TỔNG QUAN VỀ ĐẢO MÃ – PHẦN 3

Case

Đây cũng là đoạn code trong Opera 3.62. Nó chọn lựa những trang help nào hiển thị trong browser. Tôi lướt qua đoạn code này một cách ngẫu nhiên, nhưng nó là một ví dụ hữu ích cho lệnh switch/case. Trước khi bạn tìm hiểu về nó, tôi muốn bạn chú ý rằng , có 2 cách khác nhau trong phát biểu switch. Một là viết ra 1 chuỗi các lệnh nhảy jump. Điều này tương đương với "if then sldif elseif ....end" và Một nữa là loại mà Opera 3.62 sử dụng ở đây. Nó nhanh hơn loại đã đề cập trước đó (loại "if then sldif elseif ....end"). Ở đây, chương trình tính tóan một một "điểm nhảy"(jumpmark) [4*eax+004932A3] đến code tương đương, thay vì đi qua mọi dòng line để check lệnh jump như loại đầu tiên. Đây đơn giản chỉ là cách làm việc của code, bởi code có chiều dài giống nhau cho mọi nhánh – 4 byte. Thông thường đây ko phải là case, vì bạn phải đề cập một chuỗi các phát biểu "cmp jne" mà chúng tham chiếu đến loại "if then elsif elsif elsif...end". Bây giờ hảy xem đoạn code sau:

```
:00493070 mov eax, dword ptr [ebp+10]; eax = Par1
:00493073 add eax, FFFFB1DD
  ; Par1 = Par1 -  20003d "look at stringrefs"
:00493078 cmp eax, 0000008A
  ; if eax smaller than 138d jump toindex.html
:0049307D ja 0049305E
:0049307F movzx eax, byte ptr [eax+00493357]
:00493086 jmp dword ptr [4*eax+004932A3]
  ; switch statement, calculate jumpmark

* Data Obj ->"keys.htm"
            |
:0049308D push 0051DD58 ; a jumpmark
:00493092 jmp 00493063

* Data Obj ->"prefmenu.htm#print"
            |
:00493094 push 005256B0 ; another jumpmark
:00493099 jmp 00493063

* Data Obj ->"dialogs.htm#direct"
            |
:0049309B push 0052569C ; ...
:004930A0 jmp 00493063

* Data Obj ->"prefmenu.htm#sethome"
            |
:004930A2 push 00525684
:004930A7 jmp 00493063

* Data Obj ->"dialogs.htm#fileuplf"
            |
:004930A9 push 0052566C
```

```
:004930AE jmp 00493063

* Data Obj ->"dialogs.htm#hotlist"
           |
:004930B0 push 00525658
:004930B5 jmp 00493063

* Data Obj ->"dialogs.htm#locked"
           |
:004930B7 push 00525644
:004930BC jmp 00493063
; and so on ...
```

Các biến Global :

Trong phần lớn các chương trình có nhiều biến và hằng số mà chúng có khả năng truy xuất mọi lúc, mọi nơi. Đây ko phải là biến local, bởi chúng bị loại bỏ sau khi hàm kết thúc. Vì vậy chúng (biến loacal) chỉ có thể truy xuất bởi chính hàm của chúng. Các biến có thể được truy xuất đến mọi lúc, mọi nơi được gọi là các biến global (biến tòan cục) (trái với local). Vì vậy một reverser sẽ phải chú ý "theo dõi" chúng, bởi chúng chứa các cờ flags giống như registrationflags (cờ cho biết soft đăng ký chưa), demoflags (cờ xác định soft có phải là loại demo hay FULL), trialdate (cờ xác định cho dùng thử, giới hạn tgian) , ...và các dữ liệu khác như tên chương trình, các tham số, ...Xác định các biến global có thể là một công việc khó. Tôi đã tìm hiểu có 2 cách Opera truy xuất các biến của nó. Một là dùng addr trực tiếp. Điều này có ý nghĩa là , nếu biến là 00543380 chương trình truy xuất đến nó bởi chính là số này (00543380)

direct mode: mov eax, dword prt [00543380]

Một cách khác nữa, nó có thể dùng mode addr liên quan. Bạn có thể thấy một ví dụ cho trường hợp này trong phần kế tiếp. Với mode này, chương trình truy xuất biến liên quan đến một "địa chỉ nền" (baseaddress). Base này được chứa trong một thanh ghi. Nếu bạn thấy vài thứ giống như vậy, thì nó là 1 loại khó để xác định biến global. Nhưng vẫn có thể, bạn chỉ phải search giá trị của offset. Đừng rối với những thứ loại này, nó sẽ "sáng sủa" hơn trong lần thứ 2 chương trình "chạm" đến nó. Nhìn vào ví dụ sau:

```
mov eax, [esi  + 4EC]
          |      |
          base   offset
```

