The Binary Auditor™

Manual Decompilation Exercises

www.Binary-Auditing.com

Manual Decompilation

Manual decompilation is the art of converting back a fragment of an analysed binary to a high level language (usually C/C++) or pseudo code by hand. Your goal is to recover the C/C++ code or pseudo code that generated the following assembler code.

Let us take as an example the following code:

```
mov     ebx, offset src
push    n
push    c
push    ebx
push    offset dest
call    _memccpy
add     esp, 10h
```

We can decompile it the following way:

```
memcpy(dest,src,c,n);
```

Your goal is to provide a commented pseudo code or better a manually decompiled version of the following exercises in the same way we just decompiled the above given code snippet.

1 Manual Decompilation – Exercise 1

Your goal is to analyse the following compiler-generated assembly language code and to understand how it works.

```
...
mov     edx, Var1
mov     ecx, Var2
mov     eax, edx
imul    ecx
mov     edx, eax
imul    edx, eax
mov     Var3, ecx
...
```

You must retrieve the proper C/C++ code or pseudo code of this commented code.

Your solution has to contain either a full commented C/C++ code or a detailed pseudo code describing the function of the above snippet.

2 Manual Decompilation – Exercise 2

Your goal is to analyse the following compiler-generated assembly language code and understand how it works. It contains a very simple loop.

```
     ...
     mov     dword ptr [esi], 1
     xor     edx, edx
     mov     [ebx], edx
     jmp     short loc_4012F1
loc_4012E8:
     mov     ecx, [esi]
     imul    ecx, [esi]
     mov     [esi], ecx
     inc     dword ptr [ebx]
loc_4012F1:
     cmp     dword ptr [ebx], 8
     jl      short loc_4012E8
     ...
```

You must retrieve the proper C/C++ code or pseudo code of this commented procedure.

Your solution has to contain either a full commented C/C++ code or a detailed pseudo code describing the function of the above snippet.

3 Manual Decompilation – Exercise 3

Your goal is to analyse the following compiler-generated assembly language code and understand how it works. It contains two conditional branches.

```
     push    sPassword                       ; line of code
     call    _strlen

     pop     ecx
     mov     esi, eax

     mov     ebx, offset sMyPassword;        ; line of code
     push    ebx
     call    _strlen
     pop     ecx

     cmp     esi, eax                        ; small block of code
     jz      short loc_4012B2
     xor     eax, eax
     jmp     short end_proc


loc_4012B2:
     push     esi


     push    ebx                             ; line of code
     push    sPassword
     call    _strcmp
     add     esp, 8

     test    eax, eax                        ; small block of code
     jnz     short loc_4012CC
     mov     eax, 1
     jmp     short end_proc

loc_4012CC:
     xor     eax, eax

end_proc:                                    ; end of function
     pop     esi
     pop     ebx
     pop     ebp
     retn
```

You must retrieve the proper C/C++ code or pseudo code of this commented procedure. You need to explain which instruction(s) should be changed (patched) to make the function return a positive result in most common cases. Try to change as less bytes as possible!

Your solution has to contain either a full commented C/C++ code or a detailed pseudo code describing the function of the above snippet.

4 Manual Decompilation – Exercise 4

You already know how to reverse basic code snippets.

Let us take the following code:

```
push    ebp
mov     ebp, esp
add     esp, -80h
push    ebx
mov     eax, V2
mov     ebx, eax
mov     ecx, V3
imul    ebx, ecx
mov     V4, ebx
mov     edx, V1
add     edx, eax
sub     edx, ecx
mov     V1, edx
add     ebx, edx
mov     V3, ebx
```

We can analyse it the following way:

```
push    ebp             -->   Stack frame creation, no code
mov     ebp, esp
add     esp, -80h       -->   this tell us our stack frame is 128 bytes

push    ebx             -->   save ebx contents
mov     eax, V2         -->   take V2 value
mov     ebx, eax        -->   duplicate V2 value
mov     ecx, V3         -->   take V3 value
imul    ebx, ecx        -->   calculate V2 * V3 in ebx
mov     V4, ebx         -->   “V4 = V2 * V3”, ebx holds V4 then

mov     edx, V1         -->   take V1 value
add     edx, eax        -->   calculate (V1+V2) in edx
sub     edx, ecx        -->   calculate (V1+V2) - V3
mov     V1, edx         -->   “V1 = (V1+V2) - V3”
add     ebx, edx        -->   calculate (V2*V3) + V1
mov     V3, ebx         -->   “V3= V4+V1”
```

