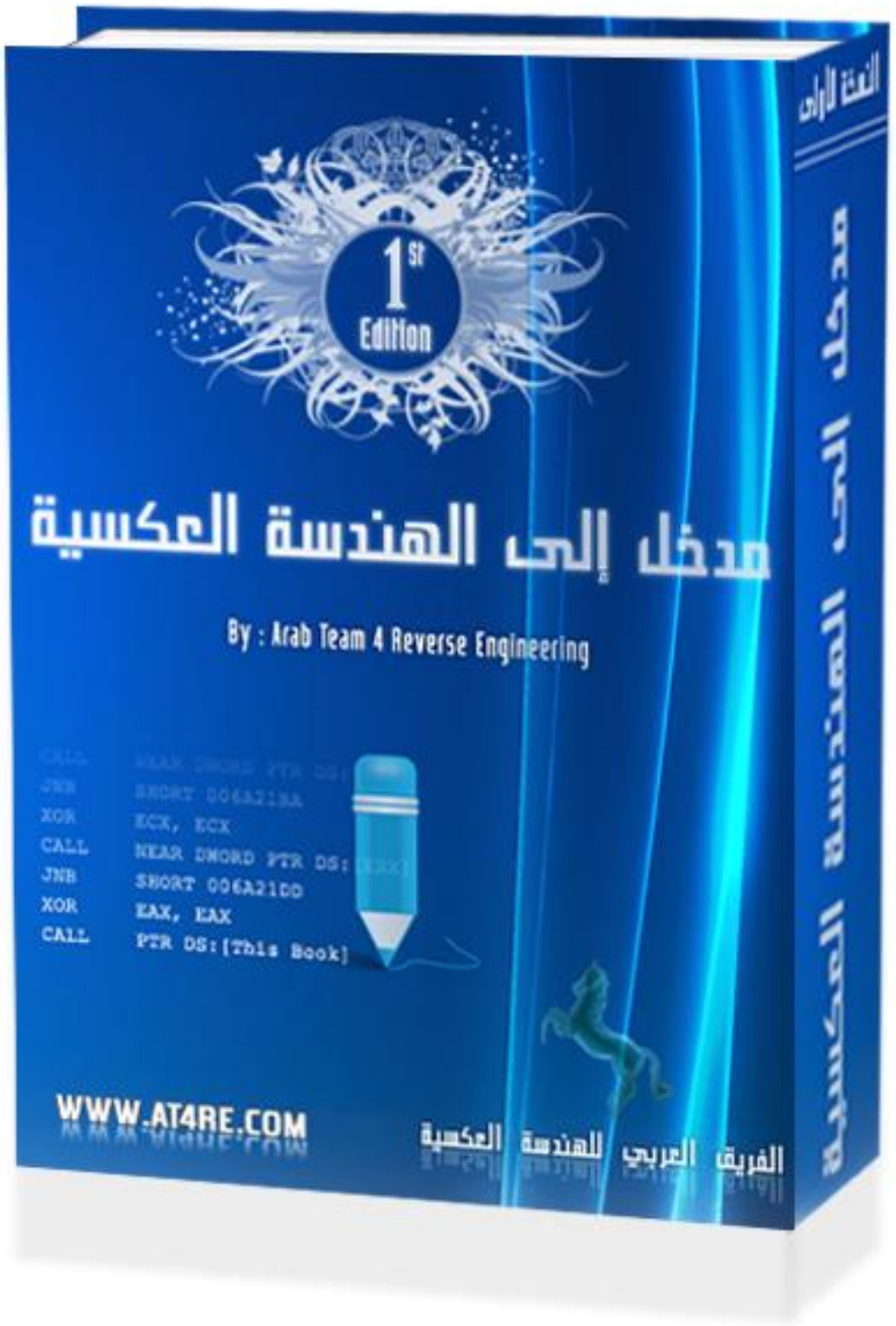

1st Edition
مدخل إلى الهندسة العكسية
By : Arab Team 4 Reverse Engineering
XOR ECX, ECX
CALL NEAR DWORD PTR DS:
JNB SHORT 006A21DD
XOR EAX, EAX
CALL PTR DS:[This Book]
WWW.AT4RE.COM
الفريق العربي للهندسة العكسية

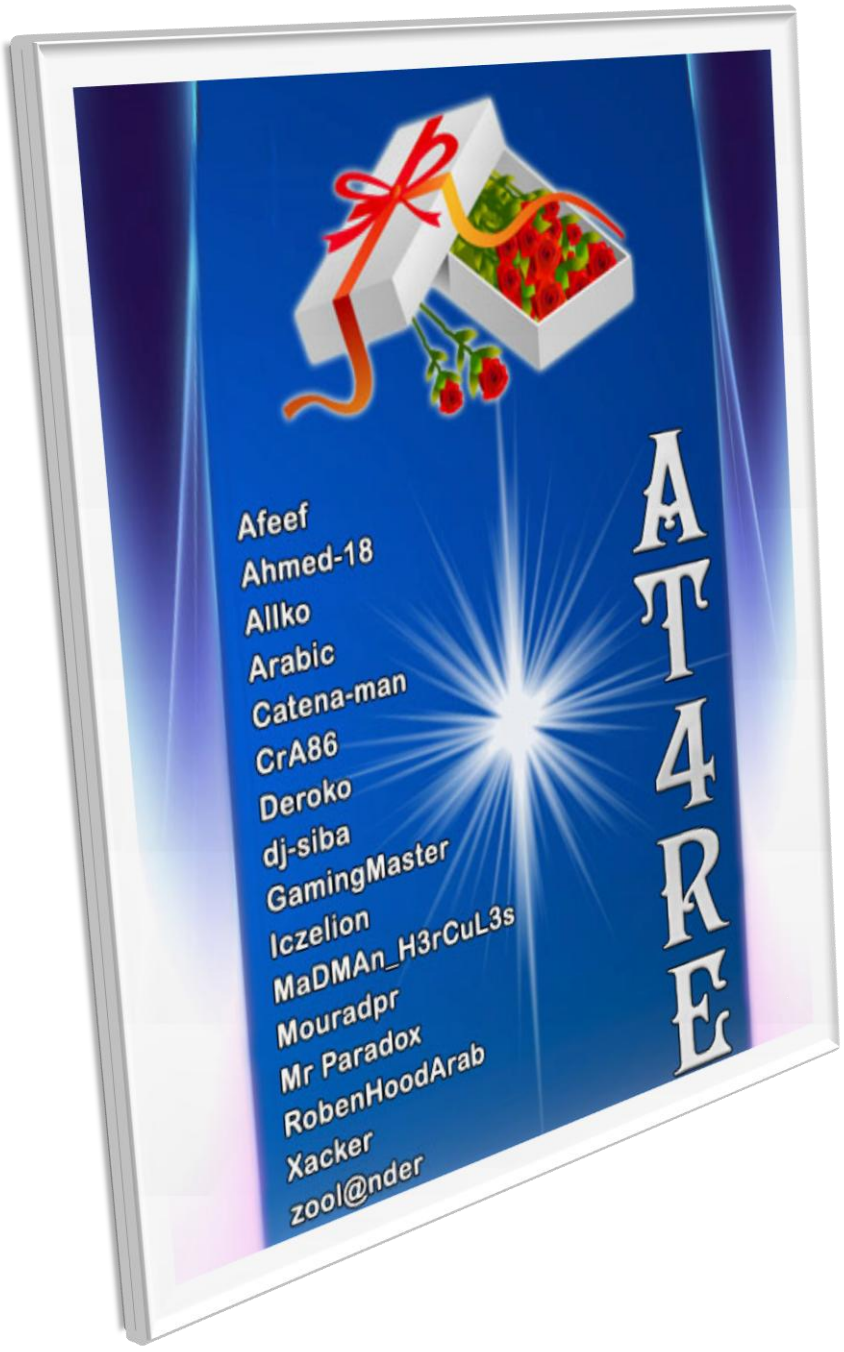

الاهـــــداء

إلى كل مسلم يهمه خدمة دينه.


شكر و تقدير

لا يسعنا سوى أن نتقدم بعظيم الشكر والامتنان للأشخاص والمواقع الواردة أسماؤهم في الأسفل، على المجهود الذي بذلوه في الوصول إلى هذا الكتاب. بعضهم ساهم بصورة مباشرة، والبعض بصورة غير مباشرة.

كنا نود ترتيب القائمة التالية حسب حجم الإسهام الذي ساهمه كل شخص في خدمة الكتاب – سواء كان ذلك بصورة مباشرة أو غير مباشرة – إلا أننا وجدنا أنه ليس من السهل المقارنة بين حجم إسهام كل شخص. لذا قررنا ترتيب القائمة أبجديا.




حقوق النسخ


كيف تستعمل هذا الكتاب

لتستعمل هذا الكتاب على احسن وجه, و تنهل منه اقصى ما يمنكن, اليك بعض التوجيهات و النصائح :

1) بعد تحميل الكتاب, حمل الملفات اللازمة التي تم عمل الدروس بالنقر على اسماءها من سيرفر منتدانا, و لا تحاول تحميلها من مواقعها الأصلية حفاظا على اكبر قدر من التوافق. فأغلب الأهداف يغيرها مبرمجوها مرات عديدة, لذا لتضمن مجاراة و فهما دقيقين لدروس هذه النسخة من الكتاب, ننصحك بتحميل المرفقات اللازمة من الروابط المخصصة في حينها.
2) اذا لم يعمل معك احد الروابط, او لم تفهم جيدا احد الأقسام, او اردت ان تسأل عن أي شئ **بخصوص الكتاب**, فنرجو منك فقط شيئين و نحن في خدمتك بما قدَّرنل عليه ربنا عز و جل:
   a. ان **تكتب مداخلتك أو سؤالك في المشاركة المخصصة** لتتبع الكتاب على الرابط التالي هنـــــــــا.
   b. ان **تختار لمشاركتك عنوانا يشير الى محتواها و نوع المشكل فيها بالضبط** , فالمشاركة من قبيل" ساعدوني" , قلَّما تثير اهتمام المشرفين.

3) استعملنا في هذا الكتاب صورا رمزية لإثارة الانتباه الى الاشياء المهمة و تسهيل التعرف على المقاطع و المرفقات و اصطلحنا على ما يلي:

	← صورة رمزية للدلالة على مرفق يجب تحميله من النت.
	← صورة رمزية للدلالة على ملحوظة, معلومة قيمة, معلومة مساعدة ...
	← صورة رمزية للدلالة على فقرة جامعة, خلاصة مركزة ...
	← صورة رمزية للدلالة على وجوب الانتباه الى معلومة او الحذر من خطأ محتمل ...
	← خطأ قاتل


جدول المحتويات


الباب الأول
الفصل الأول : مقدمه
الفصل الثاني : مدخل إلى لغة الأسمبلي
الفصل الثالث : لمحة سريعة عن برامج RE


الباب
الأول
الفصل الأول
مقدمه

1. الفصل الأول : مقدمة

قد لا يكون أغلب مستخدمي الحاسوب سمعوا بالهندسة العكسية من قبل، ربما لأنها غير شائعة كثيرا بعكس ما هو الحال مع عالم الاختراق "الهاكينج". أو ربما لأنهم يعرفونها بمصطلحها الآخر ألا وهو "الكراكينج". ويختلف الكثيرون في تعريف الهندسة العكسية، لكن باختصار يمكننا القول أن الهندسة العكسية هي "عملية تحليل شيء ما لفهم آلية عمله".هي العمل العكسي لما قام به مهندس في مجال معين، وبالتالي تقسم الهندسة العكسية إلى : هندسة عكسية للبرمجيات, او ما يطلق عليه:(Reverse Code Engineering (RCE و هندسة عكسية للهاردوير: Hardware Reverse Engineering ه نا نحن بصدد التحدث عن البرمجيات فقط، أي ال RCE.

لكن ما الحاجة إلى الهندسة العكسية أصلا ؟ يختلف هذا باختلاف المهندس العكسي .. فإذا كنت أنت كاتب البرنامج، فغالبا ستود تنقيح برنامجك (Debugging) بغرض اكتشاف الأخطاء - أو مسبباتها- في برنامجك. وبعد اكتشاف الأخطاء وتصحيحها - إن وجدت - قد ترغب في معرفة مدى قوة حماية البرنامج، أي مدى قابليته للكسر على أيدي الكراكرز(crackers) (وهم محترفو الحاسوب الذين يقضون وقتهم في محاولة كسر البرامج بحيث يصبح البرنامج غير محمي (لا يطلب رقما سريا مثلا.)) فتقوم انت في هذه الحالة، بهندسة برنامجك عكسيا بغرض حمايته من الكسر.

اما دوافع الكراكر الى تعلم الهندسة العكسية , فتختلف ايضا حسب الاشخاص (او الفرق), منهم من يتخذها اداة لتحدي الحمايات و طرق التمويه، و منهم من يحمل معها شعار "المعرفة للجميع"، فيقوم بنشر و مشاركة الآخرين فيما توصل اليه، و منهم من يحمل معها شعار "اعانة الفقراء للحصول على برامج ما كانوا ليحصلوا عليها لثمنها الباهض"، و اخيرا منهم من يتخذها اداة للتخريب على خلفية ما، مادية كانت (ككسر برنامج شركة منافسة) او دينية او ما شابه ذلك.

1.1 . هذا الكتاب :

الغرض من هذا الكتاب هو إيصال علم الهندسة العكسية للمواطن العربي وتوفير عناء البحث في الدروس والمنتديات والمواقع لتعلم هذا الفن. الكتاب مخصص للمستوى المبتدئ، وبه بعض الدروس متوسطة المستوى.

يبدأ بك الكتاب في الباب الأول، بلمحة عن لغة الأسمبلي، وهي أهم لغة برمجة يحتاجها و يبدأ بها من يريد تعلم الهندسة العكسية. وفي الباب الثاني نلقي نظرة على أهم برامج الهندسة العكسية . اما في الباب الثاني فنتطرق الى تعلم فنون وأساليب الهندسة العكسية، إبتداء بالـ Serial fishing & General RE في الفصل الأول، وانتهاء بالـ dotnet reversing و ال VB في الفصل العاشر.

الباب الثالث والأخير يتحدث عن الشق الثاني و المتزايدة اهميته في علم الهندسة العكسية، ألا وهو فك الضغط اليدوي (Manual Unpacking)،حيث نلقي نظرة مفصلة على بعض الحمايات المشهورة . الغرض من هذا الباب هو إيصال الفكرة العامة وليس شرح كل الحمايات التجارية والتي يقدر عددها بالمئات.

2.1 . الفريق العربي للهندسة العكسية :

هذا الفريق تأسس في صيف العا 2006 على يد مجموعة من الأعضاء من بينهم dj-siba و allko و mouradpr . الهدف كان إيصال الصوت العربي إلى جميع أنحاء العالم، وتعليم المواطن العربي هذا العلم، وأن نثبت للغرب أننا قادرون - على الأقل في هذا المجال - أن نكون منتجين وليس كما جرت العادة، مستهلكين.

قد لا يكون فريقنا هو الأول عربيا، لكن لعلنا الأكثر نشاطا وعملا . نحن نطمح لأن نأخذ موقعا رياديا في عالم الهندسة العكسية و نسعى الى ذلك بالعمل الجاد و الصبر.

تنويه

إن مؤلفي هذا الكتاب، وهم أعضاء الفريق العربي للهندسة العكسية، لا يتحملون أدنى مسؤولية قانونية عن أي تصرف تقوم به إثر قراءتك لبعض أو كل هذا الكتاب.

أنت وحدك سيد أمرك!!!

يداك اوكتا و فوك نفخ

تنويه

نأمل أن يكون الكتاب مفيدا.

الفريق العربي للهندسة العكسية

الأربعاء، 01 محرم، 1429

الموافق ل

Wednesday, January 09, 2008


الباب
الأول
الفصل الثاني
مدخل إلى لغة الأسمبلي

2. الفصل الثاني : مدخل إلى لغة الأسمبلي

قبل البدأ، نرجوه بكون جل ما سيقال و يكتب في هذا المقال متعلق بالبيئة 32 بايت.

لغة الأسمبلي هي لغة منخفضة المستوى (low level language) تستخدم للبرمجة، وهي عبارة عن "ترجمة" لفظية للغة الآلة (machine language) التي يستخدمها معالج ما، فهي إذن عبارة عن اختصارات (تسمى) mnemonics تساعد المبرمج على تذكر أوامر هذه اللغة.

جدير بالذكر أن لكل بنية فيزيائية لمعالج ما، هناك لغة أسمبلي مختلفة . بمعنى أن لغة الأسمبلي الخاصة بمعالجات intel تختلف عن تلك المخصصة لمعالجات Motorola.

2.1 . أنظمة العد

لمساعدتك على فهم أكبر لهذه الفقرة من هذا الباب, تجد في المرفقات :

 آلة حاسبة عددية (BIN, OCT, DEC, HEX) 

1.1.2 . النظام الثنائي BINARY :

1.1.1.2 . تعاريف و مفاهيم :

هذا هو النظام الثنائي وهو اللغة الأم للحاسوب. فالحاسوب من الداخل مكون من دوائر كهربائية. هذه الدوائر لا تعرف الـ assembly أو الـ visual basic أو أي شيء من هذه اللغات..هي تتعامل فقط مع التيار الكهربائي: وجود تيار كهربائي أو عدم وجوده، وقد اتفق على أن يرمز لحالة وجود التيار بالرمز 1 وعدم وجوده بالرمز 0.

النظام الثنائي إذن يتكون من رقمين فقط هما 0 و 1. وبالتالي فإن أساس النظام (Base). يساوي 2.

مثلا فإن تعليمة مثل هذه :

```
MOV EAX, DWORD PTR DS:[40A710h]
```

هي بالأصل يتم التعامل معها بداخل الدوائر الكهربائية للحاسوب على أنها:

```
10100001000100001010011101000000000000000
```

أما تعليمة

```
PUSH EAX
```

فهي

1010000

والآن دعنا نر كيف يتم العد بالنظام الثنائي؟ دعني أذكرك بالنظام العشري كنا نعد هكذا: 0 1 2 3 4 5 ... 9 ثم العدد 10، هل لاحظت؟

النظام العشري مكون من 10 رموز هي 0123456789، إذن بعد الوصول إلى 9 فان الرموز (الأرقام) تنتهي. ما العمل؟ نبدأ من جديد فتصبح الـ 9 صفرا، ونضيف 1 إلى الخانة التالية (خانة العشرات)، إذن 9 يليها 10. بالمثل عندما نصل إلى 399 فان خانة الآحاد قد وصلت إلى آخر رقم وهو 9 إذن نعيد العد ونستبدل الـ 9 بصفر ونضيف واحد للخانة التالية. ألخانة التالية بها 9 إن أضفت إليها 1 ستصبح 10 إذن سنكتب اصفر والواحد ينتقل للخانة التالية وهي الـ 3، فتصبح 4، أي سنصل إلى 400.

تابع معي كيف نعد بالنظام الثنائي :

0 ثم 1 ثم 10. ما الذي حصل؟ كنا نعد بشكل طبيعي. 0. يليه 1. يليه ماذا؟ لا يليه شيء فالواحد هو آخر رقم في هذا النظام. إذن استبدله بصفر وأضف واحد للخانة الجديدة فتحصل على 10. فلنتابع : 10 ثم 11 ثم 100، ماذا حصل؟ الـ 10 يليها 11.. يليها 100

كما ترى فخانة الآحاد قد وصلة إلى آخر عدد إذن استبدله بصفر و أضف واحد للخانة التالية(أي خانة العشرات)، لكن خانة العشرات بدورها قد وصلت لآخر عدد.. أذن عندما نضيف 1 للـ 1 سنحصل على 10 أي نبقي الصفر وننقل الـ 1 للخانة التالية فيصبح لدينا 100.

انظر :

decimal	binary
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001

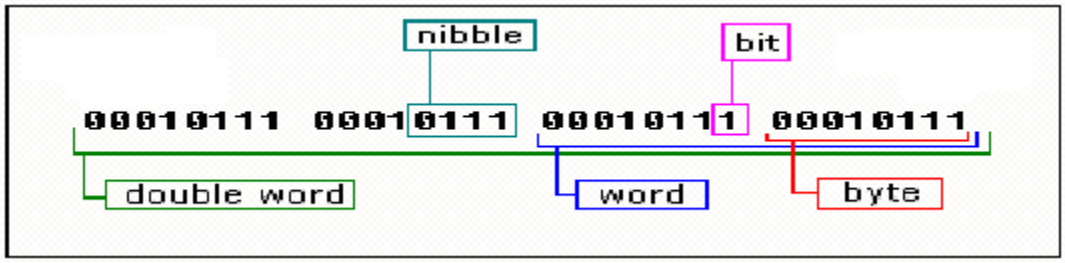
نرجو أن تكون الفكرة قد أصبحت واضحة.

2.1.1.2 . BIT و BYTE و أخواتهما، ما الخطب؟

- كل رقم (digit) في النظام الثنائي يسمى BIT،
- كل 4 bits تسمى NIBBLE،
- كل 8 bits تكوّن BYTE،

- كل two bytes تكوّن WORD،
- وختاما كل two words تكوّن DOUBLE WORD.

انظر الشكل التالي :



إن أول بت (في اليمين) يسمى low bit أو Least Significant Bit (LSB) أي البت ذو القيمة الأقل، وفي المقابل فإن آخر بت (في اليسار) يسمى high bit أو Most Significant Bit (MSB) أي البت ذو القيمة الأعلى.

عندما نقول 10100101b فإننا نعني قيمة بالباينري ولهذا وضعنا حرف b في نهاية الرقم. لكن هذا ليس شرطا.

العمليات في النظام الثنائي:

ماذا إذا أردنا جمع عددين بالنظام الثنائي؟ لا بأس دعنا نرى.

0=0+0
1=1+0
10=1+1

والآن لنر كيف نجمع 1100110 مع 1101

```
  1 1
1 1 0 0 1 1 0
0 0 0 1 1 0 1  +
---------------
1 1 1 0 0 1 1
```

كما ترى، فقد أضفنا أصفارا للرقم السفلي -هذا للتوضيح فقط- كي تسهل علينا عملية الحساب, لأنك تعلم ان الصفر على اليسار لا قيمة له

أول صفر مع أول 1 الناتج هو 1
ثاني 1 مع الـ 0 الناتج هو 1
ثالث 1 مع الـ 1 الناتج هو 10 ـ إذن نكتب الـ 0 ونضيف 1 للخانة التالية
رابع 0 مع الـ 1 مع الـ 1 السابق الناتج هو 10 - إذن نكتب الـ 0 ونضيف 1 للخانة التالية
خامس 0 مع الـ 0 مع الـ 1 السابق الناتج هو 1
سادس 1 مع الـ 0 الناتج هو 1
سابع 1 مع الـ 0 الناتج هو 1.

لن نشرح العمليات الأخرى أي الطرح والضرب والقسمة فهذه ستأخذ وقتا طويلا إضافة إلى انه لا فائدة كبيرة منها و لأننا سنتعلمها في الدروس الخاصة بالهندسة العكسية.

2.1.2 . النظام السداسي العشري HEXADECIMAL :

يتكون هذا النظام من 16 رقم (رمز) هم : (من اليمين لليسار) : 0 1 2 3 4 5 6 7 8 9 A B C D E F

أساس النظام (Base) يساوي 16. دعنا نرى كيف يمكن أن نعد باستخدام هذا النظام.

0 ثم 1 ثم 2. ثم 9 ثم A ثم B. ثم F ثم 10

ما الذي حصل؟ كما في السابق. .. بدأنا العد وحين انتهت الأرقام ووصلنا إلى أخر رقم ممكن وهو F اضطررنا إلى استبداله بصفر وإضافة 1 للخانة التالية.

انظر إلى هذا : (ابدأ التتبع من اليسار)

... 108 109 10A 10B 10C 10D 10E 10F 110 111 112 113 ...

... D389 D38A D38B D38C D38D D38E D38F D390 ...

كما في النظام الثنائي فلن أتطرق لعمليات الحساب في هذا النظام رغم أنها تتبع نفس القاعدة في النظام السابق..

2.1.3 . النظام الثماني OCTAL :

هذا النظام يتكون من ثمانية رموز هي 0 1 2 3 4 5 6 7 والعد فيه يكون كالتالي (من اليسار لليمين).

0. 1 2 3 4 5 6 7 10 11 12 13 14 15 16 17 20 21 22 23 24 25 26 27 30 31 32 33

2.1.4 . النظام العشري DECIMAL :

هذا هو النظام المألوف لجميع الناس. في النظام العشري هناك 10 أرقام (digits) :

0, 1, 2, 3, 4, 5, 6, 7, 8, 9

التحويل من نظام إلى آخر مذكور في الملحق 1 في نهاية الكتاب.

2.2 . موجب ام سالب؟ SIGNED NUMBERS :

تمعن في هذه القيمة : 0B521. هل هي موجبة أم ســـالبة؟ بالطبع لا مجال للحديث عن الموجب أو السالب فهذه قيمة لا تحمل إشارة. اذن، كيف يمكننا التمثيل اولا ثم التفريق بين عددين – في النظام 2 او 16 – دون استعمال الرمز المعتاد (-)؟

هناك طريقتان لفعل هذا: الأولى سهلة لكنها المشكلة متعلقة بتمثيل الرقم 0 و سيأتي ذكره ان شاء الله. و الثانية اكثر تعقيدا في الفهم لكننا سنحاول التبسيط ما امكن.

2.2.1 . الطريقة الاولى : المكمل الأول (FIRST COMPLEMENT)

إذا كان لدينا 8-bits (بايت) فيمكننا عمل 256 عدد مختلف (256 combinations) بما فيها الصفر. إذن يمكننا أن نفترض أن أول 128 عدد (0-127) هي موجبة والأعداد التالية (128-256) سالبة.

للحصول على الـ 1^{st} complement لأي عدد موجب فإننا ببساطة نحوله إلى ما يقابله بالـ binary.

إذن للحصول على 1^{st} c للقيمة 54 فإنا نحولها إلى binary فنحصل على 110110

أما إذا كانت القيمة سالبة، أي 54- فانا نطبق المعادلة التالية :

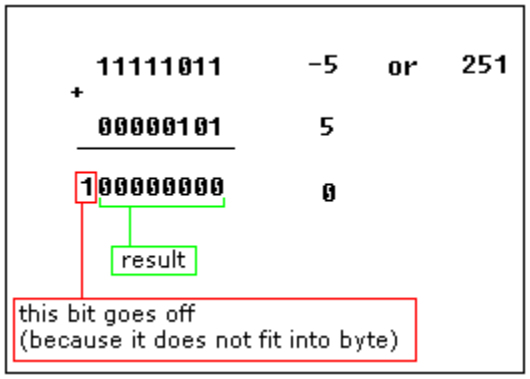
1st c = 2n – 1 – number in binary

- n التي تراها تعني عدد الخانات، فان كانت القيمة بايت فـ n=8 وأن كانت word فـ n=16 وهكذا.
- 2^n أي 2 مرفوعة للأس n.
- Number in binary أي الرقم بعد تحويلها إلى binary. لنطبق المعادلة على القيمة 54- لتحويلها إلى 1^{st} c بصيغة بايت

$$1^{st}\ c = 2^8 - 1 - 110110 == 11111111 - 110110 == 11001001$$

هناك طريقة أخرى أسهل من هذه. لو لاحظت فان القيمة 54 بالثنائي تساوي 00110110 (أضفنا صفرين لليسار كي يصبح العدد مكونا من 8 خانات) والآن اعكس كل بت فتحصل على 11001001.

لكن دعنا نتأمل قليلا. لماذا تم اختيار هذه الطريقة (أقصد طريقة الـ 1st complement) وليس أي طريقة أخرى ؟ إذا جمعت 5- مع 5 فستحصل على 0. أليس كذالك؟ حسنا هذا يحدث عندما يجمع الـ processor القيمة 5 مع القيمة 251، النتيجة هي 255، وبسبب الـ overflow فالنتيجة تعتبر صفرا.



كما ترى الناتج هو 100000000 لكن نحن جمعنا عددين من فئة byte أي 8bit لذا يجب أن يكون الجواب مكون أيضا من 8bit لكن حيث أننا حصلنا على 9bit فان هذه الحالة تسمى في علم الحاسوب overflow، ويتم تجاهل الـ MSB ويبقى 00000000 أي صفر.

نفس المبدأ ينطبق عند التعامل ما فئة word (16 bit values) حيث أن 16 bits تنشئ 65536 عددا مختلفا، أول 32768عدد (من 0 إلى 32767) تستخدم لتمثيل الأعداد الموجبة، والباقي (32768 إلى 65536) تستخدم لتمثيل الأعداد السالبة.

تبقى الاشارة الى ان المشكلة في استخدام هذه الطريقة كون الرقم 0 له تمثيلان اثنان: 00000000 و 11111111 لذا فالطريقة الثانية محبذة.

2.2.2 . الطريقة الثانية : المكمل الثاني (SECOND COMPLEMENT)

المكمل الثاني للأعداد الموجبة هو نفسه المكمل الأول والذي بدوره عبارة عن العدد بعد تحويله للـ binary.

فالـ 54 عند تحويلها للمكمل الثاني نحصل على 110110

أما المكمل الثاني للقيم السالبة فيتم الحصول عليه بإضافة 1 إلى المكمل الأول.

في المثال السابق عرفنا أن 54- بالمكمل الأول تساوي 11001001 إذن بالمكمل الثاني :

$$2^{nd}\ c = 1^{st}\ c + 1 = 11001001 + 1 = 11001010$$

لن نتعمق أكثر من هذا. يمكننا كتابة عشرات الصفحات كي نشرح هذا النظام لكن هذا ليس محور حديثنا.

2.3 . جهاز الكمبيوتر

جهاز الكمبيوتر هو آلة كهربائية تأخذ البيانات والتعليمات (inputs)، ثم تقوم بمعالجتها(processing)، وتخرج لنا المخرجات (outputs).

يقسم الكمبيوتر إلى 5 أقسام أساسية :

- **Central processing unit (CPU)**
- **Input devices**
- **Memory storage devices**
- **Output devices**
- **A communication network , called "bus" , that links all the elements of the system and connects the system to the external world.**

2.3.1 . وحدة المعالجة المركزية : CENTRAL PROCESSING UNIT (CPU)

و هي الوحيدة التي سنتحدث عنها من بين الاقسام السابقة.

الـ CPU قد تكون شريحة (chip) مفردة أو عدة شرائح متصلة مع بعضها البعض حيث تقوم بتنفيذ العمليات الحسابية والمنطقية (arithmetic and logical calculations) وتقوم بتنظيم تزامن العمليات الأخرى في الحاسوب والتحكم بها. مع تطور التكنولوجيا ظهر ما يسمى بالـ microprocessor الذي يتضمن المزيد من الدوائر الكهربائية والذاكرة داخليا. والنتيجة جهاز حاسوب بحجم أصغر بكثير.

معظم الـ CPU تتكون من المكونات الأساسية التالية :

Arithmetic logic unit (ALU)
Registers
Control section
Internal bus

-**أولا** , وحدة الحساب والمنطق Arithmetic Logical Unit (ALU): هذه الوحدة مسؤولة عن العمليات الحسابية (جمع طرح.) والمنطقية (or , XOR , and , etc...) وعمليات المقارنة بين البيانات (أكبر أصغر.).
-**ثانيا**, المسجلات Registers: إن الـ CPU يحتوي على وحدات ذاكرة صغيرة لكن سريعة جدا تستخدم لعمليات التخزين المؤقتة (temporary) للبيانات. هذه الذاكرة تحتوي على عدد من المسجلات registers، كل واحدة تقوم بعملية محددة. وسيلي شرح تفصيلي عن المسجلات لاحقا.
-**ثالثا** وحدة التحكم Control Unit (CU)
-**رابعا**, الممر الداخلي Internal Bus

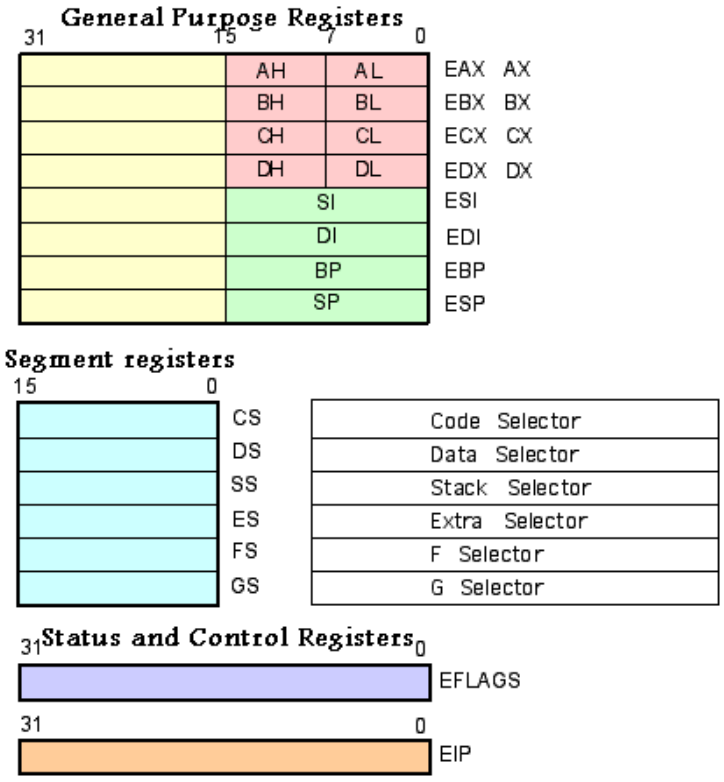
2.3.2 . نظرة تاريخية على المعالجات.

في السبعينيات طرحت شركة إنتل معالج 4004 وهو أول µP من نوع single-chip. وعندما نقول أنه 4-bit فنحن نعني أن الـ Bus Width يساوي 4 bit. كان قادرا على عنونة ذاكرة حتى حجم 640 بايت فقط. تلا هذا الطراز عدة طرازات. ثم جاء الـ 8086 في سنة 1978 وبناقل بيانات Bus باتساع 16-Bit. وناقل عناوين addresses bus باتساع 20 بت.كان قادرا على عنونة ذاكرة حتى حجم 1 ميجابايت.تلا ذلك عدة أنواع وموديلات أصبحت مزودة بناقل بيانات باتساع 32 بت، وهو الـ Pentium، وأيضا صدرت بعض المعالجات بناقل بيانات باتساع 64 بت كـ titanium، للإطلاع على قائمة بجميع الموديلات وتفاصيلها زر هذه الصفحة :

http://en.wikipedia.org/wiki/List_of_Intel_microprocessors

إن معالجي 8086 و 8088 متشابهين كثيرا من حيث الخصائص. وهذا السبب الذي يدفعنا لتسميتهم باسم x-86 family.

2.3.3 . المسجلات:



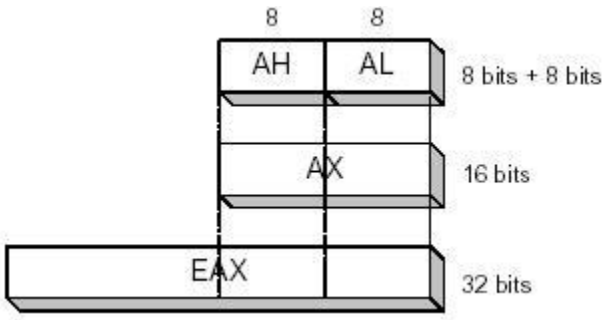
كما ترى هناك في هذا المخطط عدة أقسام للمسجلات.

2.3.3.1 . المسجلات العامة GENERAL PURPOSE REGISTERS:

إن معالجات 8086 تمتلك 8 مسجلات عامة الغرض، ولكل منها اسم خاص :

EAX - the accumulator register.
EBX - the base address register.
ECX - the count register
EDX - the data register
ESI - source index register.
EDI - destination index register.
EBP - base pointer.
ESP - stack pointer.

الواقع فإن تلك المسجلات الأربع : EDX,ECX,EBX, EAX لها التركيب التالي :



عند التعامل مع هذه المسجلات يمكنك الوصول إلى الـ 32 بت عن طريق EAX أو إلى أول 16 بت عن طريق AX أو حتى إلى أول وثاني 8 بت عن طريق AL و AH على الترتيب. لاحظ أن القسم العلوي من EAX لا يمكن الوصول إليه على انفراد. فقط القسم السفلي يمكنك التعامل معه بشكل منفرد. طبعا ما ينطبق على EAX ينطبق على EBX,ECX,EDX.

على الرغم من التسمية لكل مسجل، فإن المبرمج هو الذي يحدد استخدامات كل مسجل منها.

جدير بالذكر أن التعامل مع هذه الذواكر يتم بكل سرعة لأنها موجودة بداخل المعالج بعكس الذاكرة RAM (أو غيرها) حيث التعامل معها يتطلب استخدام الـ Buses مما يسبب تأخيرا زمنيا.

2.3.3.2 . SEGMENT REGISTERS :

CS - points at the segment containing the current program.
DS - generally points at segment where variables are defined.
ES - extra segment register, it's up to a coder to define its usage.
SS - points at the segment containing the stack.

أيضا هناك مسجلين إضافيين في المعالجات الحديثة (نسبيا) هما : FS و GS.

على الرغم من أنه يمكنك تخزين أي قيمة في الـ segment registers، فإن هذه لم ولن تكون فكرة صائبة. لأن هذا النوع من المسجلات ليس عام الغرض، بل له مهمة محددة وهي pointing at accessible blocks of memory (مؤشرات نحو مختلف قطع الذاكرة).

إن الـ segment registers تعملا جنبا إلى جنب مع الـ general purpose registers للوصول إلى أي عنوان ذاكرة.

على سبيل المثال إذا أردنا الوصول إلى عنوان الذاكرة الفيزيائي 12345h يجب أن نضع DS = 1230h و SI = 0045h. هذه طريقة جيدة لأنه هكذا يمكننا الوصول إلى عدد أكبر من العناوين بدلا من التقيد بالـ 16 بت التي تملكها الـ segment registers.

إن الـ CPU يقوم بعمليات حسابات للعنوان الفيزيائي بضرب الـ segment register بـ 10h وجمعه مع الـ general purpose register. هكذا :

```
 12300
+ 0045
-------
 12345
```

إن العنوان المكون بمسجلين إثنين يسمى effective address. بشكل افتراضي، فإن BX, SI and DI تعمل مع DS أما BP and SP فتعمل مع SS.

المسجلات العامة الغرض الأخرى لا يمكنها أن تكوّن effective address.

2.3.3 . المسجلات الاخرى:

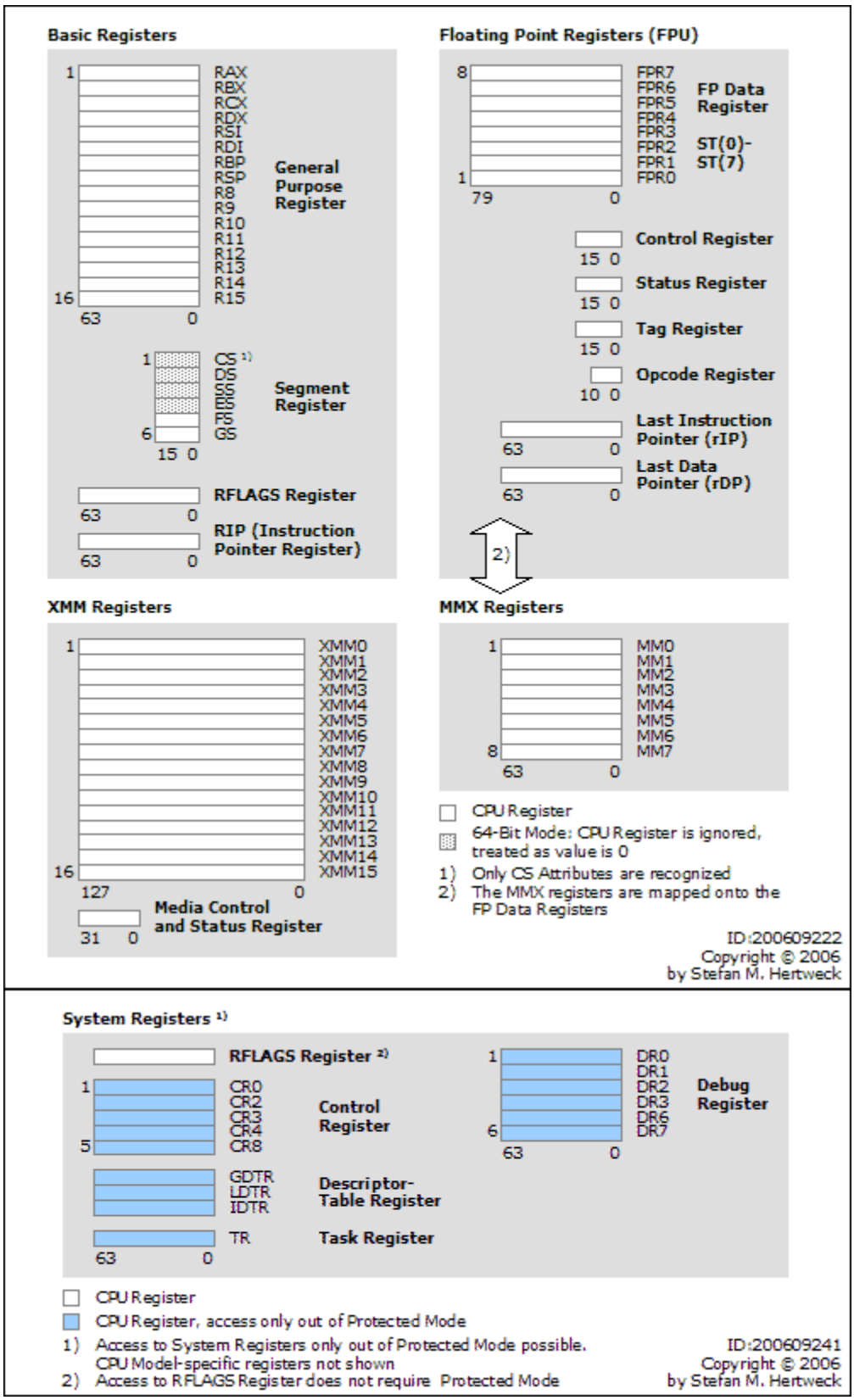
2.3.3.1 . MMX REGISTERS:

مسجلات 64 بايت و هي مشابهة ل المسجلات FPU التي تتعامل مع ال float numbers مع اختلاف طفيف في كون المسجلات MMX غير مترابطة و يمكن استعمال اي واحد منها في اي وقت.

كما ان هناك تطويرا يعتمد على نفس التقنية MMX والذي اعطى المسجلات XMM (SSE) ذات ال 128 بايتا.

2.3.3.2 . مسجلات 64 بايت:

للاشارة فقط فهناك مسجلات جديدة و اخرى تمديد للمسجلات القديمة في البروسسورات الجديدة مثل ال Itanium يمكنك الاطلاع عليها بالبحث و نعرض هنا لمحة سريعة لجردها لا غير:


Basic Registers
RAX
RBX
RCX
RDX
RSI
RDI
RBP
RSP
R8
R9
R10
R11
R12
R13
R14
R15
General Purpose Register
CS 1)
DS
SS
ES
FS
GS
Segment Register
RFLAGS Register
RIP (Instruction Pointer Register)
Floating Point Registers (FPU)
FPR7
FPR6
FPR5
FPR4
FPR3
FPR2
FPR1
FPR0
FP Data Register
ST(0)-ST(7)
Control Register
Status Register
Tag Register
Opcode Register
Last Instruction Pointer (rIP)
Last Data Pointer (rDP)
2)
XMM Registers
XMM0
XMM1
XMM2
XMM3
XMM4
XMM5
XMM6
XMM7
XMM8
XMM9
XMM10
XMM11
XMM12
XMM13
XMM14
XMM15
Media Control and Status Register
MMX Registers
MM0
MM1
MM2
MM3
MM4
MM5
MM6
MM7
CPU Register
64-Bit Mode: CPU Register is ignored, treated as value is 0
1) Only CS Attributes are recognized
2) The MMX registers are mapped onto the FP Data Registers
ID:200609222
Copyright © 2006 by Stefan M. Hertweck

System Registers 1)
RFLAGS Register 2)
CR0
CR2
CR3
CR4
CR8
Control Register
GDTR
LDTR
IDTR
Descriptor-Table Register
TR
Task Register
DR0
DR1
DR2
DR3
DR6
DR7
Debug Register
CPU Register
CPU Register, access only out of Protected Mode
1) Access to System Registers only out of Protected Mode possible. CPU Model-specific registers not shown
2) Access to RFLAGS Register does not require Protected Mode
ID:200609241
Copyright © 2006 by Stefan M. Hertweck

2.3.3.4 . المسجلات ذات الاستعمال الخاص SPECIAL PURPOSE REGISTERS :

2.3.3.4.1 . المؤشر EIP:

إن EIP دائما يعمل مع CS، وهو يشير دائما وأبدا إلى التعليمة التي يجري تنفيذها حاليا.

2.3.3.4.2 . FLAG REGISTERS : الرايات

إن الرايات يتم تغييرها تلقائيا من قبل الـ CPU بعد تنفيذ عمليات رياضية ومنطقية. إنها تسمح بمعرفة نتيجة العملية وتحديد الشروط لنقل التحكم إلى أجزاء أخرى من البرنامج.

بشكل عام لا يمكنك تغيير قيم هذه المسجلات بطريقة مباشرة.هناك طرق غير مباشرة قد نأتي على ذكرها في دروس قادمة.

31	18	17	16	15	14	13 12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved	AG	VM	RF	0	NT	IOPL	O	D	I	T	S	Z		A		P		C

تقسم الرايات إلى ثلاثة أقسام:

رايات غير مستعملة أي أنها لا تفيد في الحكم على آخر عملية وهي موجودة فقط في حالة تطوير المعالج ربما يحتاجون إلى رايات إضافية فيمكن استغلال هذه الرايات. في الوقت الحالي نحن لسنا بحاجة إليها

رايات الوضع : وهي الرايات التي تتأثر وتتغير حسب وضع العمليات التي تقوم بها وحدة الحساب والمنطق في المعالج

رايات السيطرة : وهي رايات المبرمج يتحكم بوضعها فإذا وضع بداخلها القيمة 1 تبقى هذه القيمة حتى يغيرها المبرمج في البرنامج عن طريق أوامر برمجة خاصة بها

مسجل الرايات في المعالج 8086 يحتوي على 7 رايات غير مستخدمة 6 رايات وضع و 3 رايات سيطرة اهمها:

CF	Carry flag
PF	Parity flag
AF	Auxiliary flag
ZF	Zero flag
SF	Sign flag
TF	Trap flag
IF	Interrupt flag
DF	Direction Flag
OF	Overflow flag

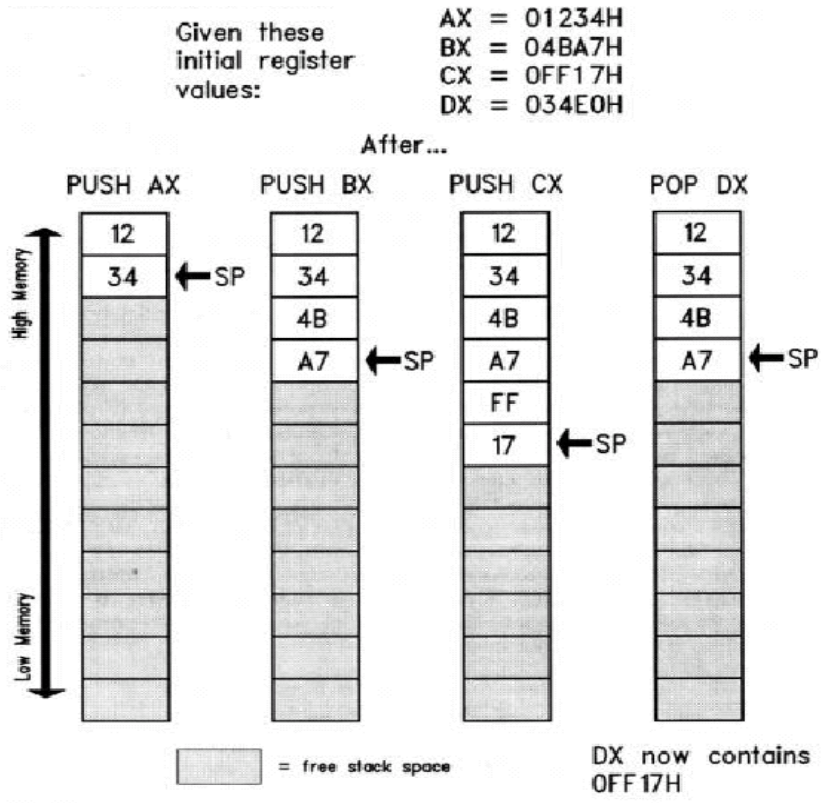
2.3.4 . المكدس STACK من النوع LIFO:

تحتاج البرامج التنفيذية إلى منطقة محجوزة في الذاكرة تدعى stack. تفيد هذه المنطقة في توفير حيز من الذاكرة لتخزين العناوين والمعطيات بشكل مؤقت، علما أن طول عنصر المعطيات الأساسي في stack هو كلمة واحدة DWORD في البيئة 32 بايت.

ينبغي على المبرمج تعريف الـ stack في برامج exe في حينه إلا انه لا حاجة لذلك في برامج com إذا يتولى نظام التشغيل تعريف مكدسة آليا. يهيئ DOS مسجل المقطع SS بعنوان بداية مقطع الـ stack، كما يهيئ المسجل ESP بحجم الـ stack ليشير إلى نهايته أو قيمته الحالية.

تتعامل مجموعة من التعليمات مع الـ stack بشكل مباشر كالتعليمات **CALL**,**PUSH**,**POP**. فالتعليمة **PUSH** مثلا تنقص محتويات المسجل ESP بمقدار كلمتين (DWORD) عند تنفيذها وتحزن قيمة ما في تلك الكلمة المحجوزة التي أصبح SPE يشير إليها. أما **POP** فهي على العكس حيث تنقل الكلمة التي يشير إليها ESP إلى موقع ما (مسجل او مكان في الذاكرة) ثم تسحبهما من قمة الـ stack و تزيد المسجل ESP بمقدار كلمتين.

والآن تمعن في الصورة التالية.هناك قيم مخزنة في المسجلات العامة الأربعة، ونريد أن نرى ماذا يحدث أثناء تنفيذ بعض التعليمات المتعلقة بالمكدس



لاحظ أننا إذا حفظنا قيم المسجلات AX,BX,CX,DX على الترتيب بواسطة التعليمات التالية :

```
PUSH ax
PUSH bx
PUSH cx
PUSH dx
```

فعلينا أن نسترجعها كالتالي:

```
POP dx
POP cx
POP bx
POP ax
```

يسمى هذا بمبدأ LIFO (Last In First Out) أي أن آخر قيمة تنقل للـ stack هي أول قيمة تخرج منه. لاحظ أن القيم التي تم استرجاعها من الـ stack تبقى موجودة فيه لكن لا يؤشر عليها المسجل SP، مما ينبغي ذكره هنا هو وجود حالتين يجب إلا يمر فيها الـ stack.

الأولى هي محاولة سحبنا قيمة ما من الـ stack وهو في حالة underflow أي فارغ يحتوي على القيمة FFFFh)

أما الحالة الثانية فهي محاولة دفع قيمة للـ stack وهو ممتلئ في حالة overflow (أي SP تحوي القيمة 0) ولذلك ينبغي توخي بعض الحذر في تعاملنا مع الـ stack. يشكل ذلك مطلبا بالغ الأهمية لعمل البرنامج في حالة تعطل مفاجئ وانتقال النظام إلى حالة غير مستقرة تستدعي غالبا إعادة استنهاض أو إقلاع الحاسب من جديد.

2.3.5 . العنونة في البرنامج : ADDRESSING

(الشرح تم على بيئة 16 بت و يبقى صالحا لل 32 بت)

تستطيع المعالجات x86 تنفيذ عمليات الوصول إلى كلمات الذاكرة بفعالية أكبر إذا كانت عناوينها زوجية ففي تعليمة **MOV** يستطيع المعالج الوصول إلى الكلمة ذات الإزاحة 0012 مثلا بشكل كامل ونسخها مباشرة إلى المسجل AX مثلا لأنها تملك عنوانا زوجيا أما إذا كانت الكلمة ذات عنوان فردي كعنوان الإزاحة 0013 عندئذ سينفذ المعالج خطوتين للوصول إلى هذه الكلمة التي سنفرض أن محتوياتها كما في الشكل التالي :

محتويات الذاكرة	xx	01	23	xx
الازاحة	0015	0014	0013	0012

في الخطوة الأولى ينسخ المعالج محتويات الحجرة ذات الإزاحة 0013 إلى المسجل AL بعد وصوله إلى كلا الحجرتين عند الازاحتين 0012 و 0014، أما في الخطوة الثانية فهو ينسخ الحجرة ذات الإزاحة 0014 إلى المسجل AH بعد وصوله إلى كلتا الحجرتين ذات الازاحتين 0014 و 0015 وبذلك تصبح محتويات المسجل AX هي 0123h.

لا تحتاج برمجة المواقع الزوجية برمجة تختلف عن الفردية / ولا إلى اهتمام بمعرفة فيما إذا كانت هذه المواقع فردية أم زوجية ففي كلا الحالتين سنحصل على نتائج صحيحة لان عملية الوصول إلى مواقع كلمات الذاكرة تضمن قراءتها وكتابتها ونقلها بشكل صحيح.

نستطيع استخدام التوجيه (directive) المسمى even في برامجنا لرصف أو توضيع كافة المعطيات في الذاكرة في المواقع ذات العناوين الزوجية لضمان فعالية أكبر للبرامج.

2.3.6 . مزيد من التوضيح حول الـ EFFECTIVE ADDRESSES

(الشرح تم على بيئة 16 بت و يبقى صالحا لل 32 بت)

قلنا بأن العناوين في المعالجات x86 تتكون من 20 بت، لكن معظم المسجلات registers في هذه المعالجات تتكون من 16 بت، إذن كيف سيتم التعامل مع عناوين ذاكرة بحجم 20 بت رغم أن المسجلات حجمها الأقصى هو 16 بت؟ الحل يكمن في استخدام مسجلين بدلا من واحد. ألمسجلين سويا حجمهما 32 بت أي أكبر من 20 وبالتالي فقد حلت المشكلة.

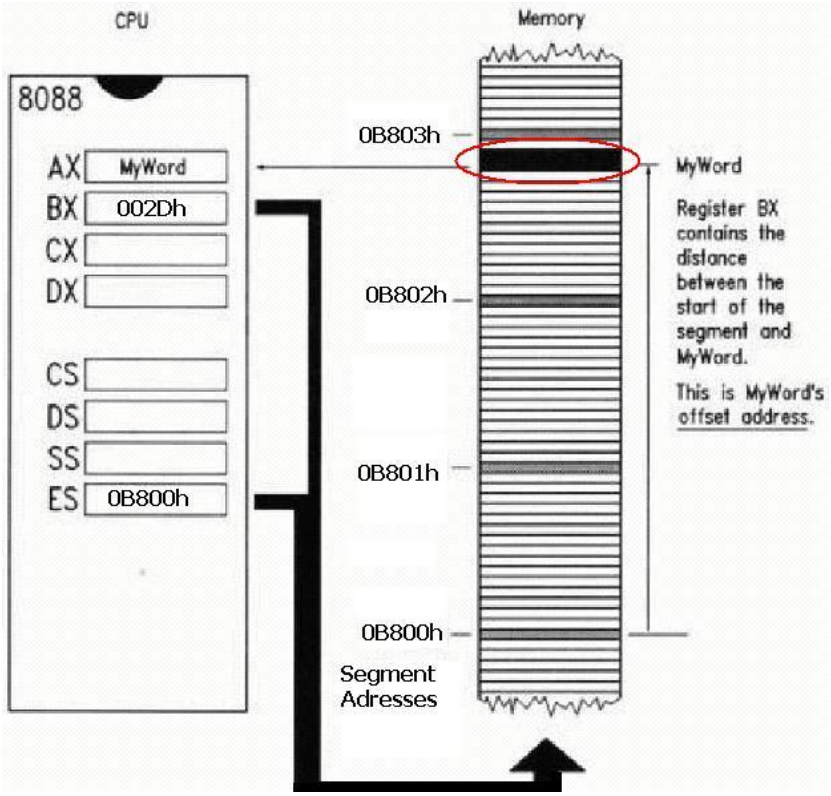
دعنا نرى ما يحدث عند تنفيذ تعليمة مثل

MOV AX, ES:[BX]

كما ترى فإننا نريد نقل بيانات معينة موجودة في الذاكرة (ولنسم البيانات هذا : MyWord). نريد أن نحدد للمعالج العنوان الذي توجد فيه MyWord، فكما ترى استخدمنا مسجلين لنصف هذا العنوان كالتالي ES:[BX]، تذكر أن الصيغة العامة للـ effective addresses هي كالتالي :

Segment: offset

وهذا يعني أن جزء الـ segment من العنوان موجود في ES وجزء الـ offset من العنوان موجود في BX ونريد من المعالج أن يذهب إلى هناك ويحضر الـ MyWord ويذهب بها إلى AX. الشكل التالي يوضح لنا كل ما يحدث أثناء تنفيذ التعليمة **MOV** AX, ES:[BX]



في اليسار ترى المعالج 8088 وترى جزءا من مسجلاته. في اليمين ترى الذاكرة (RAM) وهناك على يسارها عناوين الـ Segment Addresses (الـ Segment Addresses تكون مقسمة evenly إلى مجموعات فالـ segment تبدأ كل 16 بايت أي 10h. أو دعنا نقول يمكن أن تبدأ كل 16 بايت، فإذا تذكرنا أن 8088 يمكنه التعامل مع 1mb – أي 1048576 بايت - من الذاكرة، وبالأخذ بالحسبان أن الـ segment يمكن أن تبدأ كل 16 بايت، إذن 1048576 تقسيم 16 الناتج 65536، أي أنه هناك 65536 مكانا مختلفا يمكن للـ segment أن تبدأ عنده. كما ترى فالفرق بين عنوان كل segment هو 10h. قد تقول : لكني أرى الفرق 1h.هناك 0 في النهاية تم الاستغناء عنه من باب التسهيل، فالعناوين التي على اليسار مثل 0B800h كانت بالأصل 0B8000h لكن الصفر الأخير سيبقى صفرا لأنه كما قلت الفرق بين كل عنوان والذي يليه هو 10h إذن فهناك صفر دائما لذا استغنينا عنه)

إن MyWord التي نريد نسخها موجودة في مكان ما بالذاكرة، هذا المكان يبعد 02Dh عن الـ 0B800h segment لذا فإن عنوان هذه الـ MyWord هو 0B800 : 02D. لكن لاحظ أنه يمكننا استخدام segment أخرى، ولتكن الـ segment التي فوقها مباشرة 0B801 عندها ستختلف الإزاحة offset وستصبح 01Dh وبالتالي يمكننا أن نقول أن جميع العناوين التالية لها نفس التأثير :

0B800 : 02D
0B801 : 01D
0B802 : 00D

الصورة أصبحت أوضح الآن.

ملاحظة صغيرة عن كتابة العناوين بصيغة Segment: offset
يجب ألا تضيف اللاحقة h في نهاية الرقمين التي تدل على أن النظام هو الهكس.

فلنعد إلى الصورة مرة أخرى. الآن أصبحنا نعلم أين توجد MyWord و أين قد خزننا عناوينها الاثنين.

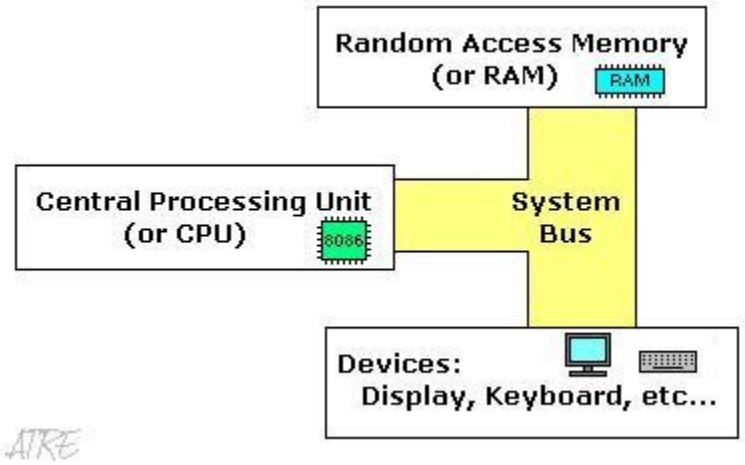
بعد تنفيذ التعليمة يكون المعالج قد ذهب إلى العنوان المطلوب و أخذ MyWord ووضعها في AX. لاحظ أنه بإمكانك استخدام أي Segment Register آخر كـ DS مثلا.

إن الـ offset يمكن وضعه فقط في BX,BP,SP,SI and DI.

2.4 . لغة الأسمبلي. ما هي؟

2.4.1 . تعريف:

هي لغة برمجة منخفضة المستوى (low level)، كي تبرمج بالأسمبلي يلزمك معرفة بالـ computer structure، يمكننا توضيح الـ computer model كالتالي :



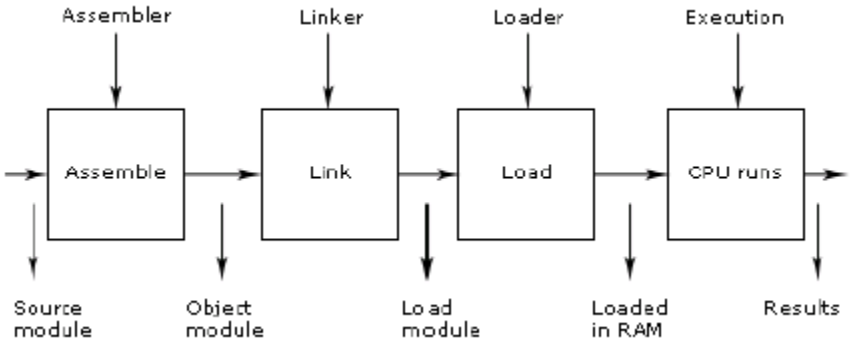
والآن دعنا نلقي نظرة مفصلة على المسجلات التي بداخل الـ CPU.

(لن أتطرق إلى الـ internal architecture للمعالج.سأكتفي بما يهمنا بطريقة مباشرة فقط)

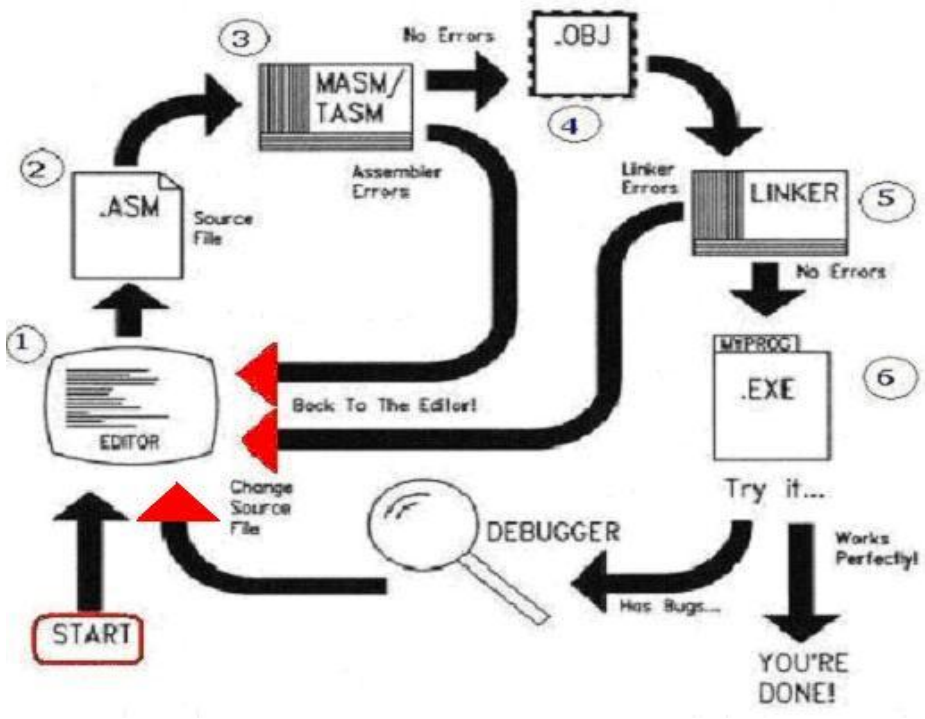
2.4.2 . ASSEMBLING AND LINKING :

ما تقوم بكتابته من تعليمات يسمى source code، وعند تجميعه بواسطة assembler كـ masm أو tasm فانك تحصل على ملف وسيط يسمى object code، ثم نقوم باستخدام احد الـ Linkers لربط ملفات الـ obj مع بعضها البعض والحصول على ملف.exe واحد.

