تعلم اكتشاف ثغرات البرامج و كتابة الإستثمارات

::: فيض مخزن المكدس :::

Stack Buffer Overflow

اعداد: Z3r0n3

الفريق العربي للبرمجة©

مقدمة:

تحدث ثغرة فيض مخزن المكدس (Stack Buffer Overflow) عندما يقوم البرنامج باستدعاء دالة تنقل بيانات إلى مخزن محجوز في المكدس و لا تقوم بفحص حجم البيانات المنقولة و المساحة المخصصة لاستقبالها. لذلك يوجد احتمال كبير لكتابة جزء من البيانات التي يتم نقلها في الذاكرة المجاورة للمخزن و إتلاف محتواها الذي يمكن أن يكون مهم في سير تنفيذ البرنامج. يمكن أن تتسبب هذه الثغرة في تغيير مجرى تنفيذ البرنامج و بذلك يمكن أن ينتج عنها أخطاء الوصول إلى الذاكرة (Memory Access Violation)، نتائج خاطئة أو انهيار البرنامج. يمكن استثمار هذه الثغرة لتغيير سلوك البرنامج المصاب و حقن ثم تنفيذ أكواد خبيثة (Malicious Cocde) تخدم مصالح مستثمر الثغرة.

1. المكدس و استدعاء الدوال | Stack and Calling Functions

توضع البيانات في المكدس باستعمال التعليمة PUSH و تسحب بالتعليمة POP. عند "دفع" قيمة إلى المكدس، يقوم المعالج بإنقاص عدد البايتات التي سيتم دفعها من مسجل مؤشر المكدس ESP ثم كتابة القيمة في العنوان الجديد الذي يشير إليه ESP. و بذلك يصبح المسجل ESP يشير إلى عنوان البيانات التي تمّ نقلها إلى المكدس. عند "سحب" قيمة من المكدس، ينقل المعالج القيمة الموجودة في قمة المكدس، ثم يضيف عدد البايتات المسحوبة الى المسجل ESP. لكي يتم استغلال هذه المساحة في تخزين قيم أخرى. تسمى هذه الخاصية بـ Last In First Out (LIFO) يعني آخر قيمة مدفوعة أول قيمة تسحب.

1. 1. استدعاء الدوال باستعمال التعليمتين CALL و RET

عند استدعاء دالة باستعمال التعليمة CALL، يقوم المعالج بالخطوات التالية:

1. دفع القيمة الحالية لمسجل مؤشر التعليمة (EIP) إلى المكدس
2. تحميل عنوان الدالة إلى المسجل EIP
3. تنفيذ التعليمات التي يشير إليها المسجل EIP

عند استعمال التعليمة RET للعودة من دالة، يقوم المعالج بالخطوات التالية:

1. سحب القيمة الموجودة في قمة المكدس (عادتا ما تكون عنوان الرجوع من الدالة) و وضعها في المسجل EIP
2. إذا كانت التعليمة RET تحتوي على المعامل الاختياري n (تكتب هكذا RET n)، يقوم المعالج بإضافة القيمة n إلى المسجل ESP ليتم تحرير n بايت من ذاكرة المكدس. عادتا ما تكون القيمة n هي حجم البارامترات الممررة إلى الدالة.
3. نقل التنفيذ إلى العنوان الموجود في المسجل EIP (عادتا ما تكون التعليمة التي تلي تعليمة استدعاء الدالة)

ملاحظة: هذه الخطوات متعلقة باستدعاء دالة قريبة (NEAR CALLS)

1. 2. إطار المكدس | Stack Frame

يقسّم المكدس إلى أطر، يمكن لكل إطار أن يحتوي على متغيرات محلية، بارامترات يتم تمريرها إلى دالة أخرى و قيم أخرى. عند استدعاء دالة، و قبل دفع أي قيمة إلى المكدس، عادتا ما تقوم الدالة بنسخ محتوى مؤشر المكدس ESP (الذي يشير إلى إطار مكدس الدالة المستدعية) إلى المسجل EBP و ذلك لتسهيل الوصول إلى البارامترات الممررة إلى الدالة و عنوان الرجوع.

مثال:

```
void MyProc()
  {
      ...
  }
```

أغلب المترجمات تقوم بتوليد الكود التالي في بداية كل دالة:

```
MyProc:
      push    ebp
      mov     ebp, esp
...
```

1. 2. تمرير البارامترات باستعمال المكدس | Passing Parameters Using The Stack

يمكن استعمال المكدس لتمرير بارامترات الدوال من خلال و ضعها في إطار المكدس التابع للدالة المستدعية. و في هذه الحالة يمكن الوصول إلى البارامترات الممررة عبر المكدس باستعمال المسجل EBP الذي هو بدوره يشير إلى إطار مكدس الدالة المستدعية.

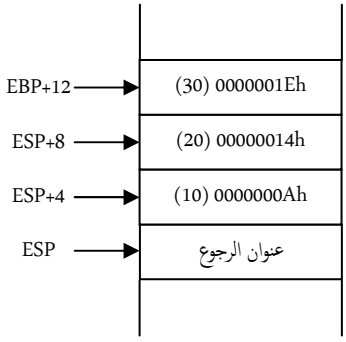
مثال:

```
void MyProc(int a, int b, int c)
{
    ...
}
int main()
{
    ...
    MyProc(10,20,30);
    ...
}
```

يكون استدعاء الدالة MyProc كالتالي:

```
push    30
push    20
push    10
call    MyProc
```

شكل المكدس بعد استدعاء الدالة:



1. 3. المتغيرات المحلية | Local Variables

تستعمل الدوال المكدس لتخزين قيم المتغيرات المحلية و يمكن حجز مساحة خاصة بها داخل المكدس عند بداية كل دالة من خلال طرح حجم كل المتغيرات من المسجل ESP.