In other words: the above lines of assembler can be manually decompiled in:

```
{
      unsigned char buffer[128];

      V4 = V2*V3;
      V1 = V1+V2-V3;
      V3 = V4+V1;
     ...
```

5 Manual Decompilation – Exercise 5

Your goal is to analyse the following procedure and recover the C/C++ code or pseudo code that generated it.

```
proc near

arg_0           = dword ptr  8
arg_4           = dword ptr  0Ch
arg_8           = dword ptr  10h

     push    ebp
     mov     ebp, esp
     push    ebx
     push    esi
     mov     ecx, [ebp+arg_8]
     mov     esi, [ebp+arg_0]
     mov     eax, [ebp+arg_4]
     mov     edx, esi
     test    ecx, ecx
     jnz     short loc_40125A
     xor     eax, eax
     jmp     short loc_401265
loc_401254:
     mov     bl, [eax]
     inc     eax
     mov     [edx], bl
     inc     edx
loc_40125A:
     mov     ebx, ecx
     add     ecx, -1
     test    ebx, ebx
     jnz     short loc_401254
     mov     eax, esi
loc_401265:
     pop     esi
     pop     ebx
     pop     ebp
     retn
endp
```

You must retrieve the proper C/C++ code or pseudo code of of this commented procedure.

Your solution has to contain either a full commented C/C++ code or a detailed pseudo code describing the function of the above snippet.

6 Manual Decompilation – Exercise 6

Your goal is to analyse the following procedure and recover the C/C++ code or pseudo code that generated it.

```
proc near

V1            = dword ptr  8
V2            = dword ptr  0Ch
V3            = dword ptr  10h
V4            = dword ptr  14h

     push    ebp
     mov     ebp, esp
     push    esi
     push    edi
     mov     edi, [ebp+V4]
     mov     esi, [ebp+V3]
     mov     edx, [ebp+V2]
     mov     eax, [ebp+V1]
     test    edi, edi
     jnz     short loc_40122C
     xor     eax, eax
     jmp     short loc_401237
loc_401219:
     mov     ecx, esi
     cmp     cl, [edx]
     jz      short loc_401227
     mov     cl, [edx]
     mov     [eax], cl
     inc     edx
     inc     eax
     jmp     short loc_40122C
loc_401227:
     mov     [eax], cl
     inc     eax
     jmp     short loc_401237
loc_40122C:
     mov     ecx, edi
     add     edi, -1
     test    ecx, ecx
     jnz     short loc_401219
     xor     eax, eax
loc_401237:
     pop     edi
     pop     esi
     pop     ebp
     retn
endp
```

You must retrieve the proper C/C++ code or pseudo code of this commented procedure.

Your solution has to contain either a full commented C/C++ code or a detailed pseudo code describing the function of the above snippet.

7 Manual Decompilation – Exercise 7

The initial numbers provided on the left of the code snippet represent the relative value of your Stack pointer within the function. It can help you recognizing the (slightly) different usage of function's parameters (hint: 3 parameters are passed to this function).

```
proc near
000                 push    ebx
004                 push    esi
008                 xor     ebx, ebx
008                 mov     [eax], ebx
008                 mov     ebx, ecx
008                 dec     ebx
008                 test    ebx, ebx
008                 jl      short loc_408135
008                 inc     ebx
loc_40810E:
008                 mov     ecx, [eax]
008                 shl     ecx, 4
008                 movzx   esi, byte ptr [edx]
008                 add     ecx, esi
008                 mov     [eax], ecx
008                 mov     ecx, [eax]
008                 and     ecx, 0F0000000h
008                 test    ecx, ecx
008                 jz      short loc_40812D
008                 mov     esi, ecx
008                 shr     esi, 18h
008                 xor     [eax], esi
loc_40812D:
008                 not     ecx
008                 and     [eax], ecx
008                 inc     edx
008                 dec     ebx
008                 jnz     short loc_40810E
loc_408135:
008                 pop     esi
004                 pop     ebx
000                 retn
sub_408100      endp
```

You must retrieve the proper code or produce a pseudo code of this slightly commented procedure.

Your solution has to contain either a full commented High-Level Language code or a detailed pseudo code describing the function of the above snippet. You should also recognize the algorithm and name it accordingly. It pertains to the basics of cryptography field.

