TỔNG QUÁT VỀ "ĐẢO MÃ"

Bài này tui tham khảo bài viết của Ignatz/stoicForce . Ở đây , tui ko thể dịch sát nghĩa như các lão ấy viết. Vì dịch sát nghĩa ra tiếng Việt thì chắc cũng ko có ý nghĩa gì hết. Vì mấy Lão này ko phải là Người Anh, nên tiếng Anh họ viết khó dịch quá. Đồng thời trình độ tiếng Anh của Benina cũng thuộc dạng amatơ, nên có gì sơ xuất mong các bạn thông cảm. Ở đây ,tui hiểu như thế nào khi đọc tut này thì tui viết lại như cách mình nghĩ mà thôi. Chỉ là phỏng dịch. Nếu có gì sai các bạn góp ý cho.
Vì TUT này rất hay, nên tui chọn tut dành để "biếu" các bạn newbie còn khó khăn khi đọc TUT về Reversing.
Đầu tiên , tui xin dịch "Reverse" là "Đảo mã" (Chứ ko phải "đào mả" nhe) cho nó gọn. Nếu có ai dịch từ này hay hơn , xin cho ý kiến để sao này "ko tái phạm". Hìhì
Nói thêm một chút, một anh Reverser khác với một anh Cracker. Hay nói nôn na một anh chuyên đảo mã khác với một anh chàng bẻ khóa. Anh đảo mã có thể trở thành một anh bẻ khóa .Còn anh bẻ khóa thì chưa chắc trở thành anh đảo mã. Vì nếu bạn đảo mã tốt thì bạn dễ dàng trở thành cracker một cách dễ dàng. Vì lý do đó , nên tui mới dịch TUT này , rồi các bạn sẽ hiểu những gì tui nói.

ĐIỂM SÁNG TRONG QUÁ TRÌNH ĐẢO MÃ CODE

(CONVENTION) TẬP QUÁN:

Khi bắt đầu đảo mã,bạn sẽ sớm thấy rằng bản chất của vấn đề là thường chúng ta dựa vào một số thói quen khi đọc code, giống như cách mà bạn đặt tên cho biến và chú thích cho vòng lặp . Khi đảo mã cũng giống vậy. Chưa bao giờ tui lượt giảm thông tin trong code, mà luôn luôn thêm thông tin vào.Ví dụ, chưa bao giờ viết đè lên 1 address mà thêm một cái tên sau nó:

```
mov [00534160], eax
```
mov [00534160_SerialFlag], eax *; like this you won´t lose vital information*

(COMMENT) CHÚ THÍCH CHO CODE

Đó là điều rất quan trọng ,bạn nên dùng những định nghĩa trong sáng cho code và đừng quan tâm khi bạn phải viết nhiều từ để mô tả code.Bạn sẽ yêu chính bạn về việc làm này khi sau này bạn quay lại đọc những ghi chú mà mình đã viết và bạn sẽ ghét bỏ chính bản thân mình khi ko ghi chú đầy đủ để sau này ko lần ra được những bước mình đã đi qua. Nó cũng sẽ làm một vài phần của code trong sáng hơn cho chính bạn . Từ đó bạn cùng sẽ dễ tìm ra mẫu code làm gì nếu bạn chú thích nó một cách đúng đắn. Rule tổng quát cho việc ghi chú thích là: ghi chú thích code làm sao mà một người chưa đảo mã đọan code đó có thể hiểu được mà ko phải "sorry, understand ". Đơn giản thế thôi, nhưng thật sự việc này ko dễ đâu các bạn ạ. Vì ghi chú thích cho mình nhớ thì dễ , nhưng khi ghi chú thích văn tắt cho một người khác hiểu thì rất khó. Bạn phải có một quá trình đọc code và đồng thời học tập các cao thủ khi họ comment code.

