuống dưới. Ta thử đặt breakpoint ở đây và step into vào**

**A screenshot of a computer

Description automatically generated**

**Nhảy vào vùng mà nó nhảy thì ta thấy nó gọi đến 1 hàm, ta thử trace vào hàm đó**

**A screenshot of a computer

Description automatically generated**

**Không phải như call ebx lúc nãy, lúc này hàm này có vẻ là 1 hàm thật vì ta thấy có push ebp và ebp, esp ở đầu. Như vây, có thể địa chỉ mà jmp ở trên nhảy vào có thể là OEP**

**A screenshot of a computer

Description automatically generated**

**Dump thử thì có vẻ đúng vì số lượng hàm import có vẻ nhiều**

**A screenshot of a computer program

Description automatically generated**

**Cho vào DIE thì thấy không còn dấu hiệu bị pack**

**A screenshot of a computer error message

Description automatically generated**

**Chạy thử thì thấy chạy thành công, chứng tỏ đã dump đúng OEP và unpack thành công**

1. **unpackme7\_1998BC713BE5132B9356438D53CAE971.exe**

**A screenshot of a computer

Description automatically generated**

**File pack bởi Yoda’s Protector**

**Bài này chạy trên Win XP**

**Ta sẽ dùng Ollydbg để phân tích**

**A white rectangular object with black text

Description automatically generated**

**Chương trình đang ở entry point của file pack**

**Theo tài liệu tìm hiểu, Yoda’s Protector sử dụng int 3 để anti-debug, nếu có debug, chương trình sẽ ngắt và gây lỗi**

**A screenshot of a computer program

Description automatically generated**

**Nên trong phần exception, ta sẽ tích hết các ô để pass exception, nghĩa là nếu gặp exception, ta sẽ pass qua và tiếp tục chương trình**

**Và cũng theo tài liệu tìm hiểu:** [**https://www.astesj.com/publications/ASTESJ\_0506142.pdf**](https://www.astesj.com/publications/ASTESJ_0506142.pdf) **, Yoda’s Protector sử dụng hàm BlockInput để anti-debug. Hàm BlockInput sẽ chặn mọi thao tác từ bàn phím và chuột nếu tiếp cận đến chương trình khi giá trị return về là True. Mà hàm BlockInput được load từ user32.dll**

**A screenshot of a computer program

Description automatically generated**

**Nên trước khi run, ta sẽ tích vào Break on new module (DLL) để khi run, khi chương trình load user32.dll, ta sẽ nhảy vào đó và tìm hàm BlockInput để patch giá trị return thành 0 để không bị block**

**A screenshot of a computer

Description automatically generated**

**User32.dll đã được load, follow vào đó**

**A computer screen with a message

Description automatically generated**

**Tìm nơi hàm BlockInput được gọi**

****

**Ở đây ta sẽ nop dòng 2 và dòng 3 và sửa dòng 1 thành mov eax, 0 để khi ret giá trị eax = 0, tức không bị block**

**A screenshot of a computer

Description automatically generated**

**Sau khi patch sẽ thành như này**

****

**F2 để đặt breakpoint ở ret để sau khi run nhảy về địa chỉ này để eax trả về 0**

**Cũng theo tài liệu ở trên, Yoda’s Protector còn gọi hàm IsDebuggerPresent và GetCurrentProcessId để anti-debug. Ta sẽ nói về hàm IsDebuggerPresent trước**

**Trước tiên, ta tìm đến vùng gọi hàm IsDebuggerPresent**

**A computer screen with a blue box and white text

Description automatically generated**

****

**Ở đây không gọi trực tiếp hàm IsDebuggerPresent mà sẽ truy cập vào TEB ở fs:[0x18], sau đó lại lấy trường ở offset 0x30, tức PEB, và lấy trường ở offset thứ 2, tức flag BeingDebugged. Nếu BeingDebugged khác 0, thì chương trình không bị debug, còn nếu khác không, chương trình sẽ bị debug. Vậy ta chỉ cần patch mov eax, 0 và nop 2 dòng còn lại để khi ret về eax bằng 0**