ملاحظة: لكي لا يحصل تداخل بين المتغيرات، يجب أن يكون حجم المساحة التي نريد حجزها من مضعفات 4.

```
sub     esp, SizeOfLocalVariables
```

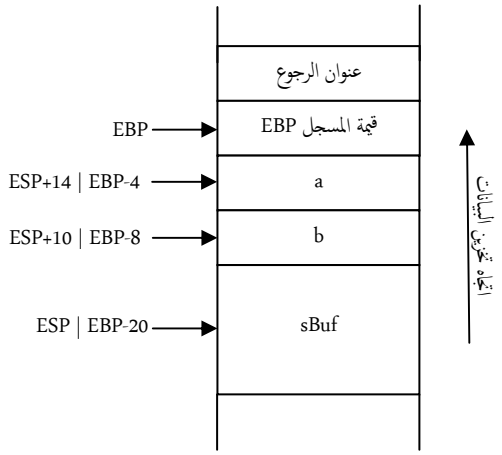
مثال:

```
void MyProc()
  {
      int a, b          //  int = 4 bytes
      char sBuf[10]     //  char = 1 byte
      ...
  }
```

سيكون مقطع كود دخول الدالة كالتالي:

```
MyProc:
      push    ebp
      mov     ebp, esp
      sub     esp, 20         ; SizeOf(a)+SizeOf(b)+SizeOf(sBuf)+2=20
      ...
```

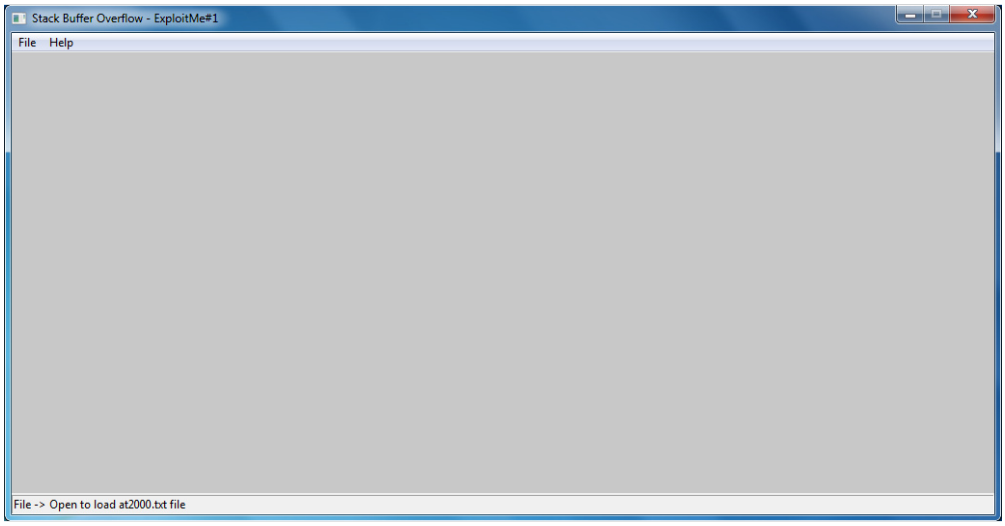
شكل المكدس:



يمكن الوصول إلى المتغيرات المحلية باستعمال مؤشر المكدس ESP الذي يشير إلى آخر متغير محلي تم حجزه أو المسجل EBP الذي يمكن استعماله ليشير إلى أول متغير محلي (EBP-4).

2. تحليل ثغرة فيض مخزن المكدس | Stack Buffer Overflow Analysis

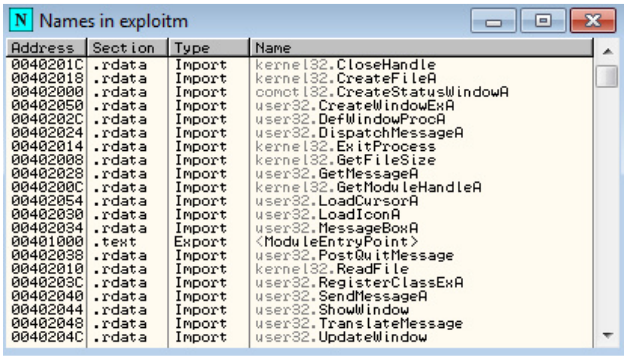
أفضل طريقة لتحليل و فهم هذه الثغرة هو التطبيق على مثال عملي. قمت بكتابة برنامج "exploitme" –تجده مرفق مع الدرس- بلغة الأسمبلي لهذا الغرض.



1-واجهة البرنامج exploitme

عند اختيار الأمر Open من القائمة المنسدلة File يقوم البرنامج بفتح الملف "at2000.txt" المتواجد في نفس مسار البرنامج و تخزين محتواه في مخزن محجوز في المكدس ثم إظهاره في رسالة (MessageBox). في حالة عدم وجود الملف يقوم بإظهار رسالة تفيد أن الملف غير متواجد.