الصورة التالية توضح المقصود :



والآن أنظر إلى الصورة التالية فهي توضح مراحل تطوير برامج الأسمبلي :



1. باستخدام أي محرر (... , notepad , radasm , winasm) نقوم بكتابة برنامج الأسمبلي.
2. بعد حفظ المشروع سنحصل على ملف (أو أكثر) بامتداد asm.
3. باستخدام أي مجمّع assembler (... , FASM , TASM , MASM) نقوم بتجميع ملف أو ملفات الـ asm.
4. بعد التجميع، نحصل على ملف يسمى relocatable object module, بامتداد obj (كل ملف asm يقابله ملف obj واحد)
5. نقوم باستخدام أي linker (... , tlink , link) لنربط جميع ملفات الـ obj ونحصل على الملف التنفيذي.
6. ها قد انتهينا والناتج هو ملف exe جاهز للتشغيل.

طبعا لاحظ أنه في كل خطوة كنا نفترض أنه لم يحدث أي خطأ لان حدوث خطا يتطلب اعادة تلك الخطوة من جديد.هل ترى السهمين باللون الأحمر؟ كل منهما يعبر عن حدوث خطأ في إحدى المراحل.

أمر آخر. إذا لم يحدث خطأ وحصلنا على ملف. exe لكنه لم يعمل فهنا نحتاج إلى استخدام debugger لتنقيح البرنامج واكتشاف موطن الخطأ فيه. وبعد اكتشاف مكان الخطأ نعود للسورس كود ونقوم بتغيير الكود الذي سبب الخطأ.

إذا أردت التوسع في مفهوم المجمع والرابط فهناك العديد من الدروس والمقالات التي تتحدث عن هذا. كما قلت سابقا لجأنا هنا للأختصار حتى لا يطول الكتاب كثيرا.

3.4.2 . التعليمات الأساسية

سنتكلم عن التعليمات الحسابية (Arithmetic) والمنطقية (Logic) وبعض التعليمات الأخرى. و قبل البدء دعنا نفصل الحديث قليلا عن الـ Flags. (في البيئة 32 بت يضاف الحرف E الى Flags ليصبح Eflags دلالة على Extended Flags و يكون بحجم 32 بت)

كما ترى فإن المسجل flags هو بحجم 16 بت، كل بت يسمى flag ويأخذ القيمة 0 أو 1.

كنا نريد ترجمة الفقرة التالية لكن رأينا أنها مكونة من مصطلحات ولا يوجد ما يحتاج الترجمة.

Carry Flag (CF) : this flag is set to 1 when there is an unsigned overflow. For example when you add bytes 255 + 1 (result is NOT in range 0.255). When there is no overflow this flag is set to 0.

Zero Flag (ZF) : set to 1 when result is zero. For none zero result this flag is set to 0.

Sign Flag (SF) : set to 1 when result is negative. When result is positive it is set to 0. Actually this flag take the value of the most significant bit (MSB).

Overflow Flag (OF) : set to 1 when there is a signed overflow. For example, when you add bytes 100 + 50 (result is NOT in range -128.127).

باقي الرايات ليس من المهم معرفتها (على الأقل بالنسبة لمبتدئ)

2.4.3.1 . المجموعة الاولى : **ADD, SUB,CMP, AND, TEST , OR, XOR**

هذه التعليمات لها المعاملات (operands) التالية

```
REG, memory
memory, REG
REG, REG
memory, immediate
REG, immediate
```

REG: EAX, AX , AL , AH , EBX , etc...
memory: [EBX], [EAX+ESI*4], variable, etc.
immediate: 5, -24, 3Fh, 10001101b, etc.

- بعد هذه التعليمات، فالنتيجة تخزن دائما في المعامل الأول. لكن تعليمة **CMP** و **TEST** تؤثر على الرايات فقط ولا تخزن أي شيء.
- الرايات التي تتأثر بهذه التعليمات هي CF, ZF, SF, OF, PF, AF
- عندما يكون هناك two operands فيجب أن يكونا من نفس الحجم، مثلا لا يجوز أن تكتب **MOV** ecx,ax لانهما ليسا من نفس الحجم.

❖ أيضا مما يجب ذكره هو أنه إذا رأيت قيمة ما كـ operand (معامل) لإحدى التعليمات، فهذه القيمة هي بالنظام العشري إذا لم يكن هناك لاحقة، مثلا **MOV** ebx,30 تعني أننا سننقل القيمة 30 عشري، بينما **MOV** ebx,30h تعني نقل القيمة 30 بالهكس وهكذا.

❖ في حالة عنوان الذاكرة فيجب أن تحدد الحجم الذي تريده. تذكّر أننا قلنا أن الـ two operands , يجب أن يكونا من نفس الحجم، لذا في حالة عنوان الذاكرة فإننا نبدأ الـ operand بالحجم المطلوب أما dword أو word أو حتى byte، ثم ptr وهي اختصار لـ Pointer أي مؤشر، ثم عنوان الذاكرة

- تعليمة **ADD** تقوم بجمع المعامل الثاني إلى المعامل الأول.
- تعليمة **SUB** تطرح المعامل الثاني من الأول.
- تعليمة **CMP** تطرح المعامل الثاني من الأول (لكن النتيجة يتأثر بها الرايات فقط).
- تعليمة **AND**: هذه تعليمة منطقية. كل عملية منطقية لها ما يسمى بـ Truth Table أو جدول الحقيقة. بالنسبة لتعليمة AND فالـ Truth Table هو : (هذه العملية هي عملية ضرب منطقي لكن بين الصفر والواحد فقط)

AND	
1 **AND** 0	0
0 **AND** 1	0
0 **AND** 0	0

- تعليمة **TEST** تقوم بنفس ما تقوم به **AND** لكن النتيجة تؤثر على الرايات فقط.
- تعليمة **OR** تقوم بعملية منطقية، الـ Truth Table هو كالتالي :

OR	
1 **OR** 0	1
0 **OR** 1	1
0 **OR** 0	0

- تعليمة **XOR** تقوم بعملية منطقية، الـ Truth Table هو كالتالي : (إذا كان المعاملان متشابهين فالنتيجة 0، و إذا كانا مختلفين فالنتيجة 1)

XOR	
1 **XOR** 0	1
0 **XOR** 1	1
0 **XOR** 0	0

2.4.3.2 . المجموعة الثانية : **MUL**, IMUL, DIV, IDIV

إن شعرت انك لم تفهم هذه التعليمات جيدا فلا بأس سوف تفهمها في الباب الثاني حيث سيكون هناك تطبيقات ملموسة.

هذه التعليمات لها المعاملات (operands) التالية :

```
REG
memory
```

- تعليمة **MUL** مسؤولة عن الضرب بدون إشارات، IMUL للضرب بإشارات. نفس الشيء ينطبق على DIV و IDIV.
- تعليمة **MUL** و IMUL تؤثران على الرايات التالية فقط : CF , OF

فعندما تكون النتيجة أكبر من حجم المعاملات، عندها تحمل هاتين الرايتين القيمة 1 (set to 1) فيما عدا ذلك فتحملان القيمة 0 (set to 0). أما بالنسبة لتعليمة DIV و IDIV فالرايات غير معروفة القيمة (undefined).

❖ القواعد التي يتم على أساسها الضرب في تعليمتي **MUL** و **IMUL** هي :

A byte operand is multiplied by AL ; the result is left in AX.

A word operand is multiplied by AX ; the result is left in DX:AX. DX contains the high-order 16 bits of the product.

A doubleword operand is multiplied by EAX and the result is left in EDX:EAX. EDX contains the high-order 32 bits of the product.

المثال الاول

```
-2 * -4 = 8 = 8 h = 1000 b

MOV AL, -2
MOV BL, -4
IMUL BL
AX = 8 h = 1000 b
```

المثال الثاني

```
1700 * 520 = 0D7D20 h = 1101011111010010000 b

MOV AX, 1700
MOV BX,520
MUL BX
DX =000D h = 0000000000001101 b
AX = 7D20 h = 0111110100100000 b
```

المثال الثالث

```
12345678h * 99999999 h = AEC33E18EAD65B8 h = 1010111011000011001111100001100011101010110101100101101110000 b

MOV EAX, 12345678h
MOV EBX, 99999999h
MUL EBX
EDX = 0AEC33E1 = 00001010111011000011001111100001
EAX = 8EAD65B8 = 10001110101011010110010110111000
```

لاحظ أنه إذا كتبت العدد كما هو فهذا يعني أنه بالعشري، أما إذا أضفت اللاحقة h فهذا يعني أنه بالهكس، بالمثل فإن إضافة اللاحقة b تعني أن العدد بالباينار ي.

❖ أما بالنسبة لتعليمة IDIV و DIV فلهما الجدول التالي الذي يوضح الحالات الثلاث للقسمة (تبعا لحجم ما تريد قسمته)

Size (الحجم)	Quotient (الناتج)	Reminder (الباقي)	Dividend (المقسوم عليه)
Byte	AL	AH	AX
Word	AX	DX	DX:AX
dword	EAX	EDX	EDX:EAX

```
11CCCEEE44BBBAAA h /33A33A33 = 583EF4DF h = 10110000011111011110100110111111 b

MOV EDX, 11CCCEEE h
MOV EAX, 44BBBAAA h
MOV ECX, 33A33A33 h
IDIV ECX
EAX = 583EF4DF h = 10110000011111011110100110111111 b
EDX = 15B96C3D h
```

2.4.3.3 . المجموعة الثالثة **INC**, DEC, NOT, NEG

هذه التعليمات لها المعاملات(operands) التالية

```
REG
memory
```

❖ تعليمة **INC, DEC** تؤثران على الرايات التالية : ZF, SF, OF, PF, AF
❖ تعليمة **NOT** لا تؤثر على أيّ من الرايات.
❖ تعليمة **NEG** تؤثر على الرايات التالية : CF, ZF, SF, OF, PF, AF.

- تعليمة **INC** تقوم بزيادة المعامل بمقدار واحد
- تعليمة **DEC** تقوم بإنقاص المعامل بمقدار واحد
- تعليمة **NOT** تقوم بعكس كل بت من بتات المعامل (الـ 0 يصبح 1 والعكس صحيح)
- تعليمة **NEG** تقوم باستبدال القيمة بالـ two's complement لها.

بعد انتهاء شرح ال Arithmatic instruction, اليك هذه الآلة الحاسبة التي تسهل عليك جميع العمليات الحسابية على الاعداد bit32 .

2.4.3.4 . المجموعة الرابعة : MOV INSTRUCTION

هذه التعليمة لها المعاملات(operands) التالية

```
MOV REG, memory
MOV memory, REG
MOV REG, REG
MOV memory, immediate
MOV REG, immediate
```

أما في حالة كان أحد المعاملات segment register، فهذا هو المسموح فقط :

```
MOV SREG, memory
MOV memory, SREG
MOV REG, SREG
MOV SREG, REG
```

هذه التعليمة تقوم بنسخ المعامل الثاني (المصدر source) إلى المعامل الأول (الوجهة destination)

والآن إلى القليل من الأمثلة. لن نكتب البرامج كاملة فنحن لا نعرف بعد كيف يمكن عمل ذلك، ما سنقوم به هو كتابة الكود الذي ينفذ ما نريده فقط وليس كل البرنامج.

الأمثلة التالية تشمل جميع الأوامر السابق ذكرها

```
9553h+35h=9588h
MOV EAX,9553H
ADD EAX,35H

111011101b * 100100001b=1000011010011111101b
MOV EAX,111011101B
MUL 100100001

01100001 AND 11011111 = 01000001
MOV CX,01100001B
AND CX,11011111B
```

لاحظ أنه يمكنك التعامل مع الأحرف، أي أن تنقل قيمة الـ ascii لحرف ما إلى مسجل ما وذلك بوضع الحرف بين علامتي تنصيص مفردتين هكذا : 'a' كما في المثال التالي :

```
01100001 AND 11011111 = 01000001
MOV AL, 'A'             ; AL = 01100001B
AND AL, 11011111B       ; AL = 01000001B ('A')

203 / 4 = 50.75
```

```
MOV AX, 203   ; AX = 00CBh
MOV BL, 4
DIV BL        ; AL = 50 (32h), AH = 3

NEG 5
MOV AL, 5     ; AL = 05h = 0000 0101
NEG AL        ; AL = 0FBh (-5) = 1111 1011
NEG AL        ; AL = 05h (5)

NOT 00011011b
MOV AL, 00011011b
NOT AL        ; AL = 11100100b
```

في هذا المثال، لنفرض أن القيمة 00011011b موجودة في الذاكرة عند العنوان DS:[2156] عندها يمكن عمل التالي :

```
NOT 00011011b
NOT [2156]    ; [2156] = 11100100b
```

5.3.4.2 . المجموعة الخامسة : JUMPING INSTRUCTIONS & VARIABLES

كما تعلم فإن المعالج يقوم بتنفيذ الأوامر بالترتيب. لكن إن أردنا نقل التنفيذ إلى جزء أخر في البرنامج فيمكننا عمل ذلك بواسطة أوامر القفز.تقسم هذه الأوامر إلى قفز مشروط conditional jump وقفز غير مشروط unconditional jump.

الجدول التالي يوضح أهم أوامر القفز المشروط وهناك الكثير غيرها سنذكرها لاحقا.

Instruction	Description	Affected flags
JZ	اقفز إذا كانت النتيجة صفر	ZF=1 ➔ jump
JNZ	اقفز إذا كانت النتيجة لا تساوي صفر.	ZF=0 ➔ jump
JE	اقفز إذا كانت النتيجة صفر	ZF=1 ➔ jump
JNE	اقفز إذا كانت النتيجة لا تساوي صفر.	ZF=0 ➔ jump
JNC	اقفز إذا الـ MSB لم تخرج 1 للخارج(carry out) او لم يحتاج لواحد من الخارج (borrow)	CF=0 ➔ jump

انتبه إلى أن jz تعمل مثل عمل JE، و **jnz** تعمل مثل عمل **jne**.

مثال / دعنا نرى هذا الكود :

```
CMP BL,2
JE ALLKO
MOV DX,44
JMP AT4RE
ALLKO:
MOV BL ,7
```

```
AT4RE:
NOP
```

في البداية يتم مقارنة BL مع القيمة 2، ثم هناك قفزة مشروطة **JE**، فإذا تساوى، سينتقل التنفيذ إلى allko، و allko هو عبارة عن label. كما ترى يمكن وضع label في أي مكان من البرنامج وذلك ببساطة بكتابة الاسم الذي تريد متبوعا بنقطتنا رأسيتان.

ماذا إن لم يتساوى؟ لن تنفذ القفزة وبالتالي سيتابع المعالج تنفيذ الأوامر بشكل طبيعي. ألأمر التالي الذي سينفذ في حالة عدم المساواة هو **MOV** DX,44. يليه قفزة غير مشروطة إلى الـ label المسمى at4re.

- إذن في حالة المساواة سنصل إلى تعليمة **MOV** BL,7 ويتم تنفيذها. ثم ننتقل إلى التعليمة **NOP** لتنفيذها.
- أما في حالة عدم المساواة سنصل إلى تعليمة nop ويتم تنفيذها.

تعليمة nop هي اختصار لـ no operation أي أنها لا تقوم بعمل شيء.

مثال / دعنا نرى هذا الكود.. وظيفته هي نقل القيمة 25H إلى تسع خلايا ذاكرة (من 401200 إلى 401208).هناك عدة طرق لعمل هذه العملية منها الطريقة التالية :

```
MOV EAX,25H
MOV ECX,8
BEGINNING:
MOV BYTE PTR DS:[401200+ECX],EAX
DEC ECX
JNZ BEGINNING
RET
```

في البداية نضع القيمة 25H في أحد المسجلات وليكن EAX ثم نضع القيمة 8 في المسجل ECX (ليس شرطا لكن جرت العادة أن نستخدم ECX كعدّاد) ثم هناك LABEL باسم BEGINNING. يليه أمر النقل **MOV**، لاحظ أن التوجيه (Directive) المستخدم هو BYTE PTR لأن القيمة التي نود نقلها (25 h) هي بحجم بايت. ثم نقوم بإنقاص قيمة العداد (ECX) بمقدار 1 وبالتالي فإن القيمة 25h سيتم نقلها أول مرة إلى 401208 = 8 + 401200 ثم هناك قفزة مشروطة فإذا وصلت قيمة ECX إلى الصفر فإن ZF=1 (بالتالي نكون قد نقلنا القيمة إلى الأماكن التسعة) ولن يتحقق شرط القفز وسنكمل إلى الأمر الذي بعده.

لكن بالطبع بعد أول عملية نقل وتنفيذ DEC ECX لأول مرة، فأن ECX=7 إذا الشرط تحقق وبالتالي سنقفز إلى BEGINNING

ونقوم بعملية النقل مرة أخرى وهكذا. ألى أن نقوم بعملية النقل في المرة الأخيرة عندها 401200=0+401200 ونكون قد انتهينا إذاً ← ECX=0 إذاً ← ZF=1 إذاً ← الشرط لم يتحقق إذاً ← ننفذ الأمر التالي وهو **RET** وهذا يعني **RET**URN أي اعادة السيطرة (control) إلى نظام التشغيل.

بقي أن نشير إلى جزء آخر من تعليمات القفز وتستخدم بكثرة. أنتبه إلى أنه في هذه التعليمات فإننا نتعامل مع أعداد ذات إشارة إما سالبة و إما موجبة. هذا يعني أن هذه الأوامر تعتبر آخر بت (أي MSB) هو بت الإشارة فإذا كان مساويا لـ 0 فالعدد موجب وإذا كان مساويا لـ 1 فالعدد سالب.

CMP OPERAND 1 , OPERAND 2

INSTRUCTION	DESCRIPTION
JG	JMP IF OP1>OP2 (JMP IF GREATER)
JNG	JMP IF OP1 <= OP2
JL	JMP IF OP1<OP2 (JMP IF LESS THAN)
JNL	JMP IF OP1>=OP2
JGE	JMP IF OP1>=OP2 (JMP IF GREATER THAN OR EQUAL)
JNGE	JMP IF OP1<OP2
JLE	JMP IF OP1<=OP2
JNLE	JMP IF OP1>OP2

أما هذه التعليمات فهي خاصة بالأعداد الموجبة فهي تعتبر أن آخر بت (أي MSB) هو جزء من العدد وليس دالا على الإشارة.

CMP OPERAND 1 , OPERAND

INSTRUCTION	DESCRIPTION
JA	JMP IF OP1>OP2 (JMP IF above)
JNA	JMP IF OP1 <= OP2
JB	JMP IF OP1<OP2 (JMP IF BELOW)
JNB	JMP IF OP1>=OP2
JAE	JMP IF OP1>=OP2 (JMP IF ABOVE OR EQUAL)
JNAE	JMP IF OP1<OP2
JBE	JMP IF OP1<=OP2
JNBE	JMP IF OP1>OP2

دعنا نوضح الأمر قليلا.

تأمل المقطع التالي :

```
MOV AL , -1
CMP AL,0
JG SECOND
FIRST:
MOV ECX,3
RET
SECOND:
XOR ECX,ECX
RET
```

إن 1- يتم تمثيلها بالنظام الثنائي (بحجم بايت) كالتالي : 1111 1111.

هناك عملية مقارنة بين الـ 1- وبين الـ 0، واضح أن الشرط لن يتحقق فالـ 1- ليست أكبر من 0، إذا لن تنفذ القفزة وبالتالي سنتوجه إلى تعليمة **MOV** ECX,3 فيصبح ECX=3 ثم هناك تعليمة **RET** مما يعني إرجاع السيطرة إلى النظام وإنهاء البرنامج.

لكن ألق نظرة على الكود التالي. نفس السابق لكن استخدمت **JA** بدلا من **JG**.

```
MOV AL , -1
CMP AL,0
JA SECOND
FIRST:
MOV ECX,3
RET
SECOND:
XOR ECX,ECX
RET
```

إن 1- يتم تمثيلها بالنظام الثنائي (بحجم بايت) كالتالي : 1111 1111 لكن بالنسبة لتعليمة **JA** فهذه لا تعني 1- لأننا هنا نتعامل مع أعداد موجبة إذن البت الأخيرة MSB ليست لتدل على الإشارة بل هي جزء من الرقم. أذن لو حولنا القيمة 1111 1111 إلى عشري سنجدها تساوي 255.

والآن هل 255 أكبر من 0؟ بكل تأكيد. أذن فقد تحقق الشرط. وسننتقل إلى التعليمة **XOR** ECX,ECX التي ستجعل ECX=0 ثم هناك تعليمة **RET** للعودة للنظام..

إذن في حالة تعليمة JG حصلنا على ECX=3، لكن في حالة JA حصلنا على ECX=0

2.4.3.6 . المتغيرات : VARIABLES

المتغير هو مكان في الذاكرة له اسم وحجم معينين. يتم تعريف المتغيرات في مقطع data. من البرنامج. أنظر إلى الجدول التالي :

db	Define byte	1 byte = 8 bit
dw	Define word	2 byte = 16 bit
dd	Define double word	4 byte = 32 bit

- يتم تعريف متغير باسم allko (مثلا) و بحجم word (مثلا) وبقيمة ابتدائية 4445h (مثلا) كالتالي :

Allko dw 4445h

- ماذا لو أردنا تعريف نفس المتغير لكن لا نريد إعطائه أي قيمة ابتدائية؟ ستكون الصيغة كالتالي: :

Allko dw ?

علامة الاستفهام تشير إلى عدم وجود قيمة ابتدائية.. لاحظ أنه إذا لم تعط قيمة ابتدائية للمتغير فعليك تعريفه في مقطع. data?

- أيضا يمكنك تعريف مصفوفة (array) كالتالي :

xxx dd 7,5,4,8,1

هنا قمنا بتعريف مصفوفة مكونة من 5 عناصر وكل عنصر بحجم dword كما هو واضح.

و للوصول إلى العنصر الثالث مثلا يمكننا عمل التالي :

Add xxx[2],400

لاحظ أن أول عنصر هو xxx[0] والثاني xxx[1] وهكذا.

لنفرض أننا أردنا حجز 4 بايتات من الذاكرة ونريد وضع القيمة الابتدائية 40 (28h أي 28 بالهكس) في كل بايت، ونريدهم جميعا تحت نفس الاسم.هذا ما علينا فعله (لاحظ انك تتعامل مع Bytes وان القيمة القصوى للبايت هي بين -255 و 255)

At4e db 40,40,40,40

لكن هناك طريقة أسهل كالتالي :

At4re db 4 dup (40)

يمكننا عمل نفس الشيء لكن دون إعطاء قيمة ابتدائية.كالتالي:

At4re db 4 dup (?)

أما النصوص فيمكن تعريفها كالتالي :

X db 'A'

لاحظ أنه يجب تعريف الأحرف والنصوص (strings) كمتغيرات بحجم بايت، أي يجب استخدام db في حالة ال ASCII (في حالة ال UNICODE نعرفها باستعمال dw مع مراعاة وضع صفرين بدل صفر واحد في الاخير)

ولاحظ أن الحرف أو الـ string توضع بين علامتي تنصيص مفردتين أو مزدزجتين:

تعريف string لا يختلف عن تعريف حرف. لاحظ :

msg db 'AT4RE is da best',0 أو msg db "AT4RE is da best”,0

الفرق الوحيد هو وجود الصفر (NULL) في النهاية ليدل على انتهاء الـ string. إن كنت قد تعاملت مع أي من لغات المستوى العلوي من قبل فأنت تعرف عن ماذا أتحدث.

أمر آخر يجب عليك معرفته، هو أنه بدلا من كتابة الحرف A (أو أي حرف آخر) يمكنك استخدام القيمة المقابلة له في جدول ASCII لكن دزن علامات تنصيص.

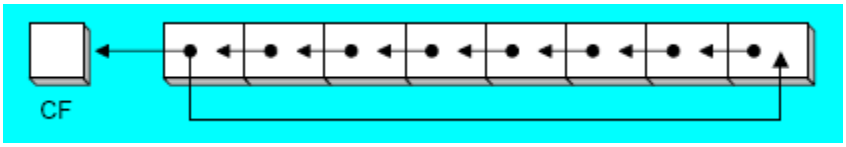
Y db 65

7.3.4.2 . المجموعة السادسة : ROTATING AND SHIFTING INSTRUCTIONS

درس اليوم يتحدث عن تعليمات الدوران والإزاحة. سنتناول بعض التعليمات فقط، لكن هناك العديد من التعليمات الأخرى. لا مجال للحديث عنها جميعا.

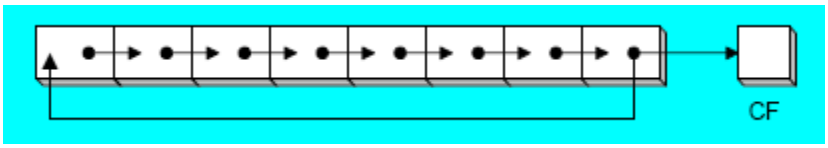
أيضا سنتعرض لبعض الأوامر الأخرى.

ROL : هذا الأمر هو اختصار لـ Rotate Left أي تدوير لليسار. لن أكثر من الكلام فالشكل التالي سيوضح كل شيء بعد تنفيذ **ROL** AX,1



لاحظ كيف أن آخر بت (MSB) تم تدويره و أصبح أول بت (LSB)، أيضا فإن هذا البت قد ذهبت نسخة منه إلى CF كما هو واضح.

ROR : هذا الأمر هو اختصار لـ Rotate right أي تدوير الى اليمين.



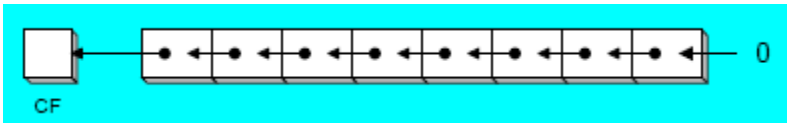
نفس الفكرة في حالة **ROL** لكن الفرق أن اتجاه الدوران هو لليمين. لاحظ ما يحدث عند تنفيذ أمر **ROR** CH,3 (هذا الأمر يعادل تنفيذ أمر **ROR** CH,1 ثلاث مرات متتالية):

الحالة الاصلية	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**1**
الدورة الاولى	1	0	0	1	1	1	1	**1**
الدورة الثانية	1	1	0	0	1	1	1	**1**
الدورة الاخيرة	**1**	**1**	**1**	**0**	**0**	**1**	**1**	**1**

CF :

1

SHL: هذا الأمر يقوم بعمل إزاحة لليسار (Shift Left). يتم إضافة 0 من اليمين (إلى LSB) ويذهب الـ MSB إلى CF.
أي هكذا :



فمثلا عند تنفيذ هذه التعليمة : SHL DL,1 يتم التالي:: Before

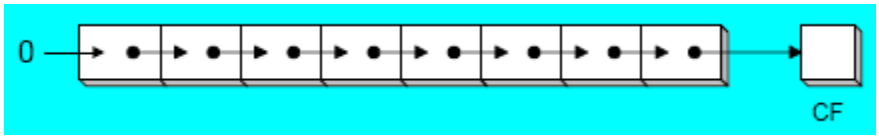
قبل		0	1	1	1	0	1	1	1
بعد	0	1	1	1	0	1	1	1	0
	CF								CF

ملاحظة: نقطة هامة في أوامر الإزاحة. إن تنفيذ أمر **SHL** لمرة واحدة يقوم بضرب العدد (الذي سيتم تدويره) بـ 2.
أنظر التالي :

```
SHL ax, 1 ; Equivalent to AX*2
SHL ax, 2 ; Equivalent to AX*4
SHL ax, 3 ; Equivalent to AX*8
etc...
```

لكن انتبه، فيجب أن يكون المسجل المستخدم في أمر **SHL** كافيا لاستيعاب نتيجة الضرب.

SHR : نفس المبدأ. لاحظ :



دعنا نرى هذا المثال : **SHR** AL,1. لاحظ كيف يخرج الـ LSB إلى الـ CF. وكيف يأتي 0 ليحل محل الـ MSB.

ملاحظة: إن تنفيذ أمر **SHR** لمرة واحدة يعني قسمة القيمة التي سيتم تدويرها على 2.

```
SHR ax, 1 ; Equivalent to AX/2
SHR ax, 2 ; Equivalent to AX/4
SHR ax, 3 ; Equivalent to AX/8
etc...
```

النقطة الاولى : هناك تعليمات قريبة لهذه التعليمات السابقة ك SAR و RCR و RCL.

النقطة الثانية : بعض التعليمات الاربع السابقة لها تصرفات خاصة في حالة الدوران مرة واحدة.

النقطة الثالثة : إن تنفيذ تعليمة مثل **SHL** ECX,32 هي مجرد مضيعة للوقت. فهذه سوف تجعل قيمة ECX صفرا. لأنه هناك 32 عملية إزاحة وكل عملية تقوم بإدخال 0 من اليمين، في النهاية ستستبدل الـ 32 بت في ECX بـ 32 صفرا. أيضا فتعليمة مثل **SHL** ECX,33 ستؤدي إلى نفس النتيجة. لختصار فإن الإزاحة لليمين أو لليسار يجب أن تتم بعدد مرات أقل من حجم المسجل الهدف (الا اذا كان هدفك التضليل مثلا).

النقطة الرابعة : بالمثل فإن تعليمة مثل **ROR** BX,16 هي أيضا مضيعة للوقت فتدوير المسجل BX 16 مرة يعني أنه سيعود إلى حالته الأولى. لاحظ أيضا أن **ROR** BX,17 تكافئ **ROR** BX,1 تماما. الفكرة نفسها تنطبق على **ROL**.

النقطة الخامسة : عملية القسمة على مضاعفات 2 تنفع في حالة كانت القيمة التي سيتم تدويرها هي unsigned أي بدون إشارة.. أما إن كانت ذات اشاراة فالنتيجة لن تكون صحيحة. في حالة قمت بعملية الإزاحة لليمين لمرة واحدة، أي قمت بعملية القسمة على 2، فإن كان هناك باقي فستجده في CF. لكن لو نفذت عملية الإزاحة (ألقسمة) أكثر من مرة فلن تستطيع الحصول على الباقي.

إن تنفيذ تعليمة مثل **SHL** AX,8 هو مكافئ لتنفيذ التعليمتين التاليتين :

```
MOV AH,AL
MOV AL,0
```

إن تنفيذ تعليمة مثل **SHR** BX,8 هو مكافئ لتنفيذ التعليمتين التاليتين : هو مكافئ لتنفيذ التعليمتين التاليتين:

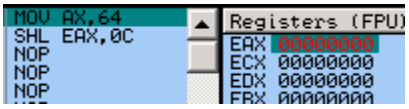
```
MOV BL,BH
MOV BH,0
```

مثال / انظر إلى هذه الصور المأخوذة من برنامج OLLY والتي توضح عملية تنفيذ هاتين التعليمتين :

```
MOV AX,64
SHL EAX,0C
```

لاحظ أنه في برامج التنقيح كـ OLLY فيفترض أن الأعداد بنظام HEX.

هذه هي الصورة الأولى توضح كيف قمت بفتح برنامج ما وغيرت أول تعليمتين فيه.



والآن F8 :

F8 مرة أخرى

أي أننا قمنا بضرب 64 h بـ ((2 مرفوعة للقوة C)) أي 64 h * 1000 h = 64000 h

أو بالعشري، ضربنا 100 بـ((2 مرفوعة للقوة 12)) أي 100 * 4096 = 409600

8.3.4.2 . المجموعة السابعة **CALL** & **RET:**

CALL : يتم استدعاء الدالة بمساعدة هذا الأمر. الأمر **CALL** يضع عنوان الرجوع في المكدس (stack). عنوان الرجوع بطول 32bit حسب البيئة.

الأمر **ret** n يستخدم هذا الأمر للرجوع من الدالة حيث يشير الرمز n يمثل عدد البايتات (Bytes) التي يجب "تنظيفها" في المكدس عند الرجوع من الدالة.

المقصود بعملية التنظيف هو القفز عن البايتات وذلك عن طريق تغيير مؤشر المكدس : المسجل ESP

إذا أردنا أن لا نقوم بعملية تنظيف بايتات في المكدس نسجل الأمر **ret** بدون n

```
CALL function
...
...
...
ret
```

وهكذا يكون الفصل الثاني قد انتهى بحمد الله وتوفيقه.


الباب
الأول
الفصل الثالث
لمحة سريعة عن برامج الـ
Reversing

3. الفصل الثالث : لمحة سريعة عن برامج RE

سنمر هنا على أهم البرامج والأدوات التي تستخدم في عالم الهندسة العكسية.

1.3 . OLLYDBG

هذا البرنامج هو الأكثر شهرة في مجال الـ Debugging (التنقيح) حيث أن شعبيته جعلت الكراكرز يتخلون عن برنامج w32dasm الذي كان شائعا في الماضي (وفي الحقيقة فالشركة المطورة توقفت عن تطويره أصلا).

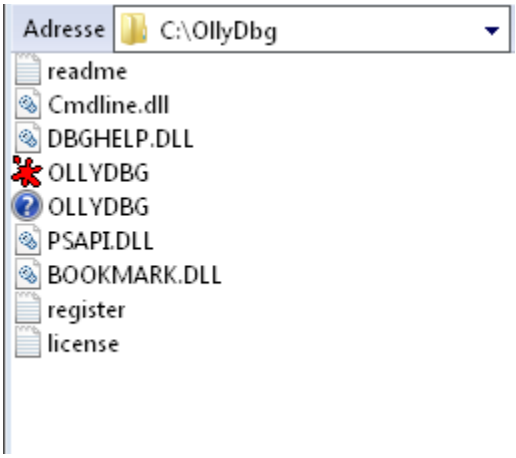
دعنا نلقي نظرة على البرنامج ككل. (سنشرح على الإصدار 1.10, علما ان النسخة 2.0 اصبحت الآن في اصدارها الفا 3, مما يعني انها على الابواب.)

3.1.1 . التحميل:

 OLLYDBG 

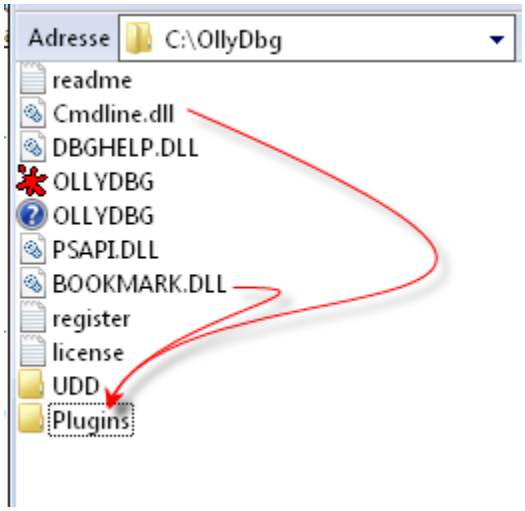
2.1.3 . التنصيب و اعداد المجلدات:

انشئ مجلدا جديدا (اسمه Ollydbg مثلا) و استخرج جميع الملفات الموجودة داخل ارشيف ال Zip الذي حملته من الرابط فوق:



الآن, انشئ مجلدين داخل السابق باسم : UDD و Plugins. هذا للتنظيم فقط.

- **UDD** : سيستعمله المنقح لحفظ ملفات تحليل البرامج التي ستنقحها.
- **Plugins** : سيستعمله المنقح لتحيل الاضافات المختلفة.

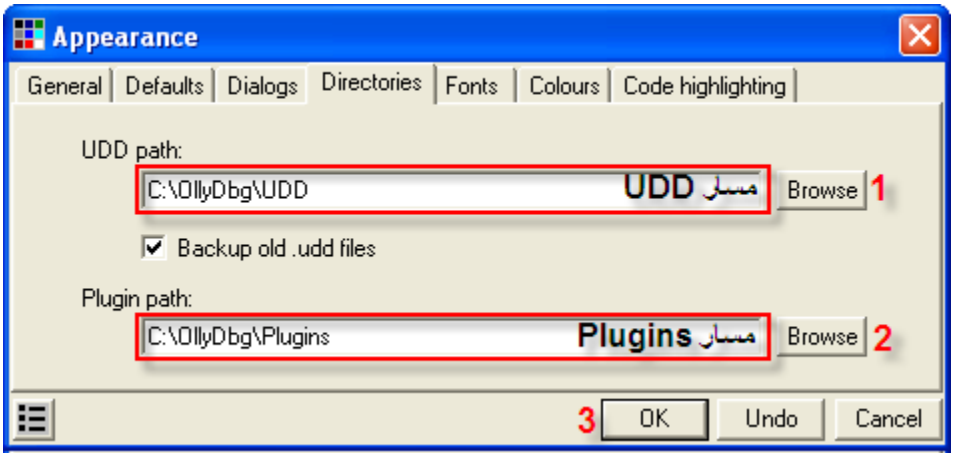


أستخرج ايضا الاضافات التالية الى المجلد : Plugins

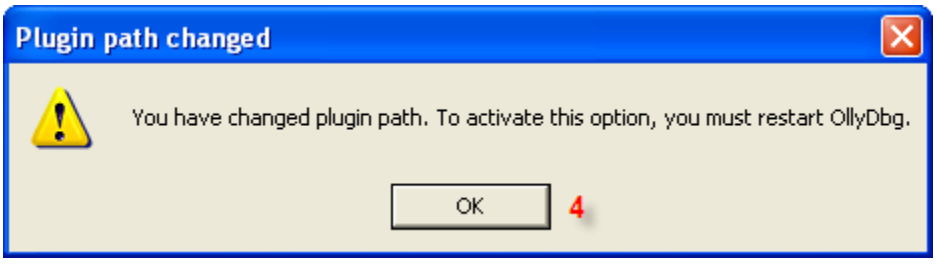
3.1.3 . اعدادات OLLYDBG الاولية:

اذا فتحت OllyDbg فستلاحظ انه لم يحمل الإضافات, و لكي يفعل ذلك, فلا بد من بعض الاعدادات.

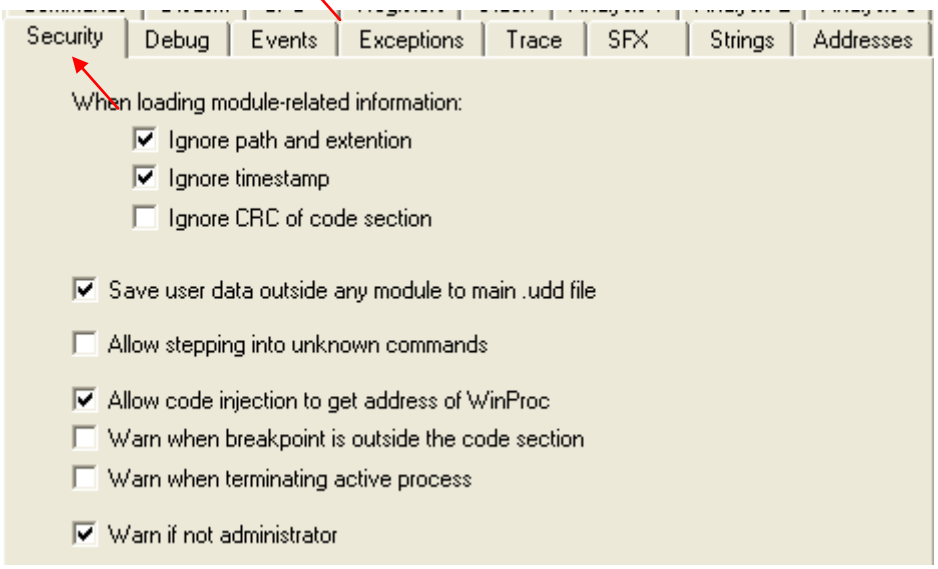
اضغط على الزر : لتفتح نافذة الاعدادات ثم توجه الى onglet الذي يحمل اسم Directories و اختر اعداداتك مثل الصورة اسفله:

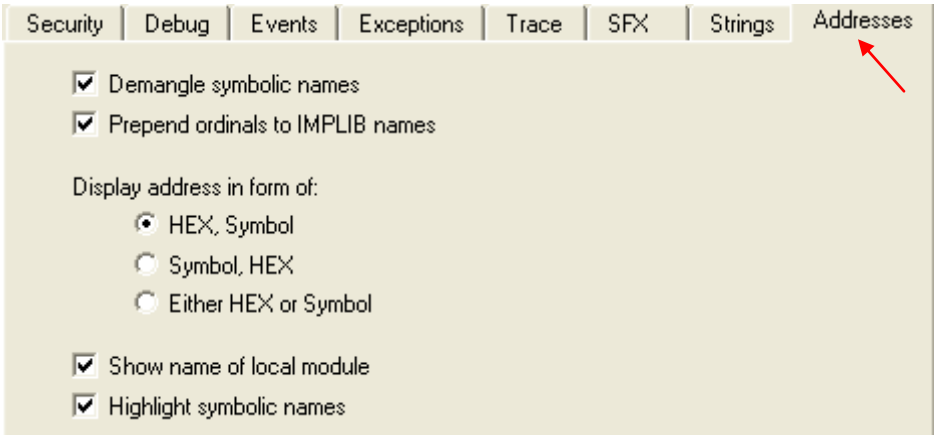


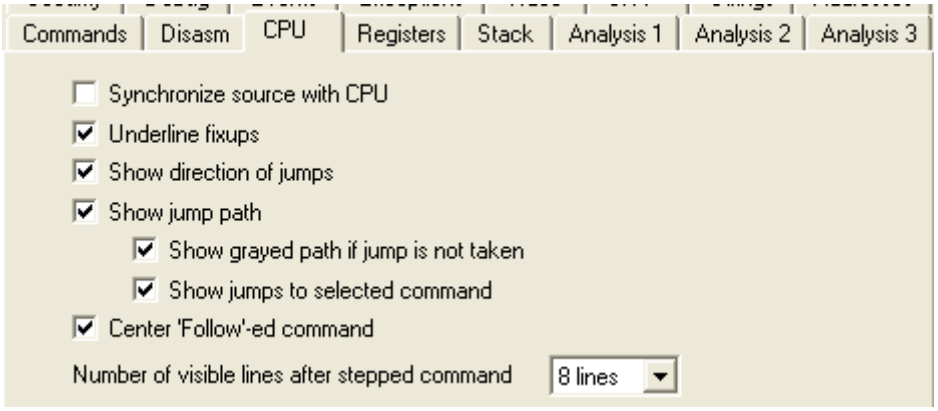
ثم:

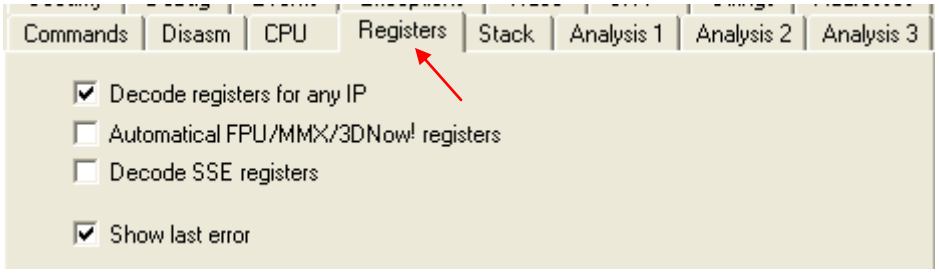


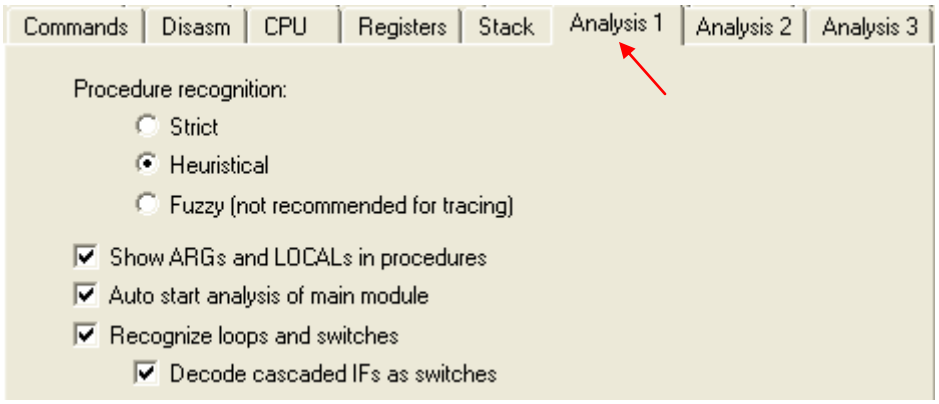
الآن, اضغط على الزر:  لتفتح نافذة اخرى للاعدادات و اختر اعداداتك كما في الصور الآتية:







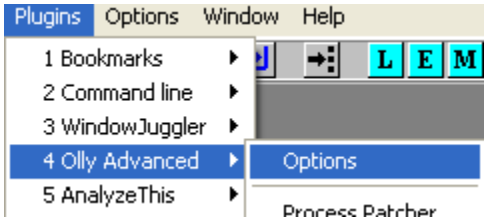




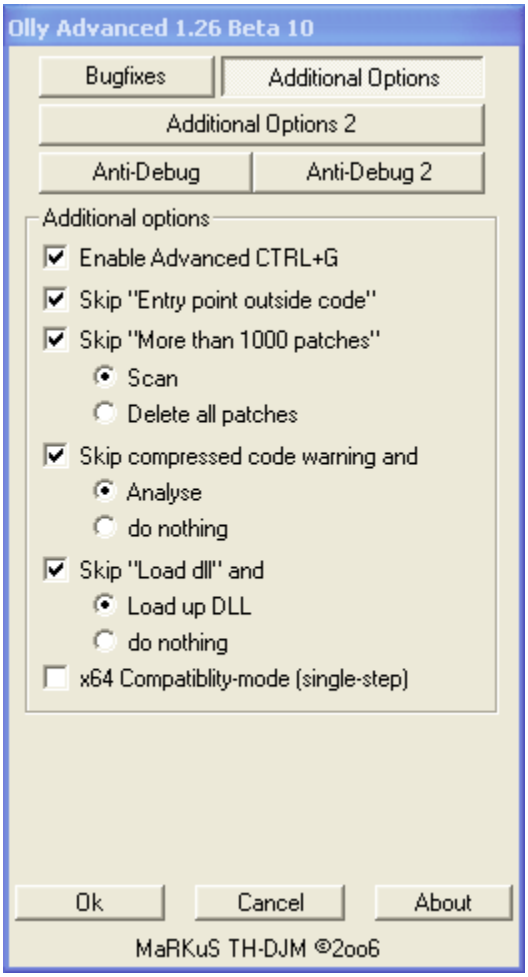
الاعدادات الاخرى ستأتي اهميتها مع الوقت و مع الدروس التي يحتويها هذا الكتاب, فقط اصبر و تابر.

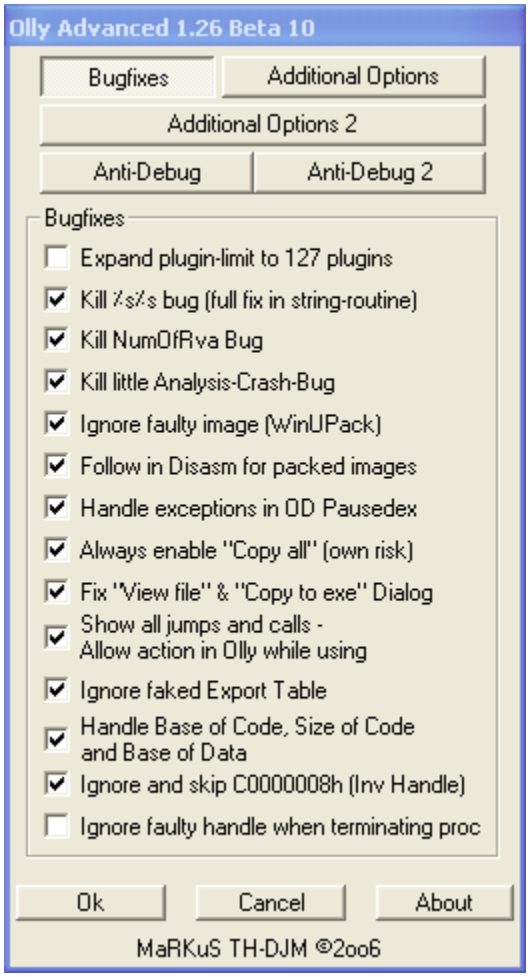
الآن اضغط أوكي ثم اعد تشغيل المنقح و ستلاحظ انه حمّل البلاغينز.

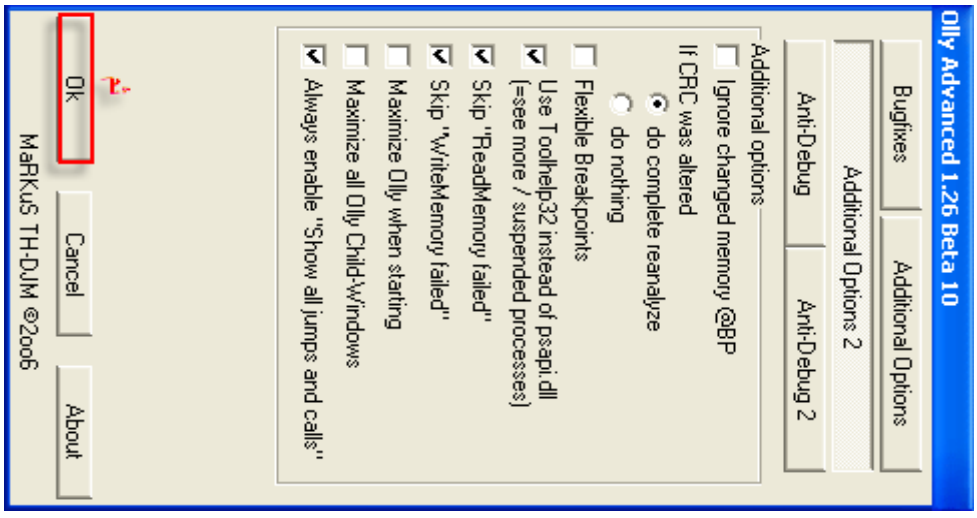
اذهب الى ال menu بلاغينز و اختر:



ثم اضبط الاعدادات كالتالي:

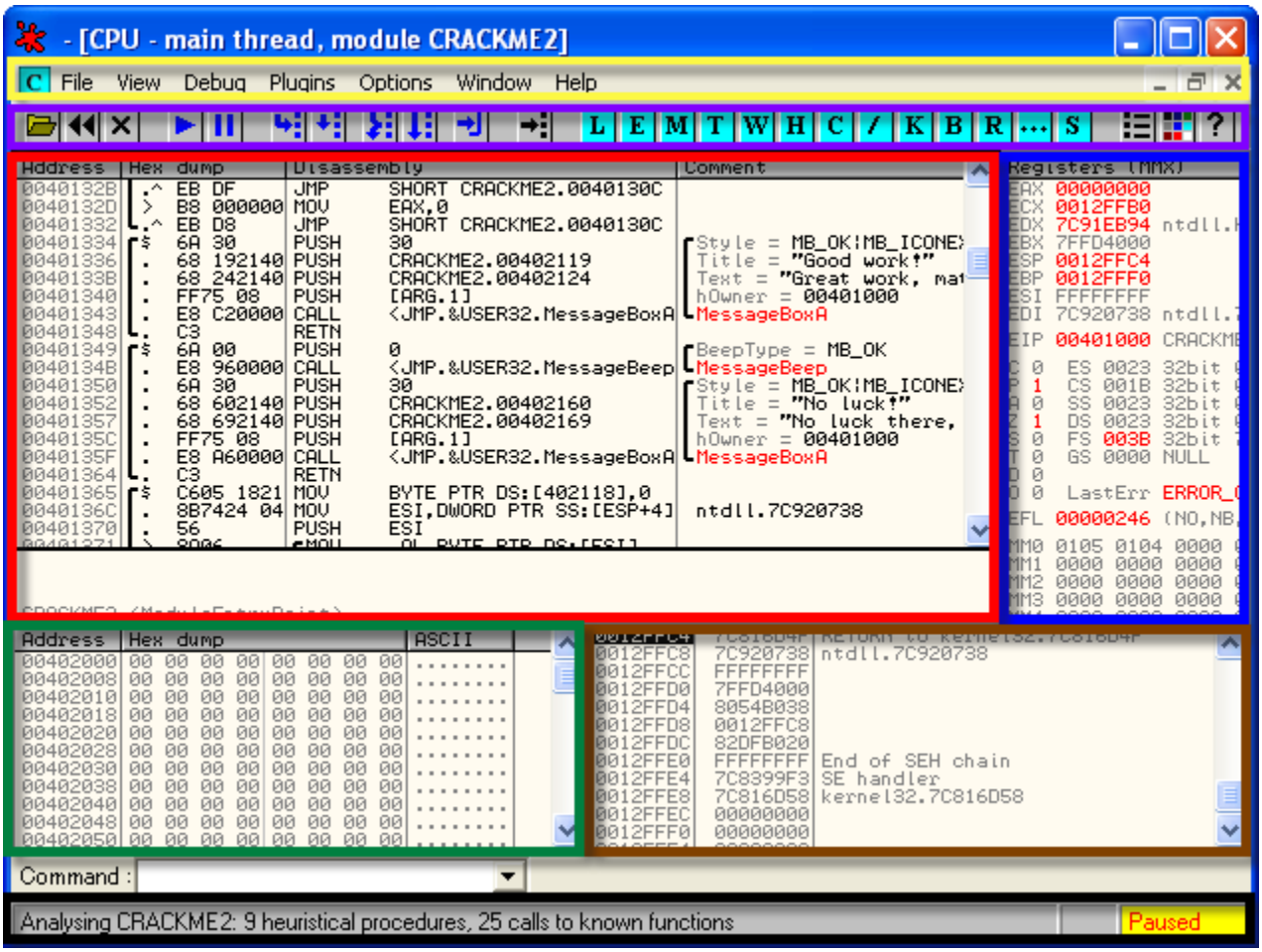

Olly Advanced 1.26 Beta 10
Bugfixes
Additional Options
Additional Options 2
Anti-Debug
Anti-Debug 2
Additional options
Enable Advanced CTRL+G
Skip "Entry point outside code"
Skip "More than 1000 patches"
Scan
Delete all patches
Skip compressed code warning and
Analyse
do nothing
Skip "Load dll" and
Load up DLL
do nothing
x64 Compatiblity-mode (single-step)
Ok
Cancel
About
MaRKuS TH-DJM ©2oo6


Olly Advanced 1.26 Beta 10
Bugfixes
Additional Options
Additional Options 2
Anti-Debug
Anti-Debug 2
Bugfixes
Expand plugin-limit to 127 plugins
Kill %s/%s bug (full fix in string-routine)
Kill NumOfRva Bug
Kill little Analysis-Crash-Bug
Ignore faulty image (WinUPack)
Follow in Disasm for packed images
Handle exceptions in OD Pausedex
Always enable "Copy all" (own risk)
Fix "View file" & "Copy to exe" Dialog
Show all jumps and calls -
Allow action in Olly while using
Ignore faked Export Table
Handle Base of Code, Size of Code
and Base of Data
Ignore and skip C0000008h (Inv Handle)
Ignore faulty handle when terminating proc
Ok
Cancel
About
MaRKuS TH-DJM ©2oo6


Olly Advanced 1.26 Beta 10
Bugfixes
Additional Options
Additional Options 2
Anti-Debug
Anti-Debug 2
Additional options
Ignore changed memory @BP
If CRC was altered
do complete reanalyze
do nothing
Flexible Breakpoints
Use Toolhelp32 instead of psapi.dll
(=see more / suspended processes)
Skip "ReadMemory failed"
Skip "WriteMemory failed"
Maximize Olly when starting
Maximize all Olly Child-Windows
Always enable "Show all jumps and calls"
Ok
Cancel
About
MaRKuS TH-DJM ©2oo6

3.1.4 . نوافذ OLLYDBG

هناك الكثير من النوافذ في هذا المنقح العملاق, و هو الشئ الذي يصيب المبتدئ بالذهول, لكن الامر و بعد قراءتك لهذه الاسطر, سيصبح اكثر و ضوحا و اقل ضبابية بإذن الله.



في الإطـــــــار الاصفر : شريط القوائم.

في الإطـــــار البنفسجي : شريط ازرار التحكم و الاعدادات

في الإطـــــــار الاحمر : نافذة الديزاسمبلي (Disassembly)

في الإطـــــــار الأزرق : نافذة المسجلات

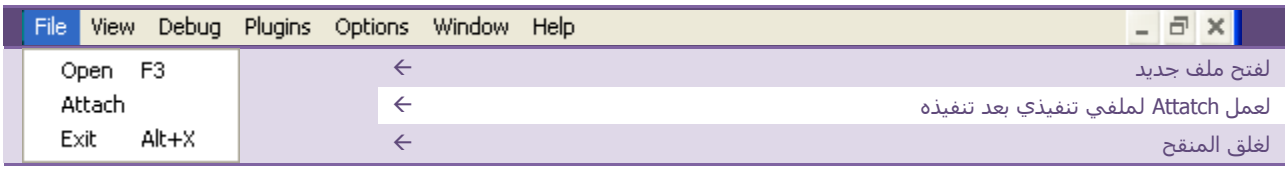
في الإطـــــــار الأخضر : نافذة الذاكرة (Dump Memory Window)

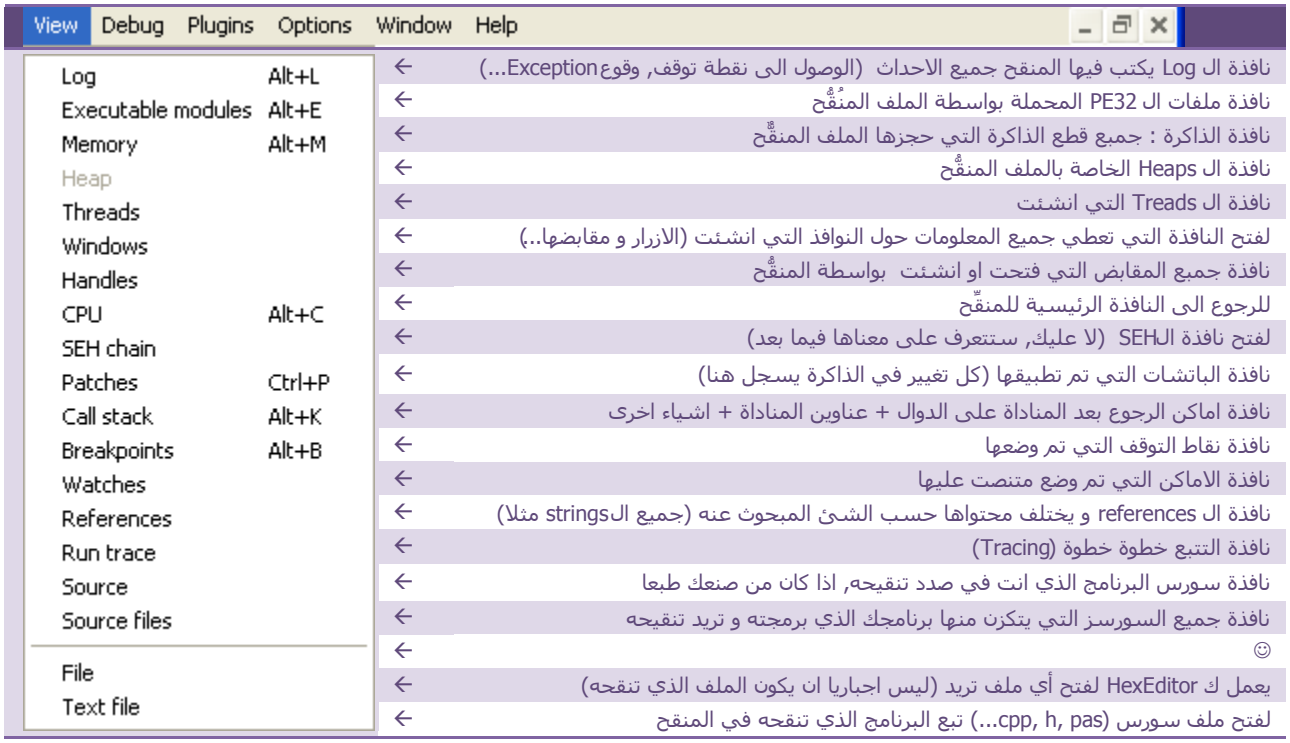
في الإطـــــــار البني : نافذة المكدس

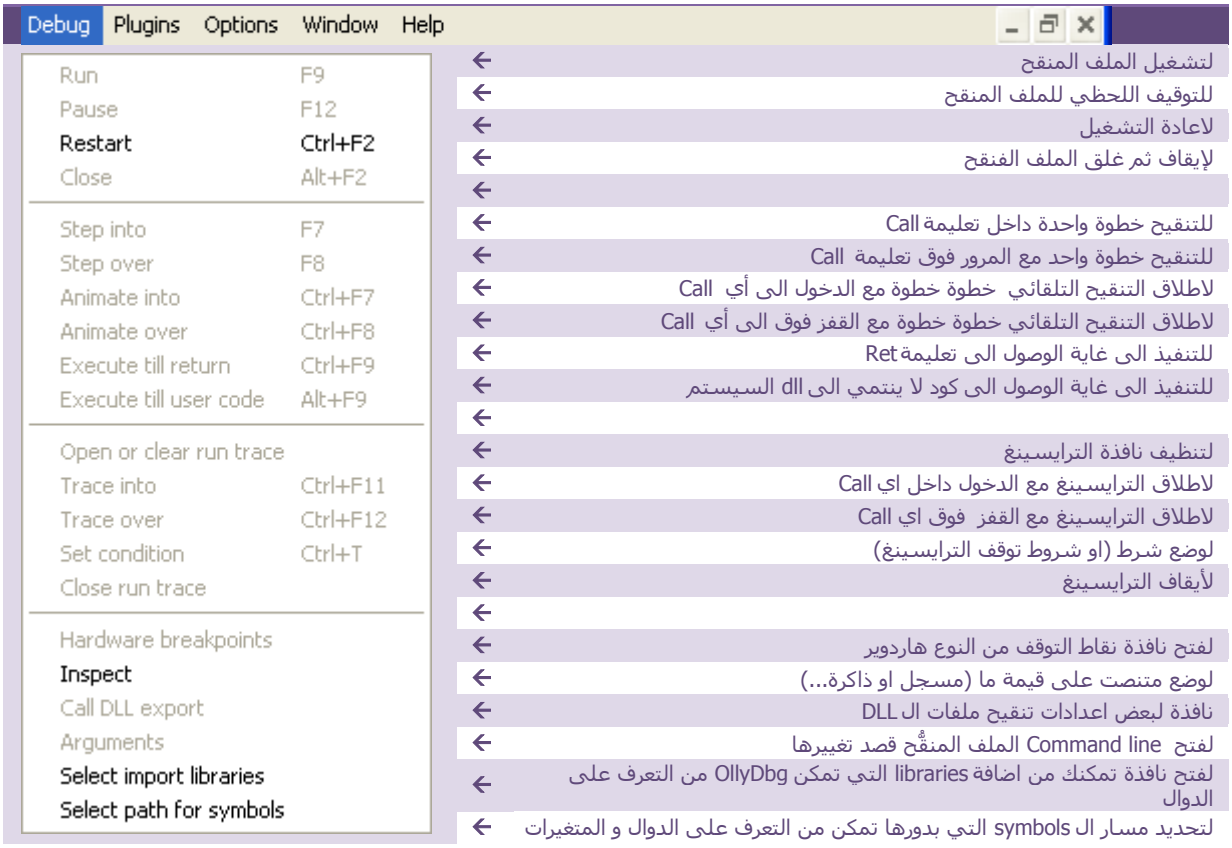
في الإطـــــــار الاسود : Status bar

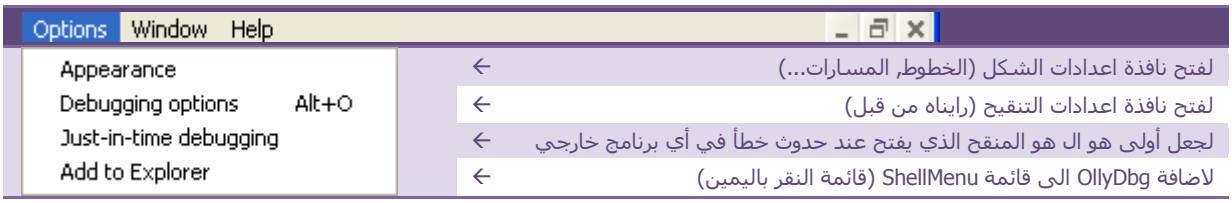
3.1.4.1 . شريط القوائم

File View Debug Plugins Options Window Help



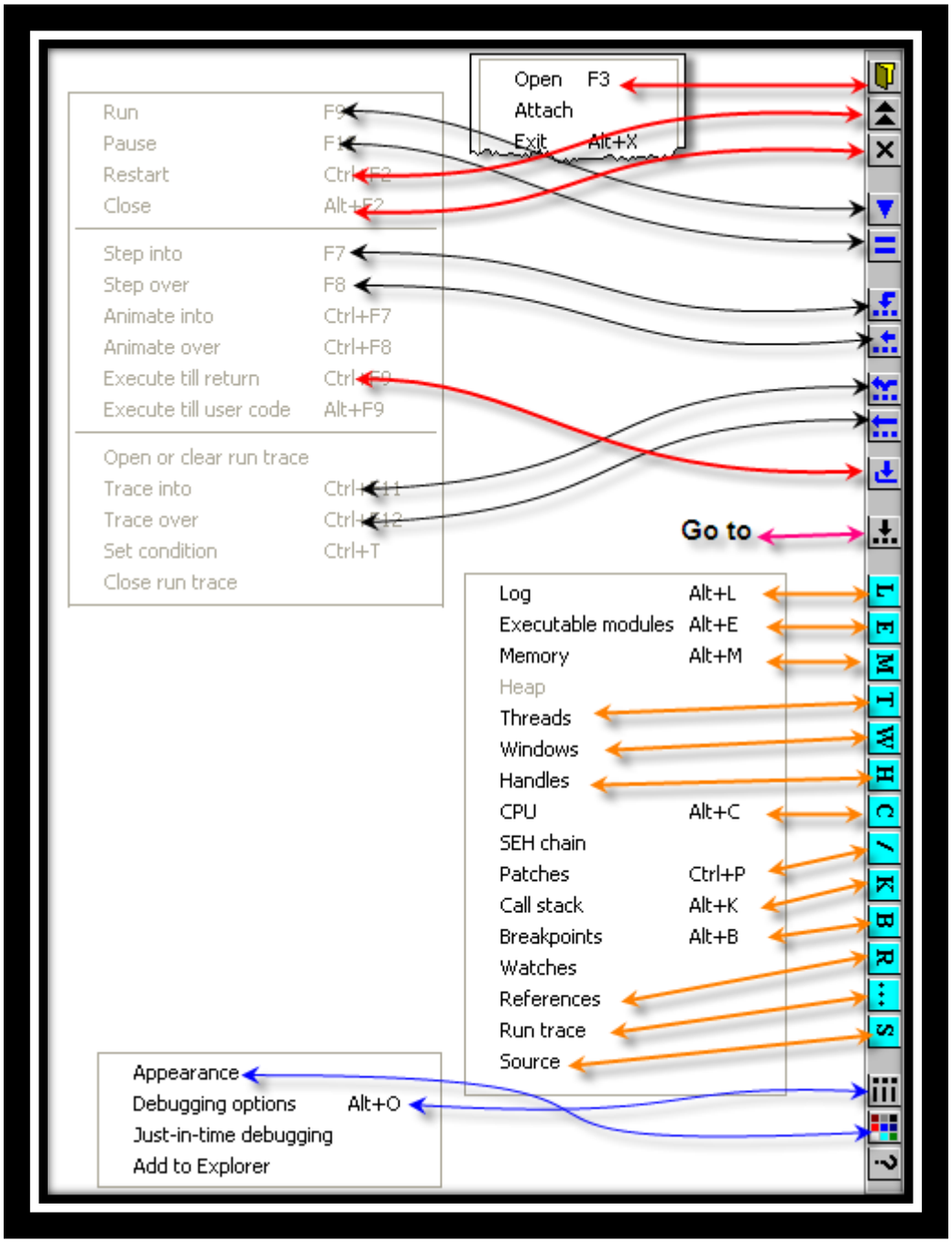



Debug Plugins Options Window Help
Run F9
Pause F12
Restart Ctrl+F2
Close Alt+F2
Step into F7
Step over F8
Animate into Ctrl+F7
Animate over Ctrl+F8
Execute till return Ctrl+F9
Execute till user code Alt+F9
Open or clear run trace
Trace into Ctrl+F11
Trace over Ctrl+F12
Set condition Ctrl+T
Close run trace
Hardware breakpoints
Inspect
Call DLL export
Arguments
Select import libraries
Select path for symbols
لتشغيل الملف المنقح
للتوقيف اللحظي للملف المنقح
لاعادة التشغيل
لإيقاف ثم غلق الملف الفنقح
للتنقيح خطوة واحدة داخل تعليمة Call
للتنقيح خطوة واحد مع المرور فوق تعليمة Call
لاطلاق التنقيح التلقائي خطوة خطوة مع الدخول الى أي Call
لاطلاق التنقيح التلقائي خطوة خطوة مع القفز فوق الى أي Call
للتنفيذ الى غاية الوصول الى تعليمة Ret
للتنفيذ الى غاية الوصول الى كود لا ينتمي الى dll السيستم
لتنظيف نافذة الترايسينغ
لاطلاق الترايسينغ مع الدخول داخل اي Call
لاطلاق الترايسينغ مع القفز فوق اي Call
لوضع شرط (او شروط توقف الترايسينغ)
لأيقاف الترايسينغ
لفتح نافذة نقاط التوقف من النوع هاردوير
لوضع متنصت على قيمة ما (مسجل او ذاكرة...)
نافذة لبعض اعدادات تنقيح ملفات ال DLL
لفتح Command line الملف المنقَّح قصد تغييرها
لفتح نافذة تمكنك من اضافة libraries التي تمكن OllyDbg من التعرف على الدوال
لتحديد مسار ال symbols التي بدورها تمكن من التعرف على الدوال و المتغيرات

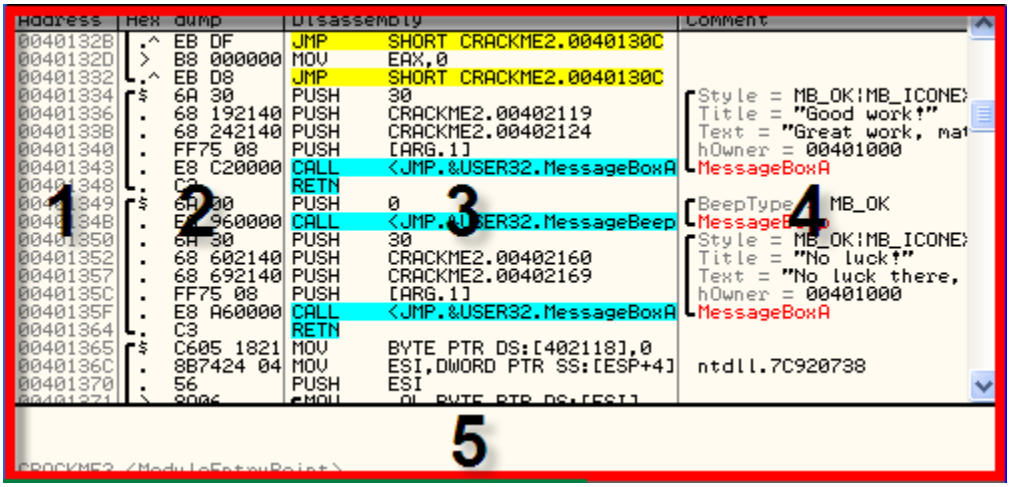

Options Window Help
Appearance
Debugging options Alt+O
Just-in-time debugging
Add to Explorer
لفتح نافذة اعدادات الشكل (الخطوط, المسارات...)
لفتح نافذة اعدادات التنقيح (راينا من قبل)
لجعل أولى هو ال هو المنقح الذي يفتح عند حدوث خطأ في أي برنامج خارجي
لاضافة OllyDbg الى قائمة ShellMenu (قائمة النقر باليمين)

3.1.4.2 . شريط الأزرار

هو فقط لربح الوقت و كل الازرار انما تقوم بما شرحناه سابقا و اليكم التوافق ازرار – قوائم:

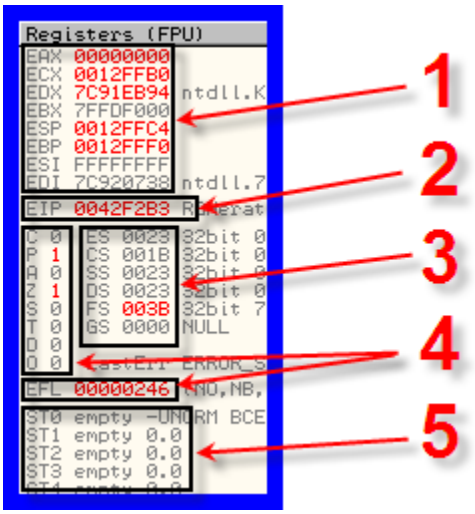


3.1.4.3 . نافذة الديزاسمبلي (DISASSEMBLY)



1 - عمود العناوين : و فيه ترى عناوين تعليمات الاسمبلي
2 - عمود التمثيل السداسي عشري لتعليمات الاسمبلي (لغة الآلة)
3 - عمود التعليمات و الاوامر (لغة الاسمبلي بالكيفية التي يمكن للانسان فهمها)
4 - عمود التعليقات/السورس/البروفايل (ترقب شرح هذه الاشياء في النسخة Intermediate من الكتاب)
5 - قسم المعلومات (اذا ضبطت OllyDbg كما في الصور فوق,فستلاحظ المعلومات التي يوفرها هذا القسم عند التنقيح)

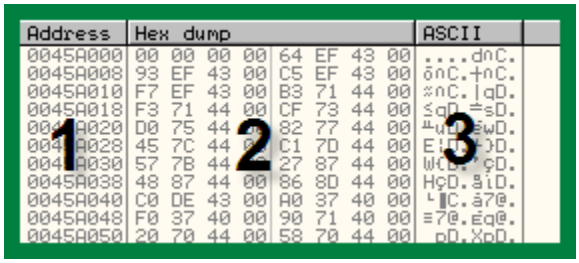
3.1.4.4 . نافذة المسجلات



1. المسجلات ذات الاستعمال العام
2. المسجل EIP الذي يشير دائما و أبدا الى التعليمة التي يتم تنفيذها حاليا.