Bây giờ điều bạn phải làm là tìm kiếm giá trị offset, nếu bạn có thể tìm nó ở những lúc khác với "nội dung" giống như thế, thì bạn đã tìm ra 1 biến global. Giống như là tôi tìm ra regflag (cờ đăng ký) ở đây:

```
; first appearence
:004CB0AF mov eax, dword ptr [esi+flg_Regged_000004EC]
:004CB0B5 cmp eax, ebx
:004CB0B7 lea edi, dword ptr [esi+flg_Regged_000004EC]

; second appearence
```

```
:004D9543 mov dword ptr [esi+flg_Regged_000004EC], eax  ; check that
:004D9549 pop ebx
:004D954A je 004D9556

; third appearence
:004D963C mov eax, dword ptr [ecx+flg_Regged_000004EC]  ; return with
regFlag
:004D9642 ret


; there are still many more occurences of this addressing but i think
you got the point
```

Biến được định trị lúc khởi tạo (Intialization):

Các biến này phải được định trị khi khởi tạo. Bạn có thể thấy điều này như thế nào trong ví dụ sau đây. Khi bạn thấy một phần code giống như vầy thì bạn sẽ biết chúng là gì.

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
; /*INIT SECTION */
|:0045BD04(U)
|
:0045BD12 mov dword ptr [esi+regName_00000138], ebx
:0045BD18 mov dword ptr [esi+0000013C], ebx     ; the variables are
addressed via esi+X
:0045BD1E mov dword ptr [esi+00000144], ebx     ; this means relative
addressing
:0045BD24 mov dword ptr [esi+00000148], ebx
:0045BD2A mov dword ptr [esi+0000014C], ebx
:0045BD30 mov dword ptr [esi+00000150], ebx
:0045BD36 mov dword ptr [esi+00000154], ebx
:0045BD3C mov dword ptr [esi+00000158], ebx
:0045BD42 mov dword ptr [esi+0000015C], ebx
:0045BD48 mov dword ptr [esi+00000160], ebx
:0045BD4E mov dword ptr [esi+00000164], ebx
:0045BD54 mov dword ptr [esi+0000038C], ebx
:0045BD5A mov dword ptr [esi+00000184], ebx
:0045BD60 mov dword ptr [esi+00000188], ebx
:0045BD66 mov word ptr [esi+00000212], 0008
:0045BD6F mov dword ptr [esi+00000218], ebx
:0045BD75 mov dword ptr [esi+0000021C], ebx
:0045BD7B mov dword ptr [esi+00000224], ebx
:0045BD81 mov dword ptr [esi+00000220], ebx
~something deleted~
:0045BE9F mov dword ptr [esi+00000358], ebx
:0045BEA5 mov dword ptr [esi+0000036C], ebx
:0045BEAB mov dword ptr [esi+00000380], edi
:0045BEB1 mov dword ptr [esi+0000037C], edi
:0045BEB7 mov dword ptr [esi+00000378], edi
:0045BEBD mov dword ptr [esi+00000374], edi
:0045BEC3 mov dword ptr [esi+00000370], edi
:0045BEC9 mov dword ptr [esi+00000384], edi
:0045BECF mov dword ptr [esi+00000388], edi
```

