Phần 2
TỔNG QUAN VỀ ĐẢO MÃ

RETURN VALUE (giá trị trả về của hàm)

Nếu hàm trả về một giá trị, nó luôn trả về trên thanh ghi eax hay trong một tham số input (nhập vào trước hàm) nào đó. Bạn sẽ thấy rằng thường thì nếu nó là tham số input, bằng cách kiểm tra nó ta thấy nó là addr hay một giá trị được cung cấp cho hàm call. Nếu nó là một giá trị , nó ko được chỉ định như một tham số , bạn cần chú ý nó sau này. Nếu nó là một addr, nó sẽ trỏ đến một string như ví dụ sau. Sau đó bạn sẽ thấy dòng kế của code miêu tả nó được sử dụng lại hay là ko. Trong phần lớn các case , bạn sẽ thấy nó được sử dụng lại một lần nữa (điều đó thật hiệu quả khi nhớ addr của nó). Bây giờ tui sẽ chỉ cho bạn một lập trình viên pascal với cách nhìn của anh ta khi create một hàm. Điều này sẽ làm cho các khái niệm rỏ ràng trong sáng hơn. Ở đây là khai báo của một hàm:

```
function MyAdd(x, y :integer) : integer; // declaration
var erg: integer;                        // local var
begin
  erg := x + y;                          // do the things
  add := erg;                            // return value
end;
```

Đây là một hàm thông thường, nó chỉ cộng 2 số int, có 2 tham số, và một biến cục bộ được sử dụng tạm thời để gán cho giá trị trả về của hàm. ASM phải cần đến là di chuyển 2 tham số này ra khỏi stack khi hàm thực thi xong, và nó phải nghịch đảo đầy đủ vùng nhớ cho biến cục bộ của hàm...Giống như đoạn mã này:

```
; *Pascal*
push ebx     ; par1
push ecx     ; par2
call MyAdd   ; call the function
             ; in this example, the function restores the
                stack (pascal)
mov ..., eax ; returnvalue in eax

; *C*
push ecx     ; par2
push ebx     ; par1
call MyAdd   ; call the function
add esp, 8   ; restore the stack
mov ..., eax ; returnvalue in eax
```

```
;-------------------------------
; *MyAdd*
;-------------------------------
push ebp      ; save baspointer
mov ebp, esp  ; get a grip on the parameters
sub esp, 4    ; make room for the local var

; adding
mov ebx, [ebp+0C]  ; get first parameter
mov ecx, [ebp+8]  ; get second parameter
add ebx, ecx      ; add
mov [ebp-4], ebx  ; move result to local var
mov eax, [ebp-4]  ; return local var
                  ; returnvalue is eax
leave         ; this instruction does the folowing:
              ; mov esp, ebp
              ; pop ebp
add esp, 8    ; ONLY for pascal!
ret
```

Ở đây bạn sẽ thấy khác nhau giưã tập quán gọi hàm giữa Pascal và C. Nó cũng cho thấy Asm làm việc như thế nào với tham số và biến cục bộ.
Bây giờ tôi sẽ chỉ cho bạn sự giống nhau của chương trình về tham so biến . Chú ý sự khác nhau thì tốt hơn.Tôi đã làm với chính ví dụ trên trong Pascal.Now, chúng ta khai báo một thủ tục. Không phải là hàm.Bới vì nó sẽ có giá trị trả về thật thụ.

```
procedure MyAdd2(var res:integer; x, y :integer) //
declaration
begin
  res := x + y;        // do the things
end;                   // no explicit returnvalue
```

Chú ý điểm này, tham số res sẽ viết đè lên bởi thủ tục với kết quả trả về. Điều này có ý nghĩa là bạn phải cho một tham số biến cho thủ tục mà nó có thể viết đè lên.Nó ko thể là hằng số hay là một số. Chúng ta hảy xem điều gì xảy ra khi nhìn vào đoạn code asm sau:

```
;* i will only do the pascal style since you already
;* saw everything important regarding the
```

```
;* c - pascal differences in the last example
lea ebx, res    ; because it´s a var you have to pass the
address!
push ebx        ; put address as 1st param on stack
push edx        ; edx contains x
push ecx        ; ecx contains y
call MyAdd2     ; call the function
mov ..., res    ; since res was overwritten by MyAdd2
                ; it now has the result

;-------------------------------
; *MyAdd2*
;-------------------------------
push ebp       ; save baspointer
mov ebp, esp   ; get a grip on the parameters
               ; no local variables
; adding
mov eax, [ebp+10]  ; get first parameter
mov ecx, [ebp+0C]  ; get second parameter
add eax, ecx       ; add
mov ebx, [ebp+08]  ; get the address into ebx
mov [ebx], eax     ; move result to variable-parameter
(erg)
leave
add esp, 0C        ; adjust stack
ret
```

Trong ví dụ này chúng ta sẽ thấy sự khác nhau giữa giá trị trả về thật thụ và một biến-tham số

LOOPS (VONG LẶP)

Chỉ ra một vòng lặp có thể là một việc khó, đặc biệt là khi nó lớn và đã được tóm tắt thì làm điều này càng khó hơn. Nhưng trong suốt những năm tôi đảo mã tôi đã rút ra rằng, cứ một lệnh jump nhảy ngược về là một cái nhảy của vòng lặp Loop. Phát biểu **if then else** thông thường, bạn sẽ thấy nó nhảy đến một code nào đó´ ở phía trước, còn đối với Loop thì phải trở về đoạn code nằm trước lệnh nhảy. Bây giờ thì bạn đã dễ dàng hơn để chỉ định được một vòng lặp rồi phải ko. Khi bạn đã chỉ ra được một vòng lặp thì bạn sẽ đảo mã được tiến trình của nó . Bây giờ chúng ta sẽ xem xét phần chính

COUNTER

Nếu bạn đã tiếp xúc với các chương trình asm rồi ,thì các bạn sẽ để ý rằng: thường thì các vòng lặp sử dụng thanh ghi ecx như là một biến đếm cho vòng lặp.Trong trường hợp khác, nếu ecx ko phải là biến đếm , thì hảy xem tại trạng thái nhảy của vòng lặp , nó phải có điều kiện kiểm tra trước khi nhảy (như test eax,eax). Hảy nhờ vào đầu óc của bạn để tìm ra biến đếm là gì.