3. المسجلات من النوع Segment registers
4. الرايات في المستطيل العمودي و تشاهد في المستطيل الافقي القيمة العامة للمسجل EFlags.
5. مسجلات من نوع آخر

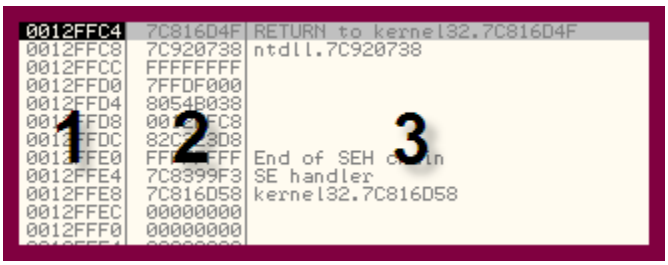
3.1.4.5 . نافذة الذاكرة (DUMP MEMORY WINDOW)



تختلف ال views التي يمكن رؤية هذه النافذة عليها لكن اهمها بالنسبة للمبتدئ هي هذه النظرة فوق (نظرة مشابهة للمحرر الهيكس). وتمكنك متابعة أي عنوان من الذاكرة يحتوي اشياء مهمة في هذه النافذة (سيريال, اسم, كتابة...) و سيأتي كيفية المتابعة فيما بعد في هذا الكتاب.

1 - عمود العناوين (VA و ليس أوفسيت كما في محررات الهيكس)
2 - عمود القيم الهيكسية
3 - عمود القيم ال ASCII القابلة لها

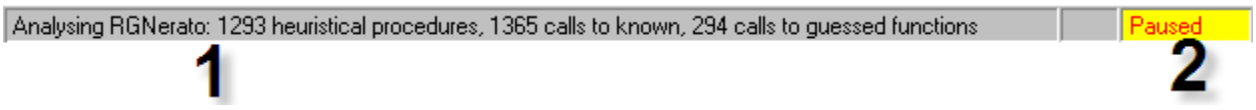
3.1.4.6 . نافذة المكدس



نافذة المكدس ذات اهمية كبيرة و تحتوي:

1 - عمود العناوين
2 - القيمة التي يحتويها كل عنوان و قيمتها 32 بت (هذا راجع لل Alignement الذي تعتمده الملفات التنفيذية)

3.1.4.7 . STATUS BAR



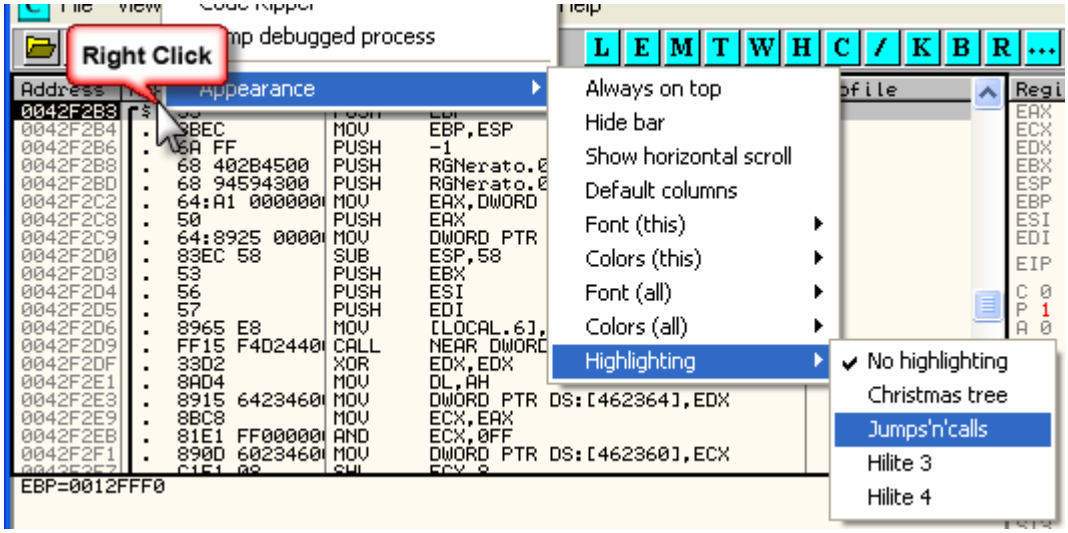
تحتوي قسمين مهمين:

1 - قسم المعلومات و فيه تظهر المعلومات المختلفو و الاخطاء التي قد تحدث (نفس هذه الامعلومات يمكنك الاظلاع عليها متى شئت في نافذة ال Log.
2 - هذا القسم يبين حالة البرنامج المنقّح, هل هو يعمل ام متوقف.

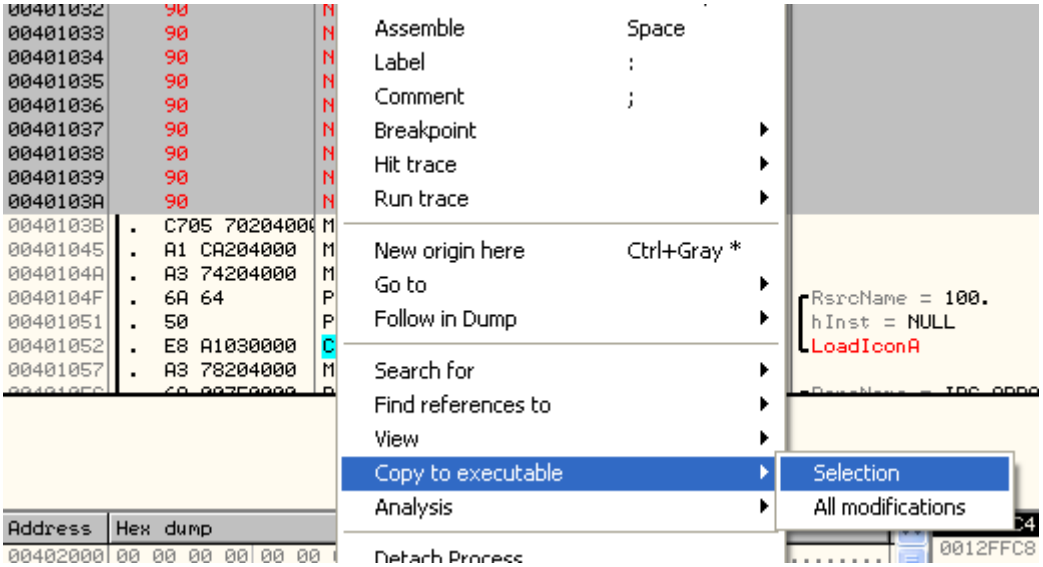
3.1.5 . اعدادات اخرى و معلومات متفرقة:

لرؤية اكثر وضوحا, اعمل ما يلي: (المعلومات التالية

1 - اضغط يمين في أي منطقة من نافذة الديزاسمبلي ثم



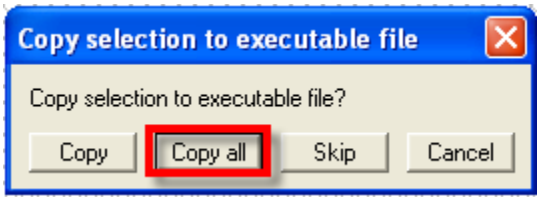
2 - لوضع نقطة توقف :
   a. اما اضغط F2 أو
   b. Double click على عمود التمثيل السداسي عشري (العمود الثاني من نافذة الدسزاسمبلي)
3 - لتغيير تعليمة ما, اشر عليها ثم:
   a. اما اضغط على الزر Space
   b. او ابدأ الكتابة فورا
4 - احفظ التغييرات لديك خياران:
   a. اما انك تريد حفظ تغيير ما في مكان واحد (او عدة اماكن متقاربة جدا), في هذه الحالة عليك بالتضليل على مكان (اماكن) التغيير ثم ضغطة يمين:



ثم

b. واما انك تريد حفظ تعديلات كثيرة متباعدة, و في هذه الحالة لا ينفعك الا (دون تضليل على الاسطر):

ثم



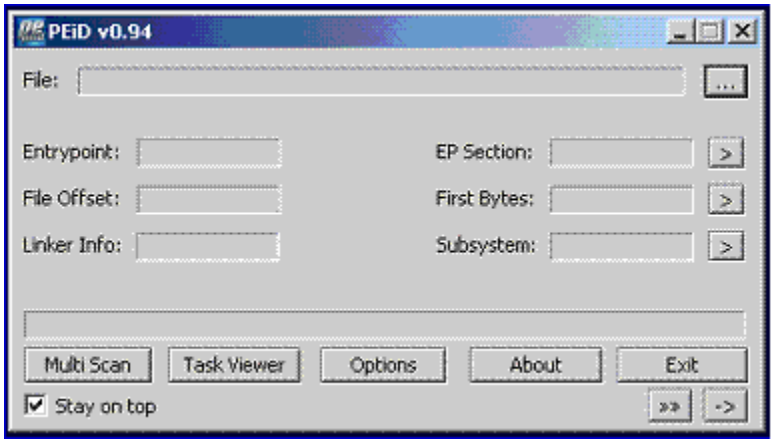
2.3. برنامج PEID V0.94

هذا البرنامج يستخدم للفحص، حيث أنه يكشف لك هل البرنامج محمي أم لا، فإن كان محميا سيحدد لك نوع واصدار الحماية وان لم يكن فسيحدد لك اللغة البرمجية التي كتب بها البرنامج.

يمكن تحميله من هذا الموقع.

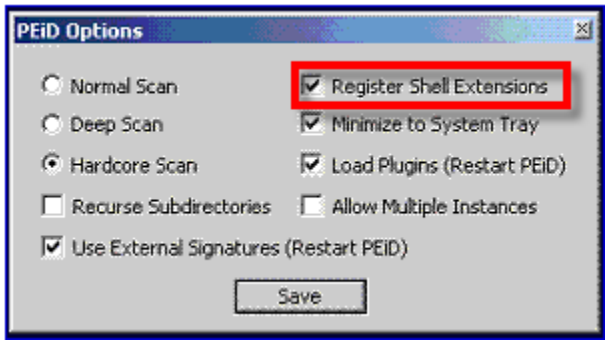
http://www.peid.info/

والآن قم بتشغيل البرنامج :



نريد الآن تعديل بعض الخيارات بحيث لا نضطر إلى فتح البرنامج ثم الضغط على browse ➔ open في كل مرة نريد فيها فحص ملف ما، وعوضا عن ذلك سنكون قادرين على فحص البرنامج بمجرد الضغط عليه بالزر الأيمن للفأرة.

اضغط على زر options ثم غير في الخيارات لتصبح كما في الصورة التالية :



والآن لفحص برنامج اضغط باليمين و:

كما ترى فبمجرد اختيار الأمر scan with PEiD نكون قد انتهينا ☺

وهكذا ينتهي الفصل الثالث.

وبانتهاء الفصل الثالث ينتهي الباب الأول، بحمد الله.


الباب الثاني
الفصل الأول : Serial Fishing & General RE
الفصل الثاني : patching
الفصل الثالث : loaders
الفصل الرابع : Code Injection
الفصل الخامس : Cryptography
الفصل السادس : KeyFile-Making
الفصل السابع : Keygening
الفصل الثامن : BruteForcing
الفصل التاسع : Miscellaneous
الفصل العاشر : Reversing « not native »languges


الباب
الثاني
الفصل الأول
Serial Fishing & General Re

1. الفصل الأول : SERIAL FISHING & GENERAL REVERSING

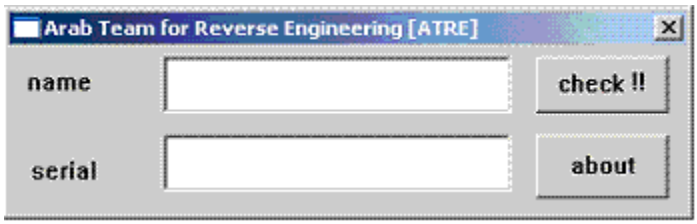
في هذا الفصل سنتعلم أولى طرق الهندسة العكسية والمسماة serial fishing. تقوم هذه الطريقة على إدخال يوزرنيم وباسوورد (بالطبع سندخل باس وورد خطأ. لو كنا نعرف الباس وورد الصحيح عندها لما كان هناك داع للهندسة العكسية !!!). بعد ذلك نقوم بالبحث في البرنامج عن النقطة التي يتم عندها مقارنة السيريال الصحيح بالخطأ. كل ما علينا هو تسجيل السيريال الصحيح و أنتهينا !!.

حسنا لنبدأ بأول مثال على الـ serial fishing

1.1. المثال الأول

ستجد في مجلد الضحايا برنامجا اسمه ch2_sec1_1

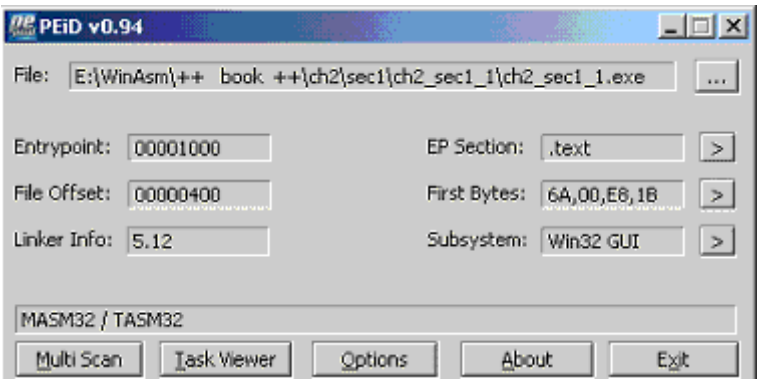
الخطوة الأولى هي تشغيل البرنامج وتفحصه :



قم بكتابة أي اسم وليكن at4re وأي سيريال وليكن 123456 ثم اضغط على check.

احفظ النص الذي تراه أمامك.

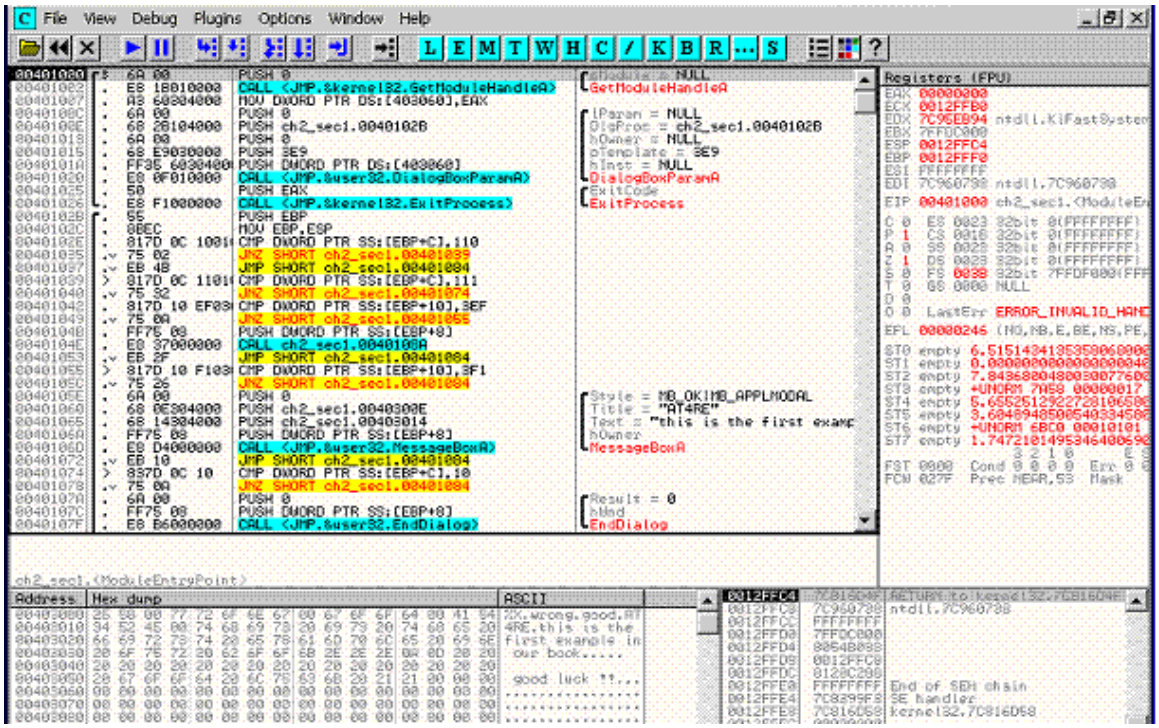
الخطوة التالية هي فحص البرنامج. هناك عدة برامج فحص لكن اليوم سنستخدم PEiD v 0.94. و بعد فحص البرنامج سنحصل على النتيجة التالية :



كما هو واضح فالبرنامج تمت برمجته بلغة Assembly 32.

البرامج المبرمجة أصلا بالاسمبلي تسهل المهمة على الكراكر لأنك ستتفحصها كما هي بدون تغيير. أما أن كان البرنامج مكتوبا بلغة من لغات المستوى العلوي High Level Language عندها سيقوم olly بتحويل البرنامج من HLL (كلغة الـ c++ مثلا) إلى لغة الاسمبلي مما يجعل البرنامج أكثر تعقيدا وغموضا.

الخطوة الثالثة هي فتح البرنامج باستخدام OllyDBG وعند عمل ذلك سنحصل على :



والآن دعنا نفكر. ما الذي نريد عمله؟

بما أن البرنامج يقوم بعمل مقارنة بين السيريال الصحيح والأخر الخطأ إذن و بكل بساطة نريد رؤية السريال الصحيح ☺

إذن دعنا نتفحص الأوامر ونرى ما يوجد لدينا. وقبل أن أكمل دعني أخبرك بالدوال المشهورة المستخدمة لجلب نص من مربع نص (أي التي تأخذ ما تكتبه في المربع الأبيض بجانب كلمة user و serial)

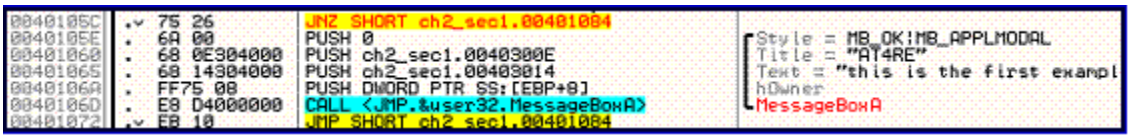
GetDlgItemTextA

GetWindowTextA

العديد من الدوال في حقيقة الامر، توجد لها نسختان:
- نسخة تتعامل مع ال Ascii strings و تلحق بالدالة في هذه الحالة الحرف A (كناية عن ASCII)
- و نسخة تتعامل مع ال Unicode strings و يلحق بها في هذه الحالة الحرف W (كناية عن Wide char)

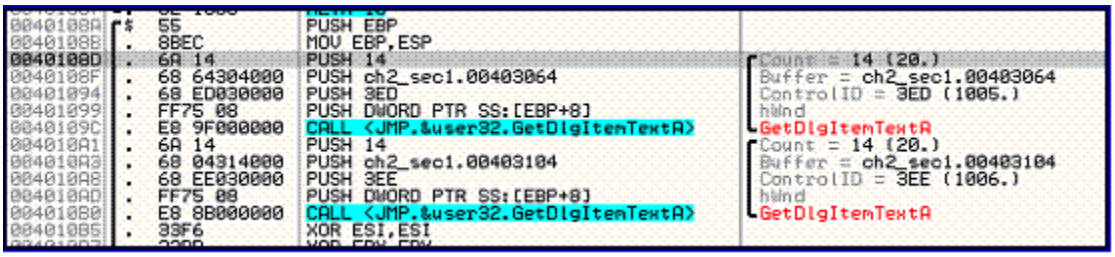
بالإضافة لذلك، تهمنا الدالة **MessageBoxA** فهي المسؤولة عن إخراج بعض انواع الرسائل إليك وطبعا نحن عادة مهتمون برسالة الخطأ.

إذن انزل للأسفل قليلا. هناك دالة **MessageBoxA**. انظر :



يبدوا أن هذه هي الرسالة التي تخرج عند الضغط على زر about. لا بأس لا يهمنا ذلك فلنتابع.

انظر :

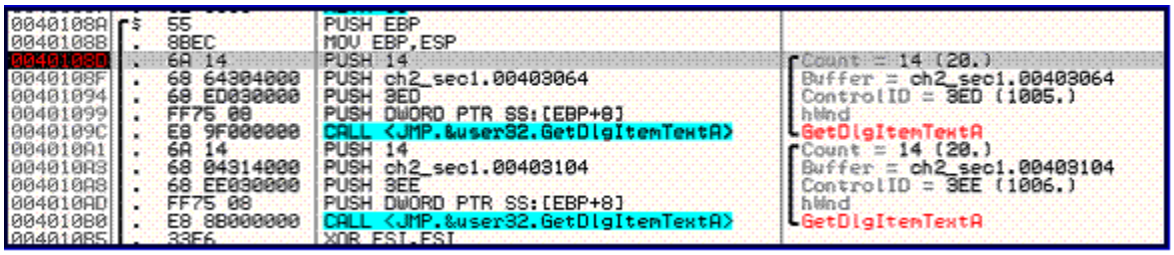


دالتان من نوع **GetDlgItemTextA** وواضح أن إحداهما لجلب اليوزرنيم والأخرى للباسوورد.

في هذا الدرس بما أننا مازلنا مبتدئين فلن أشرح البارامترات المتعلقة بالدالة والتي تراها بالصورة ضمن مجموعة واحدة. لكن يمكنك بدون معرفة ماهية تلك البارامترات أن تعرف أي الدالتين مسؤولة عن اليوزر وايهما مسؤولة عن الباسوورد. كيف؟

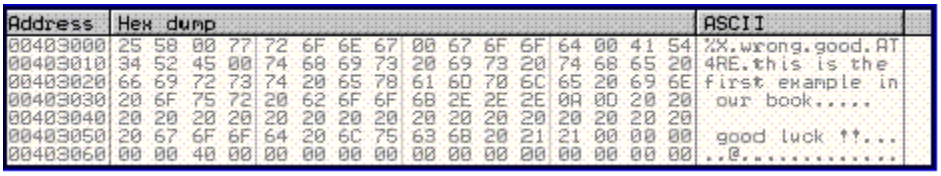
ضع نقطة توقف عند أول أمر في الدالة الأولى. أي عند العنوان 40108D، وذلك بالضغط على زر F2. والآن قم بتشغيل البرنامج وذلك بالضغط على زر run كما في الصورة :

أو بالضغط على زر F9. حسنا بعد تشغيل البرنامج ستخرج لك شاشة، ادخل اليوزرنيم at4re والباسوورد 123456 و الآن اضغط على check. ماذا تلاحظ؟ لقد عدنا إلى olly :



جميل والآن، لنقم بعملية التتبع. اضغط على F8 واستمر حتى نخرج من أول دالة (أي أننا سنضغط F8 لـ 5 مرات)

لكن انتظر، دعنا أولا نرى ماذا يوجد في قسم الـ hex dump :

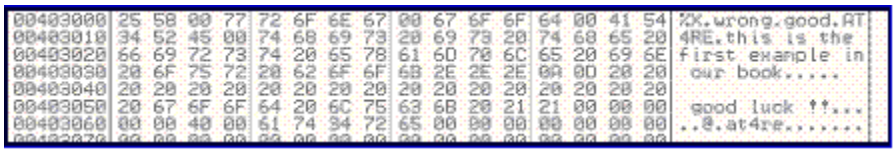


حسنا. والآن اضغط على F8 (5مرات). النتيجة :

كما ترى فإن eax=5 وهو عدد المحارف التي تم إدخالها في خانة اليوزرنيم. إذن نستنتج أن أول دالة خاصة بجلب اليوزرنيم.

كلمة محارف هي كلمة عامة تشمل : الأحرف، الأرقام والرموز
هذه النتيجة مهمة جدا وسنستخدمها كثيرا. معظم الدوال بعدما يتم تنفيذها فإن القيمة المرجعة يتم إرجاعها في eax. في حالتنا، أي في حالة الدالة **GetDlgItemTextA** فالقيمة المرجعة هي عدد المحارف المدخلة.

والآن لنر ماذا حصل في قسم الـ hex dump :



لعلك ترى أن هناك كلمة تم إضافتها (في آخر سطر). وهي at4re أي اليوزرنيم المدخل وهذا دليل آخر على أن أول دالة خاصة باليوزرنيم.

نفس الشيء بالنسبة للدالة الثانية فبعد التتبع سترى أن eax=6 وهو عدد المحارف المدخلة في خانة السيريال. بالمثل ستجد 123456 مخزنة في قسم hex dump أسفل الـ at4re بعدة أسطر.

على الرغم من أن كل ما ورد في السابق ليس له علاقة مباشرة بالطريقة التي سنحصل فيها على السيريال الصحيح، إلا أن المعلومات التي تم ذكرها تعتبر أساسية كي تهيئ لعالم الهندسة العكسية.

والآن سنستمر في التتبع. سنمر على عدة دوال, و بما أن هذا الدرس هو الأول لك فقد فضلت أن يكون مبسطا قدر الإمكان لذا لن أشرح جميع الدوال الموجودة وسأترك ذلك للدرس القادم، كل ما نريده هو البحث على دالة المقارنة والمسماة lstrcmp

أعتقد أنك رأيتها. إنها موجودة بالأسفل. كل ما علينا عمله هو التتبع بـ f8 حتى نصل إليها.

كما ترى ها هي دالة المقارنة lstrcmpA. وهي تجري مقارنة بين 123456 و 1E0. إذن؟ الأمر واضح : السيريال الصحيح هو 1E0. يمكن تجربة اليوزر at4re والباسوورد 1E0

النتيجة؟ لقد نجحنا. مبروك ☺

كما ترى لو استمررت في التتبع فستصل إلى الأمر 401103 short **jnz** والذي يعني أنه إذا لم تتحقق المساواة بين السيريال المدخل و الآخر الصحيح, فسيذهب بك إلى القمر (أقصد إلى رسالة ☺ wrong) وهذا واضح. أما إذا تحققت المساواة فلن يتم تنفيذ القفزة وسنذهب إلى رسالة good :).

آمل أن تكون قد استفدت من هذا المثال. ألمثال الثاني سيكون عن نفس البرنامج لكن مع مزيد من التعمق.

2.1 . المثال الثاني

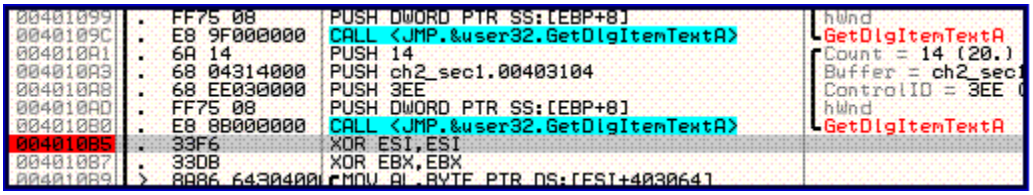
سنتحدث عن نفس المثال الذي تحدثنا عنه في الدرس السابق. لكن اليوم سنتناول أمورا جديدة لم نتكلم عنها في الدرس السابق. لذا قم بفتح الضحية (ch2_sec1_1) باستخدام olly.

ما الذي نريد الوصول إليه؟ اليوم نريد أن نعرف لماذا حصلنا على السيريال 1E0 عندما أدخلنا اليوزر at4re؟؟؟ إذن بعد فتح البرنامج سنصل إلى هنا.

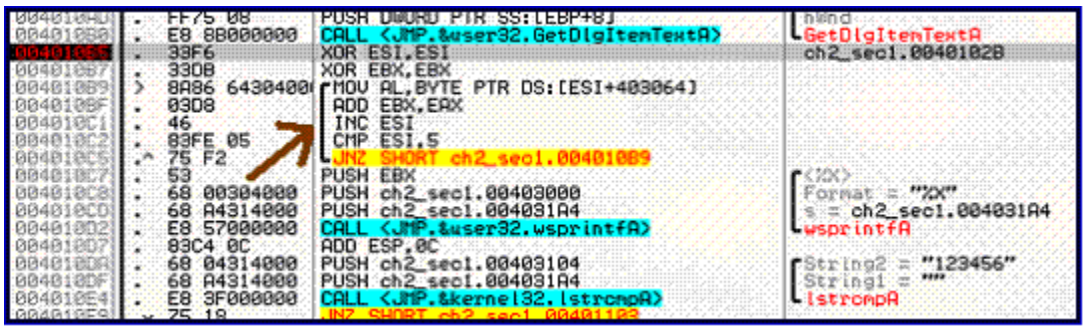
والآن نريد أن نعرف أين توجد خوارزمية التشفير؟ أين هو الكود المسؤول عن توليد السيريال الصحيح؟ بكل تأكيد لا يمكن أن توجد خوارزمية توليد السيريال الصحيح قبل أن يتم جلب اليوزرنيم إلى البرنامج. إذن, نحن على يقين أن الخوارزمية المطلوبة موجودة بعد أول دالة GetDlgItemTextA. لكن بما أن هذه الدالة متبوعة بدالة أخرى مشابهة إذن ما نبحث عنه موجود بعد هاتين الدالتين.

ليس معنى كلامي أنه يجب أن تكون الخورازمية بعد تلك الدالتين مباشرة. قد تكون بعدهما لكن ليس مباشرة، أي قد توجد دالة ما تؤدي إحدى الوظائف، يتبعها خوارزمية التوليد.

إذن ضع نقطة توقف بعد دالة GetDlgItemTextA الثانية مباشرة كما بالصورة (اذهب هنا لمعرفة كيفية وضع نقطة توقف):



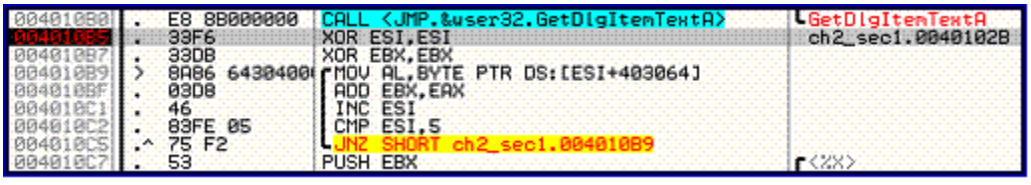
والآن شغل البرنامج (F9) ثم أدخل at4re و 123456 ثم اضغط الزر check وستلاحظ أنك عدت إلى Olly.



الحلقات (Loops) في OllyDbg و التي هي ترجمة برمجية للاوامر FOR و WHILE اذا تعرف عليها OllyDbg فهو يمثلها بخط رأسي سميك بجانب الأوامر التابعة للحلقة (يوجد سهم يشير إليها)

إذن من المرجح أن تكون تلك الحلقة جزءًا من الخوارزمية التي لطالما بحثنا عنها. ومما يؤكد ذلك هو وجود دالة lstrcmpA بعد دالة wsprintfA. أي أننا قد وصلنا إلى نهاية المطاف !

حسنا. لنتتبع جميع الأوامر من نهاية دالة GetDlgItemTextA الثانية وحتى نهاية الحلقة (الخوارزمية).



أول أمر (عند العنوان 4010B5) هو **XOR** ESI,ESI. يجب عليك قبل أن تكمل الدرس أن تكون قد قرأت القسم الذي يشرح التعليمات المنطقية **AND**, OR, NOT & **XOR** لأنني لن أشرحها هنا.

وكما تعلم فإنك عندما تقوم بعمل عملية **XOR** لنفس المتغير (أي مثلا 0 **XOR** 0 أو 1 **XOR** 1) فالنتيجة دوما صفر. إذن كلما رأيت تعليمة بهذا الشكل فاعلم أنها مستخدمة لتفريغ المسجل تمهيدا لاستخدامه فيما بعد.

التعليمة التالية هي **XOR** EBX,EBX وكما قلنا فهذه التعليمة معناها أن يتم تصفير هذا المسجل تمهيدا لاستخدامه فيما بعد.

ها قد وصلنا إلى أول أمر في الحلقة loop وهو **MOV** AL, BYTR PTR DS: [ESI+403064]

سأشرح هذه التعليمة بالتفصيل. **MOV** أي أن هذه تعليمة لنقل معلومات ما من مكان لآخر. كلمة ptr هي اختصار لـ pointer. وكما ترى يسبقها byte.

حتى أوضح لك الصورة، فتعليمة **MOV** لها معاملان (two operands) ويجب أن يكونا من نفس الحجم. فمثلا هذه الصيغ مسموحة

MOV EAX, EBX
MOV CX, DX
MOV AL, BH

كما ترى في المرة الأولى كان حجم كل من المعاملين 32 بت، في المرة الثانية كان الحجم 16 بت وفي المرة الثالثة 8 بت. الفكرة تكمن في أنه يجب أن يتساوى حجما المعاملين. حسنا لكن عندما يكون أحد المعاملين حيزا في الذاكرة (كما في حالتنا هذه) فما الذي سيحصل؟ ما هو حجم الذاكرة؟؟!! في هذه الحالة نقوم باستخدام directive (موجه) صيغته كالتالي : **byte ptr** أو **word ptr** أو **dword ptr** أي 8، 16 و 32 بت على الترتيب. هل فهمت الآن؟

أما سبب وجود :DS قبل القوسين المحتويين على عنوان الذاكرة فان كنت قد نسيت فعليك مراجعة الفصل الثاني في الباب الأول والذي يتحدث عن أساسيات الحاسوب ولغة الأسمبلي.

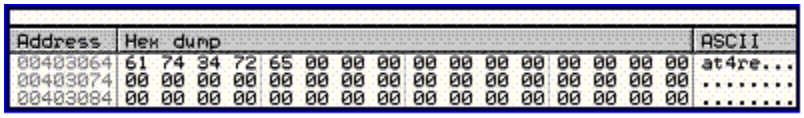
والآن نعرف أن ما بداخل القوسين [] هو عنوان في الذاكرة. لكن انا أرى شيئا غريبا !! ESI+403064

لا تستغرب, ESI سبق لنا أن قمنا بتصفيره. إذن في الدورة الأولى من الحلقة ستكون قيمته 0 وبالتالي ما بداخل القوسين يساوي 403064، ومع الأخذ بعين الاعتبار وجود عداد لـ ESI (ممثلا بالتعليمة **INC** ESI) فانه في الدورة الثانية سيكون ما بداخل القوسين مساويا 403065=1+403064، وفي الدورة الثالثة 403066 وهكذا.

حسنا لنعد إلى الخورازمية، ما فهمناه أن أول أمر سيقوم بنقل بايت واحد بدءا من العنوان 403064 في كل مرة.حسنا لنذهب إلى الذاكرة وننظر ما الذي يوجد هناك !!!

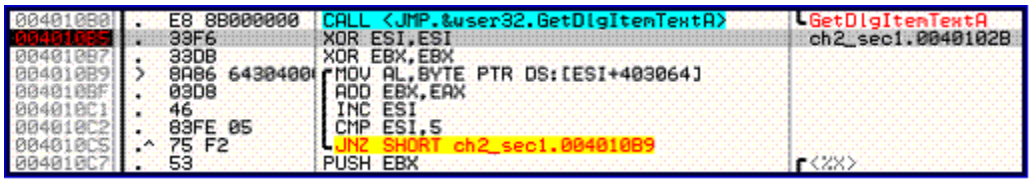
اضغط على أي مكان في الـ hex dump ثم اضغط ctrl+G و :

والآن سنصل إلى هنا :



أها, كل شيء أصبح واضحا الآن.هذه الخوارزمية تبدأ بأخذ أول بايت (أي أول محرف) بدءا من 403064 ويستمر العداد ESI في الزيادة بحيث في كل دورة نقوم بأخذ المحرف التالي.

لنستمر في تتبع الخوارزمية. ما الأمر التالي؟



إنه add ebx,eax أي أن قيمة المحرف الأول ستوضع في متغير فارغ هو ebx، ثم في الدورة التالي تضاف قيمة المحرف التالي وهكذا.

إذن لدينا حالة تشبه التالي : (انتبه فجميع القيم بالهكس وليس بالعشري)

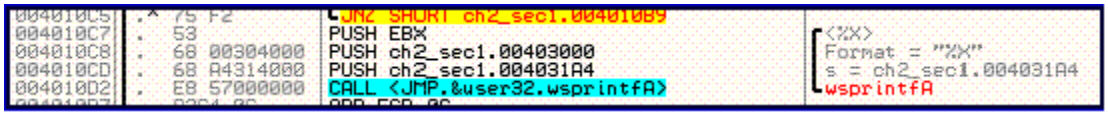
الحرف	e	r	4	t	a
المقابل بالهكس	65	72	34	74	61

61+74+34+72+65=10E

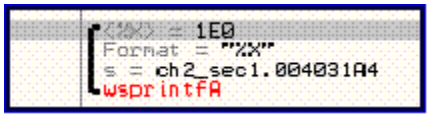
اعتقد أن كل شيء أصبح واضحا الآن. عرفنا من أين أتت تلك الـ 10E.

جدير بالذكر أنه لو كان اليوزرنيم مكونا من 7 محارف مثلا، فلن تتعامل الخوارزمية إلا مع أول 5 محارف كما ترى (هناك مقارنة بين esi و 5 ويتبعها قفزة. ألأمر واضح إذا وصلنا إلى خامس محرف فستنتهي الحلقة وسنخرج منها)

لنر الدالة التالية.

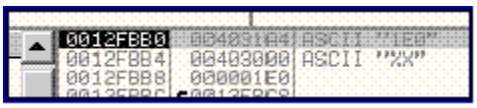


أن هذه الدالة لها عدة استخدامات تختلف باختلاف الـ format. في مثالنا هذا الـ format يساوي %X مما يعني أن هذه الدالة عندما تعطيها قيمة عددية فإنها ستحولها إلى نص يحتوي على تلك الأرقام. أي أن 1E0 ستصبح "1E0".تتبع باستخدام f8. لاحظ فور الوصول إلى أول أمر في الدالة :



كما ترى فان الدالة تأخذ القيمة العددية التي توصلنا إليها.

وعند الانتهاء سترى التالي في قسم المكدس :



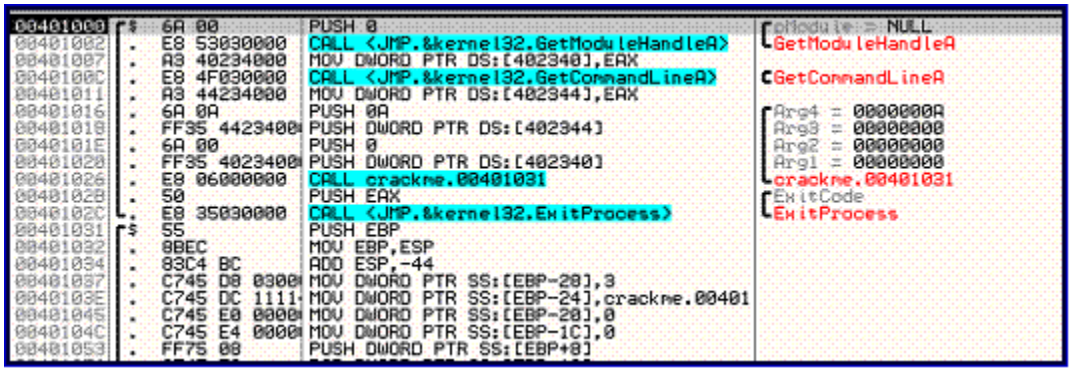
ما يلي الدالة هذه هي دالة المقارنة وسبق لنا الحديث عنها. يتبعها دالة رسالة الخطأ ودالة رسالة الصواب و إلى هنا ينتهي الدرس الثاني.

3.1. المثال الثالث :

قم بتشغيل الملف المسمى ch2_sec1_3 ولاحظه جيدا.هناك جملة unregistered version. أدخل أي رقم و اضغط على check. لاحظ انه لا توجد أي نافذة message تخبرك بان السيريال خطأ.

افحص الملف باستخدام PEiD لمعرفة ما إذا كان مضغوطا packed. لحسن الحظ فانه ليس مضغوط ولحسن الحظ أكثر فكما يظهر في الصورة فالبرنامج مكتوب بالـ assembly مما يجعل الملف أسهل وأكثر وضوحا.

أطلق OllyDBG وافتح الـ crackme.ستشاهد الصورة التالية.



والآن دعنا نبدأ.

في البداية ابدأ بإلقاء نظرة سريعة على البرنامج من أعلى إلى أسفل. وركز انتباهك على قسم التعليقات (رابع عمود من اليسار) وابحث عن أي دالة قد تبدوا مثيرة للاهتمام.

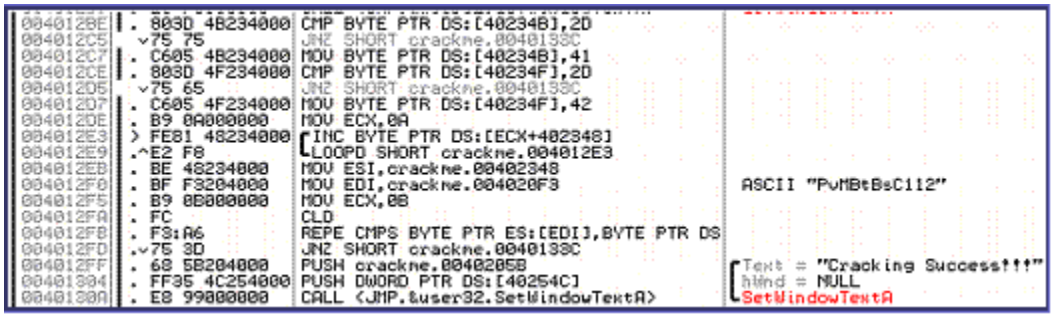
ها قد وجدنا ما قد يكون مفيدا. ألدالة SetWindowTextA التي تعرض جملة unregistered version. لاحظ أن هذه الدالة لها بارامترات 2 parameters (فمثلا احدهما هو البارامتر text ويحتوي على النص الذي ستعرضه الدالة).

ضع نقطة توقف breakpoint وذلك بالضغط على F2 (لست متأكدا من أن هذا هو الطريق الأصح لكنه صحيح بلا شك). انزل قليلا لترى دالة أكثر إثارة للانتباه. أنها GetWindowTextLengthA وكما هو واضح من الاسم فإنها مسؤولة عن حساب طول الـ string التي ندخلها. ومن المهم معرفة أن هذا الطول يخزن في EAX.

أسفل منها هناك أمر مقارنة يقارن طول الـ s/n بـ B (11 decimal) وبالتالي نكون قد عرفنا أن طول الـ s/n هو 11 خانة وهذه معلومة مهمة. أسفل منه هناك قفزة. إذا لم يكون طول السيريال 11 فلن نحصل على شاشة good cracker ثم هناك دالة أخرى مهمة أيضا وهي GetWindowTextA وكما هو واضح فإنها تأخذ ما كتبناه في المربع المخصوص.

لكن إلى أين تأخذه؟ انظر إلى البارامتر الثاني buffer. واضح أن السيريال المدخل سيخزن في 402348.

جيد والآن للنظر ما يوجد بعد هذه الدالة.

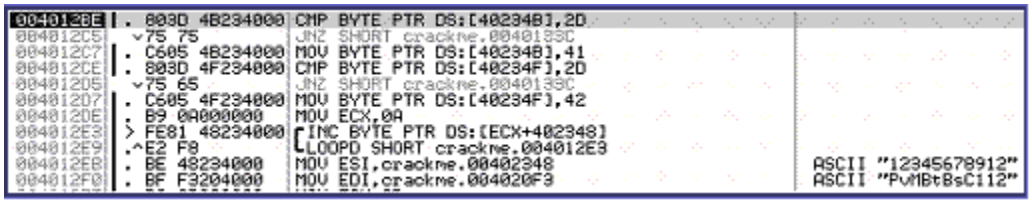


شغل البرنامج بالضغط على F9 سيخرج لك الـ crackme ادخل أي سيريال مكون من 11 خانة وليكن 12345678912

اضغط على check وفورا ستعود إلى olly. نحن الآن عند نقطة التوقف التي انشاناها.سنبدأ بعملية التتبع.

اضغط على F8 لتنتقل إلى الأمر التالي. أستمر في الضغط حتى تصل إلى أمر المقارنة أسفل دالة GetWindowTextLengthA مباشرة. انظر إلى قسم الـ registers وبالتحديد إلى EAX. ما هي قيمته؟

كما تلاحظ فان طول السيريال تم تخزينه في EAX أي 0xB (11). فلنتابع. أستمر بالضغط على F8 حتى تصل إلى الدالة GetWindowTextA. والآن اضغط F8 لتحصل على الصورة التالية



ماذا تلاحظ؟ تم تخزين السيريال في العنوان 4012EB أي بالـ buffer رقم 402348 وأسفل منه هناك stringتثير الانتباه.هل يعقل أن تكون هي السيريال الصحيح؟ اذهب لموقع البرنامج وانسخه وشغل النسخة وادخل هذا السيريال. ماذا تلاحظ؟

محاولة فاشلة!!! يبدوا أن المبرمج أذكى مما توقعنا. لا باس. أنظر إلى الأمر مرة أخرى انه مقارنة بين القيمة 0x2D والقيمة 4.كيف عرفت أنها 4؟ انظر إلى القسم 7 لتجد كل ما تحتاجه. وكما تلاحظ القيمة 4 تمثل الخانة الرابعة من السيريال. يمكننا أن نستنتج من ذلك أن الخانة الرابعة للسيريال يجب أن تكون 0x2D وباستخدام جدول ASCII تعرف أنها علامة الشرطة " – "

من علمنا أن السيريال يخزن في 402348 وان طوله 11 فيمكننا وضع التخطيط التالي (انظر إلى اليسار). وبالتالي فعندما نشاهد CMP BYTE PTR DS:[40234B],2D فهذا يعني مقارنة 0x2D مع محتوى الذاكرة الذي عنوانه مخزن بال segment DS والـ offset له 40234B أي انه الرقم 4...	402348 =1 402349 =2 40234A =3 40234B =4 40234C =5 40234D =6 40234E =7 40234F =8 402350 =9 402351 =1 402352 =2

ثم اضغط. أنتظر!!هناك قفزة. وبما أن المقارنة سلبية أي أن ما أدخلناه لا يساوي الـ – فان القفزة ستنفذ ولن نتمكن من إكمال تتبع البرنامج. أمامك حلان. أما أن تتبعه نظريا أي دون تطبيق باستخدام F8 أو الحل الآخر – وهو ما أفضله – أن تقوم بتعديل القفزة من **JNZ** إلى **JZ**. أضغط عليها right click ثم اختر assemble ومن النافذة الجديدة غيرها واضغط انتر ثم escape (لأننا نريد تغيير أمر واحد فقط) تلاحظ أن الأمر تغير لونه إلى الأحمر. بإمكانك الآن الضغط على F8. لاحظ في قسم التعليقات بالأسفل(ألقسم 7) انه يخبرك ما إذا كانت القفزة ستتم أم لا وهنا يخبرك بأنه : jump is NOT taken حسنا اضغط F8 مرة أخرى.هناك أمر **MOV** حيث ستنقل القيمة 41 (ASCII : A) إلى الخانة الرابعة.

الأوامر الثلاث التالية ينطبق عليها نفس الكلام. ألمقارنة للخانة 8 مع 0x2D أي – وبعدها نغير القفزة ثم نرى أمر **MOV** حيث تنتقل القيمة 42 (ASCII : B) إلى الخانة الثامنة.

الأمر التالي ينقل القيمة 0xA (10) إلى ECX وأنت تعلم أن هذا المسجل يستخدم كعداد لذلك نستنتج أن هناك loop لعشر مرات.

وفعلا الأمران التاليان هما الـloop المقصودة (لاحظ أن olly يضع على يسار كل loop خطا ليدلك عليها. أنظر إلى يسار هذين الأمرين). أول أمر في الـ loopهو **INC** BYTE PTR DS:[ECX+402348] أي انه سيزود آخر خانة بمقدار واحد و الآن قم بالضغط على F8 مرتين وراقب السيريال الخاص بنا.هل شاهدت؟ لقد زادت آخر خانة بمقدار واحد. أكمل الـ loop لتجد بالنهاية أن جميع الخانات تزيد بمقدار واحد عدا الخانة الأولى.

ها قد خرجنا من loop وأصبح السيريال كالتالي 134B678C:23

الأمران التاليان ينقلنا السيريال المعدل إلى ESI والسيريال الذي أثار انتباهنا إلى EDI وانظر إلى قسم المسجلات لتتأكد بنفسك. الأمران التالي يضع القيم 11 في ECX إذا هناك loop تكرر 11 مرة.

تجاهل الأمر التالي وانتقل إلى الذي بعده.كما توقعنا هناك loop مقارنة كالتالي

REPE CMPS BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]

أي أن أول بايت في ESI يقارن مع أول بايت في EDI (وهذا معنى BYTE PTR) وذلك ل 11 مرة أي أن كل خانة تقارن مع نظيرتها في الـ string : PvMBtBsC112. فإذا كانت نتيجة المقارنة ايجابية فان القفزة التالية لن تتم وسينتقل إلى جملة good boy

الخطوة الأخيرة

المعادلة التي حصلنا عليها هي كالتالي(الأرقام في الصف السفلي تجدها في جدول ASCII)

P	v	M	B	t	B	s	C	1	1	2
50	76	4D	42	74	42	73	43	31	31	32

والآن اطرح 1 من كل خانة عدا الأولى مع العلم بان الخانتين 4 و8 يجب أن تكونا (–)

50	75	4C	2D	73	41	72	2D	30	30	31
P	u	L	-	s	A	r	-	0	0	1

إذن السيريال نمبر الصحيح هو PuL-sAr-001

وعند إدخال هذا السيريال يقارن البرنامج الخانتين 4 و8 ب – فيجد أنهما متطابقتان ثم يستبدلهما ب A و B على الترتيب ثم يزود كل خانة ما عدا الأولى بمقدار واحد ويقارن النتيجة ب PvMBtBsC112 فإذا حصل التطابق فالسيريال صحيح.

كان بإمكاننا عمل patch للقفزة التي تسبق رسالة الخطأ مباشرة لكن هدفنا كان إيجاد السيريال الصحيح.

1.4 . المثال الرابع :

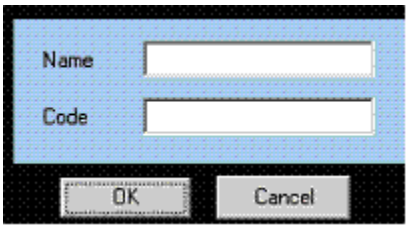
في هذا المثال سنتعامل مع برنامج تجاري واقعي. (تجده بالمرفقات باسم ch2_sec1_4 في الحقيقة البرنامج سهل للغاية واسمه (MIDITracker ويستخدمه من يقومون بعمل المقطوعات الموسيقية بصيغة xm أو mod وغيرها. سنقوم باستخدام طريقة الـ serial fishing وذلك لأنك بعد تتبع البرنامج ستجد أن الباسوورد الصحيحة يتم كتابته في مكان يمكن رؤيته بسهولة فنقوم عندها بأخذ الباسوورد – بكل ما لكلمة سهولة من معنى ☺ - وينتهي الفيلم !!

طريقة serial fishing ليست طريقة بالمعنى الحرفي للكلمة فهي إن أردنا تعريفها سنقول :

عندما ترى الباسوورد الصحيحة، خذها !!

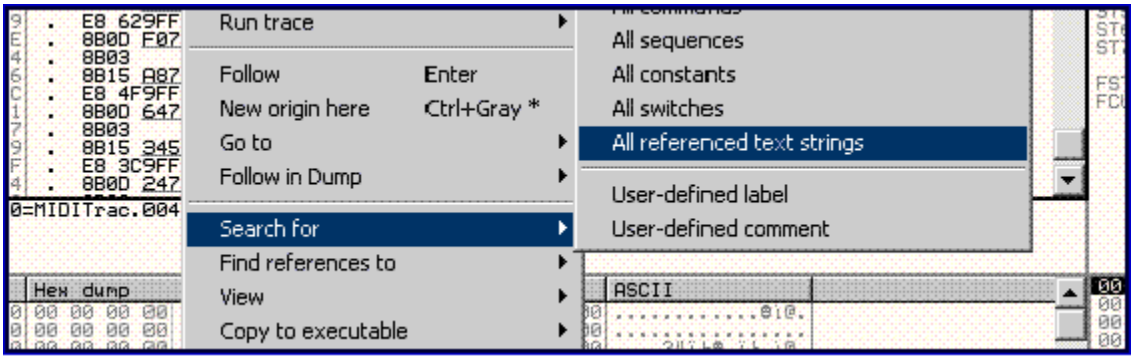
بداية لنفحص البرنامج.

إذن فهو مبرمج بالدلفي.حسنا لنقم بتشغيله.ستلاحظ وجود splash screen وبالأسفل عدة أزرار من بينها enter code. أضغط عليه.

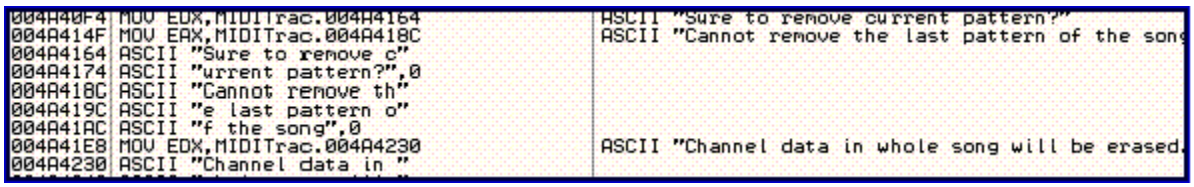


فليكن الاسم at4re وكلمة السر 123456 ثم ok

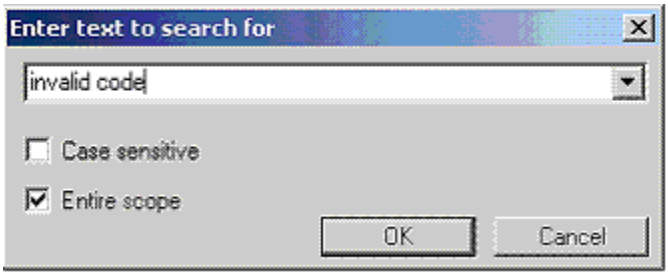
والآن حان دور olly. بعد فتح الضحية اضغط باليمين في أي مكان في olly واختر search ➔ all referenced text strings



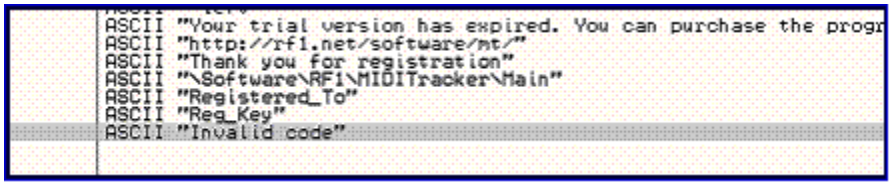
وستخرج لنا النافذة التالية :



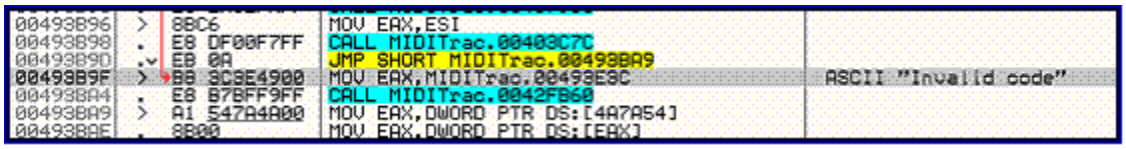
اضغط على زر home في لوحة المفاتيح حتى نصل إلى بداية النافذة، والآن اضغط باليمين واختر search for text وستخرج لك النافذة التالية :



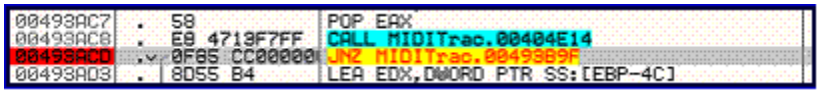
لاحظ أنني كتبت عبارة invalid code وهي العبارة التي ظهرت لنا في رسالة الخطأ وهي ما نريد البحث عنه. لاحظ أيضا أنني غيرت الخيارات كما هو واضح في الصورة. الآن اضغط ok.