```
:0045BED5 mov dword ptr [esi+000003C4], edi
:0045BEDB mov word ptr [esi+000003B4], bx
:0045BEE2 mov dword ptr [esi+000003EC], ebx
:0045BEE8 mov dword ptr [esi+000003F4], ebx
:0045BEEE mov dword ptr [esi+000003F8], ebx
:0045BEF4 mov dword ptr [esi+000003F0], edi
:0045BEFA mov word ptr [esi+000004D8], bx
:0045BF01 mov dword ptr [esi+000003DC], ebx
:0045BF07 mov dword ptr [esi+000002A8], ebx
:0045BF0D mov dword ptr [esi+000002AC], ebx
:0045BF13 mov dword ptr [esi+00000250], ebx
:0045BF19 mov dword ptr [esi+00000254], ebx
:0045BF1F mov dword ptr [esi+00000258], ebx
:0045BF25 mov dword ptr [esi+0000025C], ebx
:0045BF2B push 00000720
:0045BF30 mov dword ptr [esi+flg_Regged_000004EC], ebx  ; INIT
```

Ở đây ko cần nói nhiều. Bạn có thể nhận biết được một cách dễ dàng mỗi biến được set như thế nào . Phần lớn chúng là flags nhưng một vài là addr trỏ đến string hay dữ liệu khác.

Tạo nguồn code trong ngôn ngữ cấp cao:

Bước sau cùng trong kỹ thuật đảo mã là tạo lại nguồn code trong ngôn nhữ cấp cao. Bạn có thể sẽ hỏi làm điều đó như thế nào?. Có 2 phương pháp thông thường để làm điều này. Một là hòan tòan theo chương trình target và cũng bám sát theo các chỉ thị của nó mà đảo mã , đôi khi ngay cả ko cần biết chúng là gì, ta sẽ tìm hiểu ý nghĩa của chúng sau khi đão mã khi đọc nguồn code trong ngôn ngữ cấp cao đã tạo lại .Phương pháp thứ 2 là : hiểu code và viết lại nguồn code của chính mình, mà ko cần đảo mã mọi thứ khi bạn có thể đóan code đó làm việc như thế nào . PP thứ nhất rất rối trong việc xây dựng lại file nguồn nguyên gốc, vì vậy bạn phải viết nó trong 1 ngôn ngữ giống như code nguyên gốc. PP thứ hai chỉ là tạo nguồn code mà nó có chức năng tương tự như nguyên gốc(Ví dụ: ko quan trọng khi bạn show 1 cái nag với hàm dialogbox hay là hàm messagebox để người dùng biết rằng anh ta đã sai khi reg, nhưng chương trình đảo mã thì ko cần chính xác như thế), khi nó có thể được viết trong một ngôn ngữ nào đó. Tất nhiên cách "vẽ lại" này , bạn sẽ chưa biết chính xác chương trình làm việc có giống như khi bạn xem theo chức năng của code khi reversing hay ko – Cần nhiều cách để kiểm tra lại, phải được thực hiện trong trường hợp thứ 2 này. Nhưng nó nhanh hơn khi nhìn xóay vào mọi dòng trong asm. Thêm nữa trong pp đầu tiên là bạn có thể đôi khi tiếp tục reversing mà ko cần hiểu ý nghĩa của code. Nó có thể sẽ sáng sủa hơn sau này khi đã đảo mã ra ngôn ngữ cấp cao. Hai cách trên, tùy bạn lựa chọn khi reversing thôi.
Bây giờ xem ví dụ sau:

```
:004BCF49 push ebp                                  ; save base pointer
:004BCF4A mov ebp, esp                              ; set basepointer for
the function
:004BCF4C sub esp, 00000548                         ; make room for stack
:004BCF52 push ebx                                  ; save ebx
:004BCF53 mov ebx, ecx                              ; ebx = ecx
:004BCF55 cmp dword ptr [ebx+00000714], 00000000  ; if [ebx+714] == 0
```

```
:004BCF5C je 004BCF66                                ; contine with
function
:004BCF5E push 00000001                              ; eax = 1
:004BCF60 pop eax                                    ; .
:004BCF61 jmp 004BCFEF                               ; leave with with eax
= 1

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004BCF5C(C) /* continue with ?WriteRegFile(?) */
|
:004BCF66 push esi                                   ; push source
:004BCF67 push edi                                   ; push destination
:004BCF68 mov ecx, ebx                               ; ecx = ebx
:004BCF6A call 004BC81A
:004BCF6F mov ecx, 0000012F                          ; ecx = 0x12F (303d);
303*4 = 1212 (0x4BC)
:004BCF74 mov esi, ebx                               ; sourceaddress of
RegInfo
:004BCF76 lea edi, dword ptr [ebp+FFFFFAB8]          ; destination address
of RegInfo
:004BCF7C lea eax, dword ptr [ebp+FFFFFAB8]          ; .
:004BCF82 repz                                       ; while not finshed
:004BCF83 movsd                                      ; move the RegInfo
:004BCF84 push eax                                   ; pointer to
RegFileBuffer
:004BCF85 mov ecx, ebx                               ;
:004BCF87 call 004BCF14 - Decrypt(chr *ToDecrypt) ; encrypt the RegInfo

* Reference To: KERNEL32.SetFileAttributesA, Ord:0268h
                                    |
:004BCF8C mov esi, dword ptr [005121A4]                ; put address of
SetFileAttributes into esi
:004BCF92 push 00000080                             ; attributes to set
:004BCF97 lea edi, dword ptr [ebx+CRegFileSize_000004BC]; edi = addr of
filename
:004BCF9D push edi                                  ; address of filename
:004BCF9E call esi                                  ; Make File writable
:004BCFA0 push 00001010                             ; fuMode (action and
attribs)
:004BCFA5 lea eax, dword ptr [ebp+CRegInfBuf_FFFFFF74]; eax = address
of buffer
:004BCFAB push eax                                  ; address of buffer
:004BCFAC push edi                                  ; addres of Filename