(TRANSLATE) PHIÊN DỊCH CODE:

Đây là phần quan trọng nhất, đó chính là việc đảo code để thay những lệnh asm khó hiểu thành những dòng lệnh trong ngôn ngữ cấp cao. Nếu bạn làm tốt việc này, thì nó sẽ hết sức tầm thường khi bạn sinh ra ngôn ngữ cấp cao mà bạn biết từ code cần đảo mã. Việc phiên dịch và ghi chú thích là 2 việc khác nhau nhưng việc nào cũng quan trọng. Nếu bạn chỉ ghi chú thích , thì kết quả của nguồn code cũng ko được rõ ràng.Lấy ví dụ: có 3 cách khác nhau để thực hiện một vòng lặp. Nhưng khi bạn đọc mã asm thì bạn sẽ thấy có thể nó là loại vòng lặp while, hay repeat, hoặc là for .Hảy nhìn ví dụ dưới đây:

```
xor exc, exc
:00440000
inc ecx          ; comment: counter of loop is increased
.
.
cmp ecx, 05      ; comment: counter is compared
jbe 00440000     ; repeat this while the counter of
                 ; loop is 4 or less continue loop
-----
; reversed 1
while (i < 5) {
  still to come
}
-----
;reversed 2
for (i=0; i<5; i++) {
  still to come
}
-----
;revesed 3
i = 0
repeat
  i++;
  still to come
until (i >= 5 )
```

Những lời ghi chú thích (comments) ko là gì cả , mà nó chỉ làm cho bạn và những người khác hiểu cái gì đang diễn ra trong đoạn code mà thôi. Cách đão mã không thể giống như mã gốc mà người lập trình viết ra . Trừ phi anh ta biết đão mã code ngôn ngữ C ra ngôn ngữ ASM (hiếm có). Như ví dụ trên ta thấy có nhiều các phiên dịch một vòng lặp, ở đây giống loại repeat until hơn, nhưng điều đó ko quan trọng. Miễn sao nó đúng ý nghĩa của đoạn code là được.

FUNCTIONS (HÀM)

Một chương trình có rất nhiều hàm khác nhau, hàm này sẽ lồng vào hàm khác. Điều này cho phép chúng ta dừng lại xem xét tại một phần của code. Một lợi ích khác ,cấu trúc của

code sẽ biểu lộ ở ngòai hàm đó. Nếu bạn đảo mã một cặp hàm lồng nhau, bạn sẽ thấy cấu trúc của hàm sẽ thể hiện ở ngòai đọan code hàm. Cấu trúc này chứa: returnvalues (giá trị trở về), parameters (tham số hàm), stackadjustmenets (thông số Stack), nhiều lệnh khác và hơn thế nữa. Bây giờ chúng ta xem phải mô tả một hàm ra sao:

1. Tên hàm
2. Tham số
3. Giá trị trở về
4. Mục đích của hàm

Tìm tên cho hàm là phần quan trọng nhất. Tên được mô tả sao cho dễ hiểu và đúng mục đích của hàm. Điều này sẽ giúp bạn dễ hơn trong việc giải thích code nếu bạn tìm thấy hàm này sau đó ở đâu trong đọan code. Những tham số được mô tả bởi tên và lọai tham số, đôi khi được chêm vào những mô tả nhỏ. Điều này sẽ trỡ nên dễ hơn khi bạn đảo mã "tập quán" gọi hàm. Bạn sẽ có giá trị trả về của hàm trong thanh ghi eax. Điều này cũng có ý nghĩa một hàm có thể không trả về một string hay 1 đối tượng object. Nó chỉ có thể trả về một con trỏ, trỏ đến một string hay một object. Một khả năng khác là hàm có thể tự viết đè lên các tham số input.Vậy nó cũng có thể làm điều này với các tham số output. Vì vậy những tham số , sau cuối hàm , bạn hảy viết một mô tả ngắn cho hàm đó.
Có lẻ bạn nên nghĩ về vấn đề nhỏ sau đây, hàm này gọi hàm khác, và chúng lại gọi những hàm khác nữa vân vân.Vì vậy chúng ta nên bắt đầu từ đâu?.Bạn nên nghĩ về điều này, hảy dựa vào độ sâu khi bạn bắt đầu đảo mã. Bạn phải sử dụng chiến lược "bóp đí..t" hay "nắm đầu" đọan mã cần đảo, chúng ta sẻ xem xét các vấn đề này.