هذا ما كنا نبحث عنه.. لاحظ وجد عبارات أخرى لها صلة بالموضوع.حسنا اضغط مرتين فوق العبارة المحددة وسنصل إلى :



مممم.كما ترى هناك سهم (باللون الأحمر) يشير إلى هذا المكان.. عموما إذا صعدت للأعلى سترى باقي العبارات المتعلقة بالـ registration. لا بأس اذهب إلى القفزة التي توصلك إلى هذا المكان (أي تتبع السهم الأحمر لترى من أين يأتي) وضع هناك نقطة توقف.(أي عند 493AC0 كما في الصورة التالية)



والآن قم بتشغيل الضحية (F9) واضغط على زر enter code ثم أدخل at4re و 123456 واضغط على ok.ستلاحظ أننا عدنا إلى olly وبالتحديد عدنا إلى نقطة التوقف التي أنشأناها للتو..كما ترى في قسم التعليقات السفلي مكتوب :

```
Jump is taken
00493B9F=MIDITrac.00493B9F
```

هممم. إذن انزل للأسفل حتى نرى إلى أين ستأخذنا.

هل انتبهت أثناء نزولك إلى العبارات التي تجاوزناها؟ هناك عبارة thank you for registration وقد تخطيناها وكأن شيئا لم يكن !! هذا ليس غريبا فالسيريال المدخل خطأ إذن علينا ألا نحلم بعبارة الترحيب تلك. لكن الأمر الأهم من ذلك والذي علينا إدراكه أنه عندما وصل البرنامج إلى القفزة التي وضعنا نقطة توقف عندها كان قد ولّد كلمة السر الصحيحة وقارنها بتلك المدخلة واستنتج أنهما مختلفتين مما يعني أننا لصوص ولا نعرف ما هي الكلمة السرية الصحيحة ☺

بداية قم بإزالة نقطة التوقف التي وضعناها.

ما أريد قوله أن علينا وضع نقطة التوقف عند مكان يسبق القفزة التي وضعناها. بالنسبة لي أرى أنه من المناسب وضعها في هذا المكان :

```
00493918  .  8038 00        CMP BYTE PTR DS:[EAX],0
0049391B  .v 0F85 C0020000  JNZ MIDITrac.00493BE1
00493921  .  8B0D F8774A00  MOV ECX,DWORD PTR DS:[4A77F8]      MIDITrac.004A8C84
00493927  .  A1 74784A00    MOV EAX,DWORD PTR DS:[4A7874]
0049392C  .  8B00           MOV EAX,DWORD PTR DS:[EAX]
0049392E  .  8B15 082E4900  MOV EDX,DWORD PTR DS:[492E08]      MIDITrac.00492E54
00493934  .  E8 B7B7FEFF    CALL MIDITrac.0047F0F0
00493939  .  8D45 D0        LEA EAX,DWORD PTR SS:[EBP-30]
0049393C  .  50             PUSH EAX
0049393D  .  8D55 CC        LEA EDX,DWORD PTR SS:[EBP-34]
00493940  .  A1 F8774A00    MOV EAX,DWORD PTR DS:[4A77F8]
00493945  .  8B00           MOV EAX,DWORD PTR DS:[EAX]
00493947  .  8B98 0C030000  MOV EBX,DWORD PTR DS:[EAX+30C]
0049394D  .  8BC3           MOV EAX,EBX
0049394F  .  E8 2CABFCFF    CALL MIDITrac.0045E480
00493954  .  8B45 CC        MOV EAX,DWORD PTR SS:[EBP-34]
00493957  .  B9 483D4900    MOV ECX,MIDITrac.00493D48          ASCII "MIDI Tracker"
0049395C  .  BA 603D4900    MOV EDX,MIDITrac.00493D60          ASCII "$APPNAME"
00493961  .  E8 F2EEFFFF    CALL MIDITrac.00492858
00493966  .  8B55 D0        MOV EDX,DWORD PTR SS:[EBP-30]
00493969  .  A1 F8774A00    MOV EAX,DWORD PTR DS:[4A77F8]
0049396E  .  8BC3           MOV EAX,EBX
00493970  .  E8 3BABFCFF    CALL MIDITrac.0045E4B0
00493975  .  68 743D4900    PUSH MIDITrac.00493D74             ASCII "Trial version: "
0049397A  .  FF75 F4        PUSH DWORD PTR SS:[EBP-C]
0049397D  .  68 8C3D4900    PUSH MIDITrac.00493D8C             ASCII " left"
```

بالطبع يمكنك وضع نقطة التوقف عن أي قفزة تسبق هذه القفزة لكن ما يحكم الموضوع أنه ليس من المنطقي اختيار مكان بعيد كثيرا عن جملة الـ invalid code التي رأيناها، كما أنه ليس من الحكمة وضعها في مكان قريب من القفزة التي وضعنا عندها نقطة التوقف السابقة

(لاحظ أن هذه القفزة موجودة في منطقة الخطر ☺ أقصد قريبة من العبارات المتعلقة بالـ registration فكما ترى أسفل منها هناك عبارة trial version)

والآن أعد تشغيل البرنامج  :

ثم شغل الضحية F9 وستصل إلى نقطة التوقف الجديدة. تتبع F8 حتى تظهر لك شاشة البرنامج. قم بالضغط على زر enter code

كما تلاحظ لقد عدنا إلى olly. تتبع حتى تخرج لك شاشة إدخال معلومات التسجيل. ادخل at4re و123456 ثم ok. سنعود إلى olly.

أن كان هذا الدرس يدور حول استكشاف خوارزميات التشفير، كنت عندها سأقول لك أن هذا المكان حتما هو مكان وجود الخوارزمية فكما أخبرتك بالدرس الماضي فالخوارزمية توجد بعد المكان الذي يتم عنده جلب اليوزرنيم والباسوورد

بعد مكان جلب اليوزر والباسوورد، قد يكون هناك قفزة تؤدي إلى مكان توجد فيه الخوارزمية و يسبق مكان جلب اليوزر، إذن كيف أقول لك أن الخوارزمية توجد بعد مكان الجلب؟ لا تقلق أنا لا أخدعك. أنا اقصد أنها موجودة بعد مكان جلب اليوزر من حيث تسلسل أوامر البرنامج.. فمن غير المنطقي أن تمر على خوارزمية توليد سيريال مع أنك لم تقم بإدخال أي يوزرنيم بعد !! إذن هي موجودة بعد مكان الجلب من حيث ترتيب الأوامر وتسلسلها الزمني وليس المكاني

لكن على رسلك فهذا الدرس ليس للخوارزميات فد تكلمنا عنها (بشكل موجز) في الدرس الثاني. نحن هنا نريد إجراء serial fishing

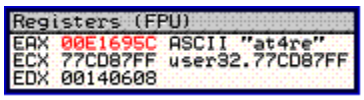
إذن تتبع وعينك على ثلاث أماكن :

- قسم المسجلات وتحديدا eax
- قسم الـ stack
- قسم التعليقات السفلي

عند وصولك إلى هذا المكان :

```
00493A96   .  E8 E5A9FCFF   CALL MIDI.trac.0045E480
00493A9B   .  8B45 BC       MOV EAX,DWORD PTR SS:[EBP-44]
00493A9E   .  8D4D C0       LEA ECX,DWORD PTR SS:[EBP-40]
00493AA1   .  BA 08000000   MOV EDX,8
```

ستلاحظ أن eax به اليوزرنيم الذي أدخلناه :

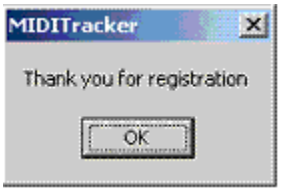


إذن علينا الآن أن نكمل التتبع لكن ببطء شديد لأننا اقتربنا. بعد تجاوزنا لأول دالة تلي ذلك المكان مباشرة كما في الصورة :

سيكون قسم التعليقات محتويا على شيء مهم جدا :

إذن ذلك الأمر الذي يلي الدالة مباشرة (أمر **MOV**) يريد نقل السترنج csoY-xf82 إلى eax. بالنسبة لي فكلما رأت سترنج بها شرطة (كهذه _) فإنني أشك في أنها سيريال. لأن أغلب السيرلات إن لم يكن جميعها تحتوي على شرطة أو أكثر !!

إذن ماذا تنتظر؟ انسخ الضحية، وافتح النسخة الجديدة وأدخل at4re وcsoY-xf82 والنتيجة :



رائع !

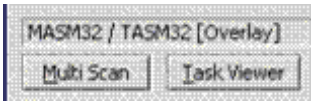
5.1. المثال الخامس :

الهدف لهذا المثال لا يحتوي على خوارزمية تشفير مختلفة عما تعلمناه. أقصد أنها ليست مميزة أو معقدة. لكن الطريق للوصول إليها يجعلنا نمر بعدة أمور مهمة سيتم شرحها، لذلك فمع نهاية الدرس ستجد نفسك – عزيزي الكراكر – وقد تعلمت العديد من الطرق والأساليب والمفاهيم الجديدة.حظا جيدا.


ستجد الضحية في المرفقات باسم ch2_sec1_5

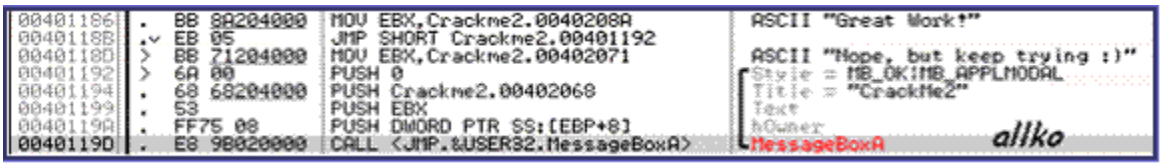
كالعادة، شغل الملف الضحية. لا شيء مميز. مكان لكتابة الـ username وآخر للـ serial number وعند إدخال السيريال الخطأ تخرج لنا شاشة مزعجة.

قم بعمل فحص للبرنامج باستخدام PEiD 0.9 والنتيجة :



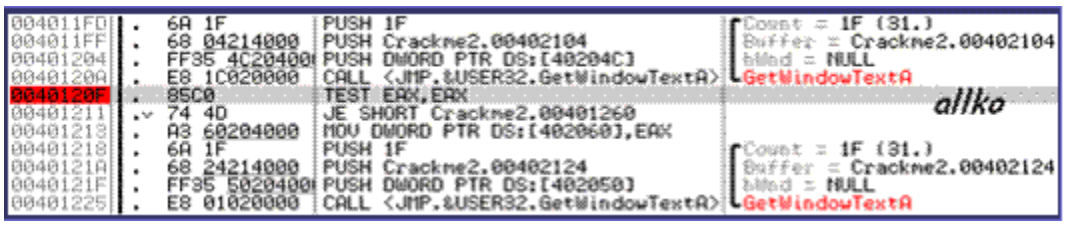
والآن حان دور OLLYDBG.

ابدأ بالبحث – في قسم الـ comments – عن دالة تهمنا.حسنا هناك MessageBoxA على عنوان 40119D كما تلاحظ فان بارامتر text فارغ لكن يوجد قبل الدالة two strings الأولى للسيريال الصحيح والأخرى للخاطئ



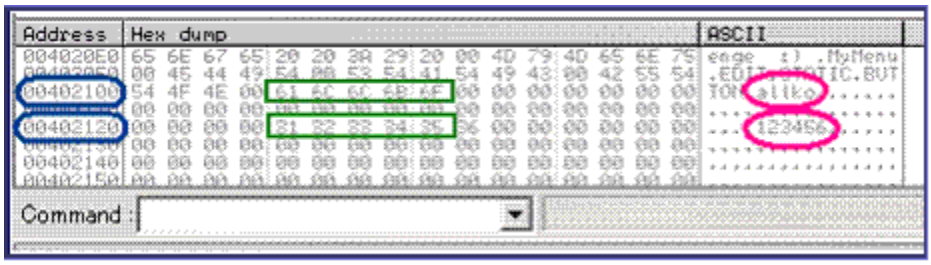

إن الـ string كما ترى تنقل إلى EBX ومقابل البارامتر text هناك أمر **PUSH** EBX. أعتقد أن الصورة أصبحت واضحة.

تابع حتى تصل إلى ما كنا نبحث عنه. GetWindowTextA.هناك دالتان من هذا النوع الأولى لليوزرنيم والأخرى للباسوورد. هذا مكان مناسب لوضع BreakPoint ضع واحدة عند العنوان 40120F كما بالصورة



هناك أمر **TEST** EAX,EAX يليه **JE** وهذان الأمران مترابطان فغالبا ما يكونان معا. والمعنى : " إذا كانت قيمة EAX صفرا فاقفز إلى العنوان." وذلك يعني انك إذا لم تدخل شيئا في خانة اليوزرنيم أو الباسوورد(لاحظ انه يوجد أمران آخران أسفل دالة السيريال) فانه لن يكمل عملية التحقق بل سيذهب إلى 401260 ولا يهمنا ألان ماذا يوجد هناك.

أين يخزن اليوزرنيم والباسوورد؟ كما علمت من الأمثلة السابقة، انظر إلى البارامتر buffer لتعرف أن اليوزر يخزن في 402104 أم الباسوورد فيخزن في 402124، والآن شغل الملف بالضغط على F9 ثم اكتب allko و 123456 أو أي شيء آخر. أضغط OK ها قد عدنا إلى Olly، تتبع – باستخدام F8 – حتى تصل إلى الأمر الذي يلي الدالة الثانية مباشرة , أي إلى العنوان 40122A والآن انظر إلى محتويات الذاكرة أي إلى القسم Data



كما تلاحظ فان الاسم خزن في 402104 (لاحظ بجانب 402100، أول 4 بايتات ليست للاسم أي أن الاسم يبدأ من 402104 وكما تلاحظ هناك الرقم 61 الذي يقابل أول حرف وهو a). وكذلك الحال بالنسبة للسيريال نمبر.

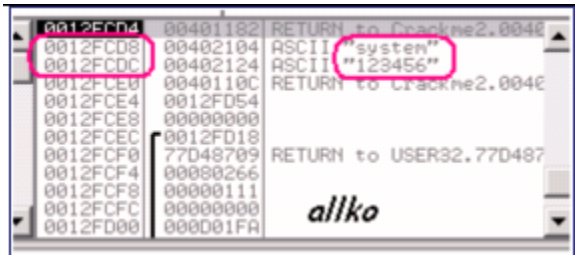
بعد ذلك هناك دالتان من نوع SendMessageA. ليستا مفيدتان لنا لذا اتركهما وتتبع حتى تصل إلى أمر **RETN** انظر إلى الأسفل. يخبرك Olly بأنه : Return to 0040116F لا بأس تتبع. نحن الآن عند العنوان المذكور و هناك أمر **TEST** EAX,EAX يليه **JE**.

هناك أمران **PUSH** لنقل اليوزر والباسوورد إلى الذاكرة عند العناوين المبينة(لاحظ أن العناوين هي نفسها التي كانت بجانب بارامتر buffer) والآن وصلنا إلى **CALL** اضغط انتر وضع BreakPoint حتى يتسنى لنا تتبع الأوامر الموجودة بداخل الـ **CALL**.كما ترى هناك loop inside loop.

فلنبدأ بفك التشفير.

لاحظ أن هناك أمر مقارنة مع القيمة 6 لذا، اعد تشغيل البرنامج واتبع نفس الخطوات السابقة لكن ادخل اسما من 6 أحرف وليكن system ونفس كلمة السر 123456.

00401261 $ A1 60204000 MOV EAX,DWORD PTR DS:[402060]	ينقل طول الاسم (6) من الذاكرة إلى **EAX**
00401266 . 3905 6420400(CMP DWORD PTR DS:[402064],EAX	يتم مقارنة طول السيريال مع القيمة 6
0040126C .v 75 2F JNZ SHORT Crackme2.0040129D	اقفز إلى **40129D** إذا لم يكونا متساويين
0040126E . 33C0 XOR EAX,EAX	صفر **EAX**
00401270 . 8B7C24 04 MOV EDI,DWORD PTR SS:[ESP+4]	ضع اليوزرنيم في **EDI** *(ملاحظة 1)*
00401274 . 8B7424 08 MOV ESI,DWORD PTR SS:[ESP+8]	ضع الباسوورد في **ESI**
00401278 > 0FB61F MOVZX EBX,BYTE PTR DS:[EDI]	انقل اول حرف من اليوزرنيم إلى **EBX** *(ملاحظة 2)*
0040127B . 0FB616 MOVZX EDX,BYTE PTR DS:[ESI]	انقل اول خانة في السيريال(فلنسمها **s1**) إلى **EDX**
0040127E . 80FA 30 CMP DL,30	قارن **s1** مع 30 (الاعداد في **ASCII** تبدا من 30 حتى39)
00401281 .v 7C 1A JL SHORT Crackme2.0040129D	اقفز إذا كان **s1** اقل من30(أي انه ليس برقم ولا بحرف)
00401283 . 80FA 39 CMP DL,39	قارن **s1** مع 39
00401286 .v 7F 15 JG SHORT Crackme2.0040129D	اقفز إذا كان **s1** اكبر من 39 (أي انه ليس رقما)
00401288 . 80EA 30 SUB DL,30	اطرح 30 من قيمة الحرف الاول
0040128B . 83E3 0F AND EBX,0F allko	هذه العملية تبقي الخانة الأولى فقط *(ملاحظة 3)*
0040128E . D0EB SHR BL,1	اقسم على 2 *(ملاحظة 4)*
00401290 . 2ADA SUB BL,DL	اطرح القيمتين من بعض...
00401292 .v 75 09 JNZ SHORT Crackme2.0040129D	اقفز إلى **40129D** إذا لم تكونا متساويتين
00401294 . 47 INC EDI	زود قيمة **EDI** بمقدار 1 *(ملاحظة 5)*
00401295 . 46 INC ESI	زود قيمة **ESI** بمقدار 1
00401296 . 803F 00 CMP BYTE PTR DS:[EDI],0	قارن **EDI** مع 0 ، أي هل انتهت احرف اليوزرنيم؟
00401299 .^ 75 DD JNZ SHORT Crackme2.00401278	إذا لم تنتهي فاقفز لبداية الـ**loop** للانتقال للحرف التالي
0040129B .v EB 04 JMP SHORT Crackme2.004012A1	اقفز إلى **4012A1**
0040129D > 33C0 XOR EAX,EAX	صفر **EAX**
0040129F .v EB 05 JMP SHORT Crackme2.004012A6	اقفز إلى **4012A6**
004012A1 > B8 01000000 MOV EAX,1	انقل القيمة 1 إلى **EAX**
004012A6 > C3 RETN	عد إلى **401182**

	ملاحظة 1 : القيمة [ESP+4] تساوي 12FCD4 + 4 = 12FCD8 أي أن SS:[ESP+4] تشير إلى عنوان في الـ stack (SS=Stack Segment) وكما ترى في الأسفل (قسم التعليقات السفلي) فهذا العنوان يحتوي على اليوزرنيم وللتأكد انظر إلى قسم الـ stack لتتأكد، ونفس الشيء بالنسبة للأمر التالي
ملاحظة 2 : لاحظ الأمر جيدا MOVZX EBX , BYTE PTR DS:[EDI] هناك كلمة BYTE وحيث أن اليوزرنيم تخزن على هيئة DWORD (انظر 401270)ونعلم أن DWORD=Double Word=32 byte إذا أول بايت يشير إلى أول حرف من اليوزرنيم، ونفس الشيء بالنسبة للأمر التالي (أي بالنسبة للباسوورد).	
ملاحظة 3 : عملية **AND** تعني الضرب وبالتالي عندما نضرب بـ 0F أي 00001111 فان كل شيء سيصبح صفرا ما عدا الخانة الأولى لأنها تضرب بـ 1111 وبالتالي تبقى كما هي، في مثالنا فان EBX يحوي الحرف الأول من كلمة system أي حرف s والذي يقابل القيمة 73 وعند تحويلها لـ Binary فإنها تساوي 0011 0111، اضربها بـ 1111 0000 ستجد أن الخانة الثانية – 7 – قد اختفت (أصبحت 0) بينما بقيت الخانة الأولى 3 كما هي.	
ملاحظة 4 : **SHR** BL,1 تعني إزاحة محتويات BL لليمين لمرة واحدة. بعد عملية **AND** أصبح BL=03 أي 0011 وعند عمل الإزاحة فان الخانة اليمنى تختفي (تحديدا، تذهب إلى CF:Carry Flag) بينما يضاف 0 من اليسار. أي أن 0011	

تصبح 0001

مثال : مثال : لو أنBL=07أي 0111 ثم عملنا BL,1 **SHR**فانه سيصبح 0011

مثال : لو أن BL=06 أي 0110 ثم عملنا BL,2 **SHL** فانه سيصبح 1000

ومن الأمثلة أعلاه يمكن استنتاج الآتي : كل إزاحة لليمين تعني قسمة على 2 وكل إزاحة لليسار تعني ضرب بـ 2 إذا BL,3 **SHR** يعني قسمة BL على 8 (كل إزاحة لليمين تعني قسمة على 2 إذا 3 ازاحات تعني قسمة على 8) **SHL** BL,2يعني ضرب BL في 4 (كل إزاحة لليسار تعني ضرب بـ 2 إذا ازاحتان تعني ضرب بـ 4) والباقي في حالة القسمة يضيع.

لاحظت أن كل خانة من خانات السيريال تمت مقارنتها : هل هي اصغر من 30؟ هل هي اكبر من 39؟ وإذا تحقق احد هذين الشرطين فإننا سنذهب إلى bad boy massage وذلك يعني أن السيريال مكون من أرقام فقط.

انظر جدول ASCII المصغر المرفق. ألأرقام تبدأ من 30 حتى 39، ال small letters تبدأ من 61 حتى 7A بينما capital letters تبدأ من 41 حتى 5A

دعنا الآن نفك الشيفرة. أول حرف من اليوزرنيم ولنسمه u1 يتم عمل **AND** له أي انه الخانة الثانية من قيمته تختفي وتبقى الأولى، ثم يتم عمل **SHR** له. أم السيريال فيطرح منه 30. ثم تتم عملية المقارنة بين القيمتين. من ذلك يمكن وضع الجدول التالي

OPERATION	s	y	s	t	e	m
in ASCII	73	79	73	74	65	6D
AND 0F	3	9	3	4	5	D
In Binary	0011	1001	0011	0100	0101	1101
SHR BL,1	0001	0100	0001	0010	0010	0110
In decimal	1	4	1	2	2	6
ADD 30	31	34	31	32	32	36
In ASCII	1	4	1	2	2	6

إذا الباسوورد الخاصة بكلمة system هي 141226

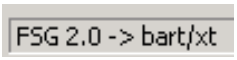
6.1 . المثال السادس

هذا المثال يتحدث عن إزالة الرسائل المزعجة والمسماة nag screen.

قم بفتح البرنامج المرفق والمسمى ch2_sec1_6

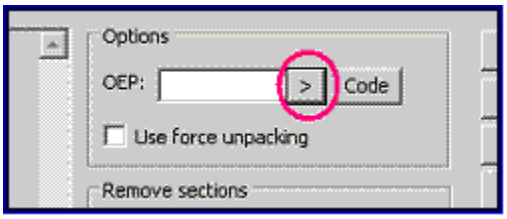
كما تلاحظ هناك nag عند بداية التشغيل وثانية عند الضغط على الزر وثالثة زمنية ورابعة عند الخروج من البرنامج.

والآن افحص البرنامج باستخدام PEiD لنر ماذا لدينا :



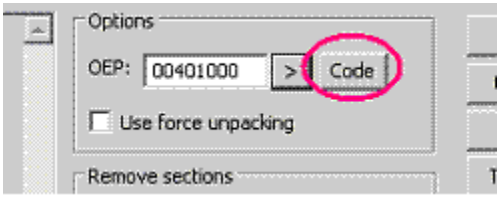
إذن فهو مضغوط باستخدام FSG 2.0 ولإزالة الضغط سنستخدم احد البرامج المخصص لذلك (لا تقلق سنتعلم فك الضغط يدويا لكن ليس الآن). سأستخدم برنامج quick unpack 1.0 فهو مشهور جدا ويصلح لفك ضغط العديد من البرامج. ستجد البرنامج بالمرفقات. افتحه ثم اختر قائمة preferences ➔ options وضع علامة صح بجانب الخيار register shell extension

والآن اضغط على الضحية باليمين واختر unpack with quick unpack.

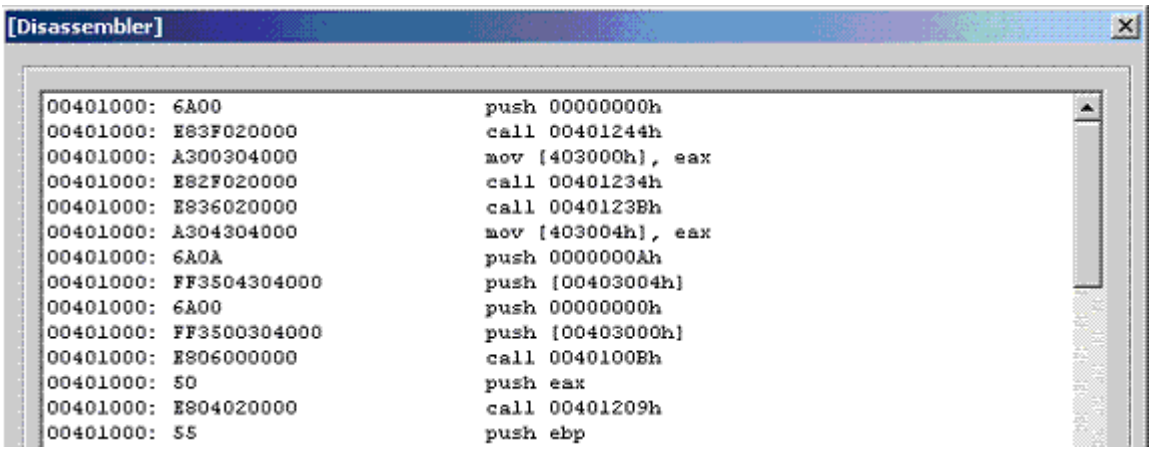


والآن :

ستجد أن الخيار الثاني تم اختياره تلقائيا.حسنا اضغط عليه مرتين (أحيانا لا تنجح عملية فك الضغط عندها اعد الخطوات لكن اختر خيارا آخر غير الخيار الثاني) والآن اضغط على زر code

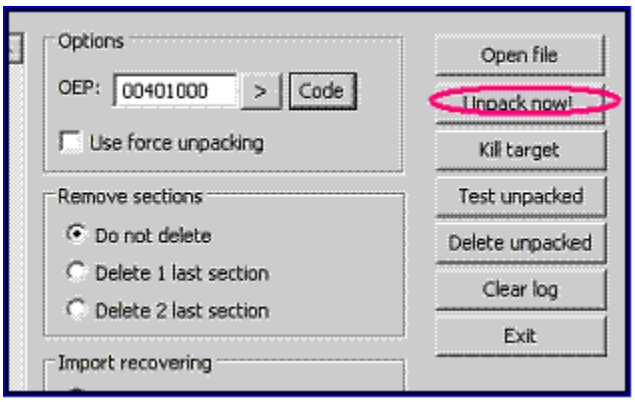


ستخرج لك هذه النافذة :



اضغط ok

والآن اختر unpack now



عندها ستخرج لك النافذة التالية :

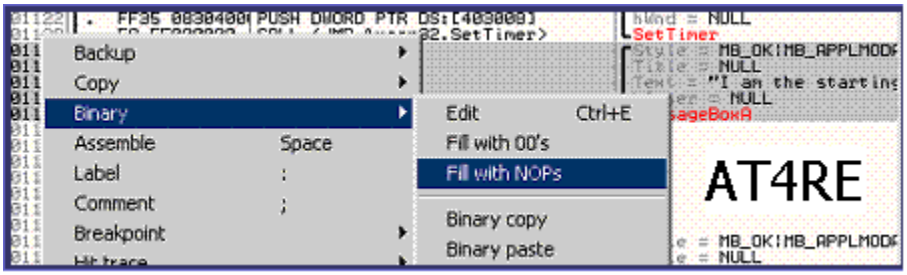
[Import Table]

N	Library	Function	Record RVA	Record section	Problem
0000	comctl32.dll	InitCommonControls	0x00002000	No.01 Name: .text	no
0001	kernel32.dll	GetCommandLineA	0x00002008	No.01 Name: .text	no
0002	kernel32.dll	ExitProcess	0x0000200C	No.01 Name: .text	no
0003	kernel32.dll	GetModuleHandleA	0x00002010	No.01 Name: .text	no
0004	user32.dll	LoadCursorA	0x00002018	No.01 Name: .text	no
0005	user32.dll	LoadIconA	0x0000201C	No.01 Name: .text	no
0006	user32.dll	MessageBoxA	0x00002020	No.01 Name: .text	no
0007	user32.dll	PostQuitMessage	0x00002024	No.01 Name: .text	no
0008	user32.dll	GetMessageA	0x00002028	No.01 Name: .text	no
0009	user32.dll	SetTimer	0x0000202C	No.01 Name: .text	no

اضغط على زر save. والآن ها قد انتهينا ستجد الملف المفكوك باسم __ch2_sec1_6 أي أن هناك شرطتين قد تمت إضافتهما إلى نهاية الاسم. والآن شغل البرنامج المفكوك لنتأكد من نجاح العملية؟ جيد انه يعمل بلا مشاكل وقد تم فك ضغطه.

افتح الهدف بـ olly. أثناء بحثنا عن شيء مثير للانتباه يلفت نظرنا دالة SetTimer يبدوا أنها ذات علاقة بالـ nag الزمنية. أسفل منها هناك MessageBoxA الخاصة بـ starting nag.حدد الدالة بالكامل (أي مع جميع بارامتراتها) ثم اضغط باليمين واختر

Binary ➔ fill wigh NOPs

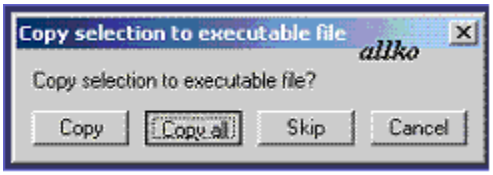


ستجد أنها أصبحت بهذا الشكل.

ثم كرر ذلك مع بقية الـ nags.(أي مع بقية دوال message box بالأسفل)

وبعد الانتهاء من الأربعة اضغط باليمين واختر copy to executable ➔ all modifications

ثم



ومن النافذة الجديدة اختر save file ثم احفظه في أي مكان. والآن قم بتشغيله.رائع!!! لم تظهر أي واحدة!!

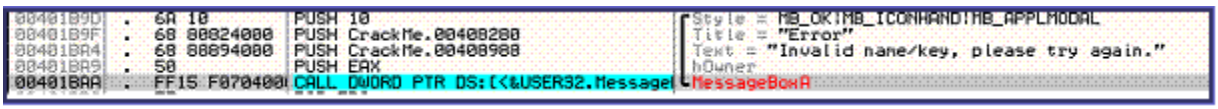
7.1 . المثال السابع

درس اليوم يختلف قليلا عن سابقيه فالبرنامج مكتوب بلغة c++ وليس بـالاسمبلي. وكما ذكرت سابقا فان الاسمبلي أسهل لنا. لان الاسمبلي تعتبر من لغات (LLL : Low Level Language) أي انه لعمل برنامج ما باستخدام الاسمبلي فانك بحاجة لأسطر برمجية اقل مما ستحتاجه لتنفيذ نفس البرنامج بالـ (HLL)..شيء آخر أود ذكره هو أن طريقة التشفير لا تختلف كثيرا رغم أنها أطول.قد يكون هذا الدرس الأخير من هذه النوعية لأنه اعتقد بانتهائك من هذا الدرس ستصبح جاهزا لكسر هذا النوع من الحمايات بنفسك.

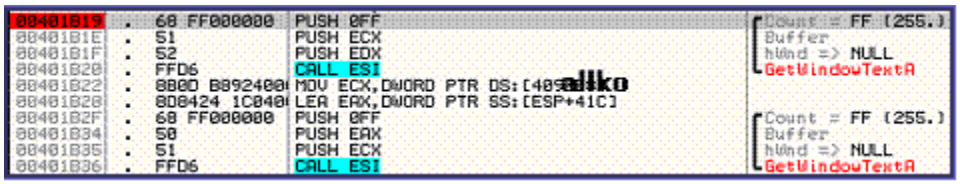
كالعادة تفحص البرنامج المرفق باسم ch1_sec1_7  باستخدام PEiD والنتيجة 

شغل البرنامج. من قائمة file اختر register.حسنا هناك message بـ username و s/n اكتب أي شيء لتظهر لك شاشة الـ error. من القائمة السابقة اختر about.كما ترى فالمبرمج سمح لك باستخدام أي طريقة (هل تفكر في الـ patching؟ لا تفرح كثيرا فلن اسمح لك بذلك!!) نريد عمل keygenerater وليس فقط تغيير هذا الأمر أو ذاك.

والآن حمّل البرنامج إلى olly. لاحظ أن البرنامج مختلف عما عهدناه مع برامج asm. لا باس.for search ثم all referenced text strings وابحث عن رسالة الخطأ. لاحظ أن هناك 3 رسائل. أختر الأولى.



والآن ابحث عما قد يكون مفيدا. أن كنت قد قرأت وفهمت الدروس السابقة فأنت تعلم ما نبحث عنه. نريد الدالة التي تستقبل الـ user والـ pass. بالنظر للأعلى قليلا ستجد اثنتين : GetWindowTextA



ضع BreakPoint كما هو واضح ثم شغل البرنامج F9 والآن ادخل اليوزرنيم allko والباسوورد 123456

ثم اضغط register. ها قد عدنا.تتبع (trace over) باستخدام F8 مرة واحدة. لاحظ بارامتر الـ buffer يبدو أن اليوزرنيم يخزن في الـ stack على عنوان 0012F7F8 والآن تتبع حتى تنتهي من الدالة الأولى. لاحظ أن EAX يحوي عدد characters أي 5.تتبع مرة أخرى حتى تنتهي من الدالة الثانية ولاحظ بارامتر الـ buffer وعنوان السيريال : 0012F9F8 وألان لاحظ الأوامر التالية (تذكر أن تنتبه إلى قسم التعليقات بالأسفل)

00401B38 MOV BL,BYTE PTR SS:[ESP+21C]	يتم نقل أول حرف من اليوزر "u[1]="a إلى BL
00401B3F XOR EDX,EDX	صفّر EDX
00401B41 MOV AL,BYTE PTR SS:[ESP+41C]	يتم نقل أول حرف من السيريال "s[1]="1 إلى AL
00401B48 TEST AL,AL	هل AL يساوي صفر ؟
00401B4A SETE DL	إذا كان مساويا للصفر فاجعل DL يساوي 1
00401B4D XOR EAX,EAX	صفّر EAX
00401B4F TEST BL,BL	هل BL يساوي صفر
00401B51 SETE AL	إذا كان مساويا للصفر فاجعل ALيساوي 1
00401B54 OR EDX,EAX	نفذ عملية OR بين EDX,EAX
00401B56 JE SHORT CrackMe.00401B81	إذا كانت النتيجة 0 فاقفز.

كما ترى إذا لم تدخل أي اسم أو لم تدخل أي سيريال أو لم تدخل الاثنين فستظهر رسالة خطا وآلا ستقفز إلى 00401B81 كما في حالتنا.

0 **OR** 1=1
1 **OR** 0=1
1 **OR** 1 =1
0 **OR** 0=0

00401B81 MOV EDI,DWORD PTR DS:[lstrlenA>]	يتم نقل عنوان الدالة lstrlenA إلى EDI
00401B87 LEA EDX,DWORD PTR SS:[ESP+21C]	الآن EDX يشير إلى عنوان اليوزرنيم
00401B8E PUSH EDX ; /String	يتم دفع اليوزرنيم كبارامتر للدالة إلى الـstack
00401B8F CALL EDI ; \lstrlenA	يتم استدعاء الدالة لحساب طول اليوزرنيم

00401B91 CMP EAX,3	النتيجة خزنت في EAX وستقارن مع القيمة 3
00401B94 JGE SHORT CrackMe.00401BBF	إذا كان اليوزرنيم > 3 فاقفز إلى 00401BBF

فعلا فان allko اكبر من 3 ☺ إذا سنتخطى رسالة الخطأ

00401BBF LEA ECX,DWORD PTR SS:[ESP+41C]	الآن ECX يشير إلى عنوان السيريال
00401BC6 PUSH ECX	يتم دفع السيريال كبارامتر للدالة إلى الـstack
00401BC7 CALL EDI	استدعي الدالة التي عنوانها موجود في EDI (نفس التي في الأعلى)
00401BC9 CMP EAX,22	النتيجة خزنت في EAX وستقارن مع 22h أي34d
00401BCC JE SHORT CrackMe.00401BF7	إذا كان الباسوورد = 22فاقفز إلى 00401BF7

حسنا ضع bp عند أمر القفز ثم اعد التشغيل ctrl + F2 ثم F9 واكتب allko و

123456789012345678901234567890abcd.. لاحظ تحول الأحرف abcd إلى ABCD فور كتابتها،تذكر هذا فقد ينفعنا، نحن الآن عند أمر القفز اضغط F8 لتنتقل إلى 00401BF7. سنبدأ بتفحص كود الحماية:

00401BF7 LEA EAX,DWORD PTR SS:[ESP+41C]	الآن EAX يشير إلى عنوان السيريال
00401BFE XOR EBP,EBP	صفّر EBP
00401C00 PUSH EAX	ادفع قيمة EAX (أي عنوان السيريال) إلى المكدس (stack)
00401C01 XOR BL,BL	صفّر BL
00401C03 MOV DWORD PTR SS:[ESP+1C],EBP	انقل قيمة EBP (أي 0) إلى الـ stack على عنوان 0012F5F4
00401C07 CALL EDI	استدعي الدالة التي عنوانها موجود في EDI (نفس الدالة في الأعلى)
00401C09 TEST EAX,EAX	هل EAX يساوي صفر ؟ (لا فهو يحوي طول السيريال أي 0x22)
00401C0B JLE SHORT CrackMe.00401C52	اقفز إذا كان صفرا
00401C0D XOR ESI,ESI	صفّر ESI
00401C0F MOV AL,BYTE PTR SS:[ESP+ESI+41C]	يتم نقل حرف من السيريال s[1]="1" إلى AL @
00401C16 INC BL	زود قيمة BL
00401C18 CMP AL,2D	قارن AL مع 0x2d (لو نظرت إلى جدول ASCII لعرفت أنها الشرطة -)
00401C1A JE SHORT CrackMe.00401C39	اقفز إذا تساوا إلى 00401C39
00401C1C MOVSX ECX,WORD PTR SS:[ESP+18]	انقل القيمة 0000=[0012F5F4] SS: إلى ECX
00401C21 PUSH EAX	ادفع قيمة EAX (أي s[1]="1") إلى المكدس (stack)
00401C22 MOV BYTE PTR SS:[ESP+ECX+20],AL	انقل قيمة AL إلى Stack SS:[0012F5F8]
00401C26 CALL CrackMe.00402510	استدعي العنوان 00402510 (اضغط F7 للذهاب إلى هناك)
التكملة تحت	

@ : انظر إلى العنوان 401C47 هناك أمر EBP **INC** يليه ESI,BP **MOV** أي : يستخدم EBP كعداد ويتم نقل قيمه كل مرة إلى ESI أي أن ESI بدوره أصبح عدادا. ولذلك فان القيمة [ESP+ESI+41C] تشير بداية إلى أول حرف، ثم مع ازدياد ESI ستشير إلى الحروف التالية.

والآن تتبع trace into باستخدام F7 لنذهب إلى ذلك العنوان

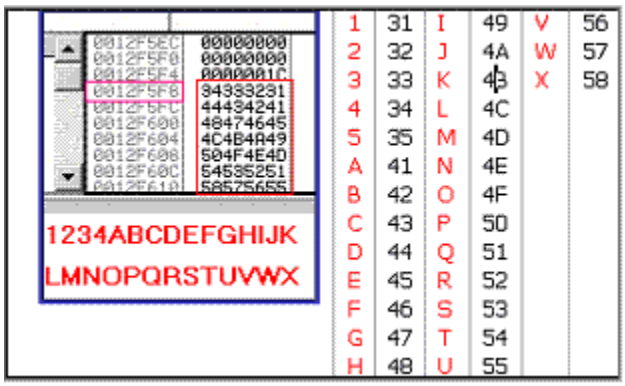
MOV AL,BYTE PTR SS:[ESP+4]	يتم نقل اول خانة في السيريال الى AL (انظر الى قسم التعليقات بالاسفل)
PUSH ESI	يتم دفع قيمة ESI الى الـ stack
XOR ESI,ESI	صفر ESI
XOR ECX,ECX	وكذلك ECX
CMP AL,5A	قارن AL مع القيمة 5A (هذه تكافئ Z أي اكبر قيمة لـ capital letters) @@
SETG CL	إذا كان AL اكبر ، فاجعل CL يساوي 1 (SET if Greater)
XOR EDX,EDX	صفر EDX
CMP AL,41	قارن AL مع القيمة 41 (هذه تكافئ A أي اصغر قيمة لـ capital letters)
SETL DL	إذا كان AL اصغر ، فاجعل DL يساوي 1 (SET if Lower)
OR ECX,EDX	نفذ عملية OR بين ECX,EDX (أي: تحقق ما إذا كان حرفا كبيرا ام لا)
JE SHORT CrackMe.0040252E	إذا كانت النتيجة 1 (أي انه اكبر او اصغر، وبالتالي فهو ليس حرفا كبيرا) فاقفز
MOV ESI,1	و إلا ، إذا كانت النتيجة 0 فزود ESI
XOR ECX,ECX	صفر ECX
CMP AL,39	قارن AL مع 39 (هذه تكافئ 9 أي اكبر قيمة لرقم)
SETG CL	إذا كان AL اكبر ، فاجعل CL يساوي 1 (SET if Greater)
XOR EDX,EDX	صفر EDX
CMP AL,30	قارن AL مع 30 (هذه تكافئ 0 أي اصغر قيمة لرقم)
SETL DL	إذا كان AL اصغر ، فاجعل DL يساوي 1 (SET if Lower)
OR ECX,EDX	نفذ عملية OR بين ECX,EDX (أي: تحقق ما إذا كان رقما أم لا)
JE SHORT CrackMe.00402541	إذا كانت النتيجة 1 (أي انه اكبر او اصغر، وبالتالي فهو ليس رقما) فاقفز
INC ESI	و إلا ، إذا كانت النتيجة 0 فزود ESI
CMP ESI,2	قارن ESI مع 2 (إذا لم يكن حرفا او رقما فـ ESI=2)
POP ESI	ارجع قيمة ESI الاصلية من الـ stack
SETE AL	إذا تساوت ESI مع 2 فاجعل AL مساويا لـ 1
RETN 4	عد من حيث بدأت

@@ : تذكر أن الحروف الصغيرة تحول تلقائيا إلى كبيرة، وبالتالي عندما نختبر الخانة الأولى(أو أي خانة أخرى) ما إذا كانت حرفا كبيرا أم لا فهذا يعني أننا نختبرها إذا كانت حرفا(بصورة عامة) أم لا.

أن هذه الـ snippet of code التي تفحصناها للتو وظيفتها بكل بساطة تفحص ما إذا كان الذي أدخلته رمزا أم لا. فهو يختبره بداية هل هو حرف؟ ثم : هل هو رقم؟ فإذا لم يكن حرفا أو رقما فهو رمز (مثلا ! أو @ أو #.) وهذا غير مقبول.
والآن دعنا نكمل الإجرائية Procedure السابقة.

التكملة	
00401C2B TEST AL,AL	هل AL يساوي صفر ؟ (انظر إلى "SETE AL" في الجدول السابق لتعرف المقصود)
00401C2D JNZ CrackMe.00401B96	تفحص راية الصفرZ إذا لم يكن صفرا (أي انه رمزا) فاقفز إلى شاشة الخطأ
00401C33 INC DWORD PTR SS:[ESP+18]	زوّد قيمة SS:[0012F5F4] بمقدار واحد.
00401C37 JMP SHORT CrackMe.00401C40	اقفز إلى 00401C40
00401C39 CMP BL,5	قارن BLمع 5 @@@ (سنصل إلى هنا إذا كان الحرف الأول شرطة -)
00401C3C JNZ SHORT CrackMe.00401C68	اقفز إلى رسالة الخطأ
00401C3E XOR BL,BL	صفّر BL
00401C40 LEA EDX,DWORD PTR SS:[ESP+41C]	انقل عنوان السيريال إلى EDX
00401C47 INC EBP	زود EBP
00401C48 PUSH EDX	ادفع قيمة EDX (عنوان السيريال) إلى الـ stack
00401C49 MOVSX ESI,BP	انقل قيمة BP إلى ESI
00401C4C CALL EDI	استدعي الدالة التي عنوانها في EDI (إنها دالة حساب الطول strlenA)
00401C4E CMP ESI,EAX	قارن قيمة EAX مع قيمة ESI (0x22 مع 0x1)

00401C50 JL SHORT CrackMe.00401C0F	إذا كان ESI اقل من 0x22(أي مازال هناك خانات في S/n) فاقفز إلى 00401C0F @@@@
00401C52 MOVSX EDX,WORD PTR SS:[ESP+18]	انسخ القيمة 0x1C إلى EDX
00401C57 LEA EAX,DWORD PTR SS:[ESP+1C]	انسخ عنوان السيريال من المكدس إلى EAX (انظر الصورة بالأسفل)
00401C5B PUSH EAX	ادفع قيمة EAX إلى الـ stack (المكدس)
00401C5C MOV BYTE PTR SS:[ESP+EDX+20],0	انسخ القيمة 0 إلى العنوان 12F614
00401C61 CALL EDI	استدعي دالة طول السترنج
00401C63 CMP EAX,1C	قارن قيمة EAX مع 0x1C(22=EAX) @@@@@
00401C66 JE SHORT CrackMe.00401C91	إذا تساوى اقفز إلى 00401C91



@@@ : انظر إلى أمر القفز على عنوان 00401C40.هذه القفزة تأتي بك هنا. ماذا يعني ذلك؟ إذا كان ال char(character) الذي يتم اختباره شرطة فاقفز إلى هنا. ثم سيتم اختبار : هل ترتيب هذا ال char يساوي 5؟ إذا نستنتج أن الخانة الخامسة هي الشرطة -. ليس هذا فقط. لاحظ أمر **XOR** BL,BL. إذا كانت الخانة هي الشرطة وترتيبها 5 سيتم تصفير BL وبالتالي في الدورة الثانية أي عند إعادة الاختبار من جديد للخانة التالية ستكون قيمته 1(قيمته 0 لكن هناك أمر **INC** bl)أي نستنتج أن كل خانة من مضاعفات الـ 5 يجب أن تكون -. أذن ضع نقطة توقف عن العنوان 401C0F ثم اعد التشغيل ctrl+F2 والآن ضع اليوزرنيم allko والسيريال 1234-abcd-efgh-ijkl-mnop-qrst-uvwx جيد. ها قد عدنا عند العنوان 401C0F تتبع trace over باستخدام F8

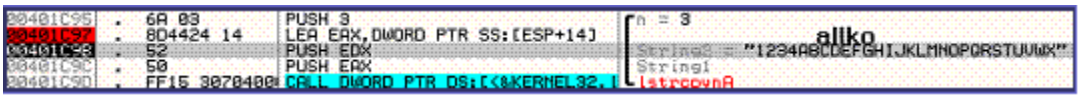
@@@@: لاختصار الوقت تتبع باستخدام F9 (مع وجود نقطة توقف عند العنوان 401C0F) وراقب ESI وعندما تصل إلى 21 فتوقف عن استخدام F9 واستخدم F8

@@@@@ : قيمة EAX تساوي 0x22 أي 34 عشري. يتم مقارنتها مع 0x1C أي 28 عشري. لاحظ العنوان 401C22 عند هذا العنوان ينقل الـ char الذي تجاوز أمر **CMP** al,2D (أي الـchar الذي ليس شرطة) إلى العنوان.هذا يعني أن السيريال بدون الشرطات يتم تخزينه هناك. لاحظ أن هناك 6 شرطات إذن طول السيريال كان 34 أصبح 34-6=28 أي 0x1c بالـ hex وهذا يفسر هذا الأمر.

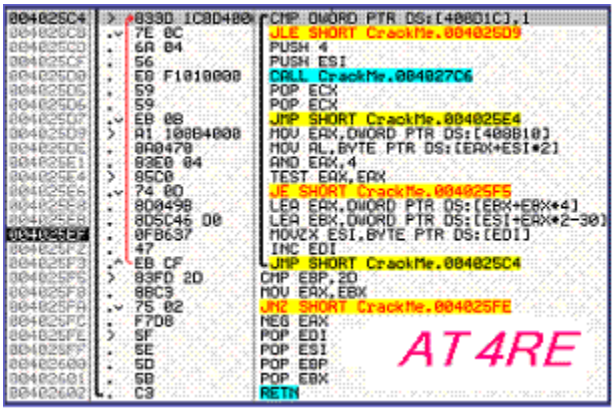
عملية التشفير الفعلية:

إلى الآن لم يكن هناك عمليات تشفير فعلية.كل ما سبق هو إجراءات للتحقق من بعض الأمور كالطول ووجود الشرطة والخلو من الرموز وغيرها.

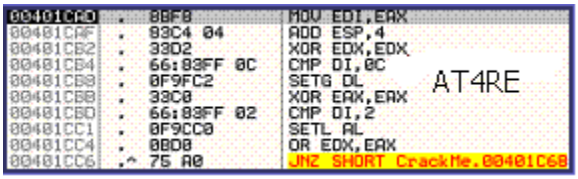
نحن الآن في كوكب المريخ عفوا, اقصد عند العنوان 401C91. الأسطر الخمس التالية تتبع دالة lstrcpynA التي تقوم بنسخ 3 chars (متضمنة 0 بالنهاية. أي عمليا تنسخ 2 فقط) من السيريال المدخل وتخزنهما في 12F5EC (لاحظ قيمة EAX)



بعد هذه الدالة بسطرين هناك اتصال 00401CA8 . E8 56090000 CALL CrackMe.00402603 على الدالة التي عنوانها 402603 والتي بدورها هي اتصال لدالة أخرى هي 402578. وهذه الصورة هي للإجرائية الثانية ضمن الإجرائية الأساسية التي تبدأ عند 402578

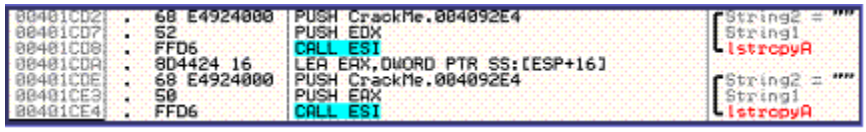


حسنا لن نقوم بشرحها تفصيليا لثلاثة أسباب. أولا هذا الدرس أصبح طويلا ولا نريدك أن تصاب بالملل. ثانيا هذه الدالة أو هذه الإجراءات غير مرتبة أو لنقل من الصعب تحديد الهدف من كل أمر بمفرده لكن يمكنك أن ترى النتيجة التي تحصل عليها بالنهاية وبقليل من المحاولة والخطأ. لاحظنا أن هذه الإجرائية بها حلقتان فقط (أي تنفذ مرتين) ولاحظنا وجود القيم 1(0x31) و 2 (0x32) وهما كما تلاحظ أول خانتين في السيريال((ضع نقطة توقف هنا ثم اعد التشغيل وحاول كل مرة بسيريال مختلف لكن لا تحاول وضع أحرف كـ v4 مثلا لان العمليات الرياضية ستنتج صفرا.))أي أنهما 12 لكن كما تعلم فـ 12 تكافئ 0xC أي أن أول خانتين يتم تحويلها إلى الصورة المكافئة بالسداسي عشري (مثلا لو كان أول خانتين 27 ستحصل على 1B بدلا من C). حسنا ها قد عدنا من هذه الدالة نحن الآن عند العنوان 401CAD



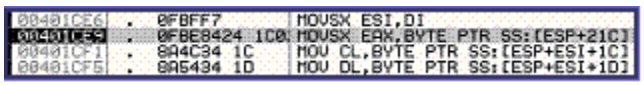
كما ترى يتم مقارنة القيمة التي حصلنا عليها للتو (0xC) مرة مع C (لا، لا تعتقد أنها تقارن مع نفسها فالقيمة c موجودة ضمن بنية البرنامج ولو غيرت السيريال لن تتأثر)

ومرة مع 2 وبقليل من الملاحظة نستنتج أنها يجب أن تكون اصغر من c واكبر من 2 إذن لو كان أول خانتين من السيريال 19 ستحولان إلى سداسي عشري أي 13 وهذه اكبر من C إذن لن يقبلها البرنامج.



هنا يتم نسخ 0 (string 2) إلى 12F5EC (string 1) في الدالة الأولى. والأمر مشابه للدالة الثانية.(string 1: 12F5F2)

أوامر الـ **MOV** الثلاثة التالي (بدءا من 401cE9) كما بالصورة :



الأول ينقل أول حرف من اليوزرنيم إلى EAX الثاني يقوم بنسخ الحرفIإلى CL. وبالنظر إلى السيريال بعد رفع الشرطات 1234ABCDEFGHIJKLMNOPQRSTUVWXنلاحظ أن حرف I هو الخانة رقم 13 (عشري). ألأمر التالي ينسخ الحرف J أي الخانة 14.

أن دققت النظر في الأمر لوجدت أن عنوان الحرف الذي يتم نسخه هو SS:[ESP+ESI+1C] حسنا القيمة 1Cثابتة إذن اتركها إما ESP فمحتواه 22 وهو طول السيريال لكن قبل إزالة الشرطات أيضا هذا لا فائدة منه إما ESI فيحتوي على C أي 12. أذن القيمة الأولى هي13= C+1 والثانية14=C+2 .

بعد ذلك بسطرين هناك دالة wsprintfA التي تقوم بتحويل char إلى قيمته بـال ascii ولو دققت النظر جيدا أثناء تتبعك لها باستخدام F8 لوجدتها تأخذ بارامترا عن طريق EAX الذي سبق وخزنا الحرف الأول من الاسم فيه. أذن الدالة ستحول حرف a ال 61 (انتبه أن اليوزرنيم بقي small letter كما هو).

أربعة اسطر أخرى وتجد دالة مقارنة تعودنا عليها. أنها IstrcmpA دقق أثناء تتبعك لها لتجد أن البارامتر الأول هو 61 والثاني IJ. ماذا تستنتج؟

أول خانتين في السيريال هما 12 أي c إذن الخانتين s[c+1],s[c+2] يجب أن يطابقا قيمة أول حرف في اليوزرنيم أي 61.

إذن غير السيريال إلى الصيغة التالية 1234-abcd-efgh-61kl-mnop-qrst-uvwx ثم اعد التشغيل وادخل السيريال المعدل.

بعد الدالة هناك التالي :

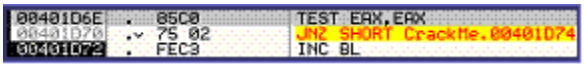
اعتقد أن كل شيء واضح. سنتجاوز القفز وستزيد قيمة BL إلى 1.

الأوامر الثلاث مشابهة للأوامر الثلاثة السابقة. بداية ينقل ثاني حرف في اليوزر إلى EAX. لاحظ أن ESI الذي يحتوي على c مضروب بـ 2.

إذن القيمة الأولى هي 2C+1 والثانية 2C+2 أي U و V. الأوامر التالية حتى 401D6C هي تكرار للأوامر السابقة.سيتم مقارنة قيمة الحرف الثاني من اليوزر

أي I=6C مع UV لذا اعد التشغيل مستخدما السيريال التالي 1234-abcd-efgh-61kl-mnop-qrst-6cwx.

ومن ثم هناك الأوامر التالية :



ولها نفس الشرح السابق.

بعد ذلك، هناك استدعاء لدالة حساب الطول. ثم تنقل الخانة قبل الأخيرة في السيريال (في مثالنا هو الحرف W) إلى cl بالأمر التالي 00401D84 . 8A4C14 1A MOV CL,BYTE PTR SS:[ESP+EDX+1A] .

وهنا 00401D95 . 8A50 FF MOV DL,BYTE PTR DS:[EAX-1] يتم نقل الخانة الأخيرة (أي X) إلى DL.

والآن الأوامر الباقية تكرر نفس الأمر. هنا 00401DAD . 0FBE940C 1B0 MOVSX EDX,BYTE PTR SS:[ESP+ECX+21B] ينقل آخر حرف في اليوزرنيم (أي o)

ليتم تحويله إلى القيمة المكافئة بـ ascii وهي 6F لتقارن الأخيرة مع WX إذن اعد التشغيل واكتب

1234-abcd-efgh-61kl-mnop-qrst-6c6F

والآن نحن هنا

لاحظ أن قيمة BL تقارن مع 3. وقد كان هناك 3 اختبارات وفي كل مرة تزيد قيمة BL.

والآن ستظهر لنا شاشة good cracker بالإضافة إلى تفعيل قائمة extra.تفحصها لتجد العديد من الأمور المثيرة للاهتمام.

الخلاصة :

اليوزرنيم يجب أن يكون 3 أحرف على الأقل.
السيريال يجب أن يكون 34 خانة بالضبط.
الخانة الخامسة ومضاعفاتها في السيريال يجب أن تكون شَرطة – (كالتي تراها فوق زر + أقصى يمين لوحة المفاتيح)
أول خانتين في السيريال#= [2]s[1]s يجب عند تحويل قيمتهما معا إلى السداسي العشري إلا تزيد قيمتهما عنا 12 (0xC) ولا تقل عن 2.

S[#+1]s[#+2]=u[1] (in hex)
S[#*2+1]s[#*2+2]=u[2] (in hex)
S[0x21]S[0x22]=u[last]

مثال 1:
دعنا نطبق الكلام السابق على اليوزرنيم control
ولنختر أول خانتين من السيريال.11 مثلا. سنحصل على #=11 ولاحظ أن أول حرف c=63 والثاني o=6F والأخير l=6C
سنحصل على :
1100-0000-0006-3000-0000-006F-006C
لاحظ أن الأصفار يمكن تغييرها كيفما شئت.

مثال 2 : اليوزرنيم Palestine
ولنختر أو خانتين 03 مثلا. سنحصل على #=03 ولاحظ أن أول حرف p=70 والثاني a=61 والأخير e=65
سنحصل على
0307-0061-0000-0000-0000-0000-0065

مثال 3 : اليوزرنيم at4re
سأختار أول خانتين ← 07
أول خانة في اليوزر هي a أي 61
إذن الخانة 07+1=8 ستكون 6
والخانة 07+2=9 ستكون 1
ثاني خانة في اليوزر هي t أي 74
إذن الخانة 15 = 1+ 07*2 ستكون 7
والخانة 16 = 2 + 07*2 ستكون4
آخر حرف في السيريال هو e أي 65
فنحصل على 0700-0006-1000-0074-0000-0000-0065
وإذا اخترنا أول خانتين لتكونا 04 مثلا فسنحصل على 0400-6107-4000-0000-0000-0000-0065
وإذا اخترنا أول خانتين 05 مثلا فسنحصل على 0500-0610-0740-0000-0000-0000-0065
وهكذا. (الأصفار يمكن استبدالهما بأي شيء)

وبهذا ينتهي أول فصول الباب الثاني. تم بحمد الله.

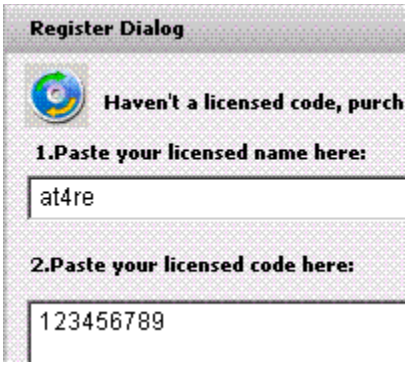

الباب
الثاني
الفصل الثاني
Patching

2. الفصل الثاني : PATCHING

1.2 . المثال الأول

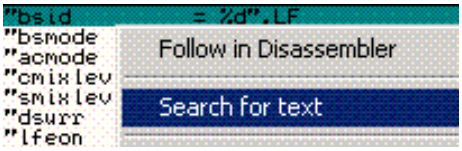
تجد الضحية في المرفقات باسم Acala DVD Copy

سنقوم بكسر هذا البرنامج السهل جدا.قم بداية بتشغيله واضغط على زر Register :

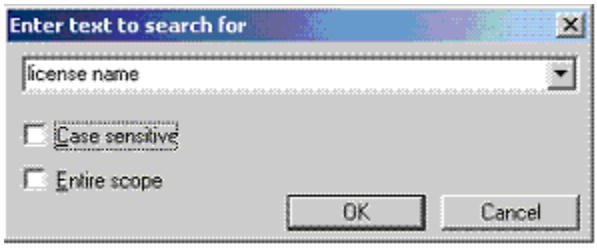


وبعدها ستظهر رسالة الخطأ التالية : Invalid license name or license code. كالعادة علينا أولا أن نقوم بفحصه بـ PEiD والنتيجة : Microsoft Visual C++ 6.0

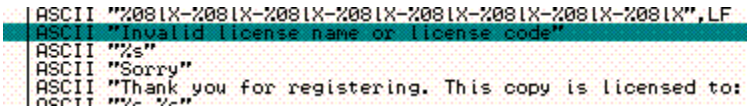
جيد لنقم بتحميل الضحية إلى olly ولنبحث عن تلك الرسالة : اضغط باليمين في وسط الشاشة واختر ➔ search for all referenced text string. بعد ظهور النافذة الجديدة، اضغط زر Home في لوحة المفاتيح للصعود للأعلى، ثم اضغط باليمين واختر search for text



ثم

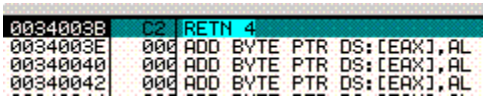


بعد الضغط على ok :

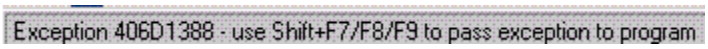


اضغط double click وستصل إلى هنا :

لنر ماذا لدينا هنا . كما ترى هناك قفزة فوق تلك التعليمة مباشرة كما يظهر بالصورة السابقة . ولو دققت فيها لوجدت أنها تأخذك إلى رسالة Thank you for registration. إذن دعنا نجرب ونعكسها . بداية قم بوضع نقطة توقف (اختصارا BP:) وذلك بتحديد القفزة (أي الضغط عليها بالزر الأيسر للماوس) ثم الضغط على زر F2 في لوحة المفاتيح . والآن دعنا نقم بتشغيل الضحية بالضغط على زر F9 في لوحة المفاتيح. ستلاحظ أنك توقفت هنا :

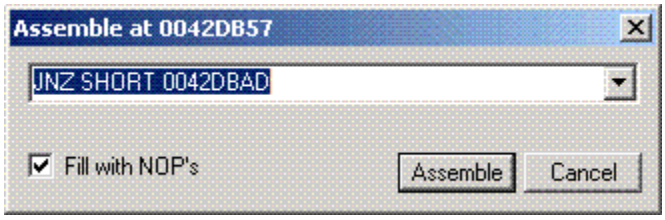


ومكتوب بالأسفل :



هذا النوع من الأخطاء يحدث كثيرا فلا تقلق. فقط اضغط Shift+F9 وستظهر نافذة التسجيل. أدخل أية بيانات ثم OK وستجد نفسك قد توقفت في olly في المكان الذي وضعنا فيه نقطة التوقف.

لنقم بعكس القفزة. اضغط زر space في لوحة المفاتيح وستظهر هذه النافذة :



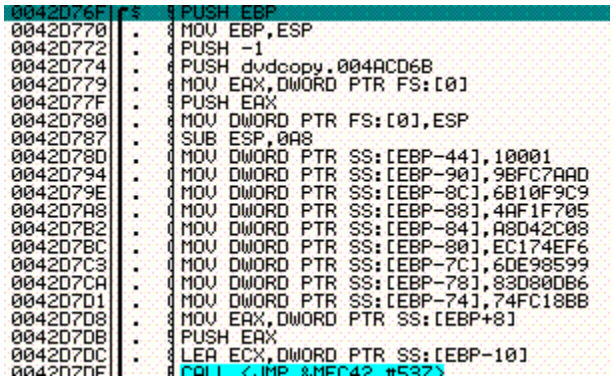
غير **JNZ** إلى **JE** (أو **JZ**.. لا فرق) ثم اضغط انتر. بعدها أغلق النافذة. والآن اضغط F9 لنر ما سيحدث. لقد ظهرت رسالة نجاح التسجيل.

لكن أغلق البرنامج (ألضحية) ثم شغله من جديد وستلاحظ أن كل شيء عاد كما كان. ما السبب**؟**

لقد غيرنا النتائج ولم نغير المسببات. رسالة النجاح أو رسالة الخطأ هي مجرد رسالة و لا يهم أي واحدة ستظهر لك فالبرنامج سيبقى غير مسجل. إذا لتسجيل البرنامج يجب أن نذهب إلى دالة الحماية التي تخبر البرنامج بأنه غير مسجل بدل الاهتمام بالمظاهر الخارجية ☺

ولو نظرت للأعلى ستجد أن أمر التحقق **TEST** EAX,EAX يتحقق من قيمة eax فإن كانت 1 فأنت مسجل وان كانت 0 أنت غير مسجل. من أين ستأتي قيمة 1 أو 0؟؟ هناك **CALL** 42D76F وبالتأكيد فهذه الدالة هي دالة التحقق. إذن ضع BP عليها وأعد الخطوات حتى تصل تتوقف عندها.