نفتح البرنامج بالمنقح OllyDbg لتتبعه و معرفة مكان الثغرة. يمكننا عمل ربط (Attach) مع البرنامج اذا كان يشتغل أو فتح البرنامج (File -> Open) لتحميله إلى الذاكرة. من لوحة المفاتيح نضغط على CTRL+N لإظهار أسماء الدوال التي يستعملها البرنامج.

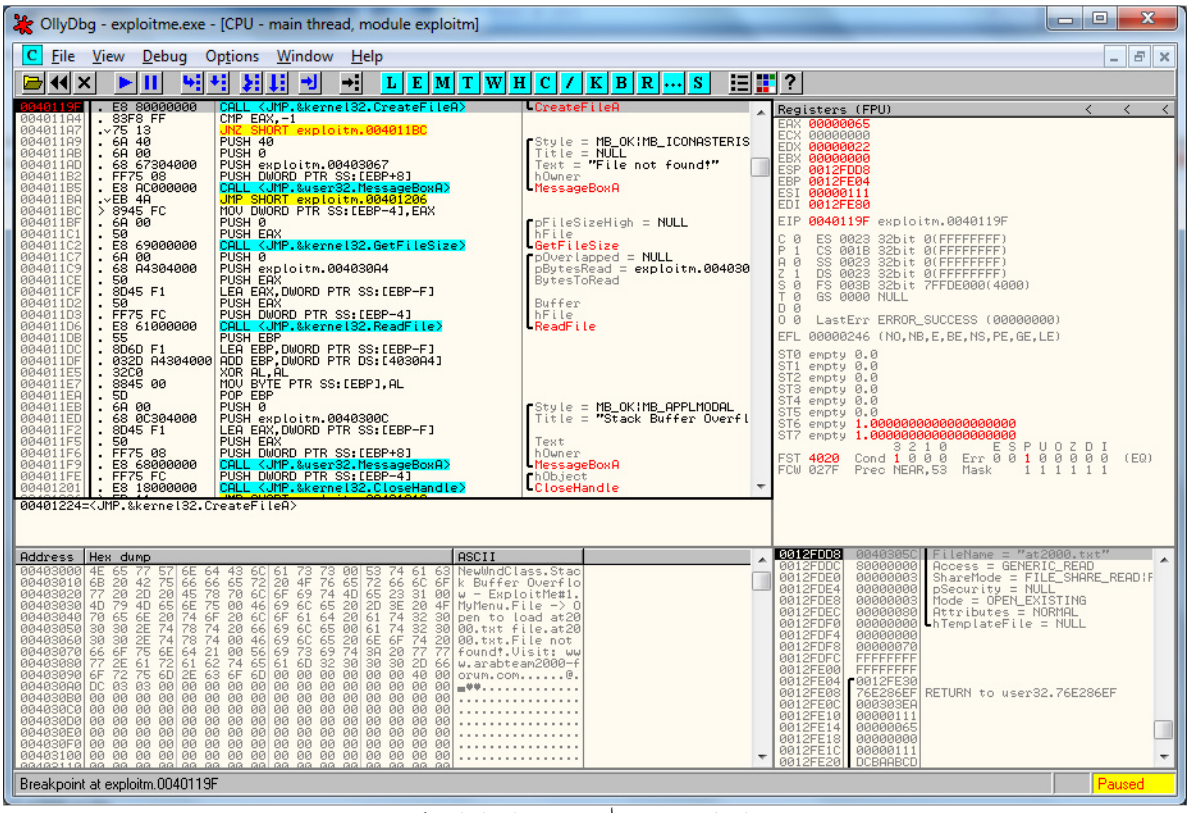


2-دوال استيراد البرنامج exploitme

بما أن الملف "at2000.txt" يعتبر المدخل الوحيد من المستعمل إلى البرنامج فإنّ الدوال المتعلقة بالتعامل مع الملفات هي التي ستساعدنا في البحث عن الثغرة:

- **CreateFile**: لفتح أو إنشاء ملف
- **GetFileSize**: لجلب حجم ملف
- **ReadFile**: لقراءة بايتات معينة من ملف و نقلها إلى مخزن
- **CloseHandle**: لإغلاق ملف

للتعامل مع أي ملف يجب أولا على البرنامج فتحه و الحصول على مقبضه و يتم هذا باستدعاء الدالة **CreateFile**. نضع نقطة توقف على هذه الدالة (الفتاح F2) ثم نشغّل البرنامج (المفتاح F9). نعود إلى البرنامج exploitme و نختار الأمر Open من القائمة File.