* Reference To: KERNEL32.OpenFile, Ord:01E8h
                                   |
:004BCFAD Call dword ptr [00512228]
:004BCFB3 mov ebx, eax                              ; ebx = hfile
OpenFile(fName,[opts]) (FileHandle)
:004BCFB5 cmp ebx, FFFFFFFF                         ; if open succeeds
:004BCFB8 jne 004BCFBE                              ; then continue with
ebx = eax
:004BCFBA xor ebx, ebx                              ; else handle = 0
:004BCFBC jmp 004BCFE6                              ; skip writing part.

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
```

```
|:004BCFB8(C)
|
:004BCFBE push CRegFileSize_000004BC            ; number of Bytes to
write
:004BCFC3 lea eax, dword ptr [ebp+FFFFFAB8]     ; eax = address of
Encrypted RegInfo
:004BCFC9 push eax                              ; Pointer to buffer
holding encrypted reginfo
:004BCFCA push ebx                              ; filehandle

* Reference To: KERNEL32._lwrite, Ord:02F7h     ; is used to write
Regfile !
                                  |
:004BCFCB Call dword ptr [005121CC]
:004BCFD1 xor ecx, ecx                          ; set ecx = 0 because
of setne cl later
:004BCFD3 cmp eax, FFFFFFFF                     ; check for errors
:004BCFD6 setne cl                              ; set if an error
occoured
:004BCFD9 push ebx
:004BCFDA mov dword ptr [ebp-04], ecx           ; save result _lwrite

* Reference To: KERNEL32._lclose, Ord:02F2h
                                  |
:004BCFDD Call dword ptr [005121C8]             ; close file
:004BCFE3 mov ebx, dword ptr [ebp-04]           ; ebx = result of
_lwrite

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004BCFBC(U)
|
:004BCFE6 push 00000021                         ; make file
WriteProtected
:004BCFE8 push edi                              ; pointer to filename
:004BCFE9 call esi                              ; SetFileAttributes
:004BCFEB pop edi                               ; resotre values
:004BCFEC mov eax, ebx
:004BCFEE pop esi

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004BCF61(U)
|
:004BCFEF pop ebx
:004BCFF0 leave
:004BCFF1 ret                                   ; bye
```

Sau khi đọc ghi chú, sẽ sáng sủa hơn cho ta biết chương trình làm việc như thế nào. Tôi sẽ cố gắng tạo 1 nguồn code C++ làm việc giống như thế, hảy xem đoạn code sau:

```
#define CRegFileSize 1212

// global Var gRegFileName
/* Name:       WriteRegInfoToFile
 * Purpose:    Writes the encrypted Reginfo into the file OUser350.dat
 *             and sets its fileattributes to writeonly
```

```
 * Returnvale: Either 1 if function fails, or it returns the result of
 *             the _lwrite function which is temporarily stored in res.
 * Remarks:    Have to find out about the first flag and the first
 *             function. Very strait forward implementation.
 */
int WriteRegIfnoToFile(void) {
 // variables
    char   *CryptRegInfo,     // Holds the encrypted
Registrationinformation
           *RegInfo,          // Holds the original
Registrationinformation
           *RegFileBuffer ;   // Filebuffer for the RegFile
    handle hFile;             // Handle to Regfile
    int    res;               // result of function

    if !unknownflag_[ebx+714] {
        exit 1;
    } else {
        (void) unknownfunction_004BC81A(uPar1, uPar2);
        (void) StrCpyN(CryptRegInfo, RegInfo, CRegFileSize);
        (void) encrypt(CryptRegInfo);
        (void) SetFileAttributes(gRegFileName, FILE_ATTRIBUTES_NORMAL);
        if (hFile = OpenFile(gRegFileName, RegFileBuffer, 10)) {
            res = _lwrite(hFile, CryptRegInfo, CRegFileSize);
            _lclose(hFile);
        }
        (void) SetFileAttributes(gRegFileName,
FILE_ATTRIBUTES_READONLY);
        return res;
    }
}
```

Đến đây là kết thúc loạt tuts này. Nếu có thời gian tui sẽ làm tiếp các phần tiếp theo để các bạn hiểu ra hơn.

Hết /
Benina (2004)