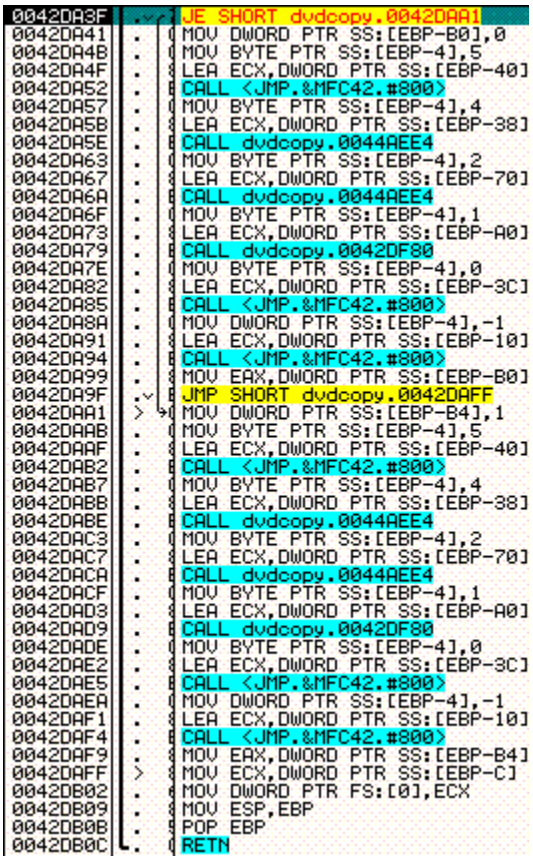
الآن اضغط F7 كي تدخل إلى داخلها.



قبل أن نبدأ التتبع نريد أن نعرف ما الذي نبحث عنه وما الذي نريد أصلا !!

نريد كسر البرنامج. وذلك يتطلب أن نبحث عن الأمر الذي يأتي بالقيمة 0 إلى Eax مما يجعل البرنامج غير مسجلا.

لاحظ أنه قد يكون أكثر من أمر يفعل ذلك لكن المفروض أن نهتم بالأوامر الموجودة في نهاية هذه الحلقة. حسنا لنفرض أننا وجدنا ذلك الأمر، ماذا سنفعل؟ سنقوم بتغييره بطريقة معينة بحيث نحصل على 1 بدلا من 0. جيد الآن دعنا نبدأ التتبع. لن نهتم كثيرا بما هو موجود في بداية الحلقة بل سنركز على نهايتها. فلنبدأ.. عند وصولك إلى هنا :



لاحظ أننا في نهاية الحلقة. وقد وصلنا إلى مفترق طرق بفعل القفزة العلوية (أول أمر في الصورة). فإما أن تنفذ القفزة ونذهب إلى الجزء السفلي من الصورة (أي الجزء بين القفزة الثانية وحتى النهاية عند **RETN**) أو أن لا نقفز ويتم تنفيذ الجزء العلوي (أي من القفزة الأولى وحتى القفزة الثانية). الوضع الطبيعي لنا – بما أننا غير مسجلون – هو أن لا نقفز. إذن المفروض أننا لو عكسنا تلك القفزة (الأولى) فسنصبح مسجلين بشكل دائم. إذن قم بعكس القفزة كما تعلمنا للتو (**JE➔JNE**)، ولنقم بحفظ التغييرات كالتالي :

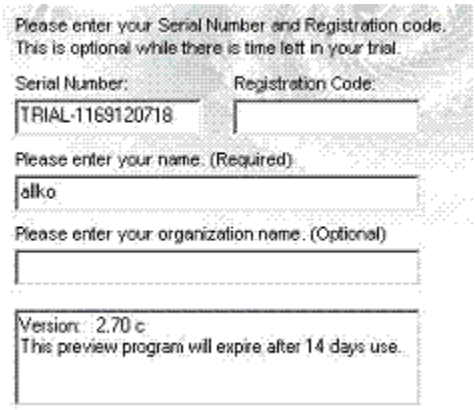
اضغط باليمين على تعليمة القفز بعدما قمت بتعديلها واختر copy to executable ➔ all modification ثم اضغط copy all وفي النافذة الجديدة اضغط باليمين واختر save file. قم بحفظ البرنامج الجديد باسم مختلف وليكن cracked.exe مثلا.

الآن دعنا نجرب واضغط على F9. رائع لقد ظهرت رسالة النجاح. لكن عند إغلاق البرنامج وإعادة تشغيله تظهر شاشة مطالبة التسجيل. لكن هذا لا يهم ما يهمنا هو البرنامج المعدل cracked.exe.شغله. ما النتيجة؟ رائع لقد نجحنا. يمكنك الآن أن تقوم بعمل patch للبرنامج (سنتعلم ذلك في الدرس القادم).

2.2 . المثال الثاني

تجد الضحية في المرفقات باسم SiteSpinner v2.7

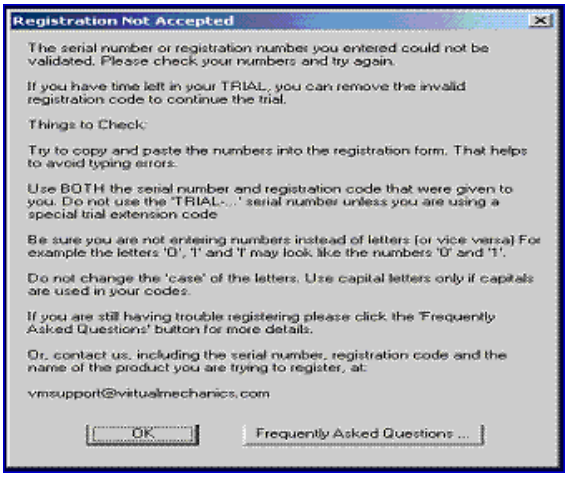
في البداية سترى splash window أولى. ثم بعدها على الفور ستأتي هذه النافذة :



في حالتك ستجد أن الرقم في الأسفل هو 15 days وليس 14، ذلك لأننا قمنا بتثبيت البرنامج في الأمس وقد مر عليه يوم كامل.

إن البرنامج منذ بدايته يقوم بعمل فحص.هل أنت مسجل أم لا؟ وبالطبع النتيجة كانت سلبية مما سبب ظهور شاشة التسجيل..

قم بكتابة أي اسم وليكن allko/AT4RE مثلا وادخل أي registration code (لا يهم ما الذي ستكتبه فنحن لا نريد عمل keygen) والآن اضغط على ok :



إذن فقد حصلنا على bad boy msg. اضغط على ok. ماذا تلاحظ ؟ عدنا إلى النافذة السابقة. عليك مسح جميع المربعات ثم الضغط على cancel كي تدخل إلى البرنامج. عموما هذا ليس ما نريد.

الفكرة هي أنه هناك فحص (في حالتنا يقع في بداية تشغيل البرنامج) حول هل البرنامج مسجل أم لا. وللوصول إلى هذا الفحص علينا البدء من تلك النافذة التي ظهرت لنا. أي شاشة طلب التسجيل (وحتى الثانية شاشة bad boy تنفع.)

ومن هنا يمكننا إتباع طريقتين لكسر البرنامج :

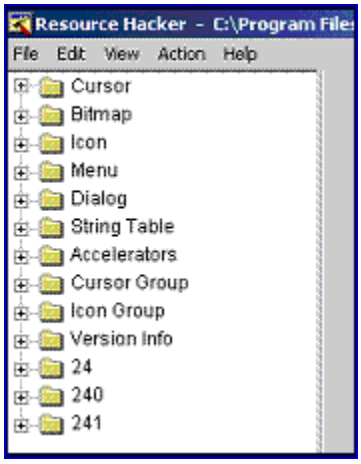
الأولى : البحث عن رسالة الخطأ (أقصد النص نفسه. أي ...search for ➔all referenced)

الثانية : البحث عن رسالة الخطأ كرسالة. أي البحث عن مكان وجود الـ MessageBoxA وذلك باستخدام أي Resource Editor

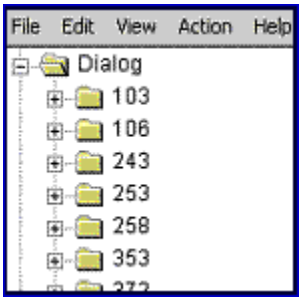
في مثالنا اليوم سنستخدم الطريقة الثانية.

أحيانا لا تنفع هذه الطريقة. عندها يجب استخدام الطريقة الأولى.

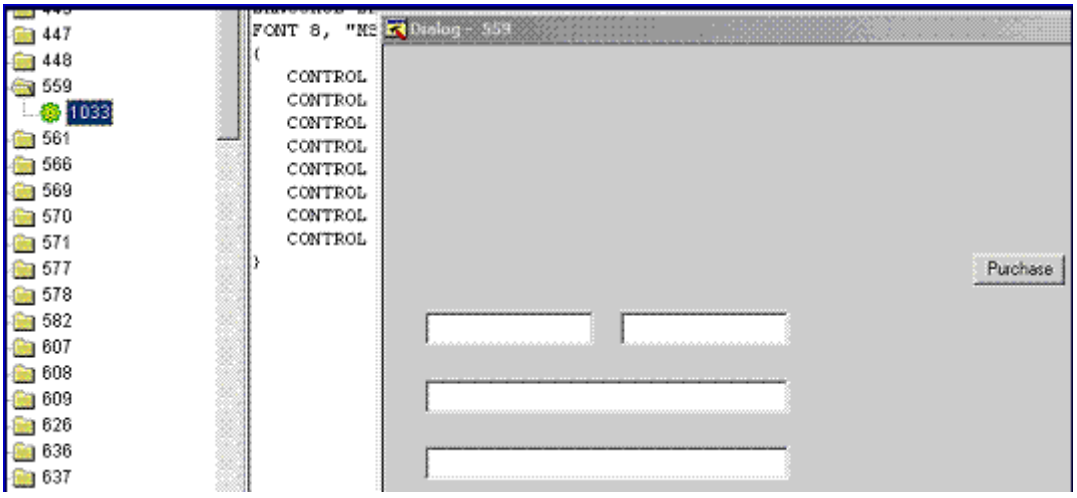
إذن قم بتشغيل برنامج ResHacker (ويمكنك استخدام برنامج PE Explore أو حتى eXescope) وبعد فتح البرنامج قم بفتح الضحية بواسطته :



والآن ما نريده هو تلك النافذة. وهذه تجدها في مجلد Dialog لذا قم بتوسيعه : (بالطبع كما قلنا، ففي بعض البرامج لن تجدها وستضطر لنسيان هذه الطريقة)



كما ترى هناك الكثير من المجلدات الفرعية. عليك توسيع كل مجلد ثم رؤية محتوياته. أستمر كذلك حتى تجد المجلد المحتوي على النافذة التي نريدها. بعد البحث المضني ستجد ما نريده هنا :

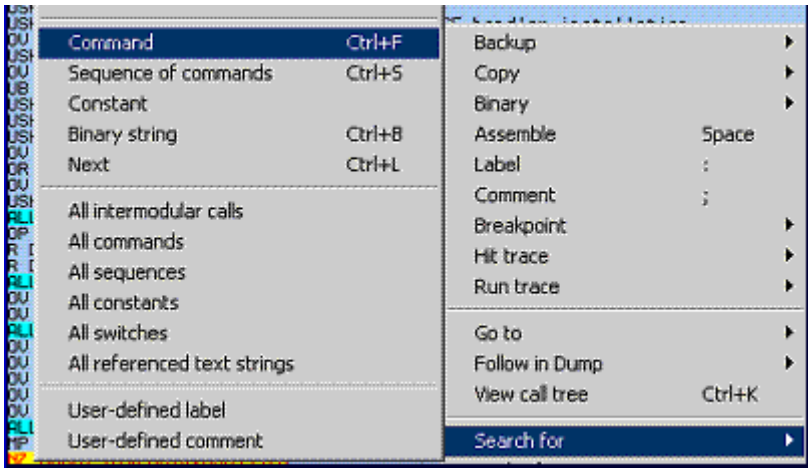


إذن شاشة التسجيل تقع في المجلد 559.طبعا هذا الرقم بالنظام العشري وعند تحويله إلى الـ hex (استخدم الآلة الحاسبة المرفقة مع الوندوز) ستجد أن 559 decimal = 22F hex

وبمتابعة البحث ستجد أن الـ bad boy msg تقع عند المجلد 3503. أي 0DAF hex.

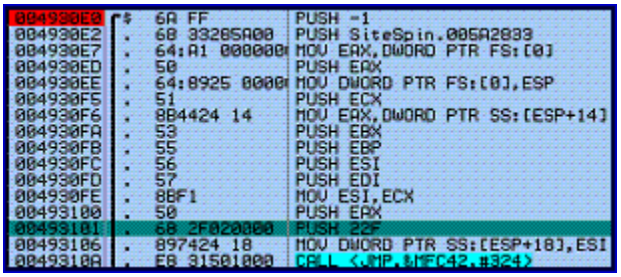
جيد والآن حان دور OllyDBG.

بعد فتح olly اضغط باليمين ثم اختر search for ثم command أو اختصارا اضغط CTRL+F



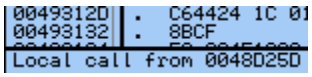
والآن الأمر المسؤول عن تلك النافذة هو 22F **PUSH** (بالمثل، الأمر المسؤول عن النافذة الثانية هو 0DAF **PUSH**)

ثم اضغط انتر :

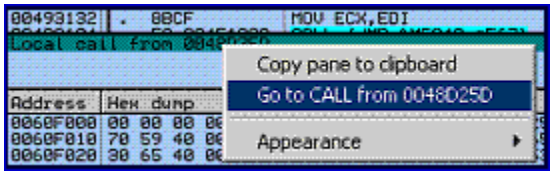


الآن لو صعدت للأعلى قليلا ستجد بداية الـ procedure (ولنسمها الـ procedure الثانية.ستعرف ما هي الـ procedure الأولى لاحقا)

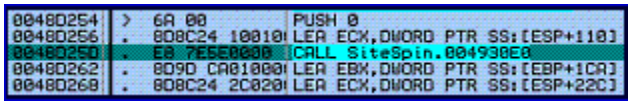
و لا مانع من وضع BP. والآن كما ترى لا يوجد قفزات تسبق ذلك الأمر – أي أمر **PUSH** 22F - حتى نتفحصها. لذا دعنا نضع المؤشر على التعليمة الأولى في هذه الـ procedure.سترى في الأسفل ما يلي :



وبالضغط على 0048D25D باليمين، اختر الذهاب إلى الدالة التي تستدعي هذه الـ procedure كما في الشكل التالي :

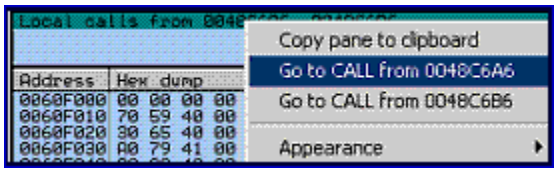


ها قد وصلنا هنا :

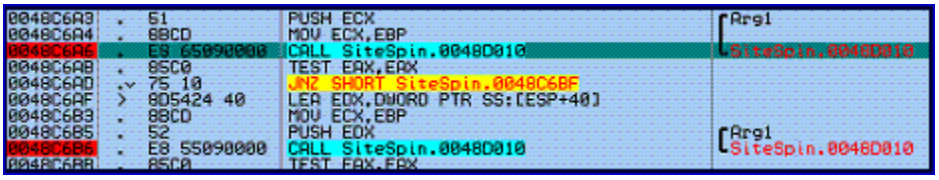


جيد حتى الآن. أذن لدينا دالة تقوم باستدعاء الـ procedure المسؤولة عن شاشة الـ registration. لو صعدت للأعلى قليلا ستجد بعض القفزات. بالطبع نحن يهمنا قفزة تمر فوق تعليمة **CALL** الموجودة عند العنوان 48D25D. لكن كما ترى جميع القفزات التي تسبق هذه الدالة والموجودة في نفس إجرائيتها لا تمر من فوقها (أي لا تتجاوزها). إذن نحن لسنا في المكان الصحيح لذا دعونا نذهب إلى بداية الـ procedure الموجود فيها هذه الدالة. وسأضع نقطة توقف عند تعليمة **PUSH** -1 عند العنوان 48D010 رغم أن هذا ليس ضروريا (أنا افعل ذلك كي لا أتوه في البرنامج)

كما فعلنا مع بداية الـ procedure السابقة (الـ procedure 2)، سنفعل مع بداية هذه الـ procedure ولنسمها الـ procedure الأولى، كما ترى هناك دالتان تقومان باستدعائها وليس دالة واحدة. لكن لو لاحظت , فهاتان الدالتان عنوانهما : 48C6A6 و 48C6B6 أي أنهما متجاورتان وليستا بعيدتين عن بعض.



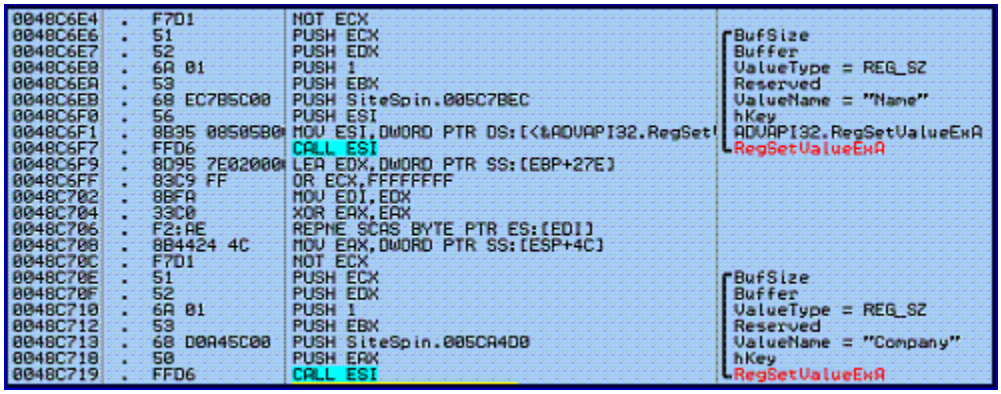
ثم :



كما ترى قمت بوضع نقطتي توقف عليهما كي تتمكن من تمييزهما في الصورة السابقة. لاحظ أيضا أن الدالتين لا توجدان ضمن procedure معينة (ببساطة وبوضوح لا يوجد ذلك الخط الجانبي الرأسي الذي تراه في الصورة قبل السابقة)

إذن قد وصلنا إلى المكان الصحيح.كل ما علينا فعله الآن هو البحث عن قفزة تتجاوز هاتين الدالتين.

لو نزلت أسفل الدالتين قليلا لرأيت التالي :

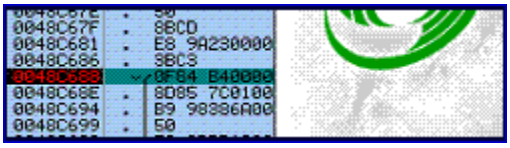


كما يتضح من اسميهما، فهاتان الدالتان لهما علاقة بتسجيل الاسم واسم الشركة. أذن يمكنا وضع التصور التالي :

في البداية يتم التحقق ما إذا كان البرنامج مسجلا أم لا. وحيث انه في حالتنا غير مسجل فإنه عند العنوان 48C6A6 يتم استدعاء الإجرائية الأولى والتي بدورها تقوم باستدعاء الإجرائية الثانية الخاصة بإظهار شاشة التسجيل. أن قمت بإدخال بيانات خطأ فستظهر لك الـ bad boy msg وبعد الضغط على ok ستقوم الدالة الموجودة عند العنوان 48C6B6

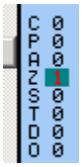
باستدعاء الإجرائية الأولى مرة أخرى لتقوم بدورها باستدعاء الإجرائية الثانية التي تظهر لك شاشة التسجيل. لكن انتبه إلى أن الاسم الذي أدخلته في المرة الأولى سيبقى موجودا.سنستفيد من هذه النقطة لاحقا.

والآن بالصعود إلى أعلى وبالابتعاد عن الدالتين (صعودا) تدريجيا، فإن أول قفزة تتجاوز الدالتين هي 048C742h **JE**عند العنوان 48C688. إذن دعنا نجرب.ضع عليها نقطة توقف ثم شغل البرنامج (F9)، أول ما يظهر هو الـ splash screen ثم نتوقف عند القفزة :



كما ترى لم أستطع تحريك الـ splash screen.

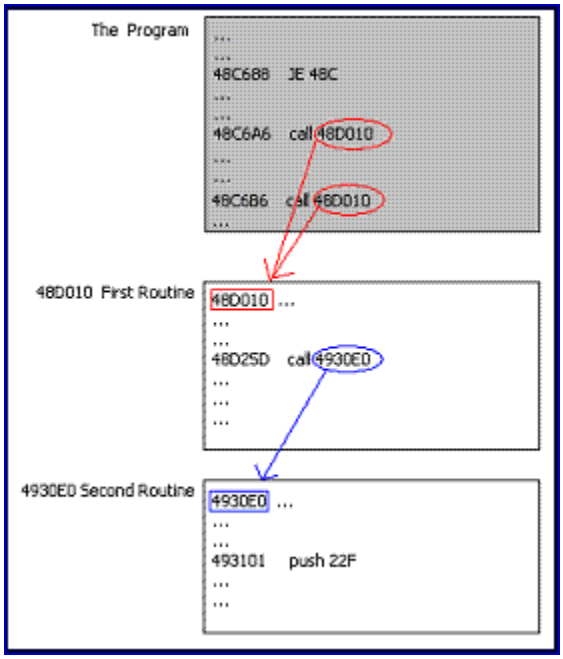
تلاحظ أن الخط الذي يشير إلى المكان الذي تقفز إليه هذه القفزة (في حال قفزت) لونه رمادي وليس احمر أي أن القفزة لن تتم (يمكن التأكد بالنظر للأسفل فسترى : Jmp is NOT taken) إذن فلنقم بعكس أحد الـ Flags وهو الـ Zero Flag : (وذلك بالضغط مرتين على القيمة 0 بجانب حرف Z)



سترى أن الـ 0 أصبحت 1. وستلاحظ تحول لون السهم بجانب القفزة إلى اللون الأحمر مما يعني أنها ستنفذ. والآن لا بأس اضغط على F9 لتشغيل البرنامج.كما ترى قد تجاوزنا الدالتين ولم تظهر شاشة التسجيل. ليس هذا فحسب بل إن المبرمج على ما يبدوا غبي قليلا.. لم يجعل برنامجه سهل الكسر بل أيضا وفر علينا عناء حقن injecting الاسم كي يصبح البرنامج مسجلا باسمنا، فكما تذكر قلت لك أن البرنامج يخزن الاسم الذي تكتبه بغض النظر عن هل السيريال صحيح أم لا. والنتيجة كما بالصورة :

رائع أليس كذلك؟

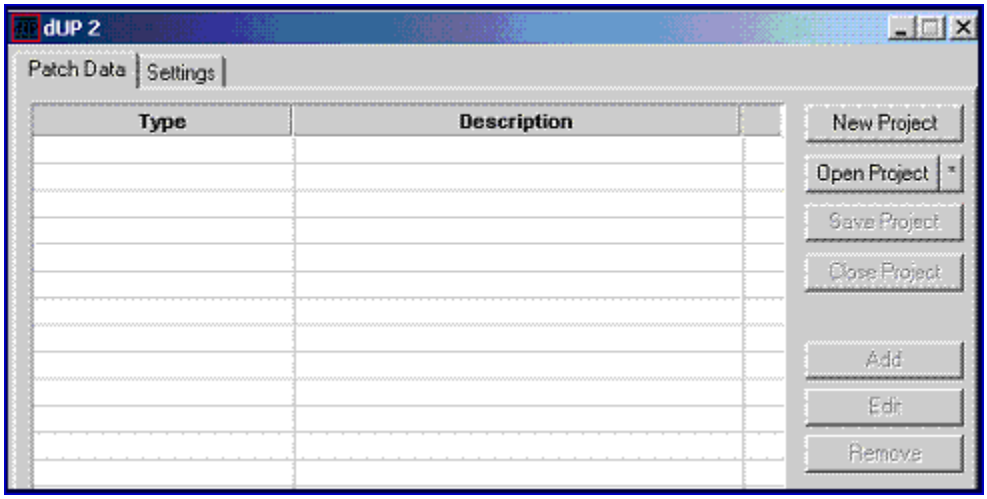
والآن ببساطة سنقوم بعكس تلك القفزة من JE إلى Jmp (حددها ثم اضغط زر space وسيخرج لك مربع التعديل. أبق كل شيء كما هو لكن غير JE إلى Jmp)ثم نحفظ التغييرات.. لن أقوم بشرح هذه الخطوة فيفترض أنك تعرفها. فقط في النهاية قمت بعمل هذه الصورة البسيطة لتوضيح مكان وجود الإجرائية الأولى والثانية و و و.



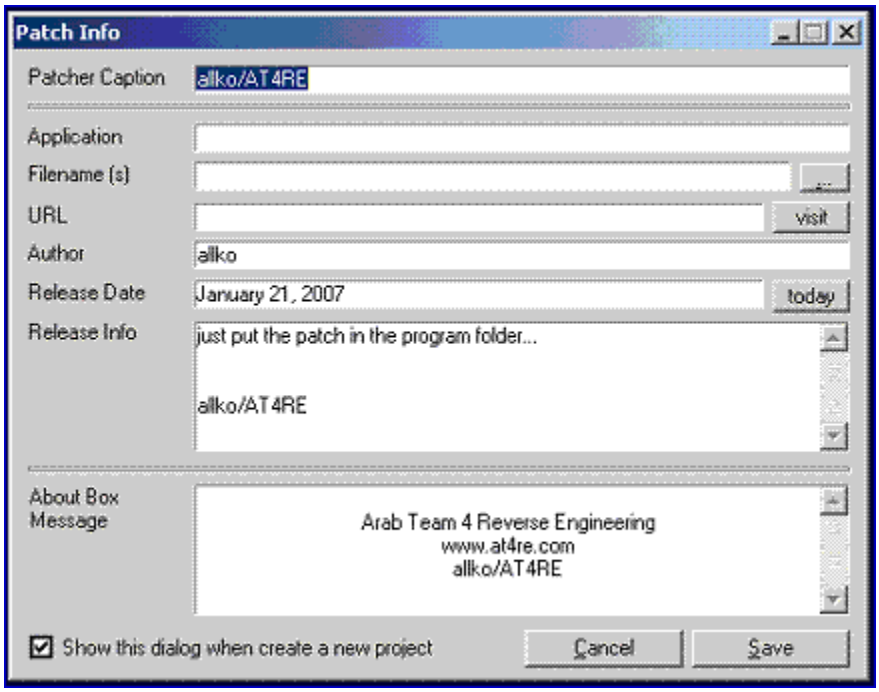
:Making the Patch

والآن كيف نقوم بصناعة الكراك (الباتش) ؟ هناك طريقتان. أما أن تبرمج الباتش يدويا ولا أعتقد أن هناك داع لها. أو أن تستخدم الطريقة السريعة وهي باستخدام برامج خاصة بهذا الغرض وأشهرها برنامج DUP وسأستخدم هنا الإصدار 2.14

بداية قم بتشغيل البرنامج لترى صورته الرئيسية كالتالي:

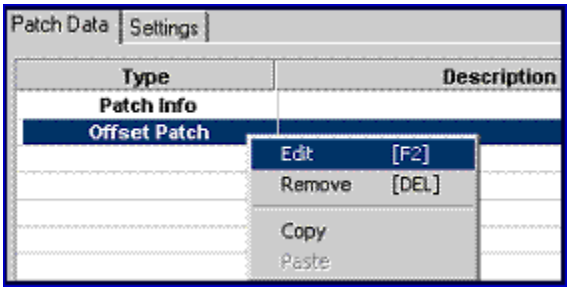


والآن اختر new project.ستظهر لك النافذة التالية :

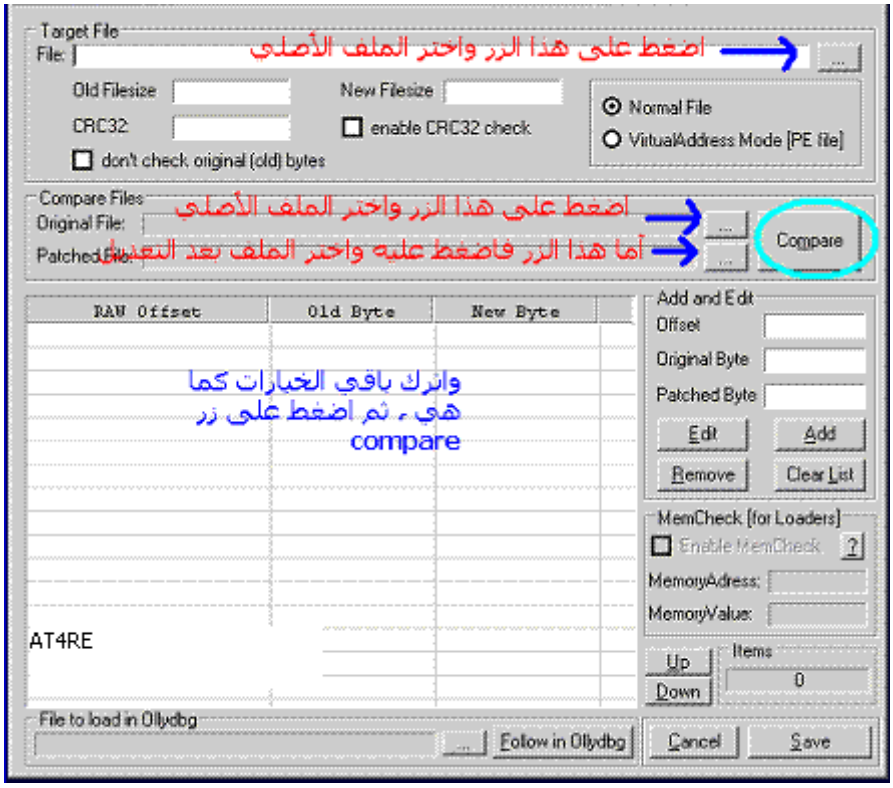


يمكنك تعبئة البيانات بالطريقة التي تحلو لك. بعد الإنهاء اضغط على زر save.ستعود للواجهة الرئيسية والآن اضغط على زر add فتظهر لك النافذة التالية :

اختر Offset Patch. الآن اضغط على كلمة offset patchباليمين :



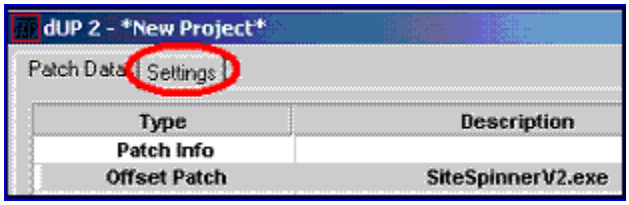
والآن ستظهر النافذة التالية :



بعد الضغط على الزر يفترض أن تصبح النافذة السابقة كالتالي :

RAW Offset	Old Byte	New Byte
0008C688	0F	E9
0008C689	84	B5
0008C68A	B4	00
0008C68D	00	90

العنوان الأول هو 8C688 (القفزة التي غيرناها). اضغط على save. لننتقل إلى صفحة الـ settings :



والآن قم بتعبئة الصفحة التالية :

المربعات التي يفضل أن تقوم بوضع علامة الصح بجانبها هي :

Custom patcher/Loader Icon : وكما هو واضح فعليك تحديد مكان الأيقونة التي تود استخدامها.

Custom Patcher Skin : هناك العديد منها, تجدها مرفقة مع النسخ الأخيرة للبرنامج.

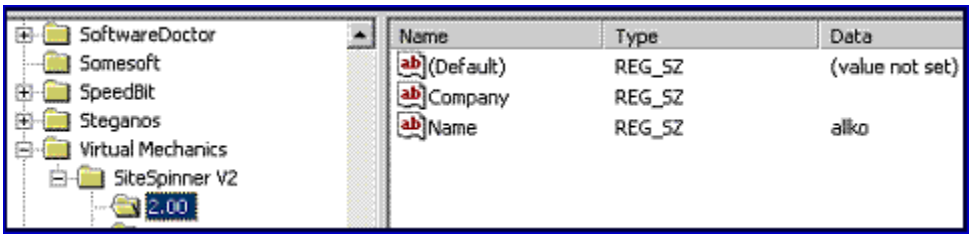
Include Trucker Module to Patcher : هذا إضافة موسيقى.

الباقي يمكنك تجربته بنفسك :) والآن هكذا نكون قد انتهينا من صنع الباتش. لكن هذا سيقوم بتغيير أمر القفز كما نريد لكن ماذا عن جعل البرنامج مسجلا باسمنا نحن؟

الحل بسيط. اتبع التالي : regedit ➔ run ➔ start وسيفتح محرر التسجيل، اتبع المسار التالي :

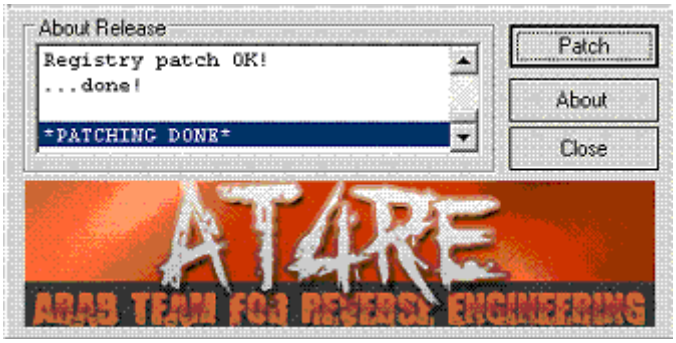
HKEY_LOCAL_MACHINE\SOFTWARE\Virtual Machine\SiteSPineer v2

والآن وصلنا إلى مجلد البرنامج. به 4 مجلدات فرعية. أبحث فيها عن مجلد يحوي قيما للاسم.ستجد أن أول مجلد وهو 2.00 هو الذي يحتوي على قيمة للاسم واسم الشركة :



اضغط على Name باليمين واختر modify وقم بتعديلها وكتابة ما يحلو لك، ونفس الشيء بالنسبة لـ company. حدد المجلد 2.00 واضغط عليه باليمين واختر export و أحفظ الملف في أي مكان تريده وبأي اسم. أغلق محرر التسجيل وقم بالضغط على الملف الذي حفظته باليمين واختر open with ثم اختر برنامج الـ notepad وانسخ جميع محتوياته.

لنعد إلى DUP. في الواجهة الرئيسية اختر زر add ثم من النافذة الجديدة اختر Registry Patch واضغط عليه باليمين واختر edit. والآن في الصفحة الجديد ألصق ما قمت بنسخه ثم اضغط على زر save. انتهينا. اضغط على زر create patch بالأسفل. وقم بحفظ الباتش بأي اسم وفي أي مكان. والآن قم بتجريبه. أنسخ الباتش وضعه في مجلد البرنامج وافتحه واختر patch :



رائع !! انتهينا !!!

3.2. المثال الثالث :


تجد الضحية في المرفقات باسم ThreatNuker v2.0.2

في درس اليوم سنتناول أحد البرامج التجارية واسمه وهو يقوم بفلترة مواقع الانترنت وحظر المواقع التي تسبب ضررا أو تشكل خطرا على جهازك. ما سنقوم به هو البحث عن النقطة التي يتم عندها اختبار هل أنت مسجل للبرنامج أم لا.

حسنا بعد تثبيت البرنامج قم بتشغيله. كما تلاحظ بالأعلى مكتوب ThreatNuker – Unregistered

وهذا الأمر كثيرا ما يتكرر في أغلب البرامج، فما الذي نستنتجه من ذلك؟ نستنتج أنه قبل تشغيل البرنامج وظهور نافذته المرئية فإنه يقوم بفحص التسجيل. جيد، دعنا نرى قائمة help :

كما ترى هناك خيار لعرض معلومات البرنامج وخيار آخر يتوسل إليك أن تقوم بالتسجيل ☺. لنر الأول :

والثاني :

قم بإدخال أي رقم للتسجيل.ستلاحظ خروج رسالة الخطأ التالية بعد الضغط على Register

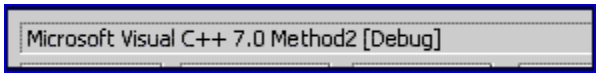
جميل، والآن دعنا نرى الخيارات المتاحة أمامنا.

نستخدم برنامج ResHacker لنعرف رقم النافذة التي تطلب التسجيل، وبالتالي نعرف أين يقوم البرنامج بطلب رسالة التسجيل وبالتتبع عكسيا نجد الأمر الذي يؤدي إلى تحديد ما إذا كان البرنامج مسجلا أم لا.

لن نستخدم هذا الخيار في هذا المثال. في أمثلة أخرى سنستخدمه بإذن الله.

نبحث عن جملة unregistered ومن ثم نتتبع عكسيا إلى أن نجد المكان الذي يؤدي إلى تحديد ما إذا كان البرنامج مسجلا أم لا.

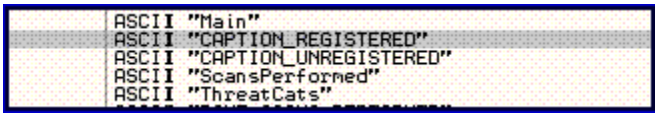
إذن دعنا بداية نفحص البرنامج بـ PEiD :



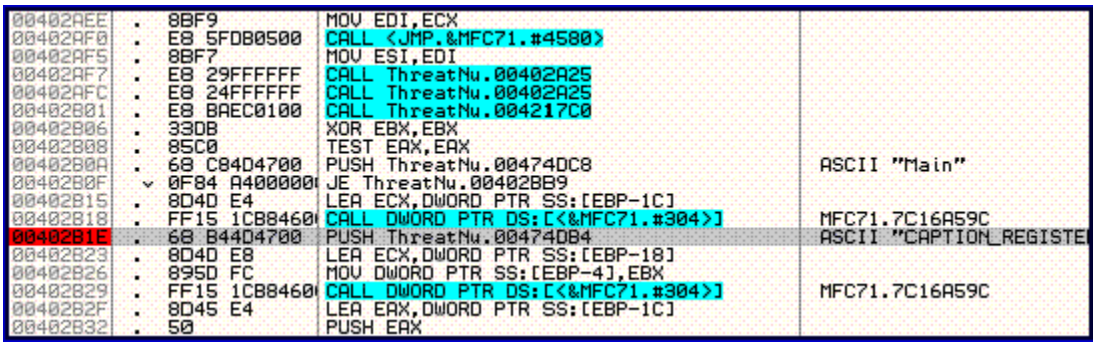
غير مضغوط أو محمي. لننتقل إلى olly.

بعد تحميل الضحية حاول البحث عن رسالة الخطأ. لن تجدها أليس كذلك؟ بالفعل. يمكنك رؤيتها أثناء تتبعك للبرنامج لكنك لن تحصل عليها بمجرد البحث في all referenced text strings ☺

إذن ما الحل. من أين سنبدأ؟ سنبدأ من الجملة المكتوبة بالأعلى : unregistered. ابحث عنها وستجدها هنا :



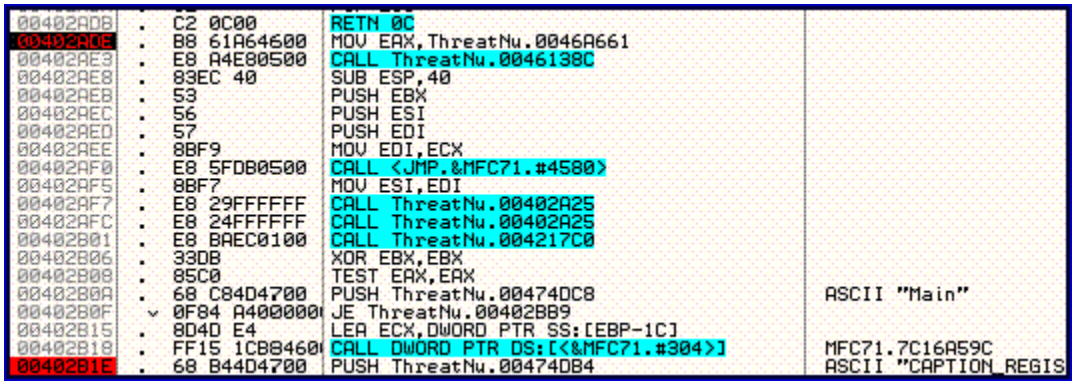
جميل. لدينا الكلام الذي سيظهر في الحالتين (إذا كان البرنامج مسجلا أم لا) إذن ضع نقطتي توقف على تلك الجملتين. ثم اضغط انتر على الأولى (المظللة بالرمادي في الصورة) وستصل إلى هنا :



رائع. أذن نريد أن نتتبع عكسيا. أي أننا سنذهب إلى الأوامر التي تسبق هذا الأمر. ما علينا فعله هو الصعود للأعلى لمسافة مناسبة ثم وضع نقطة توقف، وتشغيل البرنامج. وسنرى هل توقفنا عند تلك النقطة أم توقفنا عند النقطة السفلية؟ إذا توقفنا عند تلك النقطة فهذا يعني أن عملية التحقق من التسجيل تبدأ إما قبلها أو من عندها وإذا لم نتوقف معنى ذلك أن البداية تقع بين النقطتين أو عند النقطة السفلية.حينها نقوم بوضع نقطة توقف في المنتصف ونكرر العملية فإما أن نصل نتيجة وإما أن لا نصل حينها نكون قد ضيقنا نطاق البحث.هذا أشبه بما يعرف بالـ Binary search.

لكن انتظر لا تقلق لن نستمر في ذلك للأبد. نقوم بعمل ذلك مرة أو اثنتين على الأكثر.راقب معي:

- اصعد للأعلى. أصعد أكثر وأكثر. ماذا وجدت؟ هناك loop. جيد سنقوم بوضع نقطة التوقف أسفل منها مباشرة (أي على الأمر **PUSH** esi عند العنوان 402A4F) ثم نعيد نشغل الضحية F9.كما ترى لم نتوقف عندها للأسف. حسنا ما هو المكان الثاني الذي سنختاره لوضع نقطة التوقف.؟ لا بأس
- انظر للأمر **RETN** 8 عند العنوان 402A7E وضع نقطة توقف أسفل منه. ألنتيجة سلبية، إذن نبحث عن مكان آخر.
- انظر للأمر **RETN** 8 عند العنوان 402AA5 وكرر نفس الأمر. لا فائدة.
- انظر للأمر **RETN** 0C عند العنوان 402ADB وكرر نفس الأمر.رائع يبدو أن الأمر الذي يليه مباشرة هو فعلا بداية عملية التحقق



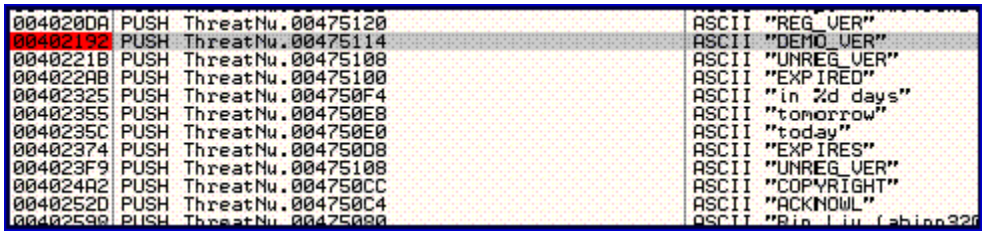
إذن ما بين هاتين النقطتين تقع عملية التحقق. لاحظ أن أمر **JE** عند العنوان 402B0F غير ملون باللون الأصفر لدي وذلك لأنني قمت بتغييره في وقت سابق فلا تقلق فلا شيء مختلف.

الدوال المسؤولة عن التحقق هي واحدة من 4 دوال تظهر في الصورة. (وقد تكون الدوال الأربع تتعاون وتكمل بعضها الآخر). إذن علينا التتبع بداخل كل منها. قد يقول قائل : لم لا نعكس تلك القفزة ببساطة !! صحيح هذا سيحل مشكلة جملة unregistered بالأعلى لكنك بعد عمل scan ومحاول الضغط على immunize ستلاحظ أن نافذة التسجيل ستخرج لك !!

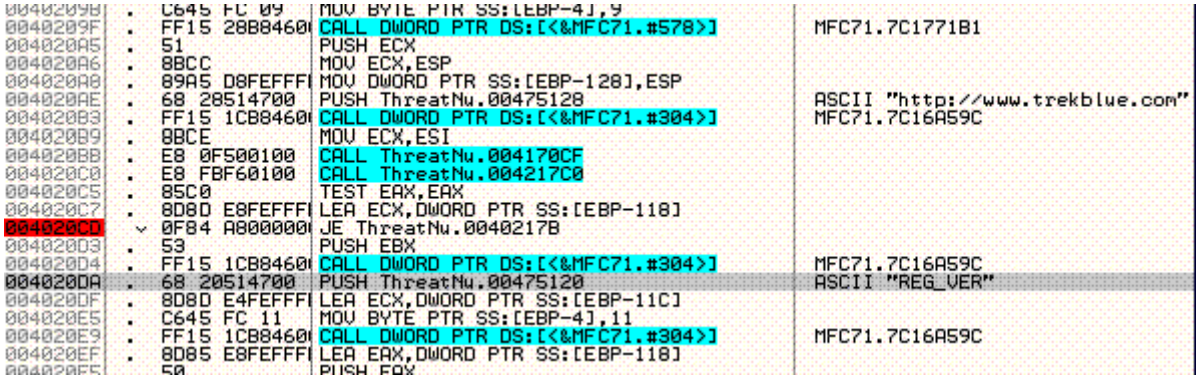
إذن دعنا نقوم بعمل التغيير في دالة التحقق نفسها وليس في أمر القفز الذي يتبعها. كما ترى أمر القفز معتمد على مقارنة تسبقه :

TEST EAX,EAX فلو كانت قيمة eax=0 هذا يعني أن البرنامج غير مسجل. والآن عملية التحقق التي بناء عليها تتغير قيمة eax موجودة في إحدى الدوال الأربع في الأعلى. هل يجب أن نتتبعها جميعها؟ في الواقع لا هناك طريقة أسهل

نحن غير قادرين على تحديد منطقة الكود المسؤولة عن زر immunize والذي بدوره يتطلب التحقق من التسجيل, لكننا قادرون على تحديد منطقة الكود المسؤولة عن نافذة about (ونحن نعلم أن تلك النافذة يجب أن تقوم بعملية التحقق كي تخبرك هل أنت مسجل للبرنامج أم لا). ارجع للصورة بالأعلى وستجد في الوسط كلمة demo version إذن اذهب إلى all referenced text strings ➔ search for ثم ابحث عن demo :

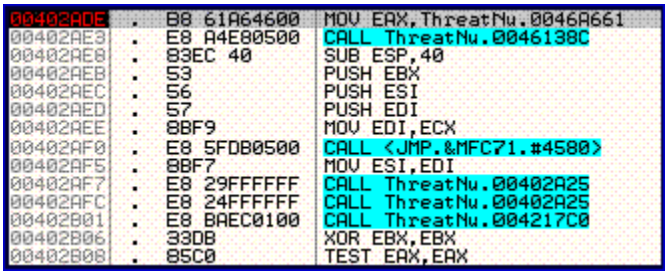


ها قد وجدنا demo. لكن لاحظ ما يوجد أعلى وأسفل منها. expired , unreg , reg وهي الجمل التي من ممكن أن تظهر لنا بناء على حالة التسجيل. لا بأس اختر reg_ver واضغط انتر عليها (نريد أن نحدد بداية هذا الكود لذا لم نقم بالضغط انتر على demo لأنها ليست في البداية)

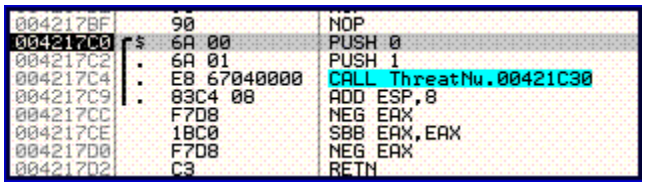


دقق النظر فيما يوجد من دوال أعلى reg_ver.كما ترى عند العنوان 4020C0 هناك 4217C0 **CALL** !! بماذا يذكرك ذلك؟ إنها نفس الـ **CALL** التي كانت ضمن الـدوال الأربع التي تحدثنا عنها. أذن؟ إذن لا داعي لفحص الدوال الأربع جميعها فهذه فقط هي التي تعنينا ☺

حسنا أبق نقطة التوقف هنا ربما نود العودة لاحقا. والآن ارجع إلى مكاننا الأصلي عند العنوان 402ADE

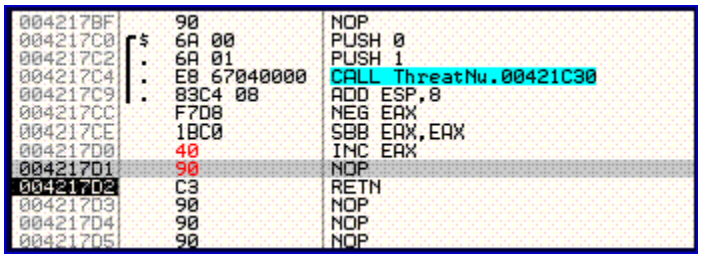


هاهو مبتغانا. أخر دالة فيهم. عندما تصل إليها اضغط F7 لتر التالي :



اعرف أن الخط الأسود الرأسي غير كامل لكن ذلك كما قلنا سببه أننا عدلنا على البرنامج في هذا المكان تحديدا.

كما ترى هذه الدالة صغيرة لا يوجد بها سوى بضعة أسطر لكن هناك دالة أخرى بالتأكيد كل عملية التحقق موجودة بداخلها. لكن انتظر لم نقوم بالدخول إلى تلك المتاهات؟ انظر ماذا يوجد أسفل من الدالة. أوامر NEG eax و sbb eax,eax ثم NEG eax..تتبع باستخدام F8 وعند تجاوزك للدالة ستلاحظ أن eax قيمته صفر. وبعد تجاوز أمر NEG eax الأول فمازالت قيمته صفر. وبعد تجاوز الأمر الثاني والثالث أيضا بقي الصفر في مكانه. جيد. كل ما نريده أن نحصل على 1 بدل الصفر. ببساطة استبدل أمر NEG eax الأخير بـ **INC** eax ولنرى ماذا سيحدث :



جيد. والآن احفظ التغييرات. قم بإعادة تشغيل الضحية ctrl + F2. ثم شغل الضحية F9 ثم اجر التغيير وأكمل تشغيل البرنامج. ماذا تلاحظ؟ اختفت جملة unregistered حسنا قم بعمل scan وكما تلاحظ يمكن الضغط على immunize بكل سهولة. إذن أستطيع أن أقول أننا قد نجحنا. يمكنك الآن حفظ نسخة من البرنامج المقرصن واستخدام أي برنامج لصنع باتش. أيضا يمكنك البحث عن العبارات الموجودة في نافذة الـ about وتغييرها إلى ايميلك واسمك بدل ايميل واسم المبرمجين إن أحببت.

4.2. المثال الرابع :

تجد الضحية في المرفقات باسم AIO Flash Mixer

في هذا الدرس ستكون عملية الكسر مختلفة قليلا عما عهدناه و لنفترض ان الضحية لا يمكن تشغيلها مع الديبغر.

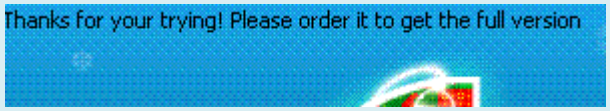
فقد اعتدنا أن نقوم برؤية رسالة الخطأ ثم البحث عنها، وبعد إيجادها نعود للخلف قليلا فنجد دالة التحقق من كونك مسجلا أم لا، فنقوم بالتغيير فيها. لكن هنا مثلا توجد فوق رسالة الخطأ دالة باسم **CALL** eax أي أن عنوان الدالة سيكون موجودا في eax مما يعني (اذا اخذنا الفتراض السابق بعين الاعتبار) أنك لن تستطيع معرفة العنوان إلا إذا تم تشغيل الضحية، وبالطبع فالضحية لن تعمل مع وجود debugger.

وان لم يكن الأمر بهذه الصورة، فالطرق التقليدية في التتبع ستكون مزعجة لعدم قدرتك على عمل BPs.

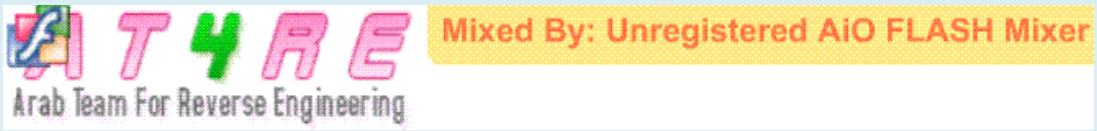
إذن حيث أننا غير قادرين على إيجاد منطقة الحماية والتحقق، فإننا سنقوم بإيجاد آثار الـ unregistration. أي الـ limitations الناتجة عن عدم التسجيل ومن ثم نقوم بالتغلب عليها كل على حدة.

لنقم بتشغيل الضحية ولاحظ وجود الـ limitations التالية :

- في الـ splash screen :

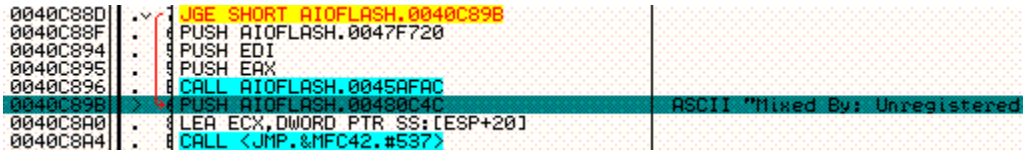


- فور تشغيل الضحية هناك نافذة اسمه why should I register وهي عبارة عن عرض فلاش.
- في قائمة file هناك خيار اسمه upload flash مخصص للمسجلين فقط.
- هناك 2 logos تظهر في عروض الفلاش بعد حفظها.
- لعمل مشروع فلاش، اختر add pictures ➔ edit واختر أي صورة ثم add ثم save ➔ file ثم save as SWF. والآن قم بفتحه وسترى الـ WATERMARK الأول هو جملة mixed by... والثاني هو أيقونة البرنامج الظاهرة في الزاوية اليسرى:



- و عند إغلاق البرنامج يطلب منك مقترحاتك للتطوير. وبالطبع لا نريد أن نقترح ☺
- خاصية download new cliparts لا تعمل.

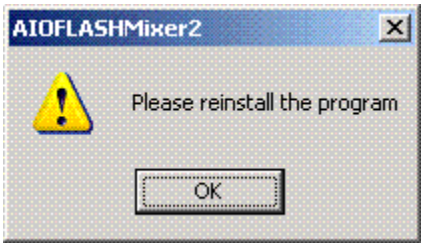
بداية يخبرنا PEiD أن البرنامج غير محمي أو مشفر. أذن دعونا نبدأ بالـ limitation رقم 4 وهو جملة mixed by. حمل الضحية إلى OllyDbg وابحث عن تلك العبارة. ها هي :



مما لاشك فيه أننا نريد البحث عن قفزة تتجاوز هذه العبارة. اصعد للأعلى وانظر إلى جميع القفزات. أستمر حتى تجد قفزة تتجاوز هذه العبارة. ألقفزة الوحيدة التي تتجاوزها هي هذه :

```
0040C5EC  .  8B0D 0CE04000  MOV EAX,DWORD PTR DS:[E
0040C5F2  .  85C0           TEST EAX,EAX
0040C5F4  .v 0F85 D2CA1200  JNZ AIOFLASH.0040CA12
0040C5FA  .  6A 00          PUSH 0
0040C5FC  .  6A 00          PUSH 0
0040C5FE  .  6A 00          PUSH 0
0040C600  .  6A 00          PUSH 0
0040C602  .  6A 01          PUSH 1
0040C604  .  6A 0C          PUSH 0C
0040C606  .  51             PUSH ECX
```

قم بتغييرها إلى jmp ثم احفظ التغييرات. شغل الضحية (خارج OllyDbg طبعا). ماذا تلاحظ؟



يبدوا أن البرنامج يستخدم crc check واكتشف أننا قمنا بتغيير أحد البايتات. ويبدوا أيضا أن المبرمج غبي قليلا لأنه أظهر لنا هذه الرسالة، حيث سنقوم بتتبعها وكشف مكان وجود الـ crc check بكل سهولة !! اذهب إلى OllyDbg وابحث عن تلك الرسالة.

كما سنرى ان شاء الله في الملحق رقم 5 ، خوارزمية crc32 هي خوارزمية تستخدم لكشف أي تعديل على البرنامج. أي أن أي عملية patch للضحية لن تجدي نفعا لأن البرنامج عند بدء التشغيل سيقوم بالتحقق من وجود أي عملية patching. الفكرة تقوم على حسابات معينة، فأحيانا تقوم هذه الخوارزميات باستخدام حسابات الحجم. فمثلا عند تغييرك لأحدى التعليمات في الضحية فالتعليمة الجديدة غالبا ستكون إما أكبر أو أصغر حجما من سابقتها (من النادر أن تكون كحجم سابقتها تماما) وبذلك فإن الحجم الجديد للبرنامج سيختلف مما يعني حدوث تغيير عليه. بالطبع هذا أسلوب واحد وهناك عدة أساليب للتحقق من كون البرنامج قد تم التعديل عليه أم لا.

جدير بالذكر أن وجود خوارزمية crc32 لا يعني أنها مستخدمة في عملية توليد السريال، فقد تكون مستخدمة في البرنامج نفسه.

```
00402A9B  .v 75 11          JNZ SHORT AIOFLASH.00402AAE
00402A9D  .  6A 00          PUSH EBX
00402A9E  .  6A 00          PUSH EBX
00402A9F  .  68 1CF34700    PUSH AIOFLASH.0047F31C          ASCII "Could not load
00402AA4  .  E8 57D80000    CALL <JMP.&MFC42.#1200>
00402AA9  .v E9 FE0F0000    JMP AIOFLASH.00402DAA
0047F31C=AIOFLASH.0047F31C (ASCII "Could not load INI file. Please reinstall
```

حسنا هاهي. وهناك قفزة فوقها. والقفزة تتجاوز الرسالة. أذن غير القفزة إلى jmp واحفظ التغييرات. ألنتيجة : يعمل بلا مشاكل ☺

لنتأكد من إزالة جملة mixed. أنشئ مشروعا جديدا. رائع لقد اختفت.

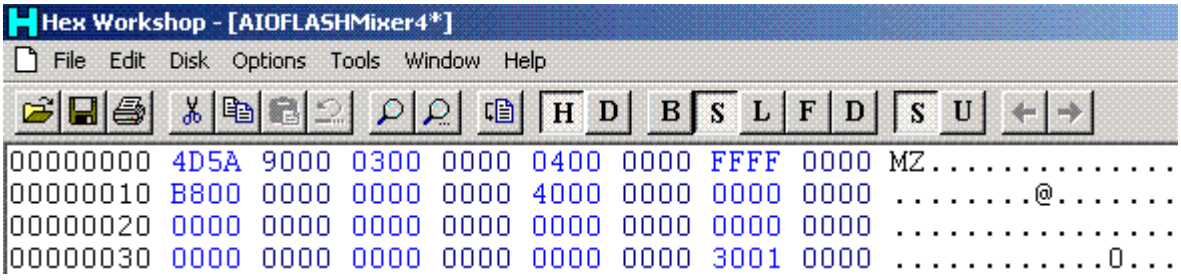
لننتقل إلى الـ limitation المتمثل بأيقونة البرنامج التي توضع في كل عرض فلاش. هناك احتمال كبير أن تكون هذه الأيقونة عبارة عن ملف فلاش موجود بداخل مجلد البرنامج. فعلا لو نظرت إلى المجلد الفرعي Resources فستجد تلك الأيقونة باسم button.swf إذن عد إلى OllyDbg وابحث عن هذه الكلمة button.swf.

بعد القيام بتغيير أول قفزة، حفظنا الملف الجديد باسم AIO2.exe، وعندما أردنا تغيير القفزة الثانية قمنا بتحميل AIO2.exe إلى OllyDbg أي قمنا بتحميل الضحية المعدلة تمهيدا لإضافة المزيد من التعديلات، وفعلا بعد تعديل ثاني قفزة حفظنا الملف الجديد باسم AIO3.exe والآن عندما نريد البحث عن button.swf فإن سنفتح AIO3.exe في OllyDbg وهكذا.

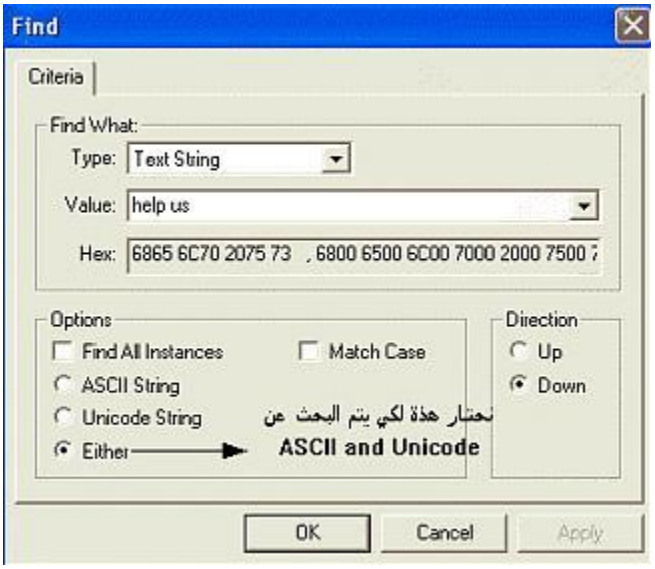
```
004250BD  .   MOV EAX,DWORD PTR DS:[EDX+D8]
004250C3  .   TEST EAX,EAX
004250C5  .v  JNZ AIOFLASH.00425155
004250CB  .   MOV EAX,DWORD PTR SS:[EBP-30]
004250CE  .   PUSH AIOFLASH.00483088              ASCII "\Resources\button.swf"
004250D3  .   ADD EAX,10C
```

جميل. غير القفزة إلى ☺ jmp قم بالتجربة للتأكد.

والآن سننتقل إلى limitation آخر وهو نافذة طلب الاقتراحات. أذن سنبحث عن help us to improve. و سنستعين ببرنامج آخر هو Hex WorkShop. بعد تحميل الضحية إليه سيبدو البرنامج هكذا :



جيد والآن دعنا نبحث عن تلك العبارة. find ➔ edit



وسنصل إلى هنا :

```
001D0F90 C000 C880 0000 0000 0800 0000 0000 F000 ................
001D0FA0 6A00 0000 0000 4800 6500 6C00 7000 2000 j.....H.e.l.p. .
001D0FB0 7500 7300 2000 7400 6F00 2000 6900 6D00 u.s. .t.o. .i.m.
001D0FC0 7000 7200 6F00 7600 6500 2000 7400 6800 p.r.o.v.e. .t.h.
```

والآن سنقوم بتصفير (حذف) هذه الـ string. وبالتالي لن تظهر تلك النافذة لأنها تعتمد على هذه السترنج والتي ستصبح غير موجودة

احتمال نجاح هذه الطريقة مرتفع، لكن بالطبع قد لا تنجح دائما

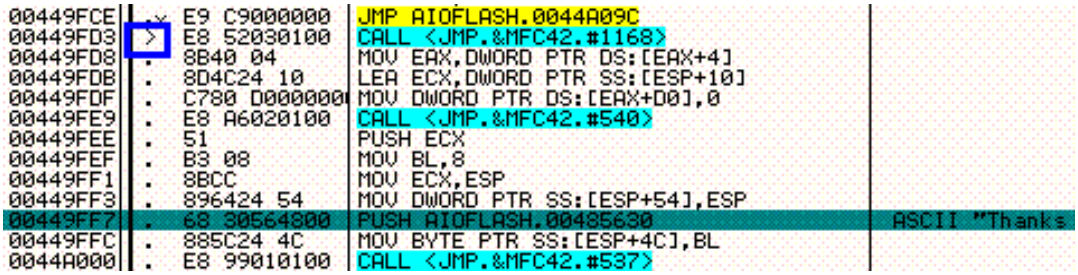
بعد تصفيرها ستصبح هكذا :

```
001D0FA0 6A00 0000 0000 0000 6500 6C00 7000 2000 j.....e.l.p. .
001D0FB0 7500 7300 2000 7400 6F00 2000 6900 6D00 u.s. .t.o. .i.m.
```

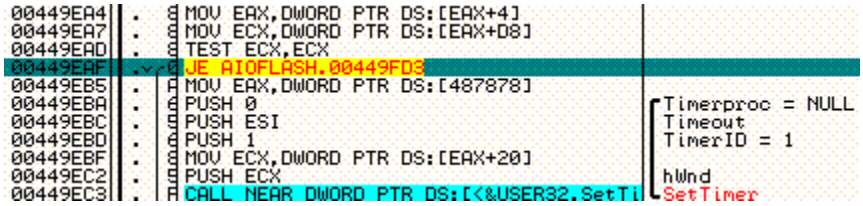
لاحظ أننا صفرنا أول حرف فقط وليس السترنج كلها. لا فرق في الحالتين. والآن لحفظ التغييرات save ➔ file. جرب. ألنتيجة : لم تظهر !! والآن لنقم بعمل نفس الشيء لكن مع نافذة order AIO flash mixer.