8 Manual Decompilation – Exercise 8

Your goal is to analyse the following procedure and recover the code or pseudo code that generated it. As prior exercise, you can see the relative stack pointer on the leftmost of each instruction.

```
sub_408138      proc near
000                 push    ebx
004                 push    esi
008                 mov     esi, edx
008                 dec     esi
008                 test    esi, esi
008                 jl      short loc_40816F
008                 inc     esi
loc_408142:
008                 xor     edx, edx
008                 mov     dl, [eax]
008                 xor     ebx, ebx
008                 mov     bl, cl
008                 add     edx, ebx
008                 test    edx, edx
008                 jge     short loc_40815B
008                 mov     ebx, 100h
008                 sub     ebx, edx
008                 mov     edx, ebx
008                 jmp     short loc_408169
loc_40815B:
008                 cmp     edx, 100h
008                 jle     short loc_408169
008                 sub     edx, 100h
loc_408169:
008                 mov     [eax], dl
008                 inc     eax
008                 dec     esi
008                 jnz     short loc_408142
loc_40816F:
008                 pop     esi
004                 pop     ebx
000                 retn
sub_408138      endp
```

You must retrieve the proper High-Level Language code or produce a pseudo code of this commented procedure.

Your solution has to contain either a full commented High-Level Language code or a detailed pseudo code describing the function of the above snippet. You should also recognize the algorithm and name it accordingly. It pertains to the basics of cryptography field.

9 Manual Decompilation – Live Code Analysis Exercise

You should reverse the following procedure extracted from a simulated protection system and analyse it. You should convert any meaningful number or constant by using either MSDN or values from Windows header files. The leftmost number represent the relative position of your stack pointer within the procedure.

```
procedure0        proc near
000                push    0
004                push    procedure1
008                call    EnumWindows
000                retn
procedure0        endp
------
procedure1        proc near

ClassName         = byte ptr -204h
String            = byte ptr -104h
hwnd                    = dword ptr -4
V1                = dword ptr  8

000                 push    ebp
004                 mov     ebp, esp
004                 add     esp, -204h
208                 push    ebx
20C                 push    esi
210                 push    edi
214                 mov     edi, [ebp+V1]
214                 push    100h
218                 lea     eax, [ebp+String]
218                 push    eax
21C                 push    edi
220                 call    GetWindowTextA

214                 mov     [ebp+eax+String], 0
214                 push    100h
218                 lea     eax, [ebp+ClassName]
218                 push    eax
21C                 push    edi
220                 call    GetClassNameA

214                 mov     [ebp+eax+ClassName], 0
214                 mov     esi, 3
214                 mov     ebx, offset Address1

Label1:
214                 lea     eax, [ebp+ClassName]
214                 call    tolower

214                 mov     edx, [ebx]
214                 call    Sysutils::StrPos(char *,char *)

214                 test    eax, eax
214                 jnz     short Label2

214                 lea     eax, [ebp+String]
214                 call    tolower

214                 mov     edx, [ebx]
214                 call    Sysutils::StrPos(char *,char *)

214                 test    eax, eax
214                 jz      short Label3
```

```
Label2:
214                 push    0
218                 push    offset aSyslistview32
21C                 push    0
220                 push    edi
224                 call    FindWindowExA

214                 mov     [ebp+hwnd], eax
214                 push    0
218                 push    0
21C                 push    1009h
220                 mov     eax, [ebp+hwnd]
220                 push    eax
224                 call    SendMessageA

214                 push    0
218                 push    0
21C                 push    0Fh
220                 mov     eax, [ebp+hwnd]
220                 push    eax
224                 call    SendMessageA

214                 push    0
218                 push    0
21C                 push    02h
220                 push    edi
224                 call    SendMessageA

214                 push    0
218                 push    0
21C                 push    10h
220                 push    edi
224                 call    SendMessageA

Label3:
214                 add     ebx, 4
214                 dec     esi
214                 jnz     short Label1

214                 mov     al, 1
214                 pop     edi
210                 pop     esi
20C                 pop     ebx
208                 mov     esp, ebp
004                 pop     ebp
000                 retn    8

procedure1        endp

aSyslistview32  db 'SysListView32'
Address1:
dd offset aRegmon       ; "REGMON"
dd offset aFilemon      ; "FILEMON"
dd offset aRegmonex     ; "REGMONEX"
```

Your solution has to contain either a full commented High-Level Language code or a detailed pseudo code describing the above function. You must provide a detailed explanation of what this procedure does and how it does that, as well as its usage within a copy protection system.