3-توقف تنفيذ البرنامج عند تعليمة استدعاء الدالة **CreateFile**

نكمل تنفيذ البرنامج خطوة بخطوة باستعمال المفتاح F8. سيقوم البرنامج من التثبت من وجود الملف "at2000.txt"، إذا كان غير موجود يظهر الرسالة "File not found !". ما عدا ذلك يقفز إلى العنوان 4011BCh لتنفيذ الدالة **GetFileSize** التي تجلب حجم الملف "at2000.txt". ترجع هذه الدالة عدد بايتات الملف في المسجل EAX. بعدها يكمل التنفيذ إلى الدالة **ReadFile** للقراءة من الملف. نموذج الدالة **ReadFile**:

```
ReadFile PROTO hFile:DWORD,\
             lpBuffer:DWORD,\
             nNumberOfBytesToRead:DWORD,\
             lpNumberOfBytesRead:DWORD,\
             lpOverlapped:DWORD
```

- hFile: مقبض الملف
- lpBuffer: مؤشر إلى المخزن
- nNumberOfBytesToRead: عدد البايتات التي نريد قراءتها من الملف
- lpNumberOfBytesRead: مؤشر لـDWORD تضع فيه الدالة عدد البايتات التي تمّ قراءتها
- lpOverlapped: غير مهمة بالنسبة لنا

يمكن للدالة **ReadFile** أن تحدث ثغرة Stack Buffer Overflow إذا توفرت الشروط التالية:

- غياب كود يفحص عدد البايتات القراءة من الملف (nNumberOfBytesToRead) و حجم المخزن المستقبل للبيانات
- المخزن محجوز في ذاكرة المكدس

نكمل تتبع البرنامج لنرى إن كانت الشروط متوفرة. استدعاء الدالة **ReadFile** في البرنامج:

```
004011A9 | . 6A 00          PUSH 0                          pOverlapped = NULL
004011AB | . 68 80304000    PUSH exploitm.00403080          pBytesRead = exploitm.00403080
004011B0 | . 50             PUSH EAX                        BytesToRead
004011B1 | . 8D45 F1        LEA EAX,DWORD PTR SS:[EBP-F]
004011B4 | . 50             PUSH EAX                        Buffer
004011B5 | . FF75 FC        PUSH DWORD PTR SS:[EBP-4]       hFile
004011B8 | . E8 61000000    CALL <JMP.&kernel32.ReadFile>   ReadFile
```

4-استدعاء الدالة **ReadFile**

لاحظ أن البرنامج قام بتمرير القيمة المرجعة من الدالة **GetFileSize** كعدد بايتات يريد قراءتها من الملف بدون التثبت منها

```
004011B0  |. 50             PUSH EAX                        ; |BytesToRead
```

ثم قام بجلب مؤشر لمخزن موجود في المكدس و تمريره كمعامل lpBuffer:

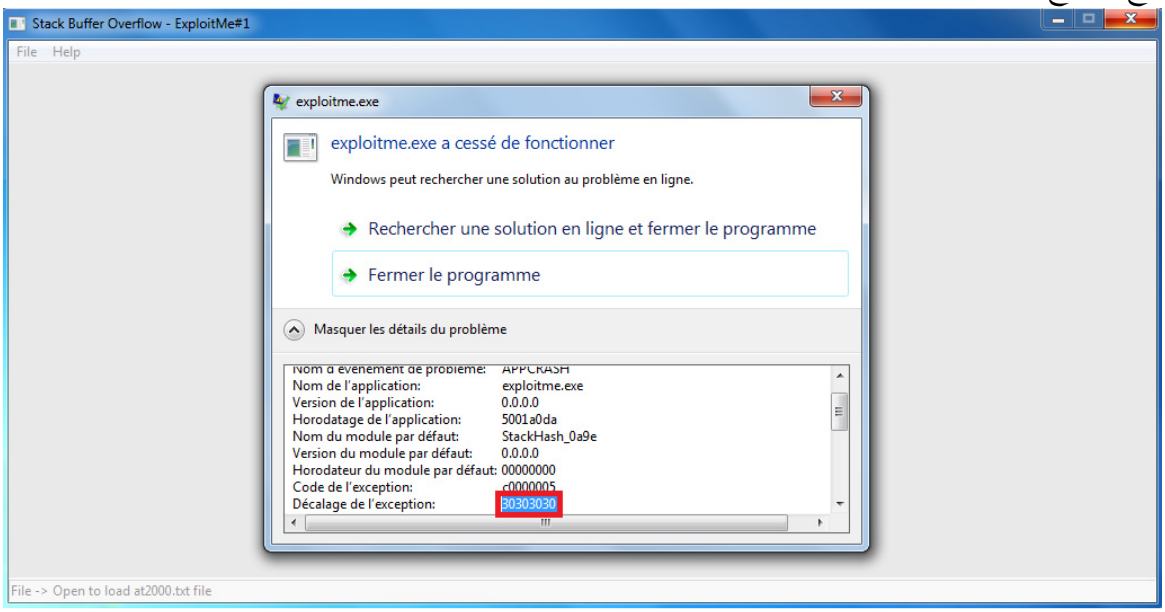
```
004011B1  |. 8D45 F1        LEA EAX,DWORD PTR SS:[EBP-F]    ; |Get Stack Buffer pointer
004011B4  |. 50             PUSH EAX                        ; |Buffer
```

إذن تحققت شروط وقوع الثغرة، و للتثبت من ذلك نفتح الملف "at2000.txt" و نضع فيه النص التالي:

00000000000000000000000000000000000000000000000000000000000000000000

ملاحظة: إذا لم يحصل خطأ نفتح الملف مجددا و نظيف إليه نصوص

نفتح البرنامج "exploitme" و نختار File -> Open و نرى إن حصل خطأ:



5-حدوث خطأ بسبب القراءة من الملف at2000.txt

لاحظ حدوث استثناء c0000005h (EXCEPTION_ACCESS_VIOLATION) عند محاولة البرنامج إكمال التنفيذ في العنوان 30303030h –النص "0000". بمعنى أن البرنامج تجاوز مساحة الـBuffer و أكمل نقل البيانات إلى الذاكرة المجاورة التي تحتوي على عنوان الرجوع و تم الكتابة عليه.