(استخدم زر f3 للتنقل بين النتائج حتى تصل إلى العبارة التي نبحث عنها).

والآن نريد التخلص من الـ limitation رقم 1. شغل OllyDbg وحمل الضحية (النسخة التي بها أحدث التعديلات) وابحث عن thanks for your trying

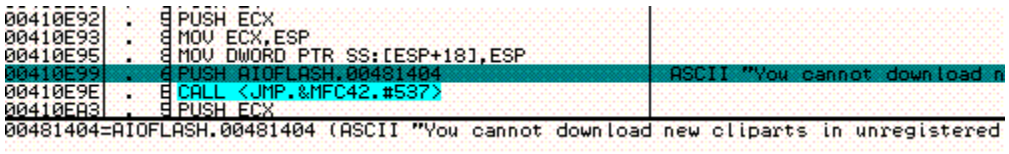


كما تلاحظ لا يوجد قفزة مشروطة فوقها مباشرة، لكن أسفل القفزة الغير مشروطة يوجد علامة (عليها مربع أزرق) تدل على أن هناك قفزة تأتي بك إلى هذا المكان. إذن فلنذهب إليها.(تتبع السهم. أو انظر إلى قسم التعليقات السفلي لتر العنوان :Jump from 00449EAF)



إذن لنقم بتغييرها إلى **jne** واحفظ التغييرات وجرب. رائع لقد تم استبدال تلك العبارة بـ registered version

الآن سنقوم بالتغلب على الـ Limitation الخاصة بالـ clip art. ابحث عن unregistered version :



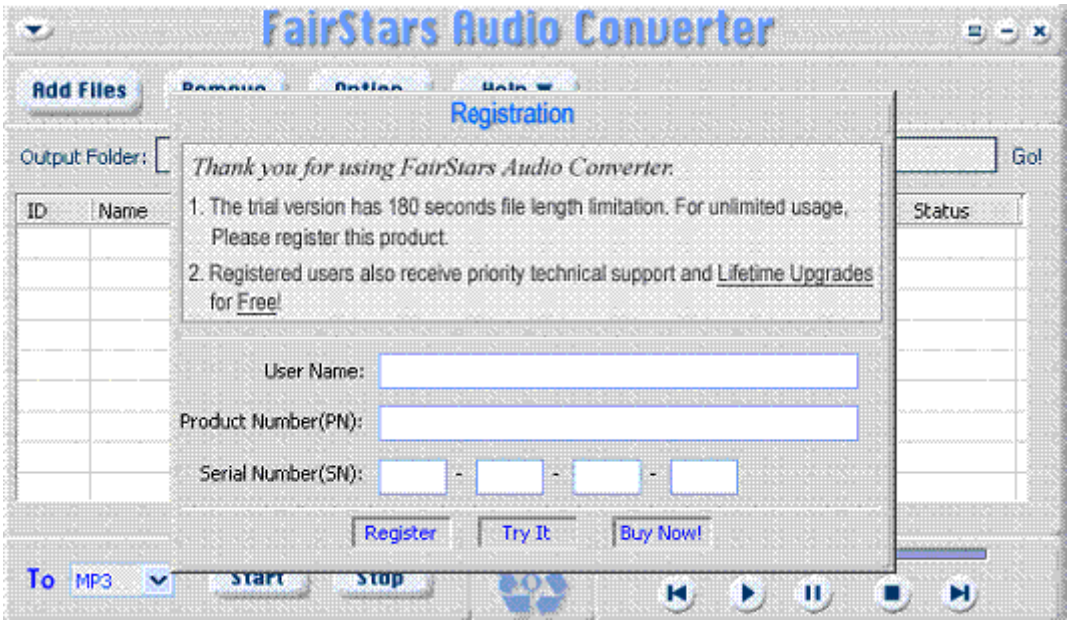
لو صعدت للأعلى ستجد قفزة. لو عكسنا القفزة فلنم نمر بهذه العبارة. إذن اعكسها واحفظ التغييرات وجرب.

هكذا نكون قد انتهينا تقريبا. بالنسبة للـ limitation الاخير المتمثل بخاصية upload flash فسنتركها لك كتمرين. بقي أن نقول أنه يمكنك تعبئة بيانات التسجيل التي تريدها في الملف المسمى settings والموجود في مجلد البرنامج.

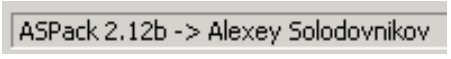
5.2 . المثال الخامس :


تجد الضحية في المرفقات باسم Fair Stars Audio converter v1.5

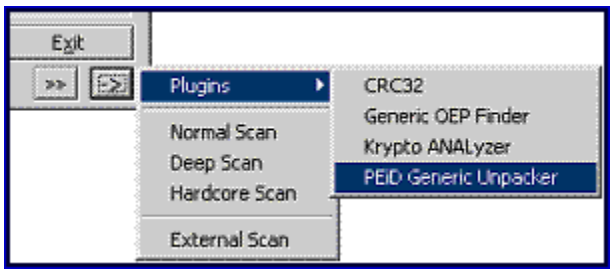
دعنا نقم بتشغيله :



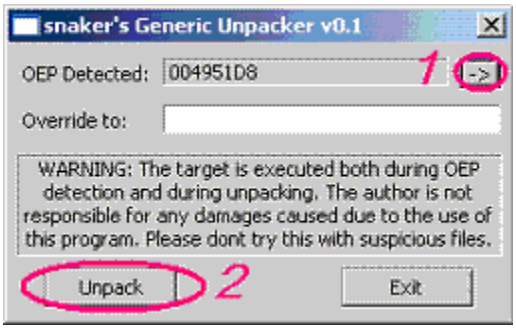
أدخل البيانات التالية : at4re ثم 123456 ثم 1111111111111 ولنرى ما سيحصل :

حسنا. لنفحص البرنامج : 

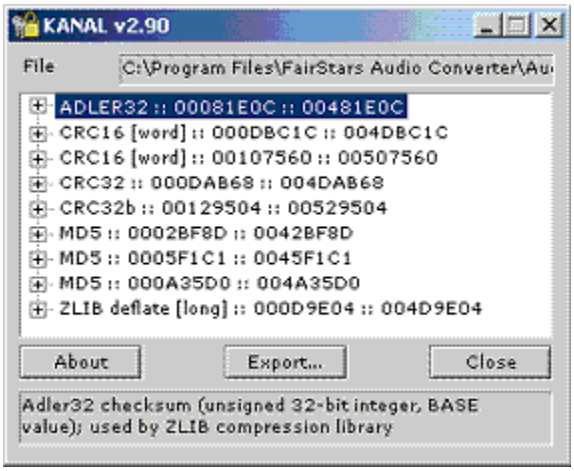
حماية بسيطة. لنقم بفكها إما باستخدام quick unpack 1.0 أو باستخدام الـ Plug in التي تأتي أصلا مع peid والمسماة peid generic unpacker



ثم



وستجد أن البرنامج بعد فكه موجود في نفس المجلد باسم AudioConverter.exe.unpacked_ والآن قم بفحصه للتأكد من وجود crc check والنتيجة :



في الواقع بعد أن انتهيت من كسر البرنامج اكتشفت أن الـ patching نجح ولا وجود للـ crc check !!! تذكر الملاحظة التي أوردتها في الدرس الماضي.

والآن لنقم بتحميل البرنامج إلى OllyDbg : (قد تظهر بعض الرسائل التحذيرية في بداية التشغيل.

دعك منها)

هام : قد تختلف العناوين في جهازي عنها في جهازك

والآن ابحث عن رسالة الخطأ PN.ها قد وجدناها :

```
00410AF8  .v 74 11        JE SHORT AudioCon.00410B0B
00410AFA  .  55           PUSH EBP
00410AFB  .  55           PUSH EBP
00410AFC  .  68 B01E5000  PUSH AudioCon.00501EB0          ASCII "PN Error!"
00410B01  .  E8 643B0800  CALL <JMP.&MFC42.#1200>
00410B06  .v E9 DF010000  JMP AudioCon.00410CEA
00410B0B  >  8B17         MOV EDX,DWORD PTR DS:[EDI]
00410B0D  .  8B42 F8      MOV EAX,DWORD PTR DS:[EDX-8]
00410B10  .  83F8 10      CMP EAX,10
00410B13  .v 0F85 D101000 JNZ AudioCon.00410CEA
00410B19  .  8D4C24 4C    LEA ECX,DWORD PTR SS:[ESP+4C]
00410B1D  .  E8 3EB20100  CALL AudioCon.0042BD60
00410B22  .  8B3F         MOV EDI,DWORD PTR DS:[EDI]
00410B24  .  C78424 F0000 MOV DWORD PTR SS:[ESP+F0],3
00410B2F  .  33C0         XOR EAX,EAX
00410B31  >  8A0C07       MOV CL,BYTE PTR DS:[EDI+EAX]
00410B34  .  884C04 2C    MOV BYTE PTR SS:[ESP+EAX+2C],CL
00410B38  .  40           INC EAX
00410B39  .  83F8 10      CMP EAX,10
00410B3C  .^ 7C F3        JL SHORT AudioCon.00410B31
00410B3E  .  8D4424 2C    LEA EAX,DWORD PTR SS:[ESP+2C]
```

فلنصعد إلى الأعلى أي إلى بداية هذه الإجرائية ونضع BP :

```
004109FE     90           NOP
004109FF     90           NOP
00410A00  r$ 6A FF        PUSH -1
00410A02  .  68 0D6B4C00  PUSH AudioCon.004C6B0D
00410A07  .  64:A1 000000 MOV EAX,DWORD PTR FS:[0]
00410A0D  .  50           PUSH EAX
00410A0E  .  64:8925 0000 MOV DWORD PTR FS:[0],ESP
00410A15  .  81EC D8000000 SUB ESP,0D8
00410A1B  .  53           PUSH EBX
```

والآن شغل البرنامج وأدخل البيانات التالية : at4re ثم 123456 ثم 1111111111111 :

ستلاحظ أننا عدنا إلى OllyDbg وتوقفنا عند الـ BP. فلنبدأ بالتتبع بـ F8

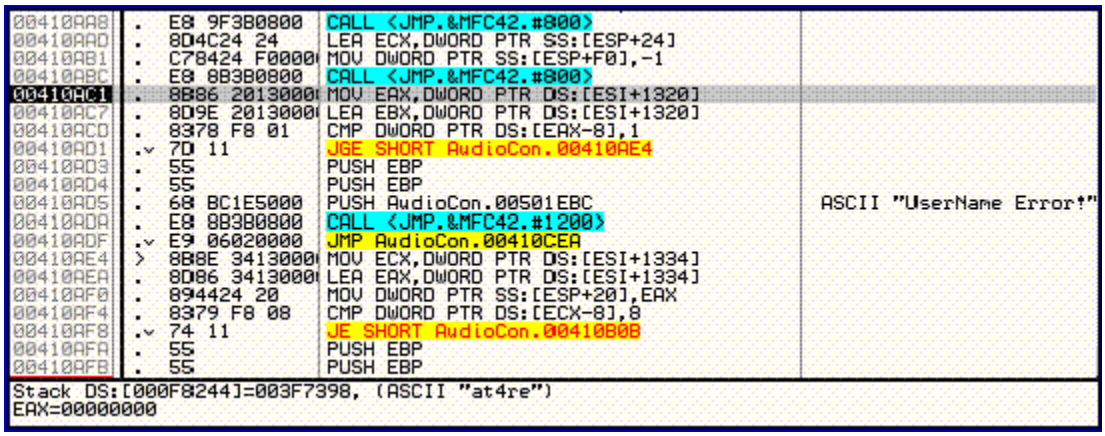
تذكر : أثناء تتبع برنامج ما بعد إدخالك لبيانات التسجيل فأهم شيء تنظر اليه هو جزء التعليقات السفلي، ويليه في الأهمية قسم الـ registers ثم قسم الـ stack.

عند تجاوزك لأول دالة تواجهك ستلاحظ أن ebx➔23456 وهذه بداية جيدة.

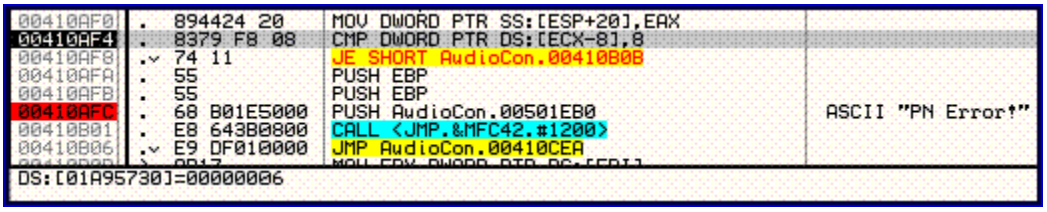
```
00410A1F  .  8BF1          MOV ESI,ECX
00410A21  .  6A 01         PUSH 1
00410A23  .  E8 8C400800   CALL <JMP.&MFC42.#6334>
00410A28  .  8D96 2813000  LEA EAX,DWORD PTR DS:[ESI+1328]
00410A2E  .  8D8E 2413000  LEA ECX,DWORD PTR DS:[ESI+1324]
00410A34  .  50            PUSH EAX
```

```
Registers (FPU)
EAX 00000001
ECX 00BF6764
EDX 01A95739 ASCII "23456"
EBX 00000001
ESP 00BF667C
```

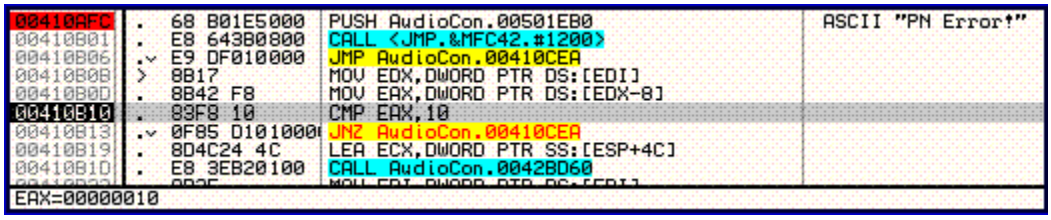
استمر في المتابعة إلى أن نصل إلى هنا :



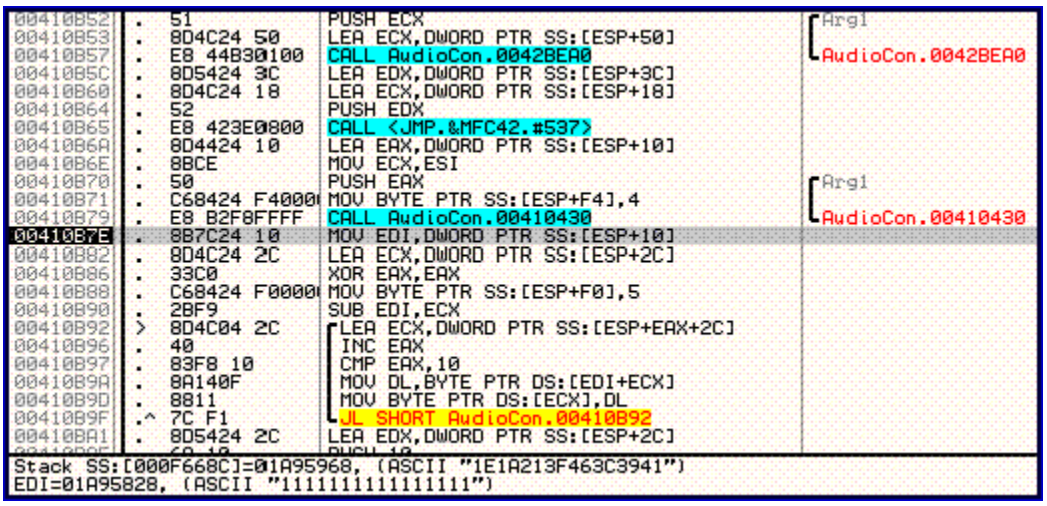
هاهو اليوزرنيم.. لاحظ عند الوصول إلى تلك المقارنة المشار اليها بالسهم، فأن الـ Pn number يتم مقارنته مع 8 :



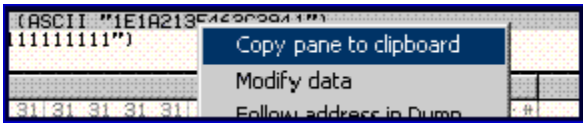
إذا قم بإعادة تشغيل الضحية وكرر نفس الخطوات مع استبدال 123456 بـ 12345678.. وبعد تجاوز ذلك المكان بقليل سنصل إلى هنا :



يتم هنا مقارنة طول السيريال (11111111111) مع العدد 10h..هذه المقارنة غير مهمة فنحن أدخلنا السيريال على حسب ما يسمح المكان المخصص ولا مجال للخطأ. فلنتابع :



يبدو لي وكأننا عثرنا على السيريال الصحيح. دعنا نعيد تشغيل البرنامج وندخله. بالطبع عليك أولا أن تقوم بنسخه كما يلي :



ومن ثم ألصقه في أي محرر نصوص وقم بإزالة أي شيء ليس له علاقة السيريال، ولنقم بإلصاقه :

(عند الضغط على register ستعود إلى OllyDbg وتتوقف عند الـ break point، قم بالضغط على F9 فنحن الآن لا نريد التتبع، بل نريد التأكد من صحة السيريال)

يبدو لي أن أمر البرنامج قد انتهى وأننا قد نجحنا. لكن اضغط موافق وأغلق البرنامج. مفاجأة !! البرنامج غير مسجل !!

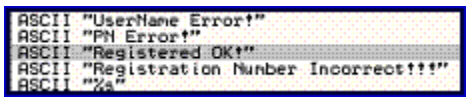
ما الذي حصل؟ بداية لا علاقة للـ crc بالموضوع فتلك الخوارزمية خاصة بكشف عمليات الـ patching ونحن لم نقم بأي منها. إذن؟؟

يبدوا لي أن السيريال صحيح بدليل عدم ظهور رسالة الخطأ وظهور رسالة الصواب، لكن على ما يبدوا أنك حين تشتري البرنامج بشكل شرعي فالشركة تقدم لك السيريال إضافة إلى شيء آخر.قد يكون دنقل dongle رغم أن هذا مستبعد جدا، و قد يكون key file أو ما يعرف باسم license file وهذا احتمال وارد. عموما سنقوم بفحص البرنامج مرة أخرى.

سنقوم الآن بعد إعادة تحميل الضحية في OllyDbg بالبحث عن كل ما له علاقة بالتسجيل وليكن كلمة register :

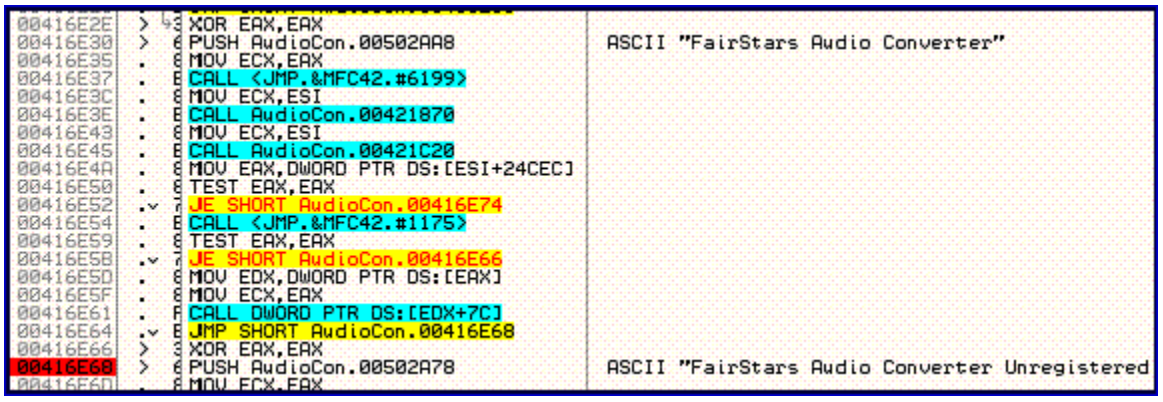
أول نتيجة :

بعد الضغط على انتر ورؤية الكود الخاص بها أعتقد أنها ليست مهمة لذا دعك منها وانتقل للنتيجة التالية (أي اضغط على ctrl + L)

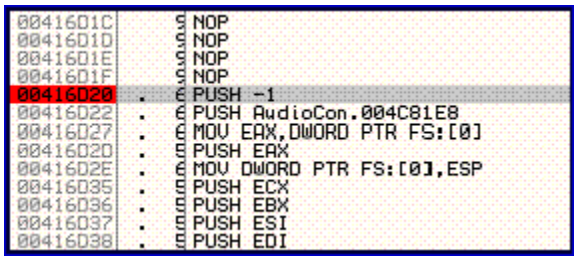


كنا هناك قبل قليل لذا دعك منها وانتقل للنتيجة التالية :

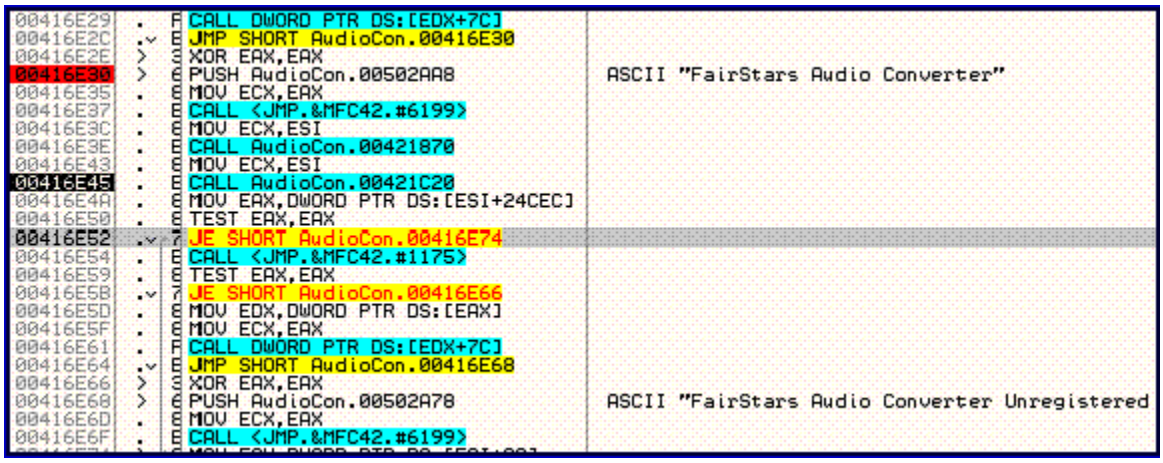
يبدوا لي مكانا جيدا.



فلنقم بوضع bp عند بداية الإجرائية :

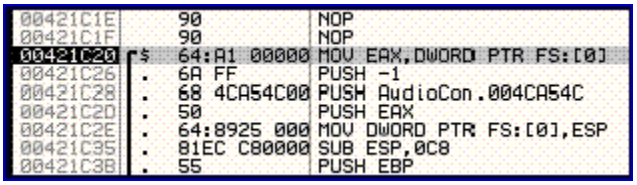


والآن شغل الضحية.تتبع حتى تصل إلى هنا :

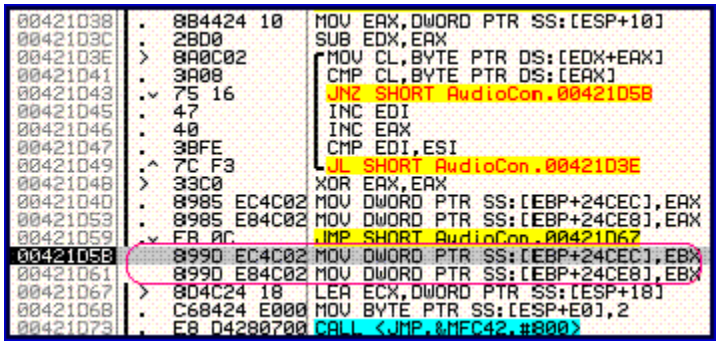


المهم في الموضوع أننا وصلنا إلى العبارة الأولى والتي تدل على التسجيل، لكن كما ترى فهناك قفزة (محددة بالرمادي) تحدد ما إذا كنا سنتوجه إلى عبارة عدم التسجيل أم لا. ومن البديهي أن يجب تجاوز تلك العبارة. أذن الحل يكمن في عكس القفزة. لكن ذلك لم ينجح. واذا نظرت إلى ما قبل القفزة بسطرين ستجد دالة يليها التعليمة التالية : **MOV** EAX,DWORD PTR DS:[ESI+24CEC] وهذا يعني أن نتيجة الدالة (والتي بلا شك هي المسؤولة عن التحقق) تخزن في ذلك المتغير. ألحل إذن هو بالدخول إلى تلك الدالة والبحث عن التعليمة أو مجموعة العليمات التي تنقل قيمة إلى ذلك المتغير.

ها قد دخلنا :



تتبع حتى تصل إلى هذا المكان :



لم نهتم بما قبل هذا الكود لأننا هنا نبحث عن تعليمات محددة ولسنا بصدد التطرق إلى الخوارزمية نفسها.

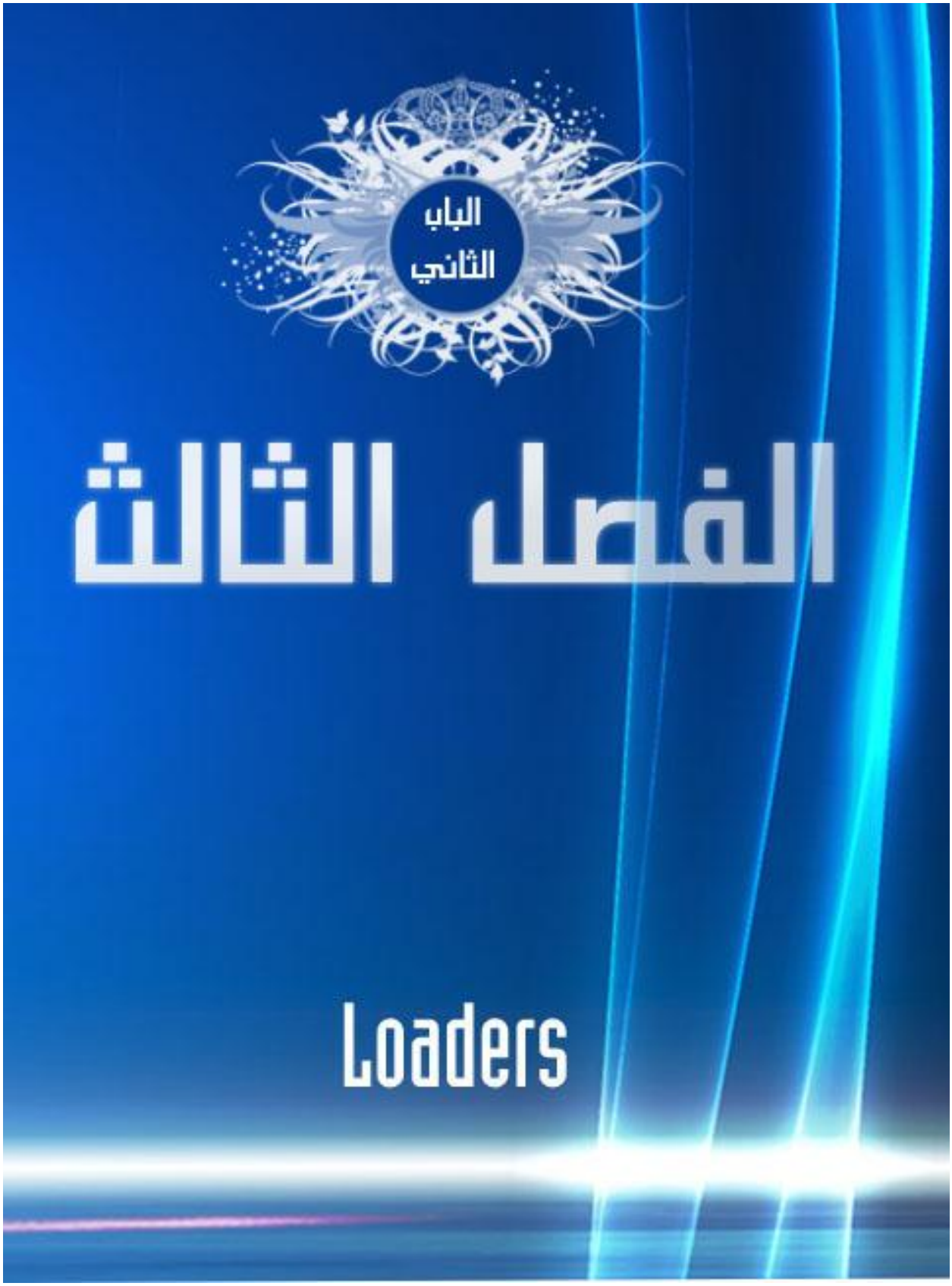
في الإطار الأحمر في الصورة هناك تعليمتان، العلوية تنقل قيمة EBX إلى المكان الذي نبحث عنه. أما السفلية فتفعل الأمر نفسه لكن إلى مكان أبعد قليلا. لذا ما يهمنا هو العلوية. ولو نظرت إلى قيمة EBX لوجدتها 1، إذن نريد أن ننقل القيمة 0 بدلا من 1، قد تقترح أن نجعل التعليمة هكذا : **MOV** DWORD PTR SS:[EBP+24CEC],0 لكن ذلك سيؤدي إلى شطب التعليمة السفلية واستبدالها بـ **NOP**.

لاحظ أنه في هذا المكان فـ edi قيمته صفر. أذن الحل يكمن في استبدال التعليمة تلك بـ **MOV** DWORD PTR SS:[EBP+24CEC],EDI. جرب واضغط F9.رائع لقد نجحنا ☺ بقي أن نتأكد وذلك بحفظ التغييرات ثم تشغيل النسخة المعدلة. والنتيجة. نجاح باهر ☺

بعض البرامج لا تعمل إذا تم تغيير اسمها لذا في حالة كتلك نقوم بتغيير اسم البرنامج المعدل إلى الاسم الأصلي للبرنامج

يمكن عمل باتش كما تعلمت بالدروس الماضية.

وبهذا ينتهي الفصل الثاني من الباب الثاني، بحمد الله وتوفيقه.


الباب
الثاني
الفصل الثالث
Loaders

3. الفصل الثالث : REVERSING VIA LOADERS

في هذا الفصل سوف نتعلم معنى الـ Loader ومتى ولماذا يستخدم.

اللودر هو برنامج يستخدم لتحميل برنامج آخر. يستخدم اللودر عندما نريد عمل patch للبرنامج وهو في الذاكرة، في حالة وجود crc check مثلا. أو حينما نريد تغيير شيء ما بالبرنامج وفي مرحلة أخرى من البرنامج نريد إعادة كل شيء إلى ما كان عليه.

أحد أشهر الأمثلة على اللودر هو الـ trainer وهو برنامج يستخدم للتغيير في الألعاب بحيث تحصل على طاقة غير منتهية وأرواح غير منتهية الخ.

بداية يجب ذكر بعض المفاهيم الأساسية.

1.3 . تعاريف و مفاهيم عامة:

1.1.3 . PROCESSES AND THREADS:

الـ Process تقدم المصادر (resources) اللازمة لتشغيل البرنامج. إنها عبارة عن تجميع لـ virtual addresses space و data و executable code و.

الـ Threadهو كود نريد أن يتم تنفيذه. يقوم المعالج بتنفيذ الـ thread وليس الـ process. إذن كل برنامج يحتوي على process واحدة على الأقل، و كل process تحوي thread واحدة على الأقل تعرف بالـ primary thread. بالطبع يمكن للـ process أن تحوي multiple threads.

جميع الـ threads التابعة لـ process معينة، تشترك في الـ virtual addresses space و الـ global variables، بينما تحتفظ كل thread بمجموعه من المكونات الخاصة بها والتي لا تشترك بها مع أي thread أخرى.

إن استخدام الـ threads يتيح لوندوز خاصية الـ multitasking التي يتمتع بها، فبدلا من متابعة تنفيذ كود معين إلى نهايته، يمكن لوندوز أن يقاطع تنفيذ الكود (الـ thread) عند لحظة ما ويقوم بتنفيذ كود آخر (thread أخرى).

المكونات التي تتحكم بالـ threads في الوندوز هي الـ scheduler و الـ dispatcher، واللذين – معا- يقرران أي الـ threads سيتم تنفيذها ولأي زمن. جدير بالذكر أن عملية مقاطعة interrupting وإعادة تنفيذ resuming للـ thread تتم بحيث لا تشعر الـ thread أنه قد جرى مقاطعتها.

2.1.3 . THE PROCESS INITIALIZATION SEQUENCE:

الخطوات التي يقوم بها الـ OS لبدء تنفيذ process معينة هي كالتالي :

- تقوم دالة CreateProcess بإنشاء process object جديد وتقوم بحجز مساحة من الذاكرة (virtual address space) له.

- ومن ذم تقوم الدالة سابقة الذكر بتوجيه ملف NTDLL.DLL وملف البرنامج التنفيذي (ذو الامتداد exe) تجاه مساحة الذاكرة التي تم حجزها.
- تقوم الدالة بإنشاء أول thread خاص بهذه الـ process كما تقوم بحجز مساحة ذاكرة من نوع stack.
- تبدأ تلك الـ thread بالعمل ضمن دالة LdrpInitialize الموجودة في ملف NTDLL.DLL
- تقوم دالة LdrpInitialize بتوجيه جميع الـ imports الخاصة بالـthread وتوجيهها نحو الذاكرة المحجوزة.
- يتم استدعاء الروتينات التابعة لدالة LdrpInitialize الخاصة بتهيئة ملفات الـ dll التي سيتم استخدامها فيما بعد.

3.1.3 . WINDOWS VIRTUAL MEMORY MANAGEMENT

تمتلك كل process الـ virtual address space الخاص بها والذي يسمح بعنونة 4GB من الذاكرة لكحد أقصى. جميع الـ threads بداخل process معينة يمكنها الوصول إلى الـ virtual address space التابع لتلك الـ process. لكن لا يمكنها الوصول إلى الذاكرة الخاصة ب process أخرى مما يمنع عملية التداخل والتعدي على ذاكرة الغير.

إن كل virtual address space يمكن تصنيفه إلى :

Free : هذا يعني أنه ليس committed ولا reserved أي أنه لا يمن الوصول إليها لكن يمكن جعلها committed أو reserved. إن محاولة القراءة منها أو الكتابة إليها يؤدي إلى حدوث الـ exception المسمى access violation.

Reserved : تعني أنه قد تم حجز مساحة الذاكرة للاستخدام المستقبلي، لكنها غير موجودة فيزيائيا بعد.

Committed : الذاكرة المحجوزة أصبحت موجودة فـــــــــيزيائيا. ولا يتم ذلك (أي لا يتم تحويل reversed إلى committed) إلا بعد أول محاولة للكتابة أو القراءة من تلك الذاكرة.

تقوم الـ process باستخدام دالة VirtualAlloc لحجز (reserve) مساحة من الذاكرة (شرط أن تقع ضمن الـ address space الخاص بها). إن عملية حجز الذاكرة بشكل فيزيائي مباشرة (أي بهيئة committed) يستنزف موارد الذاكرة بطريقة سيئة. لذا يتم حجزها بهيئة reserved أولا. يمكن استخدام نفس الدالة أي VirtualAlloc لتحويل الذاكرة من reserved إلى committed. أما دالة VirtualFree فتستخدم لتحرير مساحة الذاكرة وتحويلها إلى Free.

دالة VirtualAllocEx لها نفس آلية العمل إلا أنها تستخدم مع لحجز مساحات من الذاكرة تقع ضمن address space تابع لـ process أخرى.

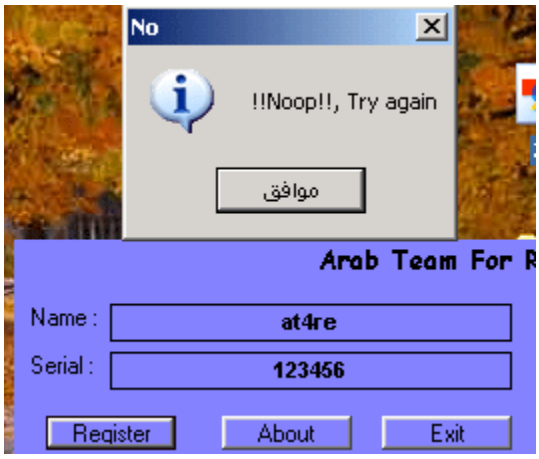
4.1.3 . لماذا نستخدم الـ LOADER؟:

بشكل أساسي يستخدم حين يكون عمل الـ patching غير ممكنا نظرا لوجود دوال حماية تستطيع كشف أي عملية patch تمت للبرنامج.حينها يوفر لنا الـ loader إمكانية عمل الـ patch لكنه يكون temporary patch أي أنه عرة إغلاق

البرنامج لو قمت بفتحه في ollydbg ستجد أن البرنامج لم يتغير. فالتغيير إذا يحدث أثناء تشغيل البرنامج، أي أثناء وجود البرنامج في الذاكرة.

2.3 . المثال الأول

ستجد الضحية في المرفقات باسم Ch2_sec3_1 . قم بتشغيلها :



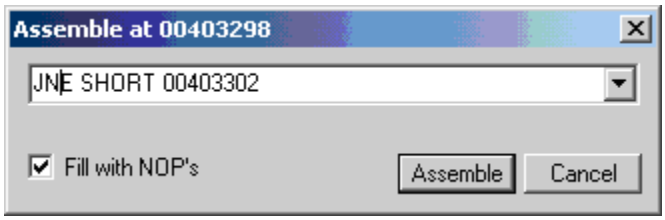
والآن لنفحصه : Microsoft Visual Basic 5.0 / 6.0 . جيد لنشغيله في OllyDbg ثم ابحث عن رسالة الخطأ وستجدها هنا :

```
004032FF   .   PUSH EAX
00403300   .v  JMP SHORT Load_Me1.00403368
00403302   >   MOV ESI,DWORD PTR DS:[<&MSVBVM60.__vbaV  MSVBVM60.__vbaVarDup
00403308   .   MOV EBX,8
0040330D   .   LEA EDX,DWORD PTR SS:[EBP-94]
00403313   .   LEA ECX,DWORD PTR SS:[EBP-54]
00403316   .   MOV DWORD PTR SS:[EBP-8C],Load_Me1.0040  UNICODE "No"
00403320   .   MOV DWORD PTR SS:[EBP-94],EBX
00403326   .   CALL ESI                                 <&MSVBVM60.__vbaVarDup>
00403328   .   LEA EDX,DWORD PTR SS:[EBP-84]
0040332E   .   LEA ECX,DWORD PTR SS:[EBP-44]
00403331   .   MOV DWORD PTR SS:[EBP-7C],Load_Me1.0040  UNICODE "!!Noop!!, Try again"
00403338   .   MOV DWORD PTR SS:[EBP-84],EBX
0040333E   .   CALL ESI
```

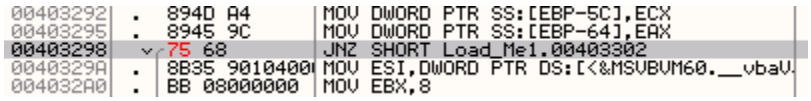
لنصعد إلى الأعلى.ها قد وجدنا رسالة الـ good boy :

```
00403292   .   894D A4      MOV DWORD PTR SS:[EBP-5C],ECX
00403295   .   8945 9C      MOV DWORD PTR SS:[EBP-64],EAX
00403298   .v  74 68        JE SHORT Load_Me1.00403302
0040329A   .   8B35 9010400 MOV ESI,DWORD PTR DS:[<&MSVBVM60.__vbaV  MSVBVM60.__vbaVarDup
004032A0   .   BB 08000000  MOV EBX,8
004032A5   .   8D95 6CFFFFF LEA EDX,DWORD PTR SS:[EBP-94]
004032AB   .   8D4D AC      LEA ECX,DWORD PTR SS:[EBP-54]
004032AE   .   C785 74FFFFF MOV DWORD PTR SS:[EBP-8C],Load_Me1.0040  UNICODE "Yes"
004032B8   .   899D 6CFFFFF MOV DWORD PTR SS:[EBP-94],EBX
004032BE   .   FFD6         CALL ESI                      <&MSVBVM60.__vbaVarDup>
004032C0   .   8D95 7CFFFFF LEA EDX,DWORD PTR SS:[EBP-84]
004032C6   .   8D4D BC      LEA ECX,DWORD PTR SS:[EBP-44]
004032C9   .   C745 84 5C27 MOV DWORD PTR SS:[EBP-7C],Load_Me1.0040  UNICODE "!!Good!!, Write
004032D0   .   899D 7CFFFFF MOV DWORD PTR SS:[EBP-84],EBX
```

إذن نريد تغيير تلك الـ **JE** إلى **JNE** ولكن دون الـ patching فهذا الدرس عن الـ loaders. اضغط زر space وغير ما يلزم :



ثم :



إذن **JE** تعادل 74 بينما **JNE** تعادل 75.

الخطوات التي يجب أن يقوم بها الـ loader هي :

- تحميل الضحية إلى الذاكرة
- إيقاف تنفيذ الضحية
- تعديل ما يلزم من بايتات
- متابعة تنفيذ الضحية
- إنهاء الـ loader (لا يعني ذلك إنهاء الضحية)

الدوال المطلوبة :

CreateProcess	
Parameter	الشرح
lpApplicationName	نضع فيه اسم الضحية أو مسارها (path)
lpCommandLine	ضعه NULL
lpProcessAttributes	ضعه NULL
lpThreadAttributes	ضعه NULL
bInheritHandles	ضعه NULL
dwCreationFlags	هذا البارامتر يحدد كيف ستنشأ الـ process. ضعه CREATE_SUSPEND
lpEnvironment	ضعه NULL
lpCurrentDriectory	ضعه NULL
lpStartupInfo	يشير إلى STARTUPINFO
lpProcessInformation	يشير إلى PROCESS_INFORMATION

WriteProcessMemory	
Parameter	الشرح
hProcess	مقبض (handle) العملية
ipBaseAddress	العنوان الذي نريد أن نكتب عليه
ipBuffer	ما نريد كتابته
nSize	الحجم
ipNumberOfBytesWritten	NULL

باقي الدوال واضحة لا داعي لشرحها. الشكل النهائي للـ loader هو كالتالي :

```
.386
.model flat,stdcall

include windows.inc
include kernel32.inc
include user32.inc
includelib kernel32.lib
includelib user32.lib

.data

stinf STARTUPINFO<>
prinf PROCESS_INFORMATION<>
path    db "DVDRipper.exe",0
base  dd 403298H
patch dd 75h

msg     db "put the loader in the target folder",0
error    db "error",0
```

كما ترى في مقطع البيانات هناك متغيرين ضروريين هما stinf و prinf (يمكنك اختيار أي اسم آخر) يليهما متغير هو path تضع فيه اسم البرنامج، ثم متغير الـ base وهو العنوان الذي سيتم التغيير عنده، وفي النهاية patch وهي القيمة التي نود كتابتها، يلي ذلك بعض الـ strings.

```
.code
start:

invoke CreateProcess,addr path , 0,0,0,0,CREATE_SUSPENDED,0,0,addr stinf , addr prinf
        .if eax==0
                invoke MessageBox,NULL,addr msg,NULL,MB_OK
        .else
                invoke WriteProcessMemory,prinf.hProcess,base1,addr patch1,1,NULL
                        .if eax==0
                                invoke MessageBox,NULL,addr error,addr error,MB_OK
                                invoke TerminateProcess,prinf.hProcess,0
                        .endif
                                invoke ResumeThread,prinf.hThread
        .endif

invoke ExitProcess,0
end start
```

أول دالة هي CreatProcess وبعد استدعائها نريد أن نعرف هل تمت العملية بنجاح أم لا، إن فشلت سيكون eax==0 وتظهر رسالة الخطأ التي تطلب وضع اللودر في مجلد البرنامج. وإن نجحت سننتقل للقسم الثاني حيث نستدعي دالة WriteProcessMemory فإن نجحت سنقوم باستدعاء ResumeThread لمتابعة تنفيذ البرنامج، وإن فشلت سنظهر رسالة خطأ ثم ننهي البرنامج واللودر معا وذلك بدالة TerminateProcess يليها ExitProcess.

طبعا إذا نجحت فإننا سنذهب مباشرة إلى ExitProcess حيث يتم إنهاء اللودر فقط.

وبذلك نكون قد انتهينا من هذا المثال.

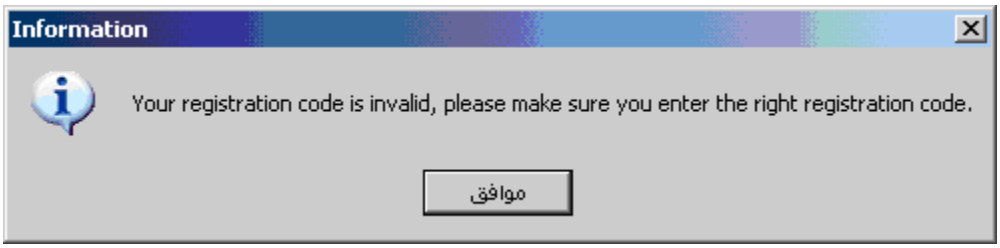
3.3. المثال الثاني

سنقوم هنا بكسر برنامج تجاري وعمل Loader له.

تجد الضحية مرفقة باسم Aimersoft DVD Ripper SE 1.0.16

عند تشغيل البرنامج ستظهر النافذة التالية :

وبعد إدخالك للـ email و الـ reg. code :



حسنا لنفحص البرنامج : Borland Delphi 6.0 - 7.0 رائع. فلننتقل إلى OllyDbg ونبحث عن رسالة الخطأ. مفاجأة : الرسالة غير موجودة.

لا بأس. لنبحث عن كلمة أخرى ولتكن reg (هذه تشمل register و registration و registered ووو). تذكر أن تستخدم ctrl + L للانتقال إلى النتيجة التالية. سترى العديد من النتائج لكن أول نتيجة يمكن أن تكون ذات أهمية هي هذه :

```
ASCII "Data"
ASCII "Error reading registration information!"
```

ضع BP عليها ولنتابع بحثنا في النتائج. ألنتيجة التالية التي قد تكون مهمة هي :

```
UNICODE "System.Name"
UNICODE "BTN_Register_Caption"
UNICODE "BTN_Order_Caption"
UNICODE "BTN_Trial_Caption"
UNICODE "LBL_Note_Caption"
UNICODE "LBL_NoteInfo_Caption"
UNICODE "LBL_Step1_Caption"
UNICODE "LBL_Step2_Caption"
UNICODE "LBL_Register_Note"
UNICODE "LBL_Register_Caption"
UNICODE "Title_Reg"
UNICODE "LBL_Registered_Caption"
UNICODE "Title_UnReg"
```

أعتقد أن هذه هي الـ string التي ستظهر في الـ caption (الشريط العلوي في البرنامج). ضع BP عليها. بعدها :

```
UNICODE "Info_Title"
UNICODE "INF_Email_Error"
UNICODE "Info_Title"
UNICODE "INF_Register_Success"
UNICODE "LBL_Register_Caption"
UNICODE "Info_Title"
UNICODE "INF_Register_Failed"
```

Success و failed و email error كلها عبارات تدل على أن هذا المكان مهم جدا. ضع نقطة توقف على ثلاثتهم.

والآن قم بتشغيل الضحية وستلاحظ أننا توقفنا هنا :

```
0052E01B  .  50          PUSH EAX
0052E01C  .  E8 9B19F5FF CALL <JMP.&WS_Log.ConfigFileDestroy>
0052E021  .  68 1CE45200 PUSH DVDRippe.0052E41C                  UNICODE "BTN_Register_C
0052E026  .  E8 4119F5FF CALL <JMP.&WS_Language.MLGetText>
```

ليس هذا هو المكان الصحيح للبدء. لماذا؟ رأيت كلمة register ولم أجد unregistered.هذا باختصار. إذن دعنا نضغط F9 مرة أخرى. كما ترى البرنامج يعمل. أدخل البيانات التالية

At4re@at4re.com و 123456 ثم اضغط Register.

سنتوقف هنا :

```
0052E936  .v EB 26       JMP SHORT DVDRippe.0052E95E
0052E938  >  6A 40       PUSH 40
0052E93A  .  68 9CE95200 PUSH DVDRippe.0052E99C                  UNICODE "Info_Title"
0052E93F  .  E8 2810F5FF CALL <JMP.&WS_Language.MLGetText>
0052E944  .  50          PUSH EAX
0052E945  .  68 2CEA5200 PUSH DVDRippe.0052EA2C                  UNICODE "INF_Register_Failed"
0052E94A  .  E8 1D10F5FF CALL <JMP.&WS_Language.MLGetText>
```

جيد. سنذهب إلى الأعلى ونضع BP عند بداية هذه الإجرائية ثم نعيد التشغيل وندخل نفس البيانات وسنتوقف هنا :

```
0052E81B      00              DB 00
0052E81C  .   55              PUSH EBP
0052E81D  .   8BEC            MOV EBP,ESP
0052E81F  .   33C9            XOR ECX,ECX
0052E821  .   51              PUSH ECX
0052E822  .   51              PUSH ECX
0052E823  .   51              PUSH ECX
0052E824  .   51              PUSH ECX
0052E825  .   51              PUSH ECX
0052E826  .   51              PUSH ECX
0052E827  .   8955 F8         MOV DWORD PTR SS:[EBP-8],EDX
0052E82A  .   8945 FC         MOV DWORD PTR SS:[EBP-4],EAX
0052E82D  .   33C0            XOR EAX,EAX
0052E82F  .   55              PUSH EBP
0052E830  .   68 8EE95200     PUSH DVDRippe.0052E98E
0052E835  .   64:FF30         PUSH DWORD PTR FS:[EAX]
0052E838  .   64:8920         MOV DWORD PTR FS:[EAX],ESP
0052E83B  .   E8 98D6FEFF     CALL DVDRippe.0051BED8
0052E840  .   84C0            TEST AL,AL
0052E842  .v  0F85 2301000(   JNZ DVDRippe.0052E96B
0052E848  .   E8 67D9FEFF     CALL DVDRippe.0051C1B4
0052E84D  .   84C0            TEST AL,AL
0052E84F  .v  0F85 1601000(   JNZ DVDRippe.0052E96B
```

كما هي العادة، سنتتبع F8 ونركز انتباهنا على منطقة التعليقات السفلية إضافة إلى منطقة الـ registers. ومما لا شك فيه أن كل قفزة مشروطة يجب أن نوليها عناية خاصة لنر إن كانت هي ما يجب تغييره أم لا. لاحظ أن أول قفزة في الصورة السابقة تأخذك إلى مكان يتجاوز عبارتي success و failed. واضح أن ذلك يحدث عندما تكون قد سجلت البرنامج بالفعل. قم بعكس تلك القفزة واحفظ التغييرات وشغل البرنامج من جديد. البرنامج سوف يعمل لكن سيغلق بعد حوالي خمس ثواني من تشغيلة وذلك بسبب خوارزمية CRC32 تقوم بكشف أي محاولة للـ patching.

حسنا لا نريد حفظ التغييرات. فقط حمل الضحية في OllyDbg واذهب إلى تلك القفزة وغيرها ثم شغل الضحية. هل تغير شيء في البرنامج؟ لا يبدوا لي ذلك للأسف.

دعنا نفكر قليلا. بالتأكيد هناك أماكن أخرى في البرنامج تقوم بالتحقق من عملية التسجيل. فالبعض مسؤول عن إظهار الـ nag التي تطلب منك إدخال الإيميل والباسوورد، والبعض مسؤول عن جملة unregistered بالأعلى (في الـ caption).

لكن. ألمنطق يقول أن دالة التحقق هي دالة واحدة فقط فليس من المنطقي أن يقوم المبرمج بكتابة دالة التحقق كلما أراد أن يقوم بالتحقق. بل أنه يوجد دالة واحدة وكلما أراد التحقق يستدعيها.حسنا نحن هنا أردنا التحقق واستدعيناها إذن في الصورة السابقة يجب أن تكون دالة التحقق موجودة. وحيث أن أمر القفز السابق لا يوجد قبله سوى دالة واحدة فقط، أذن أنا شبه متأكد أنها هي دالة التحقق.

ومع ذلك أريد التأكد قبل أن أضيع وقتي بداخلها. ببساطة ضع عليها bp وأعد تحميل الضحية، ثم قم بتشغيلها. يجب أن نتوقف عندها عدة مرات على الأقل. ألم نقل أن هناك أكثر من مكان للتحقق وجميعهم يستدعونها؟ إذن نتوقع أن نتوقف عندها عدة مرات. بالفعل لقد توقفنا عندها 4 مرات مما يؤكد أنها هي دالة التحقق. فلندخل إلى داخلها :

```
0051BED8  r$  55              PUSH EBP
0051BED9  .   8BEC            MOV EBP,ESP
0051BEDB  .   83C4 F8         ADD ESP,-8
0051BEDE  .   C645 FF 00      MOV BYTE PTR SS:[EBP-1],0
0051BEE2  .   33C9            XOR ECX,ECX
0051BEE4  .   B2 01           MOV DL,1
0051BEE6  .   A1 D0945100     MOV EAX,DWORD PTR DS:[5194D0]
0051BEEB  .   E8 E4EDFFFF     CALL DVDRippe.0051ACD4
0051BEF0  .   8945 F8         MOV DWORD PTR SS:[EBP-8],EAX
0051BEF3  .   8D45 F8         LEA EAX,DWORD PTR SS:[EBP-8]
0051BEF6  .   E8 CDFCFFFF     CALL DVDRippe.0051BBC8
0051BEFB  .   8B45 F8         MOV EAX,DWORD PTR SS:[EBP-8]
0051BEFE  .   E8 D9E7FFFF     CALL DVDRippe.0051A6DC
```

ما يهمنا هو آخر تعليمة. لا أنا لا أقصد **RETN** أو تعليمات **POP** فهذه لا أهمية لها، أنا أتحدث عن **MOV** AL,BYTE PTR SS:[EBP-1]. أنت تذكر أن الأمر الذي يلي دالة التحقق مباشرة هو **TEST** EAX,EAX إذن نحن معنيون بمعرفة قيمة EAX وواضح أن صفر تعني أنك لن تقفز أي أنك غير مسجل. ألحل ببساطة يكمن بتغيير ذلك الأمر إلى **MOV** AL,1. إذن سنحصل على :

```
0051BED8  $  55              PUSH EBP
0051BED9  .  8BEC            MOV EBP,ESP
0051BEDB  .  83C4 F8         ADD ESP,-8
0051BEDE  .  C645 FF 00      MOV BYTE PTR SS:[EBP-1],0
0051BEE2  .  33C9            XOR ECX,ECX
0051BEE4  .  B2 01           MOV DL,1
0051BEE6  .  A1 D0945100     MOV EAX,DWORD PTR DS:[5194D0]
0051BEEB  .  E8 E4EDFFFF     CALL DVDRippe.0051ACD4
0051BEF0  .  8945 F8         MOV DWORD PTR SS:[EBP-8],EAX
0051BEF3  .  8D45 F8         LEA EAX,DWORD PTR SS:[EBP-8]
0051BEF6  .  E8 CDFCFFFF     CALL DVDRippe.0051BBC8
0051BEFB  .  8B45 F8         MOV EAX,DWORD PTR SS:[EBP-8]
0051BEFE  .  E8 D9E7FFFF     CALL DVDRippe.0051A6DC
0051BF03  .  8845 FF         MOV BYTE PTR SS:[EBP-1],AL
0051BF06  .  B2 01           MOV DL,1
0051BF08  .  8B45 F8         MOV EAX,DWORD PTR SS:[EBP-8]
0051BF0B  .  8B08            MOV ECX,DWORD PTR DS:[EAX]
0051BF0D     FF51 FC         CALL DWORD PTR DS:[ECX-4]
0051BF10     B0 01           MOV AL,1
0051BF12     90              NOP
0051BF13  .  59              POP ECX
0051BF14  .  59              POP ECX
0051BF15  .  5D              POP EBP
0051BF16  L. C3              RETN
```

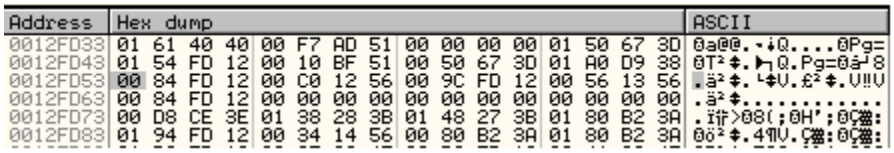
والآن شغل الضحية وجرب.رائع البرنامج يعمل ولم تظهر أي من علامات عدم التسجيل.

لكن بما أننا نريد عمل loader فكلما قلت التغييرات كلما كان اللودر أبسط. في هذه الحالة سنحتاج إلى 3 دوال WriteProcessMemory الأولى تكتب القيمة 0B0h عند العنوان 51BF10 والثانية تكتب 01 عند 51BF11 والأخيرة تكتب 90h عند 51BF12. هناك طريقة أخرى. أعد التشغيل واذهب إلى هذا المكان و ضع BP :

```
0051BED8  $  55              PUSH EBP
0051BED9  .  8BEC            MOV EBP,ESP
0051BEDB  .  83C4 F8         ADD ESP,-8
0051BEDE  .  C645 FF 00      MOV BYTE PTR SS:[EBP-1],0
0051BEE2  .  33C9            XOR ECX,ECX
0051BEE4  .  B2 01           MOV DL,1
0051BEE6  .  A1 D0945100     MOV EAX,DWORD PTR DS:[5194D0]
0051BEEB  .  E8 E4EDFFFF     CALL DVDRippe.0051ACD4
0051BEF0  .  8945 F8         MOV DWORD PTR SS:[EBP-8],EAX
0051BEF3  .  8D45 F8         LEA EAX,DWORD PTR SS:[EBP-8]
0051BEF6  .  E8 CDFCFFFF     CALL DVDRippe.0051BBC8
0051BEFB  .  8B45 F8         MOV EAX,DWORD PTR SS:[EBP-8]
0051BEFE  .  E8 D9E7FFFF     CALL DVDRippe.0051A6DC
0051BF03  .  8845 FF         MOV BYTE PTR SS:[EBP-1],AL
0051BF06  .  B2 01           MOV DL,1
0051BF08  .  8B45 F8         MOV EAX,DWORD PTR SS:[EBP-8]
0051BF0B  .  8B08            MOV ECX,DWORD PTR DS:[EAX]
0051BF0D     FF51 FC         CALL DWORD PTR DS:[ECX-4]
0051BF10     8A45 FF         MOV AL,BYTE PTR SS:[EBP-1]
0051BF13  .  59              POP ECX
0051BF14  .  59              POP ECX
0051BF15  .  5D              POP EBP
0051BF16  .  C3              RETN
```

إذن فالقيمة التي تنقل لـ al هي قيمة موجودة في الذاكرة وتحديدا في stuck. لنتتبعها :

ثم :



Address	Hex dump	ASCII
0012FD33	01 61 40 40 00 F7 AD 51 00 00 00 00 01 50 67 3D	0a@@.÷¡Q....0Pg=
0012FD43	01 54 FD 12 00 10 BF 51 00 50 67 3D 01 A0 D9 38	0T²‡.▶┐Q.Pg=0á┘8
0012FD53	00 84 FD 12 00 C0 12 56 00 9C FD 12 00 56 13 56	.ä²‡.└‡V.£²‡.V‼V
0012FD63	00 84 FD 12 00 00 00 00 00 00 00 00 00 00 00 00	.ä²‡............
0012FD73	00 D8 CE 3E 01 38 28 3B 01 48 27 3B 01 80 B2 3A	.ïî>08(;0H';0Ç▒:
0012FD83	01 94 FD 12 00 34 14 56 00 80 B2 3A 01 80 B2 3A	0ö²‡.4¶V.Ç▒:0Ç▒:

كما تلاحظ فالقيمة التي تعلوا العنوان الذي نحن فيه، تلك القيمة تساوي 1. وطبعا كل صف به 16 قيمة إذن لو استبدلنا التعليمة **MOV** AL,BYTE PTR SS:[EBP-1] بـ **MOV** AL,BYTE PTR SS:[EBP-11] يفترض أن المشكلة ستحل.

الصورتان التاليتان توضحان كود اللودر باستخدام الطريقتين ويمكنك إتباع الطريقة التي تحلو لك.

لكن لاحظ أن الطريقة الأولى مضمونة أكثر. فأنا لست متأكدا أن تلك القيمة ستكون 1 على جميع الأجهزة. لا تنسى أنك تقوم بصنع كراك يوزع لكل أنواع الأجهزة.

الطريقة الأولى :

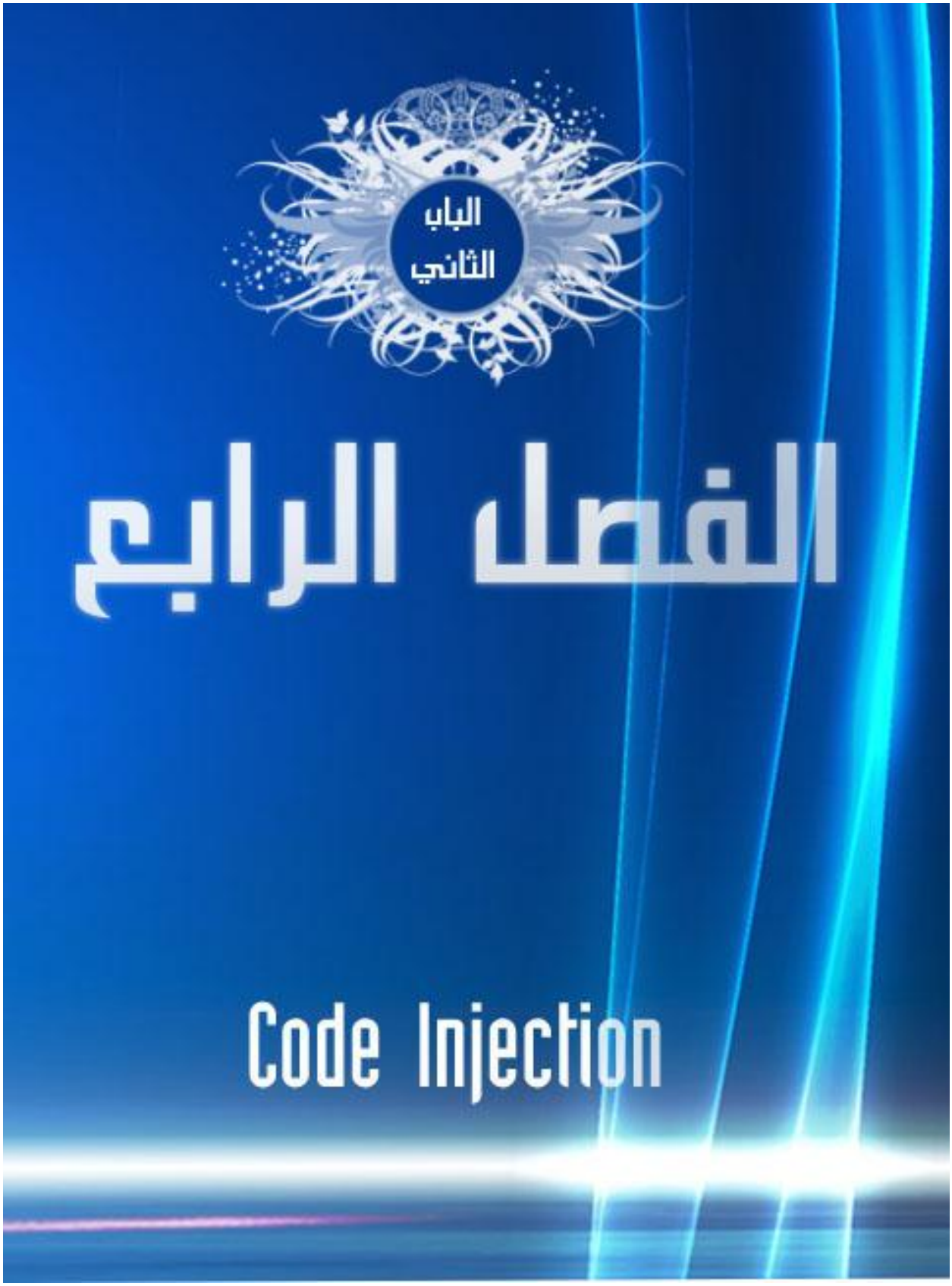
```
invoke CreateProcess,addr path , 0 ,0,0,0,CREATE_SUSPENDED,0,0,addr stinf , addr prinf
    .if eax==0
        invoke MessageBox,NULL,addr msg,NULL,MB_OK
    .else
        invoke WriteProcessMemory,prinf.hProcess,base1,addr patch1,1,NULL
            .if eax==0
                invoke MessageBox,NULL,addr error,addr error,MB_OK
                invoke TerminateProcess,prinf.hProcess,0
            .endif
        invoke WriteProcessMemory,prinf.hProcess,base2,addr patch2,1,NULL
            .if eax==0
                invoke MessageBox,NULL,addr error,addr error,MB_OK
                invoke TerminateProcess,prinf.hProcess,0
            .endif
        invoke WriteProcessMemory,prinf.hProcess,base3,addr patch3,1,NULL
            .if eax==0
                invoke MessageBox,NULL,addr error,addr error,MB_OK
                invoke TerminateProcess,prinf.hProcess,0
            .endif

        invoke ResumeThread,prinf.hThread
    .endif

invoke ExitProcess,0
```

الطريقة الثانية :

```
invoke CreateProcess,addr path , 0 ,0,0,0,CREATE_SUSPENDED,0,0,addr stinf , addr prinf
        .if eax==0
            invoke MessageBox,NULL,addr msg,NULL,MB_OK
        .else
            invoke WriteProcessMemory,prinf.hProcess,base1,addr patch1,1,NULL
                .if eax==0
                    invoke MessageBox,NULL,addr error,addr error,MB_OK
                    invoke TerminateProcess,prinf.hProcess,0
                .endif
            invoke ResumeThread,prinf.hThread
        .endif

invoke ExitProcess,0
end start
```

سورس اللودر : ASM – C - DELPHI

وبهذا ينتهي الفصل الثالث بحمد الله.