CALLING CONVENTIONS (TẬP QUÁN GỌI HÀM)

Những cái quan trọng về mục này nằm ở ngòai hàm, chương trình chuyển tham số như thế nào cho hàm và chương trình xử lý các adj như thế nào. Có 2 thói quen chính (hay tập quán cũng được). Đó là tập quán C và Pascal. Nếu bạn ko biết Stack được sử dụng như thế nào trong quá trình gọi hàm, bạn hảy tham khảo tut của tôi : "Our friend and his cousin call". Ở đây tui chỉ nhắc lại điều căn bản sau:
Những tham số được chuyển vào stack, có nghĩa rằng stack sẽ được khôi phục sau khi hàm hòan thành. Nếu ko được khôi phục,thì khi gọi hàm sẽ thay đổi stack điều này sẽ làm cho nó ko làm việc bình thường. Vì vậy tập quán gọi hàm bao gồm 2 phần:

1. Nhập giá trị vào stack
2. Khôi phục stack

Bây giờ chúng ta hảy xem xét 2 tập quán quan trong gọi hàm như trên.Bạn sẽ thấy vô cùng cần thiết khi học về chúng.

- Đối với ngôn ngữ lập trình C, tập quán gọi hàm như sau: (Dùng cả cho việc gọi hàm API):

  Trong case này, các tham số được push vào stack thứ tự khi đão mã qua ASM và khôi phục stack như ví dụ sau:

```
procedure test1(Par1, Par2, Par3: integer);
asm:
push par3    ; push in reversed order
push par2
push par1
```

```
call test1   ; call function
add esp 0C   ; restore stack value
```

- Đối với ngôn ngữ lập trình Pascal, tập quán gọi hàm như sau:

Trong case này, các tham số được push vào stack thứ tự khi đão mã qua ASM và khôi phục stack như ví dụ sau:

```
procedure test1(Par1, Par2, Par3: integer);
asm:
push par1    ; push in same order
push par2
push par3
call test1   ; call function
...          ; no add esp,0C
             ; this was already done within the test1
call
```

LOCAL VARIABLES – PARAMETERS
(BIẾN CỤC BỘ - THAM SỐ)

Khi bạn chỉ định các tham số nào truyền cho hàm , bạn có thể dễ dàng sử dụng các kiến thức này để đặt tên cho các tham số trong hàm. Hai dòng code sau đây có thể được tìm thấy cho tất cả các hàm khi mới bắt đầu:

```
(1.2.I)
push ebp        ; save the original base pointer
mov ebp, esp    ; set basepointer to the stack
(parameters!)
```

Cách này rất hiệu quả cho việc access tham số mà ko bị lỗi về "bảo tòan" stack. Bây giờ mọi tham số có thể thực hiện access qua ebp
Ví dụ về C:
par1=ebp+8
par2=ebp+0C và vân vân

Đừng quên rằng giá trị trở về của hàm và ebp được push vào stack khi hàm được gọi. Vì vậy mà tại sao tham số đầu tiên ko phải là ebp mà là ebp+8. Bây giờ thì bạn sẽ thấy dễ dàng khi làm việc với các tham số với các lệnh cần thiết là push và pop. Dòng lệnh kế tiếp thường đứng sau 2 dòng lệnh trên là là dòng lệnh dưới đây:

```
(1.2.II)
sub esp, 00000018   ; make room on stack
                    ; for local variables
```

Ở đây, Stack đã được đảo mã sang ASM, nhưng ko dùng với lệnh push và pop , các lệnh này chỉ thực hiện access qua ebp và chỉ dùng cho tham số. điều khác biệt ở đây

là , nó ko phải là ebp+8 mà là ví dụ ebp-8. Chú ý rằng ebp-8 là biến cục bộ thứ 3 (tất cả các biến đều có gía trị WORD). Bây giờ bạn hảy hình dung, stackoverflow đến từ đâu:
(Như 1.2.I được sử dụng trong hàm): ebp+X là tham số được gọi
(Như 1.2.II được sử dụng trong hàm): ebp-X là biến cục bộ

Với kiến thức này, chúng ta ko khó khăn lắm khi chỉ ra cái nào là tham số , cái nào là biến cục bộ trong hàm.

Benina 2004