**A screenshot of a computer

Description automatically generated**

**Tiếp theo là hàm GetCurrentProcessId**

**Theo** [**https://www.astesj.com/publications/ASTESJ\_0506142.pdf**](https://www.astesj.com/publications/ASTESJ_0506142.pdf)**:**

**A text on a white background

Description automatically generated**

**Yoda’s Protector sẽ gọi hàm CreateToolHelp32Snapshot để lấy PID cũng như thông tin về các tiến trình trên máy, sau đó sẽ gọi hàm GetCurrentProcessId để lấy PID của tiến trình hiện tại và so sánh. Nếu tiến trình cha là debugger, nó sẽ terminate luôn tiến trình.**

**Vì vậy, để bypass, thì PID phải luôn trả về PID của debugger bằng cách sửa giá trị trả về là PID của debugger, bởi vì khi đó, GetCurrentProceesId lấy PID của tiến trình con, mà patch thành PID của debugger, tức là tiến trình con chính là debugger, mà CreateToolHelp32Snapshot sẽ check và kiểm tra thấy explorer.exe là tiến trình cha, còn nếu không patch, thì tiến trình cha chính là debugger thì sẽ bị terminate.**

****

**Ở trên lấy trường ở offset 0x20 trong TEB chính là PID rồi gán vào eax và khi ret eax sẽ chính là PID của chương trình hiện tại**

**A screenshot of a computer

Description automatically generated**

**Mở LordPE thì thấy PID của ollydbg là 0xD1C**

**A screen shot of a computer

Description automatically generated**

**Sau khi patch sẽ trông thế này**

**Tiến hành run lại để nhảy vào breakpoint đã set ở trên**

**A white rectangular object with a grey line

Description automatically generated**

**Sau khi run, mở cửa sổ Memory**

**Thông thường, OEP sẽ nằm ở section đầu sau PE header**

**A screenshot of a computer program

Description automatically generated**

**Ta sẽ Set memory breakpoint on access vào section đầu để khi run lại, nếu có vùng nào truy cập đến địa chỉ nằm trong vùng section này nó sẽ nhảy vào đó, đó có thể là OEP**

**A screenshot of a computer

Description automatically generated**

**Sau khi run, nhảy đến 1 đoạn call 1 hàm**

**A screenshot of a computer

Description automatically generated**

**Nhảy vào hàm xem thì có vẻ đây là entry point vì có push ebp, mov ebp, esp chứ không giống 1 hàm để anti-analysis**

**A screenshot of a computer

Description automatically generated**

**Tiến hành dump ra file**

**A screenshot of a computer program

Description automatically generated**

**ImpRec để fix IAT, sau đó cut thunk những thunk có valid no, rồi fix dump**

**A screenshot of a computer

Description automatically generated**

**Cho vào DIE thì thấy file không còn bị pack**

**A screenshot of a computer error

Description automatically generated**

**File sau khi unpack thì chạy bình thường**

1. **unpackme8\_0E29BE1D445143C4135AA9108DF327C4.exe**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**File bị pack bởi ASProtect và Ntkrnl**

**Bài này ta sẽ sử dụng script để unpack**

[**https://github.com/dubuqingfeng/ollydbg-script/tree/master/ASProtect**](https://github.com/dubuqingfeng/ollydbg-script/tree/master/ASProtect) **ta sẽ sử dụng script này để unpack**

**A screenshot of a computer

Description automatically generated**

**Load vào thì nhảy entry point bị pack**

**A screenshot of a computer program

Description automatically generated**

**Ta sẽ sử dụng plugin ODbgScript để load script**

**A computer screen shot

Description automatically generated**

**Đây là script ta cần load**

**A screenshot of a computer program

Description automatically generated**

**Sau khi load script thì chương trình nhảy thẳng vào entry point**

**A screenshot of a computer

Description automatically generated**

**Tiến hành dump ra**

**A screenshot of a computer program

Description automatically generated**

**Dùng ImpRec để fix IAT**

**A screenshot of a computer

Description automatically generated**

**Sau khi dump và fix IAT, cho vào DIE để kiểm tra thì thấy không còn dấu hiệu bị pack**

**A screenshot of a computer error message

Description automatically generated**

**File sau khi unpack đã chạy bình thường**