الآن نريد معرفة إزاحة القيمة التي ستكتب على عنوان الرجوع. نضع نقطة توقف بعد تعليمة جلب مؤشر المخزن:

```
004011B1  |. 8D45 F1        LEA EAX,DWORD PTR SS:[EBP-F]        ; |Get Stack Buffer pointer
004011B4  |. 50             PUSH EAX                            ; | نضع نقطة توقف هنا
```

نضغط F9 لتنفيذ البرنامج ثم نختار File -> Open لفتح الملف "at2000.txt" عندها سيتوقف البرنامج و سيحتوي المسجل EAX على عنوان المخزن و هو 12FDF5h.

ملاحظة: يمكن أن يحصل اختلاف في العناوين من جهاز إلى آخر و من نظام إلى آخر المهم أن تكمل التطبيق على العناوين التي وجدتها

ثم نكمل تتبع البرنامج خطوة بخطوة حتى الوصول إلى التعليمة:

```
RETN 10
```

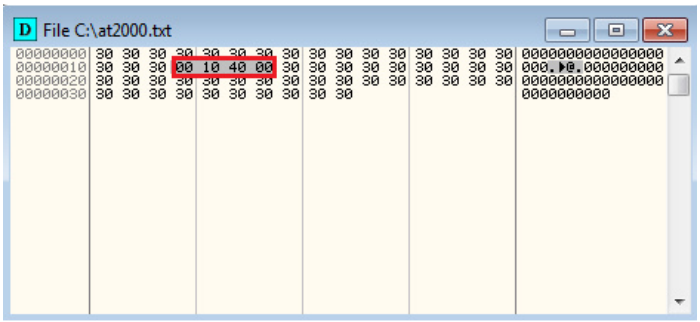
عندها سيشير المسجل ESP على القيمة التي كتبت على عنوان الرجوع من الدالة عند حصول الفيض وهي القيمة 12FE08. الآن لحساب إزاحة العنوان في الملف نقوم بطرح عنوان الرجوع من عنوان المخزن زائد 1. و في مثالنا:

```
12FDF5h-12FE08+1=14h (20)
```

بمعنى أنّ البايت رقم 20 في الملف هو بداية العنوان. الآن لتطبيق المعلومات التي قمنا باستخراجها نريد تغيير عنوان الرجوع إلى عنوان نقطة إدخال البرنامج و في حالتنا العنوان 00401000h. إذن نفتح الملف "at2000.txt" بأي محرر هكس مثل المحرر الموجود في المنقّح OllyDbg (File -> View) و نكتب العنوان من اليمين إلى اليسار ابتدءا من الإزاحة 20 في الملف.

```
[FileBase+20] <-- 00h
[FileBase+21] <-- 10h
[FileBase+22] <-- 40h
[FileBase+23] <-- 00h
```

بعدها سيكون محتوى الملف كالتالي:



-6-كتابة عنوان الرجوع في الملف "at2000.txt"

بعد تسجيل التغيرات نفتح البرنامج exploitme و نختار File -> Open لفتح الملف "at2000.txt"، إذا قام البرنامج بإظهار نافذة جديدة له يعني أننا قد نجحنا في الكتابة على عنوان الرجوع و توجيه تنفيذ البرنامج إلى نقطة الإدخال.

3. استثمار ثغرة فيض مخزن المكدس

3. 1. الشل كود | ShellCode

يمكن استثمار هذه الثغرة من خلال كتابة ShellCode وهو قطعة كود مكتوبة بلغة الآلة يستعمل غالبا في تحميل برنامج استهداف جهاز الضحية. يمكن كتابة الشل كود بلغة الأسمبلي ثم نسخ تعليمات لغة الآلة و وضعها في ملف الشل (at2000.txt في

مثالنا السابق). أهم قاعدة في كتابة الشل كود هي أن نعتبر أن كل العناوين ليست ثابتة و علينا جلبها ديناميكيا. كذلك وضع الكود و البيانات في نفس القسم.

مثال:برنامج يستعمل معامل يجلب عناوين ثابتة "offset"

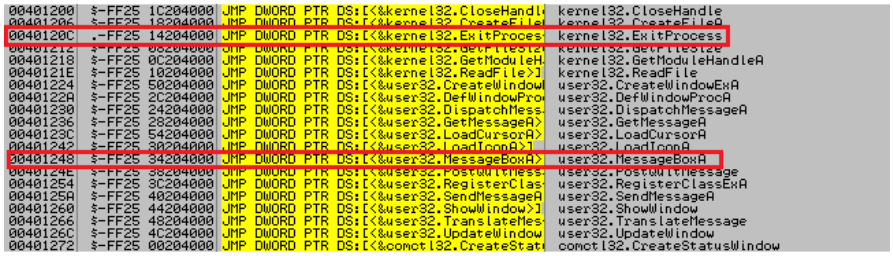
```
;+-+-+-+-+[ FileName: msgbox.asm ]+-+-+-+-+-+

.386
.model flat, stdcall
option casemap:none

;+-+-+-+-+   Include Files  +-+-+-+-+-+
          include \masm32\include\windows.inc
          include \masm32\include\kernel32.inc
          include \masm32\include\user32.inc

;+-+-+-+-+   Include Libraries +-+-+-+-+
          includelib \masm32\lib\kernel32.lib
          includelib \masm32\lib\user32.lib

.code
start:
          push MB_OK
          push offset caption
          push offset msg
          push 0
          call MessageBox

          push 0
          call ExitProcess

          caption db "ShellCode",0
          msg     db "'Offset' operator returns static address",0
end start
```