الباب
الثاني
الفصل الرابع
Code Injection

4. الفصل الرابع : CODE INJECTION

سنبدأ اليوم في قسم جديد من أقسام الهندسة العكسية يسمى بالـ code-injection أو حقن الكود.هذا يعني إضافة كود إلى البرنامج الضحية لجعله يقوم بأمر ما نريده.هذا الأمر قد يكون إظهار message box معينة. أو self-keygenning أي جعل البرنامج يقوم بإظهار السيريال الصحيح بنفسه. وهناك أمور أخرى ستعرفها بالدروس القادمة.

هذا الدرس والدروس القادمة (و خاصة الباب الثالث) تعتمد اعتمادا كبيرا على فهمك للـ PE files. إن كنت لا تعرف ما هي الـ PE files فهناك شرح وافٍ عنها في قسم الملاحق ويجب عليك قراءته وفهمه قبل متابعة قراءة الكتاب.

1.4 . المثال الأول

تجد الضحية بالمرفقات باسم ch2_sec4_1 

لن أشرح فك الحماية بالتفصيل فهي بسيطة للغاية. تفحص البرنامج لتجد انه مكتوب بالاسمبلي،كما انه يحتاج إلى كلمة السر الصحيحة فقط أي أننا لن ندخل username مما يزيد في سهولته، افتحه بـ olly وضع bp عند GetDlgItemTextA.تفحص البرنامج كما كنا نفعل في السابق.

أثناء تتبعك للبرنامج باستخدام F8 ستصل إلى تعليمة 10 **RETN** عند العنوان 401286 ستأخذك إلى الـ kernel أي ستجد العناوين أصبحت تبدأ بـ 77. لا تقلق أكمل تتبعك إلى أن ترجع إلى البرنامج. أو يمكنك وضع نقط توقف عند 401223 ثم اضغط F9.

لعلك توصلت إلى التالي: يوجد في البرنامج strings هي : s1 و s2. نقوم نحن بإدخال السيريال أي s3 (هذه الأسماء للإيضاح فقط). عملية التشفير هي كالتالي

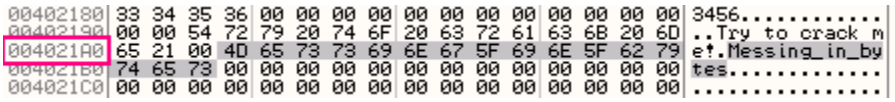
(correct Code) = s1 **XOR** s2

Our Crypted String = s4 **XOR** s2

الـ correct code:

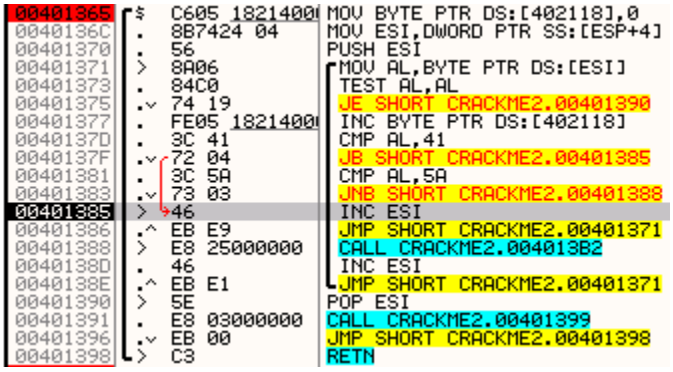
```
00402130 6D 61 74 65 21 0D 4E 6F 77 20 74 72 79 20 74 68 mate!.Now try th
00402140 65 20 6E 65 78 74 20 43 72 61 63 6B 4D 65 21 00 e next CrackMe!.
00402150 1F 2C 37 36 3B 3D 28 19 3D 26 1A 31 2D 3B 37 3E ▼,76;=(↓=&→1-;7>
00402160 4E 6F 20 6C 75 63 6B 21 00 4E 6F 20 6C 75 63 6B No luck!.No luck
```

: S2

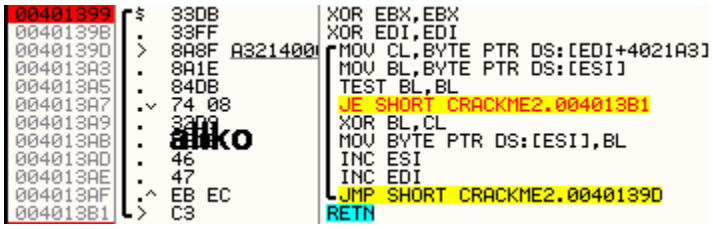


(لا يهم ايهما s1 وايهما s2. فهذه مجرد أسماء من عندي!!)

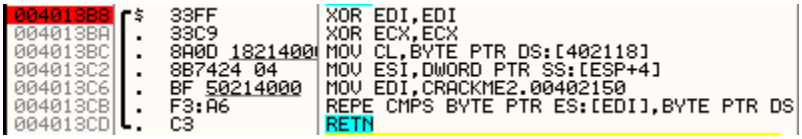
هناك ثلاث اجرائيات procedures مهمة في البرنامج الأولى عند العنوان 401365 وظيفتها فحص الـ char المدخل إذا كان رقما أم لا.



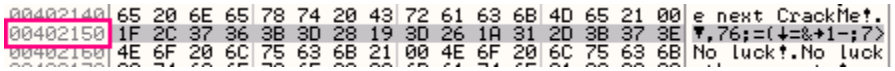
الثانية تليها مباشرة.. عند العنوان 401399 وهذه مسؤولة عن عملية التشفير الفعلية.



الثالثة عند العنوان 4013B8 وهي مسؤولة عن المقارنة بين الكود الصحيح والخطأ.



كما ترى في الصورة الثانية فان الـ char الأول في السيريال يذهب إلى bl وال char الأول من s2 يذهب إلى cl ثم هناك عملية **XOR** إذن الـ correct code هي ناتج عملية **XOR** بين s1 و s2 أي :



نعرف انه إذا كان correct code=s1 **XOR** s2 فان s1=correct code **XOR** s2

أي :

Correct Code	**1F**	**2C**	**37**	**36**	**3B**	**3D**	**28**	**19**	**3D**	**26**	**1A**	**31**	**2D**	**3B**	**37**	**3E**
S1	4D	65	73	73	69	6E	67	5F	69	6E	5F	62	79	74	65	73
S2	52	49	44	45	52	53	4F	46	54	48	45	53	54	4F	52	4D

إذن الكلمة السرية الصحيحة هي s1 وعند تحويلها إلى ASCII تصبح RIDERSOFTHESTORM

حماية سهلة أليس كذالك؟ بلى!!!

ما نريده هنا هو كتابة الـ keygen في نفس البرنامج. أي باستخدام Code Injection لذا هناك عدة خطوات يجب عملها :

- أن تجد مكان لتغيير مجرى (سريان)الأوامر أي تغيير الـ code-flow
- أن تجد مكان فارغا لحقن الكود الخاص بنا فيه (يسمى cave)
- بعد كتابة الـ code يجب إعادة السيطرة (التحكم) إلى البرنامج

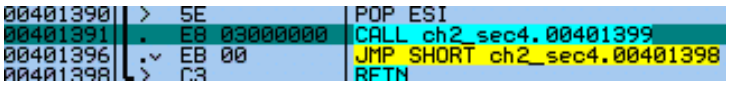
00 : تفحص الصورة الأولى.تجد انه عند العنوان 401391 يتم استدعاء crypting procedure التي عنوانها 401399.

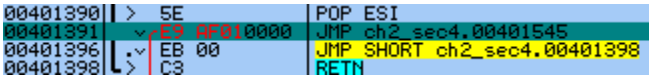
فكر بالأمر كالتالي. سنغير هذا العنوان إلى عنوان الـ cave ومن ثم فان البرنامج عندما يصل إلى هذه الخطوة سيذهب إلى الكود الذي أدخلناه بدل تلك الإجرائية ومن ثم في نهاية الكود سنعيد السيطرة للبرنامج.

01 : في البرامج المكتوبة بالاسمبلي لعلك لاحظت وجود مساحة فارغة كبيرة جدا في نهاية البرنامج.حسنا هذا هو المكان الصحيح. لكن لا تفرح كثيرا ففي البرامج الأخرى يلزمك بعض الوقت لإيجاد الـ cave.

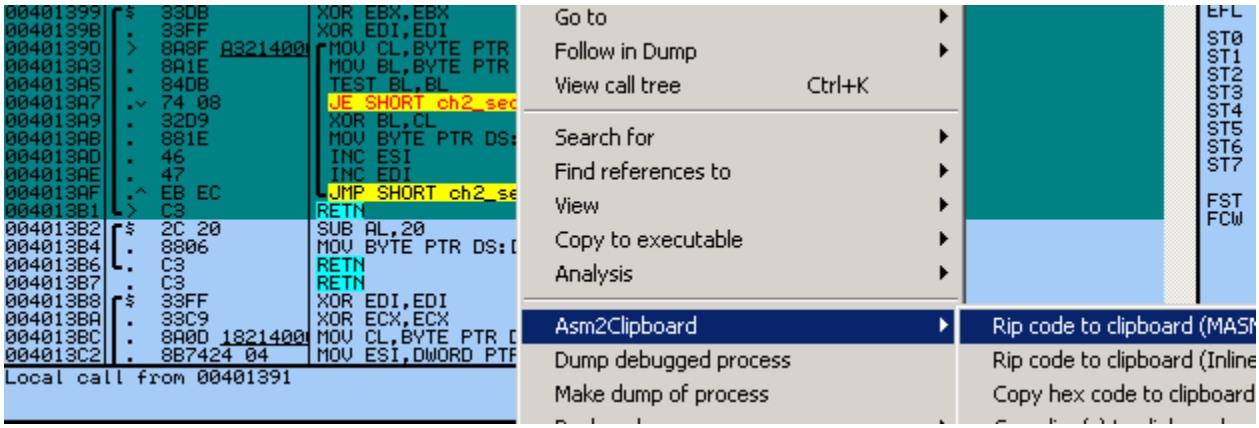
10 : الكود الذي سنكتبه هو نفسه الموجود في الصورة الثانية. لماذا؟ لأنه ينتج الشفرة الصحيحة وهذا ما نريده.

إذن اذهب للعنوان 401391 واضغط space ثم غير الأمر إلى التالي : jmp 401545 (لاحظ أن الفراغ كبير ويمكنك وضع الكود أينما تريد. أنا اخترت العنوان 401545)

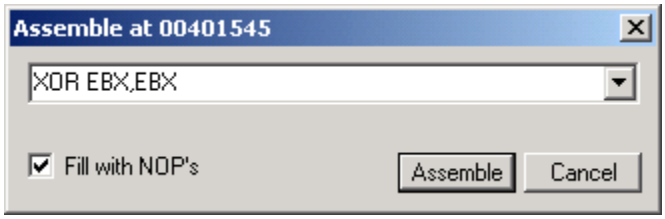




والآن حدد الإجرائية الثانية (التي تبدأ عند 401399) ثم انسخها إلى أي محرر كالـ notepad باستعمال الـ plug in المرفقة.



اذهب إلى 401545 واضغط space وادخل أول أمر كما بالصورة (انسخه والصقه. لهذا السبب قمنا بنسخ الكود كي لا نتعب أنفسنا بكتابة الأوامر من جديد)



واستمر إلى أن تنتهي من جميع الأسطر البرمجية. لاحظ عند إدخال jump أو **CALL** أن تحذف اسم البرنامج. فعند نسخه يكون هكذا **JMP** SHORT CRACKME2.00401371 احذف الاسم ليصبح هكذا **JMP** SHORT 00401371 وغير العنوان أيضا إلى 401545وأكمل جميع الأوامر..

كان من المفروض أن نغير عنوان كل jump إلى العنوان الصحيح فور إدخالها لكني أفضل عمل ذلك بعد الانتهاء من كتابة كل العناوين. وحيث أن أمر القفز يحتوي على short فلن يسمح لك olly بإبقائه كما هو.كلمة short تعني أن يقفز إلى عنوان قريب لذا تجد olly يقترح عليك إبقاء العنوان كما هو مع تغيير الصيغة إلى long. دعك من هذا وافعل كما قلت وبعد أن تصبح خبيرا اتبع الأسلوب الذي تريده.

الآن يجب أن نقوم بتعديل العناوين. أرجع إلى العنوان 401399 واضغط على كل قفزة لتر إلى أين تقفز

والآن ارجع إلى الكود الذي أدخلناه للتو وغير القفزات (أي اضغط space)..ستصل إلى النتيجة التالية :

```
00401545    33DB        XOR EBX,EBX
00401547    33FF        XOR EDI,EDI
00401549    8A8F A321400( MOV CL,BYTE PTR DS:[EDI+4021A3]
0040154F    8A1E        MOV BL,BYTE PTR DS:[ESI]
00401551    84DB        TEST BL,BL
00401553    74 0C       JE SHORT patched.00401561
00401555    32D9        XOR BL,CL
00401557    881E        MOV BYTE PTR DS:[ESI],BL
00401559    46          INC ESI
0040155A    47          INC EDI
0040155B    EB EC       JMP SHORT patched.00401549
0040155D    C3          RETN
```

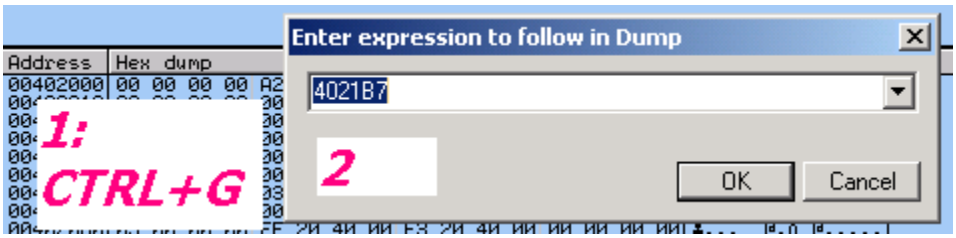
بالنسبة لعملية إرجاع السيطرة للبرنامج. فانظر إلى أمر **RETN** انه يتكفل بالمهمة.

قم بحفظ الكل: right click ➔ copy to executable ➔ all modifications ➔ copy all➔ right click ➔ save file

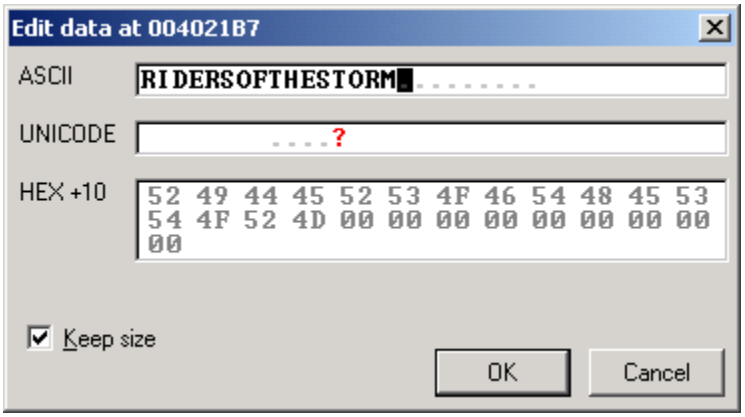
اعد تشغيل البرنامج المعدّل. ألآن اذهب إلى بداية الكود المعدل(401545) وضع نقطة توقف.شغل. وتتبع البرنامج.

ماذا تلاحظ؟ لقد فشلنا أليس كذلك؟ نعم فنحن نسخنا الكود "as is" أي كما هو. ما حصل أننا غيرنا مكانه فقط!! تذكر أن الكود يعمل **XOR** بين s2 و السيريال المدخل. أذن الحل أن نجعله يعمل **XOR** بين Correct code , s2 لذا أولا علينا إدخال الكلمة السرية RIDERSOFTHESTORM

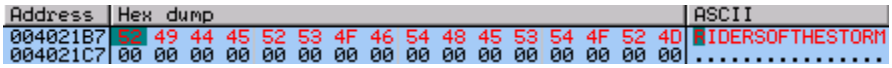
في مكان ما. وليكن مثلا 4021B7 إذن اذهب إلى هناك وحدد جميع الأصفار(النقاط على اليمين) واضغط space



وبعد التحديد والضغط على space :



لتحصل على التالي



غير أمر MOV BL,BYTE PTR DS:[ESI] إلى MOV BL,BYTE PTR [4021B7+EDI]

حيث 4021B7 هي عنوان ما كتبناه للتو.. وEDI يعمل كعداد يغير مكان المؤشر (بداية edi=0 والمؤشر يشير إلى أول حرف. ومع تزايده يصبح المؤشر يشير إلى الأحرف التالية). أحفظ التغييرات واعد التشغيل.رائع!!نجحنا!!

نم بحمد الله.


الباب
الثاني
الفصل الخامس
CryptoGraphy

5. الفصل الخامس : CRYPTOGRAPHY

إن شفرة RSA تعدى واحدة من أكثر خوارزميات التشفير استعمالا في مجال حسابات الكمبيوتر وبرامجه. هي عبارة عن Public_key_cipher صممت في معهد MIT الشهير عام 1978. في الواقع إن الاسم مقتبس من الأحرف الأولى لمصممي الخوارزمية الثلاثة : Ron Rivest , Adi Shamir & Les Andleman. منذ 2000/9/20 أصبحت هذه الخوارزمية متاحة للجميع مجانا ولم تعد حكرا على حكومة الولايات المتحدة.

1.5 . RSA

- Choose two distinct large random prime numbers p & q
- N = p * q
- Φ(N) = (p-1) * (q-1)
- Choose an integer E such that 1< E < Φ(N) and E, Φ(N) share no factors other than 1
- DE=1+KΦ(N) for any integer k.

- يسمى p بـ 1st large prime number و q بـ 2nd large prime number ويفضل ألا يختلف p,q في الحجم كثيرا.
- يسمى N بالـ Public modulus. كلما كان N أكبر، قل احتمال اختراق الشفرة.
- يسمى E بالـ Public(or Encryption) Exponent و بالطبع فإن E هو عدد أولي.
- يسمى D بالـ Private(or Decryption) Exponent
- كلمة mod اختصار لـ modulo وهذه العملية تعني إيجاد باقي القسمة، فمثلا 11 mod 2 = 1 وكذلك 23 mod 3 = 2
- الـ Public Key يتكون من N & E، أما الـ Private Key فيتكون من N & D
- من قيم E الشهيرة : 1+ [2^16] =E 65537 = و أيضا E=3, 5, 35, 10001…etc لكن يفضل اختيار قيمة كبيرة.

1.1.5 . مثال :

- p=61 , q=53
- N=61*53 = 3233
- Φ(N) = 60 * 52 =3120
- Choose E such that 1<E< Φ(N) …let's choose E=17
- DE=17(D)=1+3120(K).

if k=1 then D=183.5
if k=2 then D=367.11
…
if k=14 then D=2569.47
if k=15 then D=2753

والآن دعنا نرى المعادلة المستخدمة لتشفير رسالة ما :

C=M^E mod N
M=C^D mod N

C هي الرسالة المشفرة، و M هي الرسالة الأصلية (قبل التشفير). يجب أن يكون M<N، وإلا وجب عليك تقسيم الرسالة M إلى عدة أجزاء (blocks) بحيث يكون M<N. (أي إجراء عملية padding)

الرمز ^ يعني "أس". عملية الرفع إلى أس ومن ثم عمل mod يطلق عليهما اسم Powmod

2.1.5 . تشفير الرسائل ENCRYPTING MESSAGES :

قام أحمد بإرسال الـ public key الخاص به (أي E & N) إلى محمد، واحتفظ بالـ private key (أي N & D) معه ولم يره لأحد. يريد محمد إرسال رسالة ما (لنسمها M) إلى صديقه أحمد بحيث لا يستطيع أحد قراءة الرسالة إلا أحمد. لذلك، قام بتقسيم الرسالة إلى blocks بحيث لا يزيد حجم كل block عن N. (أي M<N).

ثم قام بتشفير كل block في الرسالة كالتالي :

C= M^E mod N

كل ما على محمد فعله الآن هو إرسال الرسالة المشفرة C (طبعا إن كان هناك أكثر من block واحد في الرسالة الأصلية، مثلا كان هناك 3، عندها سيكون هناك 3 blocks في الرسالة المشفرة، و سيقوم محمد بجمعهم بنفس الترتيب الأصلي معا ليحصل على رسالة واحدة كاملة)

بعد أن استلم أحمد الرسالة المشفرة، يتوجب عليه فك التشفير كي يتمكن من قراءتها. ببساطة سيقوم بالتالي :

M= C^D mod N

وهكذا سيحصل على الرسالة الأصلية.

دعنا نلقي نظرة على أداة RSA Tool 2 والتي قام ببرمجتها !tE. بداية عليك تغيير الـ base إلى 10 (تجده في أعلى البرنامج لليمين)

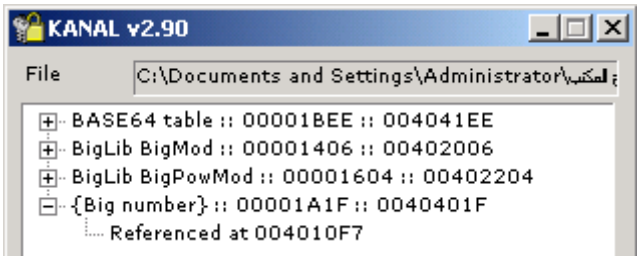
في خانة N أدخل أي عدد ثم اضغط على Factor N حتى تحصل على رقم له 2 prime factors فقط. مثلا العدد 106 له معاملان أوليان اثنان فقط هما : 2 و 53. بالمثل العدد 2005 له معاملان أوليان اثنان فقط هما 5 و 401.

يمكنك الضغط على زر Generate كي يقوم البرنامج بتوليد "حزم" من القيم وبحجم أنت تحدده. كما ترى بالأعلى هناك خانة للحجم. كلما زاد الحجم زادت قوة الخوارزمية.

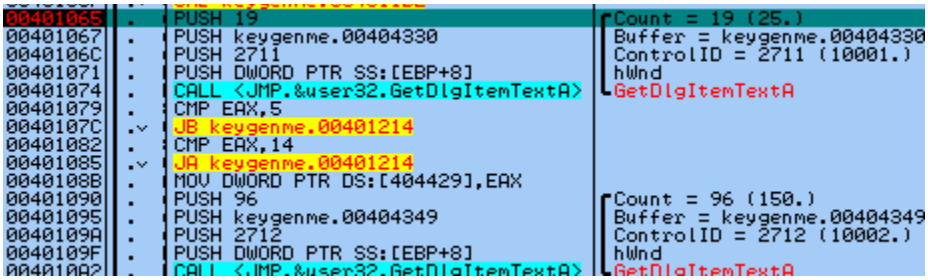
قبل الانتقال للأمثلة يجب عليك قراءة التعليمات الواردة في قائمة help ببرنامج RSA Tool 2 فلسنا هنا بصدد "نسخها ولصقها"

3.1.5 . المثال الأول :

تجد الضحية في المرفقات في المجلد المسمى ch2_sec5_1 
قم بداية بفحصه، النتيجة : music file MPEG layer II/III. دعك من هذا يبدو أن PEiD قد أصيب بالجنون ☺ (قد يظهر لديك nothing found، هذا يعني أنك لا تستعمل الـ hardcore scan. يمكنك عمل ذلك من قائمة options في PEiD). دعنا نفحص وجود الـ ciphers (شفرات) :

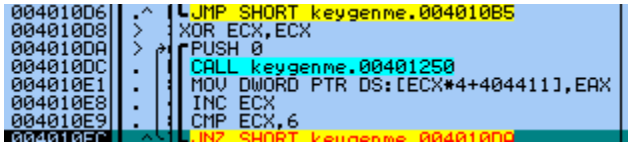


كما ترى هناك BigPowMod مما يجعل احتمال وجود شفرة RSA واردا. وإذا نظرت أسفل منها إلى Big number فهذا أيضا يزيد من احتمال وجود RSA. فلنقم بتحميل الضحية إلى OllyDbg. ضع BP عند بداية أول GetDlgItemTextA تجدها ثم شغل الضحية وأدخل البيانات التالية : 123456 , at4re.



يجب أن يكون اليوزرنيم أكبر من 5 وأقل من 14h. تابع حتى تصل إلى :

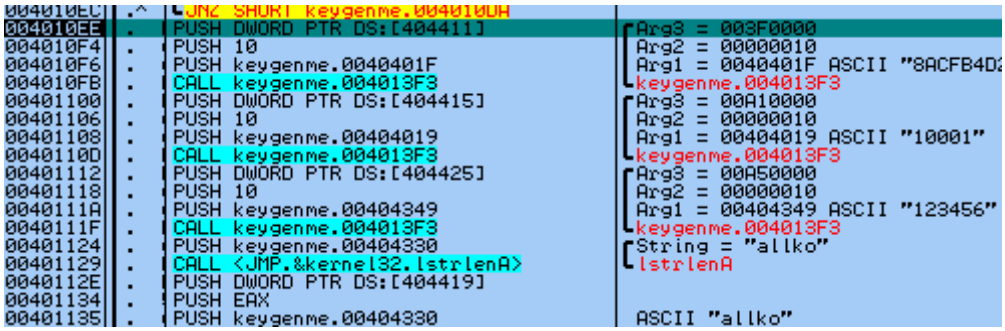
هذه الـ loop وظيفتها ببساطة التأكد من أن السيريال مكون من أرقام وحروف فقط (أي لا يوجد رموز). تجاوزها حتى تصل إلى :



هذه الـ loop وظيفتها إنشاء 6 big numbers (اختصارا bignums) وعمل initialization لهم بالقيمة صفر. بالطبع ستسأل كيف عرفت وظيفتها؟ إن كنت تتعامل مع RSA للمرة الأولى فأنا أعذرك، لكن حالما تصبح لديك القليل من الخبرة في التعامل معها ستعرف أن هذا الشكل (أقصد الـ loop) هو لإنشاء الـ bignums. ومما يدل على ذلك أنه بعد تجاوزك لتلك الدالة ستلاحظ أن قيم EAX الـ 6 هي على الترتيب :

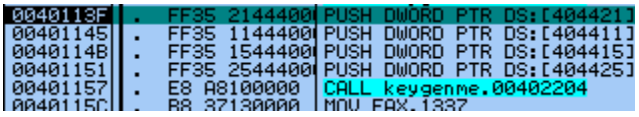
3F0000 , A10000 , A20000, A30000, A40000 & A50000

ستلاحظ أن القيمة الأولى والثانية والأخيرة يتم استخدامهم فيما بعد. دعنا نتابع :



والآن ما هذه الدوال؟ يبدوا أن الأولى تتعامل مع أحد الأرقام الكبيرة وأظنه N (modulus)، والثانية تتعامل مع رقم آخر وأظنه E والرقم الأخير هو السيريال المدخل. بعد ذلك يتم حساب طول اليوزرنيم ومن ثم هناك الدالة الأخيرة والتي على الأرجح تحول اليوزرنيم إلى صيغة bignum (لو دخلتها ستلاحظ بعد دالة VirtualAlloc أن EAX=A60000.

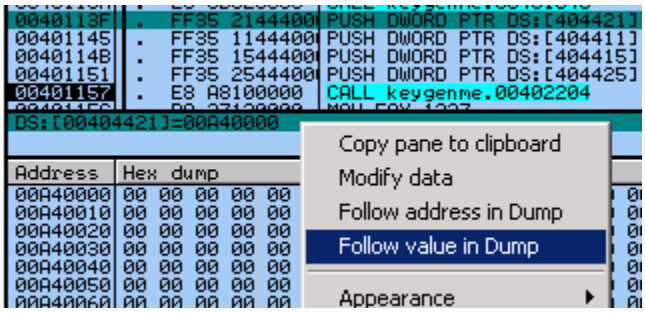
والآن سنصل إلى هنا :



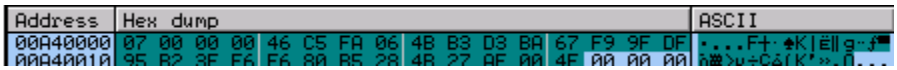
دالة بأربع بارامترات. في شفرة RSA غالبا ما يتم استخدام هذه الدالة والتي سبق أن أشرنا إليها في بداية الدرس وهي دالة powmod، وكي نتأكد أن الدالة التي نراها في الضحية هي فعلا powmod لاحظ أن البارامتر الأول هو A40000 (output 1) والثاني : 3F0000(N) والثالث A10000 (E) والرابع A50000 (our serial) أي أنها كالتالي

PUSH output1 PUSH N PUSH E PUSH entered_serial CALL powmod	powmod (entered_serial, E, N, output1)

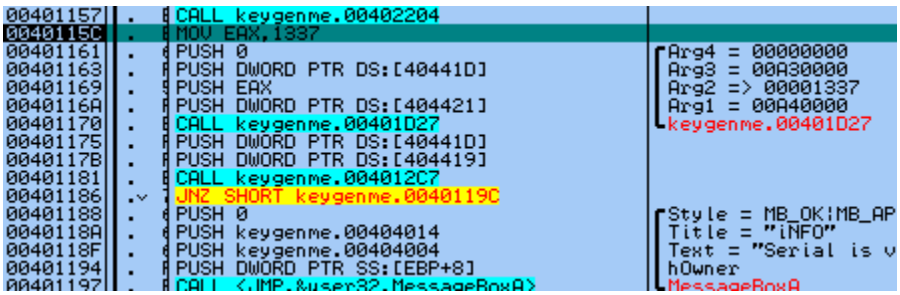
يعني أننا نفذنا C=M^E mod N حيث M➔ entered_serial. الآن اذهب إلى أول بارامتر :



وبعد الانتهاء من الدالة سترى التالي :



فلنتابع.



كما ترى هناك دالة أخرى بأربع بارامترات أيضا.تقوم بقسمة ناتج الدالة السابقة (والذي سميناه output1) على القيمة 1337h والناتج (فلنسمه output2) يخزن في A30000 (كيف عرفت؟ بالخبرة. أنتظر حتى نهاية الدرس وسنقوم بكتابة كيجين. عند رؤيتك للدوال التي سنستخدمها في كتابة الكيجين ستعرف أنها هي نفسها التي استخدمها المبرمج في كتابة التحدي). أعتقد أن آخر دالة (قبل المسج) واضحة كالشمس فهي بكل تأكيد للمقارنة بين الجواب الصحيح والجواب الخطأ.

قم بفتح برنامج RSA tool وأدخل قيمة N التي وجدناها وقيمة E كذلك واضغط على Factor N (أبق الأساس كما هو أي 16)

سيستغرق البرنامج بضع دقائق حسب سرعة جهازك. بعد الانتهاء اضغط على زر calc D. في النهاية سنحصل على النتيجة التالية :

p= 970E1A438A10E069571BDCCBB
q= EB3FFE9F5C761995147C7A28B
N= 8ACFB4D27CBC8C2024A30C9417BBCA41AF3FC3BD9BDFF97F89
E= 10001
D= 32593252229255151794D86C1A09C7AFCC2CCE42D440F55A2D

لاحظ أنك عند الضغط على زر Exact size فالنتيجة هي 200، أي أن حجم N يساوي 200 لذلك فإن هذه الخوارزمية تسمى .RSA-200 إذن لتوليد السيريال الصحيح علينا فعل التالي:

- إدخال اليوزرنيم وتحويله إلى صيغة Big
- ضرب اليوزر بالقيمة 1337h
- إضافة قيمة إلى الناتج بحيث أن هذه القيمة أكبر من 0 وأصغر من 1337h
- M = C^D mod N

أعتقد أن الخطوة الثالثة بحاجة إلى توضيح. تذكر قبل كل شيء أن دالة القسمة التي رأيناها لا تأخذ الباقي بعين الاعتبار.

والآن فلنرى الأمثلة التالية (جميعها بالنظام العشري لكن الفكرة تنطبق على أي نظام):

42/7=6
43/7=6.14
44/7=6.28
45/7=6.42
46/7=6.57
47/7=6.71
48/7=6.85

315/105=3
316/105=3.009
317/105=3.01
...
417/105=3.97
418/105=3.98
419/105=3.99

طبعا اختياري لتلك الأرقام بالذات ليس له معنى فهذه مجرد أمثلة يمكنك اختيار أي رقم تشاء. ما أريدك أن تنتبه إليه هو التالي :

في المثال الأول كنا نقسم أعدادا ما على 7 وكما تلاحظ هناك 7 أعداد ناتج قسمتها على 7 يساوي 6 مع تجاهل الباقي، وهذه الأعداد هي 42و 43و 44و 45و 46و 47و48. أي أن عدد هذه الأعداد يساوي المقسوم عليه.

في المثال الثاني حصلنا على نفس النتيجة، فعدد الأعداد التي ناتج قسمتها على 105 يساوي 3 مع تجاهل الباقي هو 105 أعداد تبدأ من 417 وتنتهي بـ 419

إذن في مثالنا ما يهمنا هو ناتج القسمة بغض النظر عن الباقي إذن لدينا عدد من الخيارات يساوي المقسوم عليه، والمقسوم عليه هو 1337h أي ما يعادل 4919 بالعشري. لذلك يمكنك إضافة أي رقم من 0 حتى 4919 وسيبقى الناتج النهائي صحيحا.

الخطوة الأخيرة سبق شرحها.

والآن دعنا نرى هذا الكيجين الذي يقوم بتوليد السيريال الصحيح. بالطبع لن نقوم باستعراض السورس كود بأكمله، فقط سنستعرض خوارزمية التوليد وباقي الأمور روتينية أصبحت تعرفها بنفسك..

Pushad	نقوم بحفظ قيم المسجلات جميعا.
invoke GetDlgItemTextA, Hwind, 4, addr user, 35	يتم جلب اليوزرنيم وتخزينه في المتغير user
TEST ax,ax	هل تم إدخال يوزرنيم أم لا؟
jnz begin	تم إدخال ← إذن لنتابع العمل
invoke SetDlgItemTextA, Hwind, 5, addr error	إذا لم يتم إدخال يوزر فستعرض رسالة الخطأ
popad	بعد عرض رسالة الخطأ نقوم باسترجاع قيم المسجلات
ret	ومن ثم نغادر بسلام.
begin:	هذا الـ label يشير إلى بداية الخوارزمية الفعلية
PUSH eax	فلنخزن قيمة eax لاسترجاعها لاحقا
XOR ecx, ecx	يتم تصفير ecx تمهيدا لاستخدامه
big_numbers:	هذا الـ label يشير إلى بداية loop
invoke _BigCreate, 0	يتم استدعاء دالة إنشاء big_num بقيمة ابتدائية 0.
MOV dword ptr [big_n+ecx*4], eax	انظر ملاحظة 1 بالأسفل.
INC ecx	يتم زيادة ecx (يستخدم كعداد لهذه الـ loop)
CMP ecx, 5	نريد إنشاء Big_num 5 - انظر ملاحظة 2
jne big_numbers	هل أنشئنا جميع الـ big_num؟؟
POP eax	يتم استرجاع قيمة eax السابقة
invoke _BigInB256, addr user, eax, big_m	يتم تحويل الـ user إلى صيغة big وتخزن النتيجة في big_m. انظر ملاحظة 3
invoke _BigMul32 , big_m ,1337h , big_mf	يضرب big_m بـ 1337 والنتيجة تخزن في big_mf
invoke _BigIn, addr N, 16, big_n	يتم تحويل N إلى صيغة big بالأساس 16 وتخزن النتيجة في big_n
invoke _BigIn, addr D, 16, big_d	يتم تحويل D إلى صيغة big بالأساس 16 وتخزن النتيجة في big_d
invoke _BigPowMod, big_mf, big_d, big_n, big_c	دالة powmod والتي سبق شرحها
invoke _BigOutB16, big_c, addr serial	دالة تحويل النتيجة من big إلى الصيغة العادية.
XOR ecx, ecx	تصفير ecx
destroy:	هنا سنبدأ بloop خاصة بتدمير جميع الـ big_num
Invoke _BigDestroy, dword ptr [big_n+ecx*4]	تم شرح تعليمة مطابقة لها بالأعلى.

INC	ecx	واضح أن ecx يستخدم كعداد.
CMP	ecx, 5	سندمر الـ big_num الخمسة جميعهم.
jne	destroy	هل انتهينا؟
invoke	SetDlgItemTextA, Hwind, 5, addr serial	دالة إظهار السريال.
popad		استعادة الحالة الأولية للمسجلات

ملاحظة 1 : حيث أن لدنيا 5 big_num فكما ترى في كل دورة في هذه الـ Loop يتم تهيئة مساحة من الذاكرة بحجم dword ليتم شغلها بالـ big_num الأول. وبما أن الـ dword = 4 byte، لذا في الدورة التالية يتم تهيئة الـ dword التالية والتي سيكون عنوانها هو نفسه عنوان سابقتها لكن مضافا إليه 4. وفي الدورة التالية نضيف 2*4 وفي الثالثة 3*4 وهكذا.

ملاحظة 2 : الـ big_nums هي :

```
big_n       dd ?
big_d       dd ?
big_c       dd ?
big_m       dd ?
big_mf      dd ?
```

ملاحظة 3 : هذا الأمر خاص بتهيئة اليوزرنيم حيث كما هو واضح يتم استخدام الأساس 256.

ملاحظة 4 : تدمير الـ big_nums بعد الانتهاء ليس ضروريا. لكن حيث أن هذا النوع من الخوارزميات يستخدم في أغراض الحماية والأمن، فمن المنطقي اتخاذ أقصى درجات الحماية والمتمثلة بمحو " آثار" هذه الخوارزمية بعد الانتهاء من استخدامها.

ملاحظة 5 : يوجد في مجلد الكيجين، مجلد فرعي باسم biglib وبداخله ملف نصي باسم fleur-biglib فيه شرح لجميع الدوال المستخدمة.

سورس الكيجين: C – ASM

وبهذا يكون الفصل الخامس قد تم بحمد الله.


الباب
الثاني
الفصل السادس
KeyFile Making

6. الفصل السادس : KEYFILING

6.1 . تعاريف و مفاهيم:

إن مفهوم الـ key_file بسيط للغاية. وهو كالحمايات التقليدية تماما لكن بدل أن تكون الواجهة التي تكتب فيها معلومات التسجيل هي نافذة تخرج لك مع بدء تشغيل البرنامج، فإن هذه الواجهة تم استبدالها بملف تقوم الشركة الصانعة للبرنامج بإعطائك إياه بعد شرائك للبرنامج.

إذن فالبرنامج به نفس حماية من نوع معين، لكن فك هذه الحماية ليس بكتابة سيريال في نافذة من نوافذ البرنامج بل كتابة هذا السيريال في ملف يوضع في مجلد البرنامج.

قبل البدء بالأمثلة سنقوم باستعراض بعض الدوال الهامة والتي ستواجهنا في حمايات Key_file

CreateFile (	هذه الدالة تقوم بإنشاء أو فتح ملف.
LPCTSTR lpFileName,	اسم الملف الذي تريد إنشائه أو فتحه.
DWORD dwDesiredAccess,	إما Read أو Write أو All. في حالة كان Read فلن يستطيع البرنامج أن يكتب شيئا في الملف.
DWORD dwShareMode,	لا تزعج نفسك واجعله Write.
LPSECURITY_ATTRIBUTES lpSecurityAttributes,	0
DWORD dwCreationDisposition,	إما Create_Always حيث يتم إنشاء ملف جديد، وفي حال كان الملف بنفس الاسم موجود فسوف يتم الكتابة فوقه. أو Create_New والتي تشبه سابقتها لكن في حال كان الملف موجود فإنها لا تفعل شيئا. أو Open_always التي تقوم بفتح الملف وفي حال كان غير موجود فإنها تنشئه ثم تفتحه. أو Open_existing والتي تشبه سابقتها لكنها لا تقوم بإنشاء ملف في حال عدم وجوده. أو TRUNCATE_EXISTING والتي تقوم بمحو محتويات الملف ويصبح حجمه 0.
DWORD dwFlagsAndAttributes,	هناك archive و encrypted و normal و hidden و offline و readonly و system وغيرها . سنستخدم normal عند كتابة الـ keymaker
HANDLE hTemplateFile);	0

ReadFile (	هذه الدالة تقوم بقراءة محتويات ملف ما.
HANDLE hFile,	مقبض (handle) الملف
LPVOID lpBuffer,	المكان الذي ستوضع فيه البيانات المقروءة.
DWORD nNumberOfBytestoRead,	عدد البايتات التي ستتم قراءتها.
LPDWORD lpNumberOfBytesRead,	متغير لعدد البايتات التي تمت قراءتها.
LPOVERLAPPED lpOverlapped);	0

WriteFile(	هذه الدالة تكتب نصا (string) ما في ملف.
HANDLE hFile,	مقبض الملف
LPCVOID lpBuffer,	عنوان المتغير الذي يحتوي على النص
DWORD nNumberOfBytestoWrite,	عدد البايتات التي ستتم كتابتها
LPDWORD lpNumberOfBytesWritten,	عدد البايتات التي تمت كتابتها

LPOVERLAPPED lpOverlapped);	0

2.6 . المثال الأول :

تجد الضحية بالمرفقات في المجلد المسمى ch2_sec6_1 

سوسر الكيفايل مايكر: ASM 

هذا المثال بسيط جدا. دعونا نفحصه أولا. يبدوا أن محمي بواسطة uPack (إذا كانت النتيجة لديك nothing found فعليك تغيير إعدادات PEiD بحيث تختار HardCore scan). قم بفك الحماية مستخدما Quick Unpack 1.0 أو أي برنامج آخر يحلو لك.

لنقم بفتحه في Olly. ها قد وجدنا دالة CreateFileA. واضح أن اسم الملف (الـ key_file) الذي يتوجب علينا إنشاؤه هو at4re.key. أسفل منها هناك دالة ReadFile. أي أن CreateFileA ستفتح الملف و ReadFile ستقرأ محتوياته.

```
00401037  .  PUSH KeyFileM.004030C6          pBufferSize = KeyFileM.0040
0040103C  .  PUSH KeyFileM.004034E4          Buffer = KeyFileM.004034E4
00401041  .  CALL <JMP.&kernel32.GetComputerNameA>   GetComputerNameA
00401046  .  PUSH 0                          hTemplateFile = NULL
00401048  .  PUSH 80                         Attributes = NORMAL
0040104D  .  PUSH 3                          Mode = OPEN_EXISTING
0040104F  .  PUSH 0                          pSecurity = NULL
00401051  .  PUSH 1                          ShareMode = FILE_SHARE_READ
00401053  .  PUSH 80000000                   Access = GENERIC_READ
00401058  .  PUSH KeyFileM.004030D4          FileName = "at4re.key"
0040105D  .  CALL <JMP.&kernel32.CreateFileA>   CreateFileA
```

يظهر لنا أيضا وجود دالة GetComputerNameA. ضع BP عند أول قفزة، أي عند 401035 ثم F9 وستلاحظ أن القفزة سيتم تنفيذها وسنتخطى CreateFileA وما يجاورها. لا بأس اضغط F9 مرة أخرى وستلاحظ أنه هذه المرة لن نقفز وسنذهب إلى تلك الدوال. حسنا تابع وتجاوز دالتي GetComputerNameA و CreateFileA ولنرى ماذا يوجد في دالة ReadFile.

```
00401062  .  PUSH 0                          pOverlapped = NULL
00401064  .  PUSH KeyFileM.004042E4          pBytesRead = KeyFileM.00
00401069  .  PUSH 0E                         BytesToRead = E (14.)
0040106B  .  PUSH KeyFileM.004030E4          Buffer = KeyFileM.004030
00401070  .  PUSH EAX                        hFile
00401071  .  CALL <JMP.&kernel32.ReadFile>   ReadFile
00401076  .  CALL KeyFileM.004010F7
0040107B  .  PUSH KeyFileM.004030E4          String2 = ""
```

كما ترى فإن عدد البايتات التي ستتم قراءتها يساوي 14. هناك دالة بعد دالة ReadFile مباشرة (المعلمة). وبعدها يوجد مقارنة. إذن على الأرجح فتلك الدالة هي الخوارزمية التي نبحث عنها.

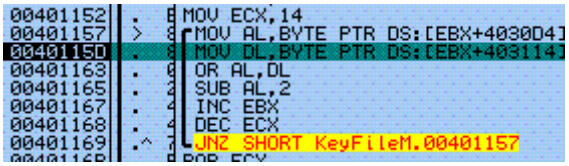
دعنا نرى ما يوجد بداخلها.

في البداية هناك دالة lstrlen دعك منها. ثم هناك loop :

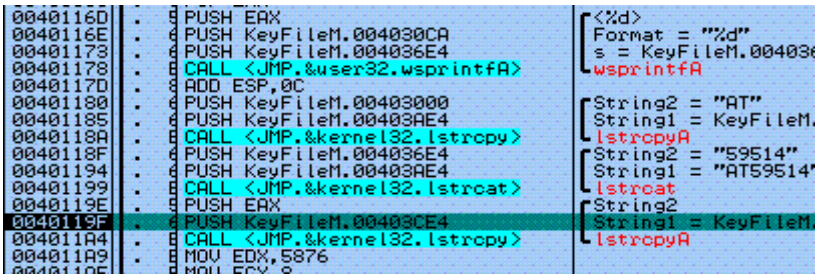
لكن هذه الحلقة ليست لها أية أهمية وهي على الأرجح مجرد trick من المبرمج. لماذا؟ انظر إلى أول تعليمة وتتبع المتغير المستخدم فيها، وذلك بالضغط باليمين ثم follow in dump➔ address constant. ستلاحظ وجود string كبيرة ABCD... حسنا ماذا عن المتغير الخاص بالتعليمة الثانية؟ إنه يشير إلى السترنج : at4re.key. ثم هناك تعليمة **XOR** و mul. لكن من الواضح أن نتيجة هذه الحلقة لا تتأثر بالمستخدم. أي أن اسم جهاز الكمبيوتر لا يدخل في الحسبان وبالتالي فالنتيجة في جهازي وفي جهازك لن تتغير !! إذن ببساطة انتظر حتى تنتهي هذه الـ Loop وسترى أن القيمة النهائية هي eax = 0E87A. فلنتابع. هناك دالة lstrlen ثم حلقة ثانية :

أيضا هذه لا أهمية لها فالـ string الأولى هي جزء من تلك التي رأيناها في الحلقة السابقة، والـ string الخاصة بالتعليمة الثانية كما هي لم تتغير. النتيجة النهائية ecx=5885h

فلنتابع.

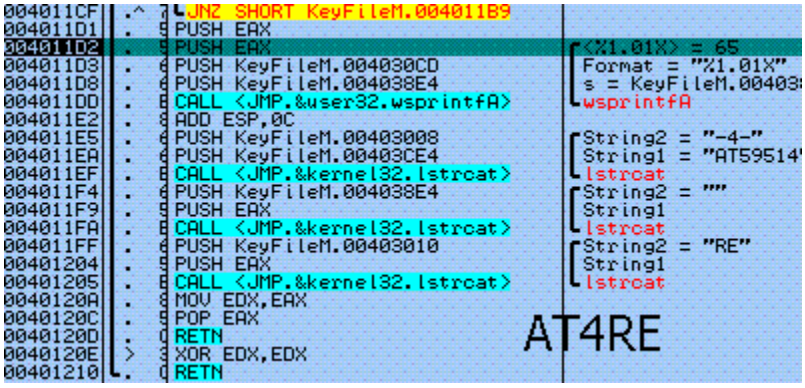


لا أهمية لها. النتيجة النهائية : eax = 0FE. فلنتابع



باختصار، الدالة الأولى تحول القيمة 5885h إلى 59514d. الدالة الثانية تنسخ AT والثالثة تضيف 59514 إلى AT فنحصل على AT59514. بعد ذلك هناك أمر غبي وهو **MOV** edx,5876 بعده بقليل هناك **XOR** edx,edx ولا أعرف ما الحكمة من نقل قيمة إلى edx ثم تصفيره !! لكن انتبه إلى أن ecx=8 وسوف يستخدم كعداد في الحلقة التالية. فلنتابع:

في التعليمة الأولى هناك السترنج ALLKO وهي اسم جهازي (بالطبع ستختلف في حالتك !) ثم في التعليمة الثانية هناك السترنج at4re.key. وأخيرا وجدنا حلقة لها معنى ! النتيجة في هذه الحالة ستختلف من جهاز لآخر. لاحظ أن الحلقة تتكرر 8 مرات فقط أي أن أول 8 محارف من اسم جهازك هي فقط ما سوف نتعامل معها. في حالتي كان الناتج النهائي 65h. فلنتابع :



يتم تحويل 65 إلى نص (ASCII). ثم تضاف السترنج -4- إلى AT59514 فنحصل على AT59514-4- ثم تضاف إليها النتيجة 65h وفي النهاية تضاف إليهم السترنج RE فيكون الحل بالنسبة لي AT59514-4-65RE. انتهينا ☺. للتأكد يمكنك إنشاء ملف نصي(بامتداد txt مثلا) وإضافة الحل في حالتك إلى الملف، ثم من folder option اختر view ثم أزل علامة الصح بجانب Hide extension of known file types. عندها سيظهر الامتداد ويمكنك تغييره من txt إلى key. لا تنس طبعا تغيير اسم الملف إلى at4re.

والآن نريد كتابة Key_maker. كالعادة لن أشرح الكود بأكمله فهو واضح لكن سأشرح الجزء الهام.

بداية سنستعدي دالة GetComputerName	invoke GetComputerName,addr pc_name , addr sizea
يتم تصفير ebx	XOR ebx,ebx
نقل القيمة 8 إلى ecx تمهيدا لاستخدامه كعداد	MOV ecx,8
هذا الـ label هو بداية الحلقة	:begin
ننقل أول محرف من اسم الجهاز إلى al	MOV al,byte ptr ds:[ebx+ pc_name]

MOV dl,byte ptr ds:[ebx+ at4re]	ننقل أول محرف من at4re.key إلى dl
XOR al,dl	عملية XOR بين al و dl.
MOV byte ptr ds:[ebx+ buffer],al	نقل النتيجة إلى biffer
INC ebx	تزويد ebx.
dec ecx	وإنقاص ecx.
jnz begin	هل انتهينا من المحارف الـ 8 أم لا؟
invoke wsprintfA,addr buffer2 , addr format , eax	تحويل النتيجة من value إلى ASCII
invoke lstrcatA,addr str1,addr buffer2	إضافة النتيجة السابقة إلى AT59514-4-
invoke lstrcatA,addr str1,addr str2	إضافة RE إلى ما سبق.
invoke CreateFile…	نستعدي CreateFile لإنشاء ملف جديد.
invoke WriteFile,eax ,addr str1,14d,addr buffer3,NULL	نقوم بكتابة كل ما نريده في الملف. لاحظ أن طول البايتات هو 14 عشري.

1 : كما تلاحظ هناك متغيرات فارغة تم تعريفها كـ space dd NULL. في الواقع حدث تداخل بين الـ strings فاضطررنا الى وضع متغير فارغ بينهم لمنع التداخل.

2 : كما ترى يوجد str2 dd "ER",0 مع أن المفروض str2 dd "RE",0. حسنا جرب الاثنتين وأخبرنا بالنتيجة ☺


الباب
الثاني
الفصل السابع
Keygening

7. الفصل السابع : KEYGENING

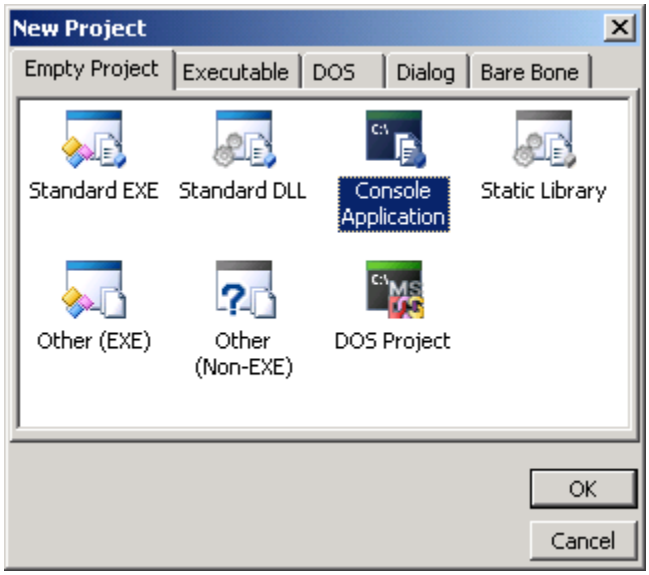
بداية، سنشرح كيفية كتابة keygenerator بلغة الاسمبلي لكن دون واجهات مرئية GUI.. نتمنى أن تستفيدوا.

7.1 . برمجة قالب الكيجين

7.1.1 . CONSOLE

ما نعنيه بهذه الكلمة أن البرنامج عند عمله ستفتح لك نافذة كنافذة الدوس يعمل منها البرنامج. لا يوجد واجهات مرئية.سنركز على البرامج ذات الواجهات المرئيةGUI لكن لا مانع من فهم أساسيات الـ console.

ما نحتاجه بداية هو أن تقوم بتثبيت برنامج masm32. أيضا يلزم تثبيت برنامج WinAsm. أن لم تكن تعرف كيفية عمل ذلك فهناك شرح لهذه الأمور في الملحق. والآن شغل winasm ثم من قائمة File اختر new project لترى التالي :



كما ترى عليك اختيار Console Application. ثم اضغط انتر أو OK. والآن في المكان المخصص لكتابة الكود قم بكتابة الكود الذي تراه في الصورة التالية

كان من الممكن إن نضع الـ source code بالمرفقات وبالتالي تقوموا بعمل copy و paste بكل سهولة لكن نريدكم إن تتعودوا على الكتابة بالاسمبلي

```
.386
.MODEL flat, stdcall
OPTION CASEMAP:NONE

Include windows.inc
Include kernel32.inc
Include masm32.inc
IncludeLib kernel32.lib
IncludeLib masm32.lib

.data
HelloMsg DB "Hello World", 0
CRLF     DB 00Ah, 00Dh, 0
ExitMsg  DB "Enter to Exit", 0

.data?
buffer  DB ?

.code
Start:
invoke StdOut, addr HelloMsg
invoke StdOut, addr CRLF
invoke StdOut, addr ExitMsg
invoke StdIn, addr buffer, 1
invoke ExitProcess, 0
End Start
```

سنقوم بتحليل الكود سطرا سطرا.

386.	هذا السطر يخبر masm32 إن يستخدم مجموعة التعليمات الخاصة بالمعالج 80386. هذا السطر لن نغيره. (أحيانا يتم استخدام .486 بدلا منه)
.model flat, stdcall	لن نتكلم عن ماهية الـ model. ولا عن الـ stdcall. اطمئن فهذا السطر أيضا لن يتغير في جميع برامجنا.
option casemap:none	هذا السطر يعني إن الـlabels (ستشرح لاحقا) ستكون حساسة لحالة الأحرف.
include windows.INC	يخبر masm32 بأن يقوم بتضمين windows.INC في ملفنا (وكأننا كتبنا محتوياته هنا).
include kernel32.INC	نفس الشيء
include masm32.INC	نفس الشيء
includelib kernel32.lib	يخبر الـ linker أن يربط برنامجنا مع ملف kernel32.lib
includelib masm32.lib	نفس الشيء
data	تعني بداية مقطع البيانات
HelloMsg db "Hello World",0	هنا نقوم بتعريف string (سلسلة من الأحرف) اسمها MsgBoxCaption ومحتوياتها هي ما بين علامتي التنصيص. ألصفر بالنهاية يعني أن الـ string ستنتهي بـ 0 (كما هو الحال في الـ c++)
CRLF db 00Ah, 00Dh , 0	هنا نقوم بتعريف متغير باسم CRLF (CARRIAGE RETURN , LINE

	(FEED وهاذان المتغيران معا يقومان بإنزال مؤشر الكتابة سطرا واحدا للأسفل.
ExitMsg db "Enter to Exit",0	سبق شرح سطر مماثل له.
.data?	تعني بداية مقطع البيانات (المتغيرات) التي لا قيمة ابتدائية لها.
Buffer db ?	متغير ليس له قيمة ابتدائية.
.code	تعني بداية مقطع الكود التنفيذي
start:	كلمة start عبارة عن label (لأنه يوجد بعدها نقطتان رأسيتان) وهي تشير إلى بداية الكود. يمكن اختيار أي كلمة أخرى كـ beginning مثلا.
invoke StdOut , addr HelloMsg	هنا نستدعي دالة StdOut المسؤولة عن إظهار string ما، أما addr HelloMsg فهي عنوان السترنج التي نود عرضها. أي أن هذا هو البارامتر الوحيد لهذه الدالة.
invoke StdOut , addr CRLF	نفس الشيء.هنا السترنج التي سنعرضها هي CRLF أي أننا سننزل مؤشر الكتابة للأسفل بمقدار سطر واحد.
invoke StdOut , addr ExitMsg	سبق شرح سطر مماثل.
invoke StdIn , addr buffer,1	هنا نستدعي دالة StdIn الخاصة باستقبال النص من المستخدم. في الواقع نحن لا نريد استقبال أي نص لكن لولا هذه الـ "إضافة" لأغلق البرنامج فورما يتم تشغيله لأن عرض السترنج لا يستغرق إلا ثوان معدودة. أما الـ 1 فهو البارامتر الثاني للدالة وهو يشير إلى أننا سنستقبل محرفا (character) واحدا من المستخدم.
invoke ExitProcess, 0	هنا نستدعي دالة ExitProcess الخاصة بإنهاء البرنامج ونقوم بإعطائها البارامتر الوحيد وهو 0.
end start	هذه تشير إلى نهاية البرنامج. لاحظ أننا كتبنا start لأننا كتبناها في الأعلى. فما يكتب بالأعلى نكتبه هنا لكن بدون النقطتان الرأسيتان.

والآن دعنا نجرب برنامجنا.سترى بالشريط العلوي بعض الأزرار كهذه

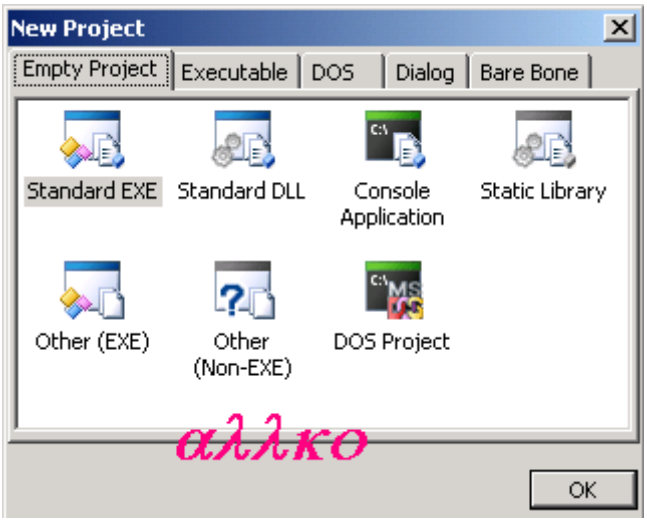
اضغط على زر Go ALL الموضح بالصورة.. أو يمكنك فتح قائمة Make ومن ثم اختر Go All (هذا الخيار يشمل خيارين اثنين : الأول assemble والثاني link). بعد الضغط عليه ستخرج لك نافذة تطلب منك حفظ الملفين (الملف الأول ملف المشروع والثاني ملف السورس كود). الأفضل إن تحفظ الملفين بنفس الاسم وفي نفس مجلد WinAsm.

والآن بعد الحفظ سترى التالي :

7.1.2 . GUI

الآن سنقوم بكتابة kg بواجهة مرئية.

قم بفتح مشروع جديد.



إن خيار Standard EXE هو الـ default أي يكون محددا تلقائيا لذا اضغط ok.

والآن تظهر شاشة فارغة.سنقوم بعمل أول برنامج لنا. برنامج message box. لذا قم بكتابة الكود الذي تراه في الصورة التالية :

```
.386
.model flat, stdcall
option casemap:none

include windows.inc
include kernel32.inc
include user32.inc
includelib kernel32.lib
includelib user32.lib

.data
MsgBoxCaption db "     at4re.com     ",0
MsgBoxText    db "allko",

.code
start:
push MB_OK
push offset MsgBoxCaption
push offset MsgBoxText
push NULL
call MessageBox

push NULL
call ExitProcess

end start
```

AT4RE

سنقوم بتحليل الكود سطرا سطرا. لكن سنشرح فقط الأسطر الجديدة والتي لم يتم شرحها في البرنامج السابق.

PUSH MB_OK	هنا يتم دفع البارامتر الرابع من بارامترات دالة MessageBox. سنتحدث عنه لاحقا.
PUSH offset MsgBoxCaption	هنا يتم دفع عنوان (offset) الـ string التي ستظهر في أعلى المسج. هذا هو البارامتر الثالث
PUSH offset MsgBoxText	هنا يتم دفع عنوان (offset) الـ string التي ستظهر في وسط المسج. هذا هو البارامتر الثاني.
PUSH NULL	هذا أول بارامتر. كلمة NULL تعني صفرا أو لا شيء. يمكن استبدالها بـ 0.
CALL MessageBox	هنا يتم استدعاء الدالة بعد إن دفعنا بارامتراتها الأربعة (يتم تخزين البارامترات في الـ stuck)
PUSH NULL	هنا يتم دفع البارامتر الوحيد لدالة ExitProcess
CALL ExitProcess	هنا نستدعي دالة ExitProcess الخاصة بإنهاء البرنامج

والآن دعنا نجرب برنامجنا.

اضغط على زر Go ALL. بعد الضغط عليه ستخرج لك نافذة تطلب منك حفظ الملفين (الملف الأول ملف المشروع والثاني ملف السورس كود). الأفضل إن تحفظ الملفين بنفس الاسم وفي نفس مجلد WinAsm.

والآن بعد الحفظ سترى التالي :

إذا كان هذا أول برنامج أسمبلي لك. فـ مبارك !!!

إذا أردت التفاصيل عن السطر الثاني فاقرأ كتاب اسمبلي.. أو اقرأ مقالات Xacker في منتديات الفريق العربي للبرمجة فمقالاته مختصرة ومفيدة. إن دالة MessageBox التي استخدمناها تأخذ 4 بارامترات كما لاحظت. أول بارامتر يتعلق بالـ style لهذه الـمسج (النافذة).

الآن سنعمل تعديل على الكود. لاحظ أننا استخدمنا توجيه offset للحصول على عنوان للـ string. هناك طريقة أخرى كالتالي :

```
PUSH MB_OK
LEA eax,MsgBoxCaption
PUSH eax
LEA eax,MsgBoxText
PUSH eax
PUSH NULL
CALL MessageBox
```

إن تعليمة LEA تعني load effective address أي أنها تستخدم للحصول على عنوان لـ string أو غيرها.كما ترى فإننا نخزن العنوان في المسجل eax (يمكنك اختيار أي مسجل آخر) ومن ثم ندفع تلك القيمة إلى المكدس(stuck) بواسطة أمر الـ **PUSH**.

لاحظ أننا استخدمنا طريقة **PUSH** و **CALL** لاستدعاء الدالة سواء في الأسلوب الأول أو الثاني. لكن حيث إن المجمع (assembler) الذي يستخدمه برنامج winasm هو masm32 (بالمناسبة فهو يدعم مجمع fasm أيضا) فيمكننا استخدام صيغة أخرى. في الواقع فان masm32 يقدم صيغا تجعل الاسمبلي في منتهى السهولة. لاحظ التالي :

يمكن الاستغناء عن :

```
PUSH MB_OK
PUSH offset MsgBoxCaption
PUSH offset MsgBoxText
PUSH NULL
CALL  MessageBox
```

بسطر واحد. وهو :

```
invoke MessageBox, NULL, addr MsgBoxText, addr MsgBoxCaption, MB_OK
```

لاحظ مدى سهولة هذه الصيغة.كلمة **invoke** هي عبارة عن توجيه (directive) ولا يمكن استخدامها مع المجمعات الأخرى كـ tasm مثلا. فهي مخصصة لـ masm. لاحظ الترتيب الذي تكتب فيه البارامترات. أيضا لاحظ كلمة addr التي تشير إلى دفع عنوان الـ string. انتبه إلى انه لا يمكنك استخدام كلمة offset في صيغة **invoke**.