EXITING CONDITION (Điều kiện thóat vòng lặp)

Khi chỉ định được biến đếm bạn cũng thấy ở đâu trong đọan code biến đếm sẽ được kiểm tra (check) .Sau đó bạn phải tìm ra tại phát biểu nào thì vòng lặp sẽ nhảy và thóat

TYPE OF LOOP (Phân lọai vòng lặp)

Điều này ko quan trọng khi bạn có thể nghĩ ra .Bạn có thể chuyển đổi một đoạn code ra nhiều loại vòng lặp. Chỉ chú ý rằng điều kiện kiểm tra ở sau cùng hay ở trước thôi. Ví dụ , vòng lặp **while** thì đk kiểm tra nằm ở đầu đoạn code, còn **repeat until** thì đk ở sau cùng đoạn code. Ko khó lắm để làm việc này

CONTROL STRUCTURES

Cấu trúc điều khiển rất quan trọng cho các cracker. Chương trình kiểm tra xem bạn đã đăng ký hay chưa, Nó sẽ rất "nặng nề" khi sử dụng cấu trúc điều khiển. Vì vậy các reverser rất cần thiết phải hiểu về chúng, thường thì họ có thể chia ra các trường hợp if-then-elseif-else và case . Không quan trọng để chỉ ra cầu trúc điều khiển và tìm hiểu chúng làm gì .Nhưng rất cần thiết để tìm hiểu ra những thay đổi các phát biểu này có nghĩa là chương trình thực thi như thế nào, chỉ những kiến thức đố bạn mới có thể giải thích và phân xử một phát biểu điều khiển một cách đúng đắn. Tôi chỉ điểm qua một vài ví dụ ở đây:

IF THEN ELSE

Đây là một phần của code trong Opera 3.62. Mục đích của code này set hiển thị tên chương trình trong thanh tool bar hoặc là <Opera 3.62 (Unregistered version)> cho người chưa đăng ký hay <Opera 3.62> cho người đã đăng ký.

```
if (regged) progname = "Opera 3.62"
else progname = "Opera 3.62 (Unregistered version)"
```

Nhưng nó lamì gì như đọan code dưới đây:

```
cmp byte ptr [PrgDisplayName_00531640], bl
  ; do the following only if PrgDisplayName=""
jne 0045FE49
mov esi, 00000080
  ; esi = 80 (maxCount)
push esi
  ; maxCount (maximum chars to append)
push edi
  ; append at edi = PrgDisplayName

String Resource ID=20092: "Opera 3.62"
      |
push 00004E7C
  ; str to append
mov ecx, ebp
call 00460BD8 - AppendStr(ToApp,EndOfOrig,Count)
push edi
call 00501E10 - int strLen(str S)
  ; calculates lenght of the prgDisplayName string
pop ecx
sub esi, eax
  ; calcuate maxCount
mov dword ptr [005315F4], eax
push esi
  ; maxCount
lea eax, dword ptr [eax+PrgDisplayName_00531640]
  ; eax = end of DisplayName
push eax
  ; EndOfOrig

String Resource ID=21428:" (Unregistered version)"
      |
push 000053B4
  ; str ToAppend
mov ecx, ebp
call 00460BD8 - AppendStr(ToApp,EndOfOrig,Count)

* Referenced by (C)onditional Jump at Addresses:
|:0045FDD3(C), :0045FE12(C)
|
cmp dword ptr [ebp+flg_Regged_000004EC], ebx;=0
  ; check regFlag
mov eax, dword ptr [005315F4]
  ; length of Opera 3.62 string
je 0045FE5E
  ; if regFlag = 0 then set end of PrgDisplayName
  ; right behind "3.62", this means
```

```
  ; discrad the "(Unregistered version)" part
mov byte ptr [eax+PrgDisplayName_00531640], bl
  ; mov 00 after the Opera 3.62!
  ; "thats why the first  chracter of manually
  ; entered Names vanished  because the lenght
  ; function was 0"
jmp 0045FE65
```

Như bạn thấy, chương trình đi theo một hướng khác. Nó ko khác nhau giữa ko đăng ký và đã đăng ký tôi nghĩ như vậy. Nó luôn luôn put phần ko đăng ký trong thanh tool bar. Điều này làm tui rối cho đến khi tui thấy nó làm việc như thế nào. Chương trình viết đè string đến 3 lần nếu bạn được reg. Rồi nó set cuối string là zero.Vì vậy phần ko đăng ký đã được bỏ qua. Hảy xem đoạn này:

```
s = "Opera 3.62\0";
a = " (Unregistered version)\0";
append_this_to_at(a, s, endOf(s));
if regged then
  s[10]="\0";
end;
```

(còn tiếp phần 3)
Benina (2004)