لو قمنا بنسخ قسم الكود و حقنه في برنامج آخر ثم تنفيذه لن يشتغل لأنّ العناوين ثابتة لا يمكن نقلها من برنامج لآخر.
لحل هذه المشكلة، نستعمل التعليمة call التي تقوم بدفع عنوان الرجوع أثناء وقت التشغيل (Run-Time) ثم القفز إلى العنوان المحدد. عند هذا العنوان نستعمل التعليمة POP لسحب العنوان الذي تمّ دفعه في المكدس.
نموذج استعمال هذه التقنية:

```
start:
;|---------------=[ shell code section ]=---------------=|
      ...
      jmp _VarName    ; jmp to get VarName dynamic address
_rVarName:            ; now [ESP] contain the dynamic address
      pop REG32       ; pop VarName address in 32-Register (EAX, EBX,...)
      ...

;|---------------=[ shell data section ]=---------------=|
      ...
_VarName:
      call _rVarName
      VarName VarType Value
      ...
```

3. 2. التطبيق على البرنامج "exploitme"

ما سنقوم به هو كتابة شل كود يظهر رسالة (MessageBox) ثم يغلق البرنامج. بعدها نكتب الشل كود في الملف "at2000.txt" و أخيرا نغيّر القيمة التي ستكتب على عنوان الرجوع إلى عنوان الشل كود داخل المكدس.
أولا نجلب عناوين الدوال المستعملة في الشل كود من جدول استيراد البرنامج المستهدف.



7-جدول استيراد البرنامج exploitme

نحدد الدالة التي نريد جلب عنوانها ثم نضغط المفتاح Space لإظهار تعليمة القفز إلى تلك الدالة. سيكون شكل التعليمة كالتالي:

```
JMP DWORD PTR DS:[Address]
```

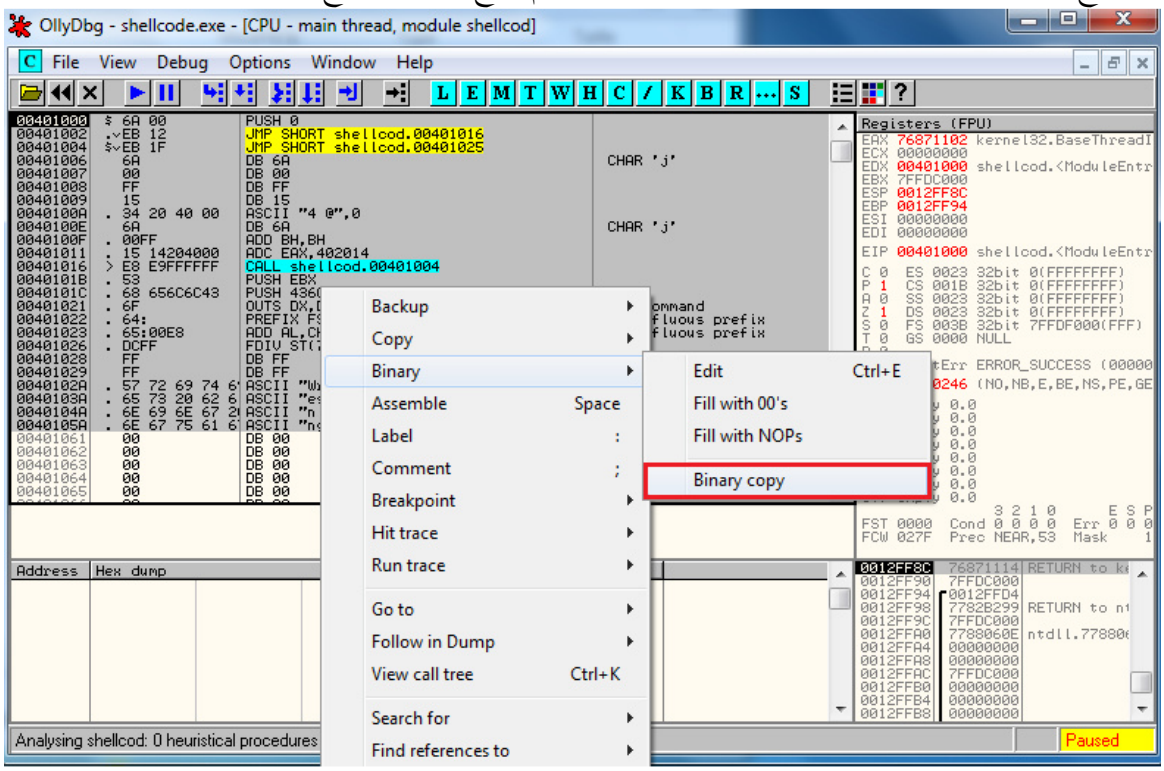
نحتفظ بالعنوان Address الذي يشير إلى عنوان الدالة:

- **MessageBox**: 402034h (دالة إظهار رسالة)
- **ExitProcess**: 402014h (دالة غلق البرنامج)

الآن نأخذ مقطع كود الملف "msgbox.asm" و نطبق عليه تقنية جلب العناوين و تغيير عناوين الدوال ثم نقوم بتجميعه باستعمال MASM.

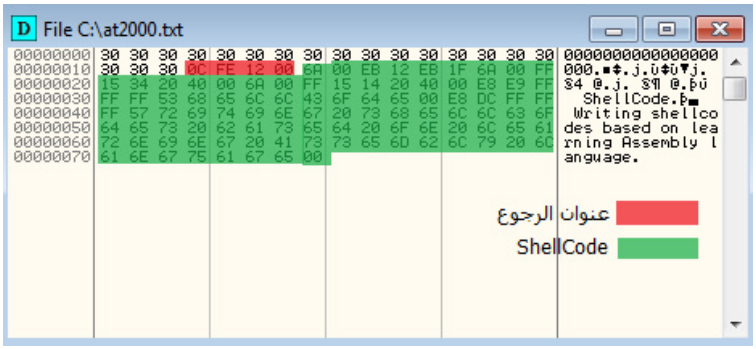
```
;+-+-+-+-+[ FileName: shellcode.asm ]+-+-+-+-+-+

.386
.model flat, stdcall
option casemap:none

.code
start:
;|=---------------=[ shell code section ]=---------------=|
        push 0          ; OK button
        jmp _caption
_rCaption:
        jmp _msg
_rMsg:
        push 0          ; hWnd
        call DWORD PTR DS:[402034h] ; <=> call MessageBox

        push 0
        call DWORD PTR DS:[402014h] ; <=> call ExitProcess

;|=---------------=[ shell data section ]=---------------=|
_caption:
        call _rCaption              ; <=> push offset caption
        caption db "ShellCode",0
_msg:
        call _rMsg                  ; <=> push offset msg
        msg     db "Writing shellcodes based on learning Assembly language",0
end start
```

بعد تجميع الملف "shellcode.asm" نفتحه داخل OllyDbg و نقوم بنسخ تعليمات البرنامج



-8-نسخ محتوى البرنامج shellcode

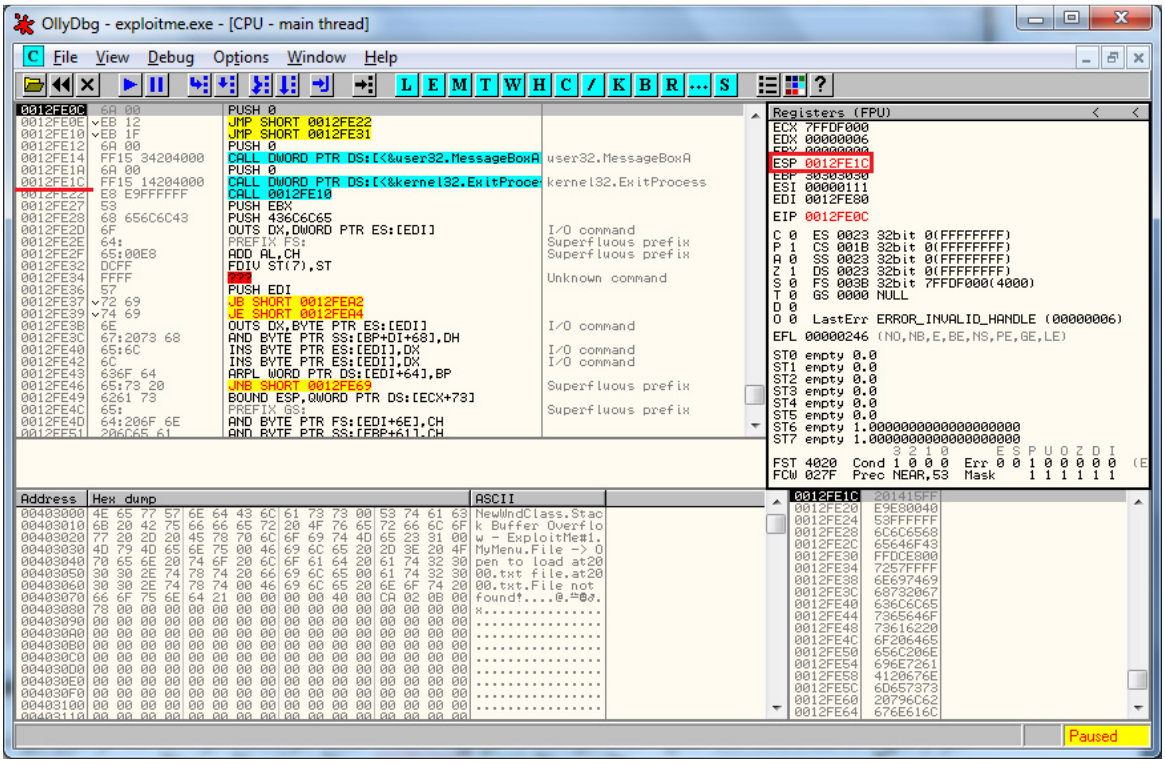
الآن نفتح الملف "at2000.txt" داخل أي محرر هكس و نقوم بلصق محتوى البرنامج shellcode مباشرة بعد عنوان الرجوع ثم نغيّر القيمة التي ستكتب على عنوان الرجوع إلى عنوان الشل كود داخل المكدس. يمكن حساب هذا الأخير من خلال إضافة إزاحة الشل الكود داخل الملف "at2000.txt" (17h) إلى عنوان المخزن (12FDF5h) الذي سيستقبل محتوى الملف "at2000.txt". يعني أن عنوان الشل كود داخل المكدس يساوي 12FE0Ch



-9-كتابة الشل كود و تغيير عنوان البرجوع

الآن كل شيء جاهز للتجربة، نفتح البرنامج exploitme و نختار الأمر Open من القائمة File لنرى ماذا سيحدث.

نعم لم تنتهي الأخطاء بعد، ماذا نفعل؟ نفتح البرنامج داخل OllyDbg و نقوم بتتبع البرنامج لحد الوصول إلى تنفيذ الشل.



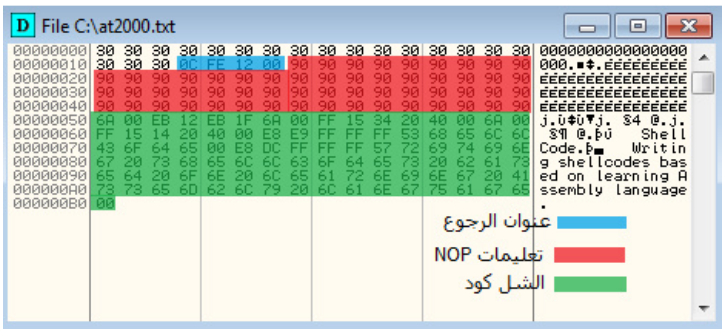
-10-قيمة المسجل ESP قبل تنفيذ أول تعليمة للشل كود

لاحظ أن المسجل ESP يحتوي على القيمة 12FE1Ch و التي هي نفسها عنوان التعليمة:

```
0012FE1C   FF15 14204000 CALL DWORD PTR DS:[<&kernel32.ExitProces>]; kernel32.ExitProcess
```

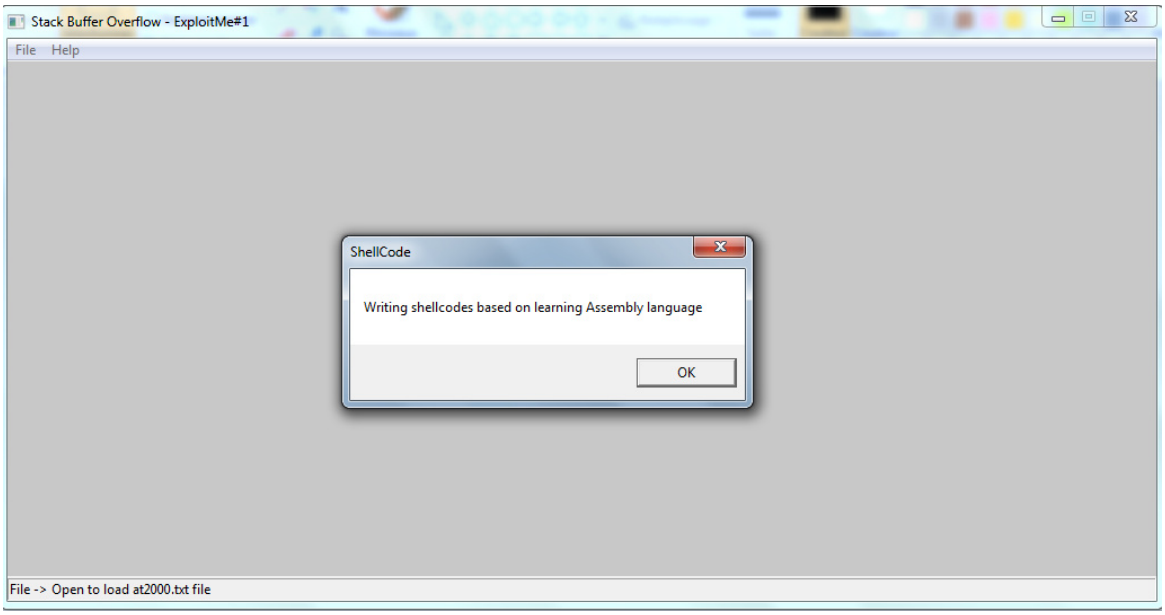
بمعنى أن أول قيمة ستدفع إلى المكدس ستكتب على تعليمة استدعاء الدالة **ExitProcess** و باقي القيم التي سيتم دفعها ستكتب على العناوين التي تسبق هذه التعليمة بمعنى أنّ تعليمة استدعاء الدالة **MessageBox** أيضا سيتم الكتابة عليها. بذلك سيتلف الشل الكود و يصبح غير قابل للتنفيذ لذلك ظهرت لنا رسالة الخطأ.

هنا يأتي دور استعمال تقنية NOP Sled و هي وضع تعليمات NOP (90h) في بداية الشل كود لإبعاده من مجال تخزين البيانات التي يقوم بدفعها إلى المكدس في نفس الوقت يتم تنفيذ هذه التعليمات التي لا تقوم بأي عملية سواء الانزلاق للوصول إلى تعليمات الشل الكود.



-11-الشكل النهائي للملف "at2000.txt"

نسجّل التغيرات ثم نشغّل البرنامج:



-12-نجاح استغلال الثغرة