نفس الشيء ينطبق على الدالة الأخرى فيمكن كتابتها كالتالي :

```
invoke ExitProcess,0
```

الآن قم بالتجريب. أحذف الصيغ السابقة واستبدلها بهذه.كما ترى حصلنا على نفس النتيجة.

أما بخصوص MB_OK فهي كما قلت عبارة عن نمط الـ messagebox. جرب إن تستبدلها بـ MB_OKCANCEL

ستجد أنه بدل وجود زر ok فقط أصبح هناك ok و cancel. الآن جرب

MB_YESNO or MB_ICONASTERISK or MB_DEFBUTTON2. كلمة or يمكنك استبدالها بإشارة +. أي أننا سننفذ الأنماط الثلاثة معا. لاحظ انه اثناء كتابتك لكل نمط سيخرج لك مربع من قبل winasm بحيث يمكن اختيار النمط الذي تريد من قائمة بجميع الأنماط المتوفرة. الآن نفذ السابق لتحصل على :

لاحظ. ألنمط الأول يعني وجود زرين yes,no والنمط الثاني يعني وجود أيقونة خطأ (التي تراها باللون الأحمر) والنمط الثالث يعني أن الزر الثاني هو الزر المختار افتراضيا.

قلت أن مجمع masm له العديد من المزايا. من بينها هو إمكانية استخدام بعض الـ directives التي تستخدم مع لغات المستوى العلوي.هذه المجموعة تشمل.if و.elseif و.while وغيرها. أكثر ما يهمنا هو الأولى والثانية.

لاحظ معي:

The C version :	The MASM version:

if (var1 == var2)
{
//code goes here
}
else if (var1 == var3)
{
//code goes here
}
else
{
//code goes here
}
	.if (var1 == var2)
	
	;code goes here
	
	.elseif (var1 == var3)
	
	;code goes here
	
	.else
	
	;code goes here
	.endif

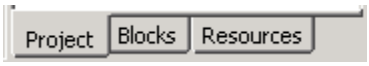
جملة الشرط تبدأ بـ.if وتنتهي بـ.endif كما هو واضح. إذا كان هناك حالات أخرى فنستخدم.elseif و ستتضح الصورة أكثر في الدرس القادم حينما يتم استخدام هذه الصيغ في كتابة dialog procedure

7.1.3 . MENUS, BUTTONS , ETC..

دعونا نتعلم شيئا جديدا.سنرى كيف يمكن عمل قوائم وأزرار وغيرها بالاسمبلي.

أنشئ مشروعا جديدا كما فعلنا في الدرس السابق. والآن من قائمة Project اختر add new Rc. لاحظ ظهور شاشة جديدة بخلفية زرقاء. أيضا فانه في الـ explorer bar سترى التالي: (إن لم يكن الـ explorer bar ظاهرا لديك فمن قائمة view اختر explorer

لاحظ إن المشروع الآن يتكون من قسمين.قسم مخصص للـ source code (ASM Files) وقسم مخصص للـ Resources (Resources Files). الآن في أسفل الـ explorer bar سترى التالي :



اختر Resources. الآن تغير الـ explorer bar وأصبح كما في الصورة التالية :

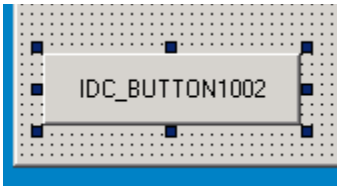
والآن اختر أول زر من اليسار (Add new dialog).. يفترض أن ترى التالي في شاشة الـ resources :

الآن نريد إضافة بعض الأمور لهذا الـ dialog. من قائمة view اختر Tool Box و Dialog.

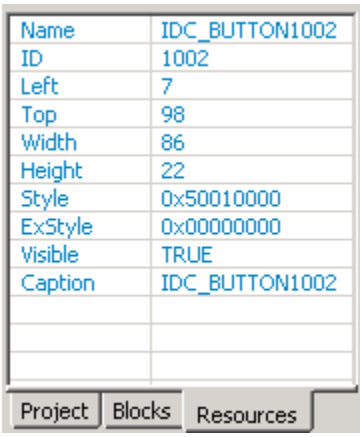
سيظهر شريط الـ Tool Box و شريط الـ Dialog.

اختر سادس زر من اليسار في شريط ToolBox وهو خاص بإضافة الأزرار. ستلاحظ إن مؤشر الماوس تغير فأصبح على شكل علامة +. قم برسم الزر في أي مكان من الـ dialog وليكن في الأسفل.

ستلاحظ أنه أصبح بالشكل التالي :



أما في نافذة الـ explorer bar فسترى التالي :



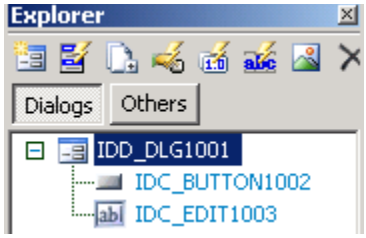
أول خانة هي خانة الاسم. لا تغيرها. ثاني خانة هي الـ ID الخاص بالزر. فكل resource له ID خاص به. آخر خانة Caption هي الكلام المكتوب على الزر. اخترها وامسح النص واكتب أي شيء وليكن : hello.

نعود إلى الـ toolbox والآن اختر رابع عنصر في الشريط (من اليسار) وهو عنصر edit. والآن ارسم مربع في وسط الـ dialog فوق الزر.

تلاحظ ظهور مربع تحرير (edit) والآن من الـ explorer bar اختر أخر عنصر ألا وهو text وامسح الكلام المكتوب واتركه فارغا (لا تكتب شيئا). سيصبح الـ dialog بهذا الشكل

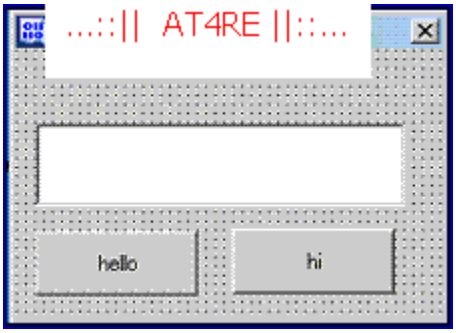
يمكنك تغيير حجم الـ dialog. أنقر في أي مكان عليه لترى بأن الزوايا أصبح بها نقاط..ضع الماوس عليها وتحكم بالحجم وقم بتصغيره أو تكبيره كما تشاء.

الآن دعنا نغير النص الموجود بالأعلى (IDD_DLG1001) من الـ explorer bar سترى بالأعلى التالي :



أول عنصر هو الـ dialog والثاني الزر والثالث مربع التحرير. انقر على الأول واذهب إلى عنصر Caption

وغير الكلام إلى : . أضف زرا آخر – بنفس الطريقة – باسم hi.



والآن قد تريد إضافة الـ dialogs من نماذج جاهزة فما الذي يجب فعله؟ انظر أسفل قائمة Make هناك زر كالتالي :

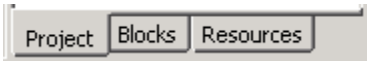
اضغط الزر اليمين.كي ننتقل إلى وضع visual mode : off. ستلاحظ اختفاء الـ dialog وبدلا منه هنالك التالي :

```
#define IDD_DLG1001 1001
#define IDC_BUTTON1002 1002
#define IDC_EDIT1003 1003
#define IDC_BUTTON1004 1004

IDD_DLG1001 DIALOGEX 0,0,158,92
CAPTION "www.at4re.com"
FONT 8,"MS Sans Serif"
STYLE 0x10cc0000
EXSTYLE 0x00000000
BEGIN
    CONTROL "hello",IDC_BUTTON1002,"Button",0x50010000,7,62,60,22,0x00000000
    CONTROL "",IDC_EDIT1003,"Edit",0x50010080,7,25,137,28,0x00000200
    CONTROL "hi",IDC_BUTTON1004,"Button",0x50010000,80,62,60,22,0x00000000
END
```

αλλκο

إذن لإضافة زر يمكنك كتابة بضعة اسطر برمجة. لكن بلا شك فالعمل من خلال الوضع المرئي أسهل بكثير. أيضا إذا كان هناك dialog جاهز تريد إضافته فانسخه والصقه هنا.

الآن من أسفل شريط الـ explorer bar اختر Project



والآن في أعلى الـ explorer bar اختر ملف untiteled1.asm. الآن ما نريد عمله هو عندما نضغط على ذلك الزر نريد إن تظهر جملة في المربع الأبيض. الكود المطلوب سيكون كالتالي :

```
.386
.model flat ,stdcall
option casemap:none

include windows.inc
include kernel32.inc
include user32.inc
includelib kernel32.lib
includelib user32.lib

mydialog proto :DWORD,:DWORD,:DWORD,:DWORD

.data
msg1 db "allko is the best",0
msg2 db "hello",0
.data?
hinstance HINSTANCE ?

.code
start:
    invoke GetModuleHandle,NULL
    mov hinstance,eax
    invoke DialogBoxParam,hinstance,1001,NULL,addr mydialog,NULL
    invoke ExitProcess,NULL

mydialog PROC hwnd:HWND , uMsg:UINT,wParam:WPARAM, lParam:LPARAM
mydialog EndP

end start
```

كالمعتاد سأشرح الكود سطرا سطرا. أول سطر غير مألوف هو

```
mydialog proto :DWORD,:DWORD,:DWORD,:DWORD
```

في لغات البرمجة الأخرى، عندما تريد عمل function مثلا فيجب إن تقوم بالإعلان (declaration)عنها، أيضا هنا يجب أن نعلن عن هذه الإجرائية(procedure). في هذا السطر عرفنا إجرائية سميناها mydialog ومن ثم كتبنا التوجيه proto ومن ثم البارامترات لهذه الإجرائية، وكما ترى جميع البارامترات من نوع DWORD=Double Word.

هذا السطر لن يتغير لجميع الـ dialogs.

السطر غير المألوف التالي هو

```
.data?
hinstance HINSTANCE ?
```

إن data? هو مقطع للبيانات مثله مثل.data لكن الفرق أنه مخصص للـ uninitialized data أي البيانات التي ليس لها قيمة ابتدائية، فكما تلاحظ فان المتغير hinstance بجانبه علامة؟ دلالة على أنه لا يحمل قيمة ابتدائية.

إن دالة GetModuleHandle تقوم بإرجاع مقبض (handle) للـ module المحددة.تعريف الدالة (بالنسبة للـ c++) هو:

```
HMODULE WINAPI GetModuleHandle( LPCTSTR lpModuleName );
```

أي أن البارامتر لهذه الدالة هو مؤشر(pointer) على string بها اسم الـ module الذي نريد مقبض له. وبالتالي إن أردنا مقبضا للـ kernel32.dll مثلا فإننا نخزن هذه الـ string (أي kernel32.dll) في مكان ما بالبرنامج ثم نعرّف مؤشرا عليها. لكن كما تلاحظ في حالتنا فإن البارامتر كان صفرا. في هذه الحالة فإن الدالة تقوم بإرجاع مقبض للملف الذي استخدم في إنشاء الدالة. أي الملف الذي توجد دالة GetModuleHandle فيه. جدير بالذكر أن المقبض المرجع يخزن في eax.

أما الدالة الثانية DialogBoxParam فتعريفها (بالنسبة للـ c++) هو كالتالي :

int **DialogBoxParam** (	
HINSTANCE hInstance,	مقبض للـ module التي يحوي ملفها التنفيذي على الـ dialog box.
LPCTSTR lpTemplateName,	هو مؤشر إلى الـ dialog box.
HWND hWndParent,	مقبض للنافذة الأم. أي النافذة الأصلية التي تفرع منها هذا الـ dialog
DLGPROC lpDialogFunc ,	مؤشر إلى إجرائية(procedure) الـ dialog box.
PARAM dwInitParam);	يحدد القيمة التي ستمرر الـ dialog box (حيث سيستقبلها عن طريق بارامتر lParam من المسج(الرسالة) المسماة WM_INITDIALOG.

الدالة الثالثة سبق شرحها وهي لإنهاء البرنامج. الآن دعنا نرى محتويات تلك الـ procedure :

```
mydialog PROC hwnd:HWND , uMsg:UINT,wParam:WPARAM, lParam:LPARAM
    .if uMsg==WM_COMMAND
        .if wParam==1002
            invoke SetDlgItemText,hwnd,1003,addr msg1
        .elseif wParam==1004
            invoke SetDlgItemText,hwnd,1003,addr msg2
        .endif
    .elseif uMsg==WM_CLOSE
        invoke EndDialog,hwnd,0
    .endif
xor eax,eax
Ret
mydialog EndP

end start
```

αλλκο

أول سطر هو تعريف الإجرائية. إن تعريف الإجرائية هو كالتالي :

INT_PTR CALLBACK **DialogProc** (	
HWND hwndDlg,	مقبض للـ dialog Box
UINT uMsg,	يحدد الرسالة الواصلة إلى الـ dialog
WPARAM wParam,	يحدد معلومات إضافية عن الرسالة الواصلة إلى الـ dialog،
LPARAM lParam);	يحدد معلومات إضافية عن الرسالة الواصلة

إن هذه العناصر الأربعة هي messages. فالـ dialog يتعامل مع النظام عن طريق رسائل.

عندما نقول رسالة واصلة فنحن نعني بذلك أن يضغط المستخدم على زر ما –مثلا- أو أن يغلق البرنامج من زر x بالأعلى.

والآن لنر تعريف دالة SetDlgItemText كما هو موجود في مكتبة MSDN :

```
BOOL SetDlgItemText( HWND hDlg, int nIDDlgItem, LPCTSTR lpString);
```

أول بارامتر هو مقبض للـ dialog box. ثاني بارامتر يحدد العنصر الذي سيعرض النص. الثالث هو مؤشر إلى الـ string التي سوف يتم عرضها.

جدير بالذكر أنه هنالك خيارات فيما يتعلق بتحديد العنصر –سواء بالنسبة للدالة هذه أو لبقية الدوال – فإما أن تكتب اسم العنصر، أي DC_BUTTON1002 مثلا، أو أن تكتب الـ ID الخاص به. بالتأكيد كتابة الـ ID أسهل بكثير وهذا ما فعلناه.

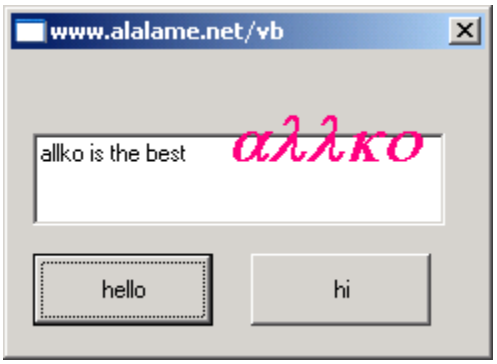
أول سطر في تلك الإجرائية هو : **if uMsg==WM_COMMAND** : وهذا يعني أنه إذا كانت الرسالة الواصلة من المستخدم هي أمر فنفذ الأوامر التالية. ضمن هذه الـ if هناك 2 if الأولى تقول أنه إذا كانت المعلومات التابعة للرسالة الواصلة، صادرة عن العنصر ذي الرقم 1002 <الزر الأيمن> فاستدعي دالة SetDlgItemTextواطلب منها أن تعرض

msg1 في العنصر ذي الرقم 1003 (أي في مربع التحرير edit). الثانية تقول أنه إذا كانت المعلومات التابعة للرسالة الواصلة، صادرة عن العنصر ذي الـرقم 1004<الزر الأيسر> فاستدعي دالة SetDlgItemTextواطلب منها أن تعرض 2 msg في العنصر ذي الرقم 1003 (أي في مربع التحرير edit). بعد هاذين الشرطين هناك endif التي تنهي عمل if.

بعد ذلك هناك : .**elseif uMsg==WM_CLOSE** : وهذه تعني أنه إذا كانت الرسالة الصادر من المستخدم هي طلب إغلاق فنفذ التالي : استدعي الدالة EndDialog التي سوف تقوم بإغلاق الـ dialog box. وبعد ذلك هناك endif التي تنهي عمل if.

بعدها تأتي تعليمة بتصفير المسجل eax, أي **XOR eax,eax** : وهذه الخطوة احتياطية. ثم هناك تعليمة **RET** التي تعيد السيطرة إلى النظام. وفي النهاية نجد mydialog end معلنة إنتهاء الاجرائية.

الآن اضغط على زر Go All في الأعلى. أحفظ الملفات في مكان مناسب (يفضل نفس مجلد winasm) والآن يفترض أن ترى التالي : (هذه الصورة مأخوذة بعد الضغط على الزر الأيمن) :



إن ظهر لك خطأ فراجع الخطوات جيدا...في المرفقات هناك ملفان الأول باسم asm والثاني باسم resource أنشء مشروعا جديدا وانسخ محتويات ملف asm إلى قسم السورس كود ثم من قائمة make اختر new rc والأن انتقل إلى الوضع الكتابي بالضغط على زر الذي أسفل قائمة make وانسخ محتويات ملف resource اليه. اضغط على زر go all واحفظ الملفات في نفس مجلد winasm. الأن سترى النتيجة بلا أخطا.حاول أن تجد خطأك بمقارنة ما كتبته مع ما هو مرفق.

وبهذا يكون درسنا لهذا اليوم قد انتهى آمل أن تكونوا قد استفدتم.

2.7 . المثال الأول

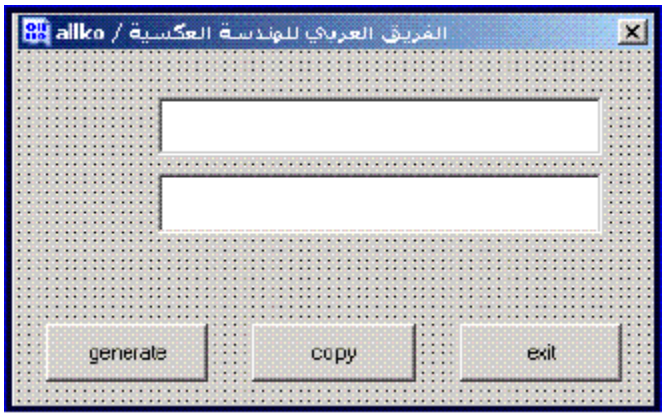
في هذا المثال سنتعلم كيفية صنع كيجين. ألضحية تجدها بالمرفقات باسم ch2_sec7_1

حمل الضحية إلى olly، كما ترى هناك دالة GetDlgItemTextA عند 401172، ضع bp عليها ثم F9. أدخل أي اسم وليكن allko والسريال 123456. اضغط زر register وستلاحظ أننا عدنا إلى Olly. تتبع F8 حتى تصل إلى 4011B6 عندها اضغط F7 لنر ما يجري هناك.

هذه الـ loop تستمر حتى عدد معين من المرات يساوي طول اليوزر ناقص 3، فان كان طول اليوزر 5، فالحلقة ستستمر لدورتين اثنتين، أي يتم عمل عملية الـ **XOR** للحرف الأول والثاني من اليوزر، ثم يتم عمل عملية تحول النتيجة السابقة من مجرد قيم هكس إلى رموز بصيغة ascii. في النهاية يتم إضافة هذه الرموز إلى FIT-. ملخص الموضوع :هناك 4 أحرف في أول كل سريال بغض النظر عن اليوزرنيم المدخل وهي FIT-

الآن شغل برنامج winasm. أنشئ مشروعا جديدا. من قائمة project اختر new rc

قم بإنشاء dialog جديد (أول زر من اليسار في شريط explorer). الآن أنشء مربعي تحرير وثلاث أزرار كما بالصورة التالية :



انتبه إلى أن الـ ID لكل resource هي كالتالي : (غيرها إن لم تكن لديك هكذا)

الـ dialog الرئيسي : 1001
مربع التحرير العلوي : 1002
مربع التحرير السفلي : 1003
زر generate : 1004
زر copy : 1005
زر exit : 1006

الآن فلنذهب إلى قسك السورس كود لنكتب كود الكيجين :

كما تعلم فأول أسطر هي :

```
386.
model flat,stdcall
option casemap:none
include windows.INC
include kernel32.INC
include user32.INC
includelib kernel32.lib
includelib user32.lib
```

يليها تعريف الـ procedures الاثنتان. واحدة للـ dialog والأخرى للـ generation :

```
mydialog        proto   :DWORD,:DWORD,:DWORD,:DWORD
generation proto
```

لاحظ أن إجرائية توليد السريال لا تستقبل شيء. بينما الأولى تستقبل 4 بارامترات. الآن في مقطع البيانات لدينا التالي :

```
.data
special1 db "FIT-",0
special2 db "%d",0
```

الـ string الأولى لإضافة تلك المحارف الأربعة، أما الثانية فستستخدم في دالة wsprintf. ثم لدينا :

```
.data?
hIstance HINSTANCE ?
NameBuffer 32 dup(?)
SerialBuffer 32 dup(?)
mystr 32 dup(?)
```

الأولى هي متغير سنستخدمه لنضع فيه مقبض (handle). هذه الصيغة (صيغة التعريف) احفظها كما هي. إن أردت المزيد من التفاصيل فابحث على الانترنت. الثانية تحجز مساحة فارغة بحجم 32 بايت خاصة بالمتغير NameBuffer. لو كتبت

```
NameBuffer dup('d')
```

فسيتم حجز مساحة بحجم 32 بايت بها 32 حرف d. طبعا هذا ما لا نريده. لذلك نستخدم علامة الاستفهام التي تدل على أن المساحة فارغة (لاحظ أن علامة الاستفهام لم توضع بين علامتي تنصيص).قد تتساءل : ألم يكن المفروض أن نكتب ذلك الأمر هكذا

```
NameBuffer db 32 dup(?)
```

لا مشكلة فنحن نتعامل مع uninitialized variables. يلي ذلك الأسطر التالية :

```
.const
IDC_KEYGEN     equ 1001
IDC_NAME       equ 1002
IDC_SERIAL     equ 1003
IDC_GENERATE equ 1004
IDC_COPY       equ 1005
UDC_EXIT       equ 1006
```

هذا هو مقطع الثوابت.كل ما فعلناه أننا أخبرنا المجمع assembler بأن زر الكيجين له ID رقمها 1001 وهكذا. الآن لدينا التالي :

```
.code
start :
invoke GetModuleHandle, NULL
MOV hInstance,eax
invoke DialogBoxParam, hInstance, IDC_KEYGEN, NULL, addr DlgProc, NULL
invoke ExitProcess,eax
```

- أول سطر يعني أن مقطع الكود قد بدأ.
- ثاني سطر يعني بداية الكود التنفيذي.
- يلي ذلك استدعاء لدالة GetModuleHandle التي تعيد مقبض الملف التنفيذي الحالي ثم يحفظه في المتغير hInstance.
- الخامس سيستدعي الإجرائية الخاصة بالـ dialog. البارامترات الأربعة هي :
  - الأول مقبض الملف التنفيذي الحالي
  - الثاني اسم الـ dialog ويمكنك استبداله بالـ ID الخاص به أي 1001،
  - الثالث مقبض للنافذة الأم (ضعه 0).
  - الرابع مؤشر إلى إجرائية الـ dialog box.
  - الخامس يحدد القيمة التي ستمرر الى dialog box حيث سيستقبلها عن طريق يارامتر lparam)
- أما الدالة الثالثة فهي خاصة بإنهاء البرنامج. الآن سنكتب إجرائية الـ dialog :

```
DlgProc proc hWnd:HWND, uMsg:UINT, wParam:WPARAM, lParam:LPARAM
.if uMsg == WM_INITDIALOG

.elseif uMsg == WM_COMMAND
    mov     eax,wParam
    .if eax==IDC_GENERATE
            invoke GetDlgItemText,hWnd,IDC_NAME,addr NameBuffer,32
            call Generate
            invoke SetDlgItemText,hWnd,IDC_SERIAL,addr SerialBuffer
    .elseif eax==IDC_COPY
            invoke SendDlgItemMessage,hWnd,IDC_SERIAL,EM_SETSEL,0,-1
            invoke SendDlgItemMessage,hWnd,IDC_SERIAL,WM_COPY,0,0
    .elseif eax==IDC_EXIT
            invoke SendMessage,hWnd,WM_CLOSE,0,0
    .endif

.elseif     uMsg == WM_CLOSE
    invoke EndDialog,hWnd,0
.endif
   xor      eax,eax
   ret
DlgProc endp
```

لن أشرح مرة أخرى فقد سبق وشرحت كودا مشابها في الدرس السابق.بشكل عام هناك ثلاث messages (باللون الأزرق) أساسية . الأولى وكما ترى فارغة ، وهي خاصة بعملية الـ initialization (البدء) . يمكنك هنا أن تضع الكود الخاص باستخدام icon للكيجين مثلا...أو الكود الخاص بتشغيل موسيقى (كما سنرى لاحقا) . مؤقتا اتركه فارغا.

ربما التغيير الجديد هو في الكود الخاص بزر copy .

```
.elseif eax==IDC_COPY
        invoke SendDlgItemMessage,hWnd,IDC_SERIAL,EM_SETSEL,0,-1
        invoke SendDlgItemMessage,hWnd,IDC_SERIAL,WM_COPY,0,0
```

حسنا لا داعي لإطالة الوقت في شرح تلك الدوال فأنا نفسي لم أذهب الى msdn كي أفهمهم. يكفي أن تحفظهم كما هم وإن أردت التفاصيل فعليك بـ msdn . فقط انتبه الى البارامترات...البارامتر الثاني (في الدالتين) هو المكان الذي تريد أن تنسخ منه وهو هنا IDC_SERIAL .

أما زر generate فتجده أسفله الكود :

```
.if eax==IDC_GENERATE
        invoke GetDlgItemText,hWnd,IDC_NAME,addr NameBuffer,32
        call Generate
        invoke SetDlgItemText,hWnd,IDC_SERIAL,addr SerialBuffer
```

أي ان الضغط على ذلك الزر يعني : 1- استدعاء دالة GetDlgItemText التي ستأخذ اليوزرنيم(IDC_NAME) وتخزنه في NameBuffer . بالطبع فهي لن تأخذ الا الـ 32 حرفا الأولى كما هو واضح من البارامتر الرابع. ثم يتم استدعاء إجرائية توليد السيريال ، وفي النهاية يتم استدعاء دالة SetDlgItemText التي تعرض السيريال في المكان المخصص.

والآن لنرى الكود الخاص بإجرائية توليد السيريال (قمت بتلوين الكود لتسهيل عملية التتبع) :

```
1  Generate proc
2  mov edi,offset NameBuffer
3  invoke lstrlen, edi
4  cmp eax,5
5  jl NOINPUT
6  invoke lstrcat,addr SerialBuffer,addr special1
7  xor eax,eax
8  xor ecx,ecx
9  invoke lstrlen, edi
10 mov ecx, eax
11 sub ecx,3
12 mov edx,4E6AF4BCh
13 xor eax,eax
14 xor ebx,ebx
15 begin:
16 mov eax,dword ptr [ebx+edi]
17 inc ebx
18 xor edx,eax
19 dec ecx
20 jnz begin
21 invoke wsprintf,addr str3,addr special2,edx
22 invoke lstrcat,addr SerialBuffer,addr str3
23 NOINPUT:
```

```
24 ret
25 Generate endp
```

السطر 1 و 25 يوضحان أين تبدأ تلك الإجرائية وأين تنتهي...
السطر 2 يقوم بأخذ عنوان اليوزرنيم ويضعه في edi (غالبا يستخدم edi لمثل هذه المهام) .
السطر 3 يستدعي دالة lstrlen ويدفع لها مؤشرا الى السترنج الخاصة باليوزرنيم . هذه الدالة تحسب طول السترنج التي تعطيها مؤشرا عليه (نحن أعطيناها مؤشرا على اليوزرنيم) وتقوم بإعادة الطول في EAX .
السطر 4 يفحص ما إذا كان السيريال المدخل طوله اقل من 5 أم لا...إذا كان صفرا (أي لم يتم إدخال أي شيء) أو كان اقل من 5 فان السطر 5 به تعليمة jl (أي jump if low) فسيتم الذهاب الى NOINPUT (هل عرفت الآن ماذا قصدت في الدرس السابق بقولي : label ؟ إن NOINPUT و begin هما مثالان على ذلك) وبعد الوصول الى هناك سيتم تنفيذ الكود الذي هناك..ببساطة ما يوجد هناك هو تعليمة ret التي تعيد السيطرة الى البرنامج و تنهي هذه الإجرائية .
السطر 6 يستدعي دالة lstrcatA التي ستضيف FIT- الى بداية السيريال .
السطر 7 و 8 يقومان بتصفير eax و ecx تمهيدا لاستخدامها.
السطر 9 يستدعي دالة lstrlen مرة أخرى لوضع الطول في eax . (((ملاحظة : كان بالإمكان عندما حسبنا الطول للمرة الأولى أن نضعه في متغير ما...ونستخدمه كلما احتجناه في البرنامج...))) .
السطر 10 ينقل الطول من eax الى ecx .
السطر 11 يطرح 3 من الطول .(لماذا 3 ؟ لا علاقة لي !!! هكذا يريد المبرمج اسأله !! إنها مجرد قيمة اختيارية :) .
السطر 12 يضع القيمة 4E6AF4BCh في edx تمهيدا لاستخدامه في الـ loop .
السطر 13 و 14 يصفران eax و ebx تمهيدا لاستخدامهما في الـ loop .
السطر 15 هو label سنتحدث عنه بعد قليل.
السطر 16 هو بداية الـ loop . كما ترى يتم نقل محتويات الذاكرة التي عنوانها يساوي ebx+edi الى eax ، 4 بايت ف المرة الواحدة (أي DWORD) . لاحظ أنه كلما رأيت تلك الصيغة فإن ما يوجد بين القوسين يشير الى عنوان ذاكرة قد يكون مثلا هكذا [ecx+8] أو هكذا [3* edi + ebx] أو أي صيغة أخرى...أما DWORD PTR فكلمة DWORD هي اختصار double word وكل word تساوي 2 بايت ، إذن DWORD تساوي 4 بايت. انتبه فلا يمكنك أن تكتب التالي:

```
mov eax,word ptr [ebx+edi]
```

لأن eax حجمه 32 بت أي 4 بايت أي dword وبالتالي من الخطأ أن تنقل إليه word . أيضا العكس خطأ فالصيغة

```
mov ax,dword ptr [ebx+edi]
```

ستتسبب في خطأ فليس من المنطق نقل 32 بت الى مسجل بحجم 16 بت فقط. يجب أن يكون الـ source والـ destination متساويين بالحجم...طبعا هذا يخص تعليمة mov لا أعرف إذا ما كان هناك تعليمات تخرج عن هذه القاعدة.
جدير بالذكر أنني اخترت ebx كعداد لليوزرنيم ...اخترت edx كمكان لوضع القيمة 4E6AF4BCh ، اخترت edi كمؤشر الى اليوزرنيم...لماذا؟ يمكنك نظريا اختيار ما تريد...لكن هناك بعض الضوابط...حاول الابتعاد عن eax فكما ترى نحن نستخدمه مع الدوال فمعظم الدوال تعيد قيمها فيه...أما ecx فهو من اسمه أي extended counter اعتدنا استخدامه كعداد. edi و ebi يستخدمان غالبا كمؤشر وبالتالي لا تضع فيهم قيما ثابتة...ضع فيهم عناوين ذاكرة فهذا هو السبب الأساسي الذي دفع Intel لصناعتهما :) .
السطر 17 يقوم بزيادة ebx بمقدار واحد. لاحظ أنه بداية كان edi يشير الى أول حرف (راجع السطر الثاني) و ebx قيمته صفر إذن المجموع سيشير الى أول حرف...لكن في الدورة الثاني من الـ Loop نريد أن نتعامل مع ثاني حرف وبالتالي يجب زيادة ebx بمقدار واحد فيصبح المجموع يشير الى الحرف الثاني وهكذا.
السطر 18 هو أساس الـ loop وهو الذي سيولد السيريال...ببساطة يتم عمل xor لكل حرف مع القيمة المخزنة في edx وهي 4E6AF4BCh (طبعا حرف h ليس جزئا من القيمة إنما هو علامة على أن الرقم بصيغة هكس). لاحظ أن القيمة النهائية تخزن في edx.
السطر 19 ينقص ecx بمقدار واحد.
السطر 20 فيه قفزة (Jump if NOT Zero). لاحظ أن ecx يتناقص تدريجيا.. فورما تصبح قيمته صفر فإن راية الصفر (Zero Flag) سترتفع أي تصبح قيمتها واحد دليلا على أن احد المسجلات أصبحت قيمته صفرا. ما تفعله تلك القفزة هو النظر إلى هذه الراية هل هي مرتفعة أم لا؟ إذا كان نعم فإنها لن تقفز وستتابع البرنامج كالمعتاد وإلا فإنها ستقفز. إلى أين ستقفز؟ إلى الـ label المسمى begin. وبالتالي ستعيد الحلقة مرة أخرى.. سنستمر هكذا إلى أن ينتهي العد ويصل ecx إلى صفر.

السطر 21 : يستدعي دالة wsprintf.. نعطيها قيمة معينة. و Format Control معين هو %d في حالتنا. وتعطينا الناتج النهائي.
أول بارامتر لها هو المكان الذي سنخزن النتيجة فيه. وهو str3 (هل يجب أن نقول : يمكنكم اختيار أي اسم آخر؟)
ثاني بارامتر هو الـ format control والثالث هو القيمة التي نود " فك تشفيرها" إن صح التعبير وهي هنا edx لأنه كما قلنا يحوي نتيجة تلك الـ Loop.
السطر 22 يضيف ما حصلنا عليه للتو وخزناه في str3 إلى السريال (المبدوء بـ FIT-)
السطر 23 و 24 تم شرحهم.
السطر 25 هو نهاية الـ procedure
والآن يجب أن ننهي البرنامج. ببساطة تحتاج إلى

end star

اضغط زر Go All. احفظ ملفات البرنامج بأي اسم (يفضل في نفس مجلد البرنامج).. والآن جرب واستمتع !

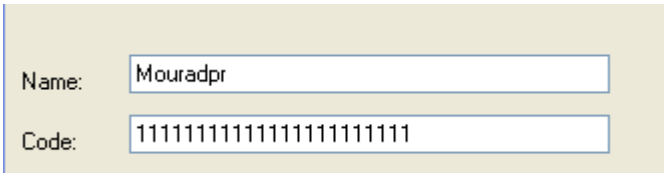
3.7. المثال الثاني

في هذا الدرس سنقوم بكسر حماية برنامج FloderView 2.2 باتباع الخطوات العامة التالية :

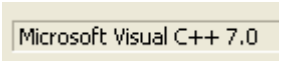
- معرفة ما المطلوب.
- تنقيح البرنامج و تحديد موقع الخوارزمية.
- طريقة الحصول على كود شبه جاهز باستعمال Code Ripper Plugin
- البداية في استغلال الخوارزمية وبرمجة KeyGen.
- تجربة KeyGen.

معرفة ما المطلوب.

أول شيء يجب أن تعرفه عن البرنامج بعد فتحه هل يطلب اسم ورقم تسجيل أم اسم فقط أم رقم تسجيل فقط وتتساءل هل بينهما علاقة، أي هل يدخل الاسم في توليد رقم التسجيل أم يتدخل شيء آخر في ذلك. حسنا لنفتح البرنامج ونضغط على زر Enter Registration :



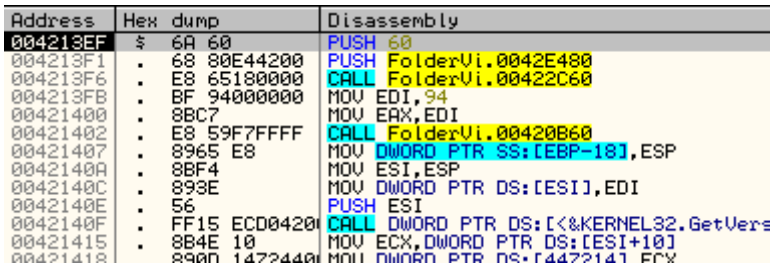
بعد الضغط على ok ستظهر رسالة الخطأ التالية : « Sorry, you have entered an incorrect registration code. »

بعد فحص البرنامج 

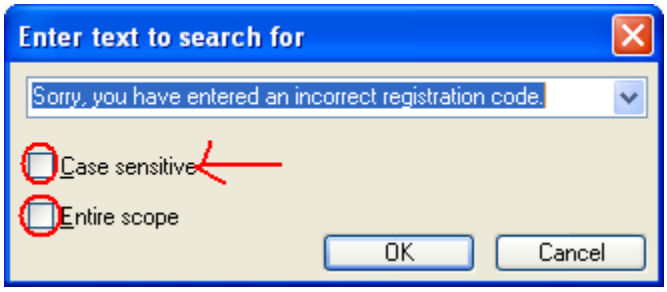
يظهر أنه غير مشفر أو مضغوط

7.3.1 . تنقيح البرنامج و تحديد موقع الخوارزمية.

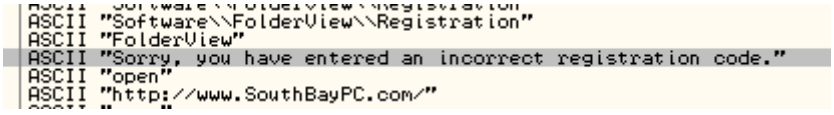
حمل الضحية في ollydbg :



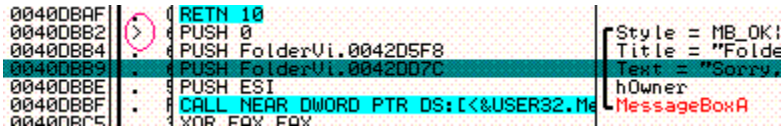
اضغط بالزر الأيمن ثم اتجه إلى « Search For → All Reference text strings »



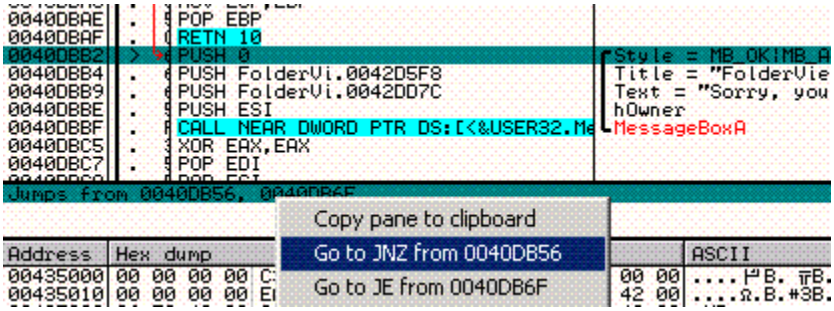
يمكنك أن تكتب جزءا منها فقط وستجدها بالبحث المتتالي. بالنسبة لـ Entire Scope فيمكنك وضع علامة الصح بجانبه إن لم ترد إزالتها.



بعد الضغط عليها مرتين ستتجه إليها في قسم الكود :



السهم المشار إليه بدائرة حمراء يعني وجود قفزة تأخذك إلى هذا المكان. لنذهب إليها : :

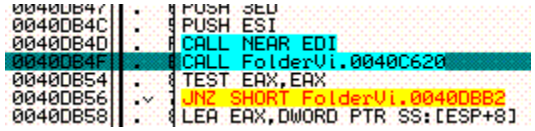


لاحظ معي أنه يتم القفز لهذه الرسالة من عنوانين. في الغالب نختار أصغرها قيمة في العنوان

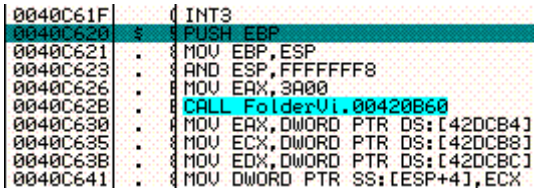
(وبعدها ننتقل للآخر)، مثلا في هذه الحالة لدينا :

0040DB56
0040DB6F

نتجه لأصغرها أي 0040DB56. كما تلاحظ فيوجد أمر **TEST** EAX,EAX بعد أمر **CALL** 0040C620. وكالمعتاد : إذا خرجنا من هذه الـ **CALL** 0040C620 بقيمة 0 في EAX فإنه سيقفز لرسالة الخطأ والعكس في حالة eax=1 حيث لن يقفز.



لنر ما يوجد بداخل الـ **CALL** تلك :



تابع النزول ستجد مجموعة من المعطيات ☺ :

```
0040C8A8|  .  MOV ECX,0A
0040C8AD|  .  MOV ESI,FolderVi.0042DC0C          ASCII "Super Cleaner 2.21 Keygen by Saiyaj
0040C8B2|  .  LEA EDI,DWORD PTR SS:[ESP+C10]
0040C924|  .  MOV ECX,6
0040C929|  .  MOV ESI,FolderVi.0042DBC0          ASCII "SuperCleaner v2.4 *KeyGen*"
0040C92E|  .  LEA EDI,DWORD PTR SS:[ESP+1010]
0040C935|  .  REP MOVS DWORD PTR ES:[EDI],DWORD P
```

فلنتابع للأسفل. رائع ها قد وصلنا للخوارزمية الآن ضع BP في أول الحلقة ثم F9 :

```
0040D31F|     INT3
0040D320|$    SUB ESP,100
0040D326|.    PUSH EBX
0040D327|.    PUSH EBP
0040D328|.    PUSH ESI
0040D329|.    PUSH EDI
0040D32A|.    MOV BYTE PTR SS:[ESP+10],0
0040D32F|.    XOR EAX,EAX
0040D331|.    MOV ECX,3F
0040D336|.    LEA EDI,DWORD PTR SS:[ESP+11]
0040D33A|.    REP STOS DWORD PTR ES:[EDI]
0040D33C|.    STOS WORD PTR ES:[EDI]
0040D33E|.    STOS BYTE PTR ES:[EDI]
0040D33F|.    MOV EDI,DWORD PTR SS:[ESP+114]
0040D346|.    PUSH EDI                          String
0040D347|.    CALL NEAR DWORD PTR DS:[<&KERNEL32.  lstrlenA
0040D34D|.    MOV ESI,EAX
0040D34F|.    XOR ECX,ECX
0040D351|.    XOR EAX,EAX
0040D353|.    TEST ESI,ESI
0040D355|.v   JBE SHORT FolderVi.0040D36D
0040D357|.    MOV EDX,DWORD PTR DS:[4370F0]
0040D35D|.    LEA ECX,DWORD PTR DS:[ECX]
0040D360|>    MOVSX EBX,BYTE PTR DS:[EAX+EDI]
0040D364|.    ADD EBX,EDX
0040D366|.    ADD ECX,EBX
0040D368|.    INC EAX
0040D369|.    CMP EAX,ESI
0040D36B|.^   JB SHORT FolderVi.0040D360
0040D36D|>    MOV EBX,DWORD PTR SS:[ESP+118]
0040D374|.    PUSH ECX                          <%u>
0040D375|.    PUSH FolderVi.0042DD1C            Format = "%u
```

7.3.2 . دراسة الخوارزمية

افتح البرنامج مرة اخرى و اضغط على زر Enter Registration :

ستخرج لك نافذة التسجيل أدخل Mouradpr و 1111111111. هذه المرة لم يعطي الرسالة المزعجة إذ كما ترى فإنه توقف عند الـ bp. حان وقت التتبع. لنتتبع البرنامج ونرى كيف يولد السيريال وذلك بالضغط على الزر F8. وانتبه على قسم التعليقات و المسجلات و المكدس.

كما تلاحظ فبعد تجاوزك لدالة lstrlenA فإنه يسجل طول الاسم في EAX في مثالنا Mouradpr طوله 8 أحرف. الآن سندخل لأول حلقة سأشرحها بالتفصيل:

```
0040D35D  .  LEA ECX,DWORD PTR DS:[ECX]
0040D360  >  MOVSX EBX,BYTE PTR DS:[EAX+EDI]
0040D364  .  ADD EBX,EDX
0040D366  .  ADD ECX,EBX
0040D368  .  INC EAX
0040D369  .  CMP EAX,ESI
0040D36B  .^ JB SHORT FolderVi.0040D360
0040D36D  >  MOV EBX,DWORD PTR SS:[ESP+118]
```

MOVSX EBX,BYTE PTR DS:[EAX+EDI]	يتم نقل أول بايت من محتوى EDI (M في مثالنا) إلى EBX طبعا كلما زاد EAX كلم انتقلنا للحرف التالي.
ADD EBX,EDX	سنضيف EDX ل EBX و نضع الناتج في EBX
ADD ECX,EBX	سنضيف EBX ل ECX وتضع الناتج في ECX
INC EAX	سنزود eax بمقدار 1.
CMP EAX,ESI	مقارنة بين EAX و ESI إن EAX يزيد بمقدار 1 في كل دورة و ESI يحمل طول الاسم وبالتالي فإذا تجاوزت EAX طول الاسم فإنه سيخرج من الحلقة عن طريق الأمر التالي.
JB SHORT 0040D360	اقفز إذا أقل.

بالنسبة لـ wsprintF فهذه الدالة فقط للتحويل بين الموازين إن صح التعبير و لها ثلاث بارامترات الأول هو مكان تخزين الناتج، الثاني هو أمر التحويل، الثالث هو ما تريد تحويله. ويختلف نوع التحويل باختلاف البارامتر الثاني المسمى Format

في حالتنا فهو يستخدم %u يعني An unsigned integer argument.

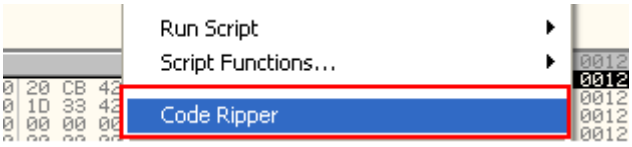
الأمر يبدوا واضحا الآن فكما ترى في آخر الخوارزمية تجد السيريال الصحيح للاسم Mouradpr. وذلك بعد أربع حلقات مختلفة النتائج:

```
Registers (FPU)
EAX 0012E9EC ASCII "1242-33680-1082-9262"
ECX 77E7A7F1 kernel32.77E7A7F1
EDX 00000005
EBX 00000000
ESP 0012E8DC ASCII "9262"
EBP 0012FD04
```

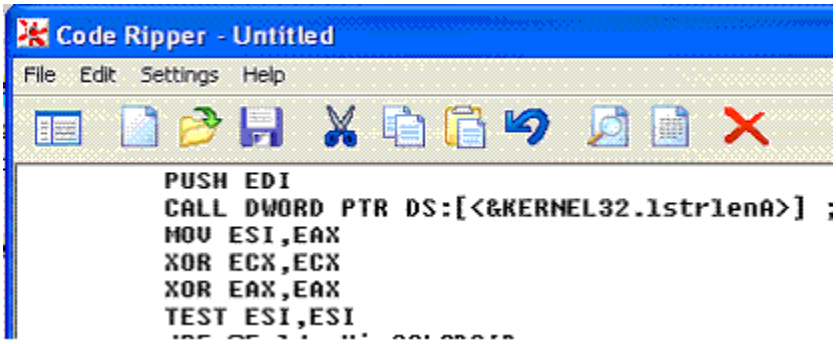
وماذا عن باقي الخوارزمية ألن نشرحها؟ لا ☺. تابع القراءة وستفهم كل شيء :)

طريقة الحصول على كود شبه جاهز باستعمال Code Ripper Plugin

سنقوم بنسخ الخوارزمية كما هي ووضعها في الكيجين.هذا سيوفر الكثير من الوقت.كل ما سنحتاج عمله هو تغيير بعض العناوين والقفزات. ظلل الخوارزمية أي من 40D320 وحتى 40D448. ثم اضغط باليمين واختر Code Ripper



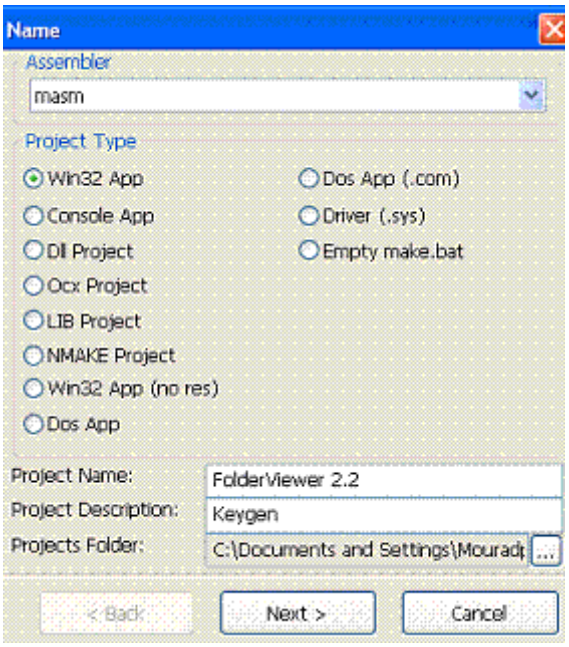
ستظهر لك نافذة الPlugin :



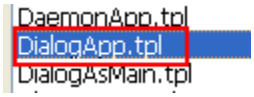
الآن انسخ الكود ولنجري التعديلات عليه : نحذف ما ليس ضروري ونصحح قيم EDX.

البداية في استغلال الخوارزمية وبرمجة KeyGen

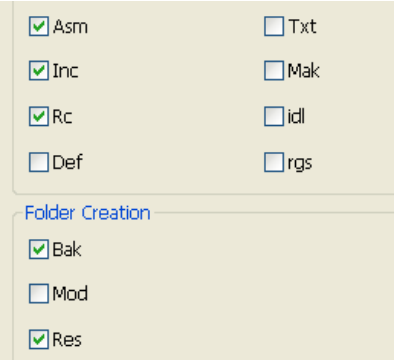
سأستعمل RadAsm نظرا لمرونته ومساعدته لك: الآن افتح برنامج RadAsm ثم New Project ← File.



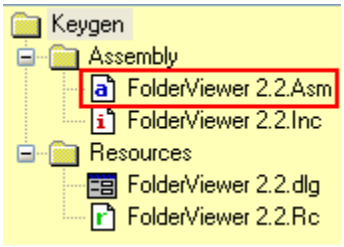
ثم Next :



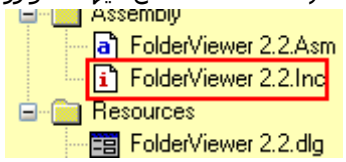
ثم :



اضغط Next ثم Finish. الآن اضغط على ملف ASM :



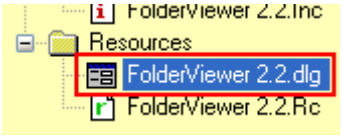
سيظهر لك الكود. سنضيف proc جديدة باسم Generate لنضع فيها الخوارزمية،اضغط على ملف **INC**



ثم أضف هذا السطر كما ترى بالصورة :

```
DlgProc         PROTO   :HWND,:UINT,:WPARAM,:LPARAM
Generate proto
```

الآن سنجهز شكل الكيجن، اضغط على ملف.dlg



ثم جهزه على الشكل الذي تريد كهذا مثلا:

ما يجب أن تتذكره هو ID لبعض الـ Controls. سنغيرها لقيم أصغر كمثال:

خانة الاسم : ID = 1، خانة التسجيل : ID=2، زر التوليد ID=3 :، زر حول : ID=4، زر الخروج : ID= 5. أما الـ Dialog أبقه كما هو 1000. الآن لندخل قسم كتابة الكود. سنستخدم بداية دالة GetDlgItemTextA لجلب النص و هذه الدالة تطلب أربع بارمترات الأول يجب أن يكون HWND إذا استعملت Dialog Proc وقد تكون NULL إذا لم تستعمل Dialog proc. أما الثاني فهو ID الخاص بال Controls في حالتنا أعطيناه القيمة 1. والثالث هو متغير (buffer) لحفظ الاسم. أما الرابع فهو الحد الأقصى لعدد البايتات سنعطيه 32 كحد أقصى.

الآن ماذا بعد أخذ الاسم؟ سنتجه للخوارزمية. أنظر الصورة :

```
DlgProc endp
Generate Proc
    |

    ret

Generate endp
end start
```

الآن سننسخ ما حفظناه في المفكرة، سننسخه إلى المكان الموضع في الصورة السابقة.

بعد نسخه، سنعدل لكن ليس كثيرا فقط سنغير شكل الدوال وكذلك سنعوض المسجلات بمتغيرات جديدة وأيضا سنعطي EDX قيمها قبل كل حلقة من الحلقات الأربع (أي 32h قبل الأولى و 28h قبل الثانية و 0E1h قبل الثالثة و 0B0h قبل الرابعة)

سنحتاج للرسالة التي ستظهر عند زر About وكذلك رسالة خطأ إذا كان إذا لم يتم كتابة الاسم سننبه لذلك. وسنحتاج بعض الدوال الأخرى كدالة طبع المحتوى بالإضافة إلى دالة النسخ ودالة المسح.. أعتقد أن هذه الأمور واضحة فبمجرد رؤيتك لها في السورس كود ستفهمها بنفسك.

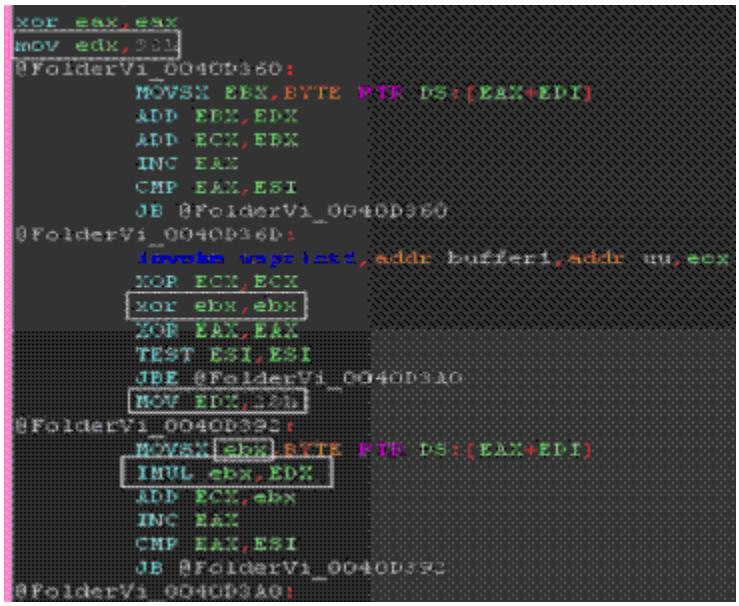
دعنا نرى طريقة تعريف دالة SetDlgItemText :

```
SetDlgItemText (
  HWND hDlg,              // handle of dialog box
  int nIDDlgItem,         // identifier of control
  LPCTSTR lpString        // text to set
);
```

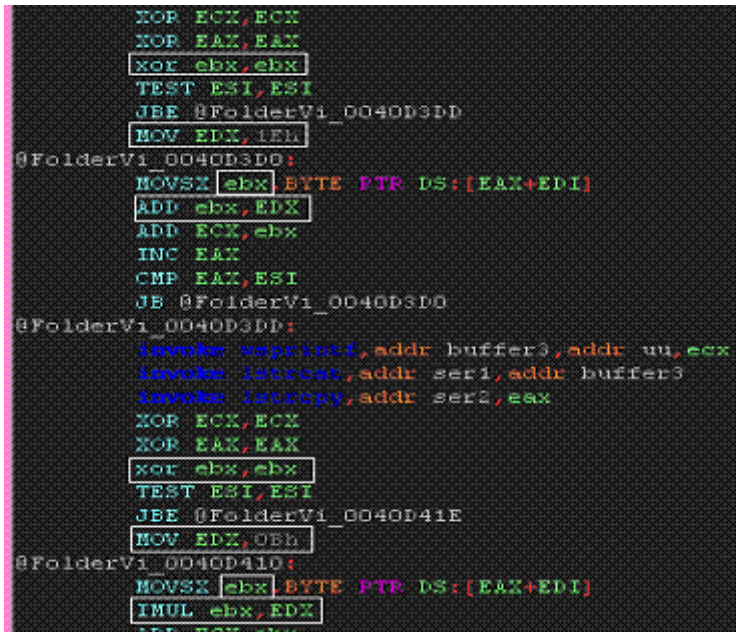
لن أشرح مجددا فهي واضحة.

وكذالك ستواجه خطأ برمجي أثناء تنفيذ الكيجن والسبب هو استعمال مسجل EBP في الخوارزمية وبالتالي تتغير قيمته وبما أن دالتي الطبع والحفظ: SetDlgItemText و GetDlgItemText تحتاج هذا المسجل لذلك سنعوض هذا المسجل بآخر بعد تصفيره طبعا.

إذن بعد تصحيح الكود لدينا التالي :



و



بعد الخروج من الخوارزمية سنقوم بطبع المحتوى في خانة السيريال.. باقي التعليمات في البرنامج يمكنك فهمها بنفسك فهي لا تحتجا إلى ذكاء ☺

الآن لنجمع الملف ونرى هل توجد أخطاء وذلك بالضغط على الزر :  أو اتجه إلى :

Make → Go

ظهر الكيجين الآن أكتب أي اسم ثم Generate وقم بالتجربة..رائع لقد نجحنا ☺

وبهذا ينتهي الفصل السابع.


الباب
الثاني
الفصل الثامن
BruteForcing

8. الفصل الثامن : BRUTE FORCING

بداية، لا تسأل عن ترجمة لعبارة brute-force. دعنا الآن نتعرف على مفهومه الـ brute-forcing. الـ bruter هو برنامج أشبه ببرامج تخمين كلمات سر الملفات المحمية أو برامج تخمين كلمة سر الإيميل. لكنه يقوم بنفس المهمة - مع بعض الفروقات - لبرنامج ما. حيث يقوم بتخمين السريال نمبر

المقابل لاسم ما، والآن، هناك سؤال يطرح نفسه. ما الحاجة إلى كتابة bruter؟ لم لا نستخدم الطرق التي تعلمناها سابقا وتحديدا الـ serial fishing؟

لأنه وببساطة : لا يمكنك ذلك.

عليك أن تفرق بين نوعين من الخورازميات. الأول يمكن أن تعكسه، أما الآخر فلا يمكن عكسه.

1.8 . المثال الأول

قمنا بكتابة برنامج اسمه BRUTE_ME  تجده في المرفقات مع السورس كود له. والآن لننظر إلى آلية عمله:

```
generate proc Hwnd:HWND
invoke lstrlen,addr s_buffer
LEA edx, s_buffer
MOV bl,byte ptr ds:[edx]
add bl,byte ptr ds:[edx+eax-2]
add ebx,3
PUSH ebx
MOV bl,byte ptr ds:[edx+2]
MOV al,byte ptr ds:[edx+1]
XOR eax,ebx
POP ebx
add ebx,eax
CMP ebx,70h
JE good1
ret
good1:
invoke MessageBox, Hwnd ,addr good,addr good , MB_OK
ret
generate EndP
```

بداية يتم حساب طول السريال المدخل (الطول يخزن في eax)، ثم ينقل الحرف الأول إلى bl ثم يتم إضافة الحرف الذي ترتيبه eax-2 إلى bl، (مثال : إذا كان الطول 8، عندها سينقل الحرف السادس). ثم نضيف القيمة 3 إلى المجموع ونخزن قيمة ebx في المكدس للرجوع إليها لاحقا.

الآن يتم نقل الحرف الثالث (edx+2) إلى bl ثم ينقل الحرف الثاني (edx+1) إلى al ويتم اجتراء عملية **XOR** بينهما. (كما تلاحظ هذه العملية ناتجها يخزن في eax)

والآن يتم اعادة قيمة ebx إلى ما كانت عليه، ومن ثم نضيف نتيجة عملية الـ **XOR** إلى ebx لنحصل على الجواب النهائي. الآن ببساطة تتم مقارنة النتيجة مع القيمة 70 هكس.

إذن نحن نبحث عن عدد عند إجراء العمليات السابقة عليه يكون الناتج 70 هكس. هذا العدد لا نعلم من كم خانة يتكون.

وكما ترى فإن هذه الخوارزمية من المستحيل عكسها. لماذا؟ لأنها تستخدم السريال المدخل. أي أنك إذا رأيت برنامجا يطلب اليوزرنيم والسريال وعند تنقيحه وجدت أن خوارزميته تستخدم اليوزرنيم فقط للحصول على سريال صحيح، فهذه - على الأرجح - يمكن عكسها، أما ان رايت أنها تستخدم السريال

المدخل واليوزرنيم معا لتوليد السريال الصحيح فهذه - على الأرجح - لا يمكن عكسها. وليس أمامك في حالة كتلك إلا الـ brute forcing أو الـ patching

قبل شرح الـ bruter دعونا نلقي نظرة على هذه الملاحظة الجميلة.

- كما ترى فهذه الضحية BRUTE_ME تطلب منك إدخال سيريال فقط. لكن هذا لا يعني أن فكرة الـ brute forcing تنطبق على الضحايا من هذا النوع فقط فبعد هذا المثال هناك مثال (ضحية تجارية) تطلب يوزرنيم وباسوورد وسنقوم بعمل brute-forcing لها.
- الكود المستخدم في كتابة الـ bruter قد يبدوا غريبا قليلا. ألسبب هو أننا لم نستخدم dialog فيه، فقط هناك كود يظهر لك message بها النتيجة. لا يوجد قوائم ولا أزرار ولا مربعات ولا أي شيء.هكذا يصبح البرنامج أصغر حجما وبرمجته تتطلب وقتا أقل.

```
pusha
next_pass:
    mov edi,offset Serial
begin:
    inc byte ptr[edi]
    .if byte ptr[edi]==03ah
        mov byte ptr[edi],030h
        inc edi
        .if edi==offset Serial_end
            invoke MessageBox,NULL,addr Fail,addr Fail,MB_OK
            popa
            ret
        .endif
        jmp begin
    .endif
```

هذا هو الجزء الأساسي في الـ Bruter، والذي سيتكرر في أي Bruter آخر. السطر الأول a**PUSH** هو لدفع قيم جميع المسجلات إلى الـ stack.

السطر التالي **MOV** edi,offset Serial واضح. يلي ذلك عداد counter يقوم بزيادة القيمة التي يشير إليها edi أي Serial.

لاحظ في قسم.code أن الـ SErial تساوي 00000000. هكذا سنشمل أكبر عدد ممكن من السريالات. لأننا سنبحث عن أعداد مكونة من 8 خانات فما دون ذلك، بإمكانك مثلا أن تستبدل ذلك الرقم بـ 00 أي أن تبحث عن أعداد من خانتين فما هو أقل لكنك لن تجحد أي عدد مكون من خانتين (أو أقل) يحقق الخوارزمية..

إذن بعد هذا الأمر سنحصل على 10000000 ثم يلي هذا الأمر جملة شرطية (.if byte ptr[edi]==03A) أي نريد أن نعرف هل وصلنا إلى A03 أم لا.

قد تعتقد أنه يفترض أن نستبدل هذه القيمة بـ 039 أي ما يعادل الرقم 9 وهو آخر رقم في النظام العشري، لكن ذلك سيجعلنا ننهي العد عند 8 وليس عند 9، إذن كي تكون الـ 9 (hex 039) مشمولة بالعد علينا أن نجعل الشرط يتحقق من القيمة التي تليها مباشرة أي A03.

حسنا وصلنا إلى 9.. ماذا بعد ذلك؟ يجب إعادة هذه الخانة إلى 0 ويتم ذلك بالأمر (**MOV** byte ptr [edi],030h) ومن ثم نقوم بزيادة مؤشر العد (**INC** edi) كي تكون الزيادة خاصة بالخانة الثانية من اليسار وليس الاولى. أي أن الـ 90000000 ستصبح 01000000.

بعد ذلك نبدأ بزيادة الخانة الأولى كالمعتاد. فنحصل على 11000000 ثم 21000000 ثم 31000000 وفي نهاية الحلقة نحصل على 91000000 ثم يليها 02000000 ثم 12000000 ثم 22000000 وهكذا.

قد تبدوا هذه الطريقة بالعد غريبة.صحيح لأننا بدأنا العد من الخانة الأكبر أي الخانة رقم 8 (MSB). السؤال الذي يطرح نفسه لماذا نقوم العد من الخانة الأمر مع أن المفروض أن نبدأ العد بالخانة الأصغر (خانة الآحاد)؟ باختصار هذه الطريقة تعطي نفس النتيجة لكنها أسهل بكثير.حاول إعادة برمجة هذا الجزء بحيث يصبح العد من اليمين وستلاحظ الفرق.

الأمر التالي هو **if edi==offset serial_end** يعني باختصار : هل انتهينا من العد؟ اذا انتهينا فستخرج رسالة Nothing Found وان لم ننتهي فسنعود للعد من جديد.

والآن بعد أن اطلعنا على الجزء الخاص بالعداد دعونا نرى الجزء الثاني.هذا الجزء يجب أن يكون موجود في جميع الـ bruters لكن بالطبع سيختلف من واحد لآخر.

```
xor ebx,ebx
xor eax,eax

invoke lstrlen,addr Serial
lea edx,  Serial
mov bl,byte ptr ds:[edx]
add bl,byte ptr ds:[edx+eax-2]
add ebx,3
push ebx
mov bl,byte ptr ds:[edx+2]
mov al,byte ptr ds:[edx+1]
xor eax,ebx
pop ebx
add ebx,eax
cmp ebx,70h
je good1
jmp next_pass
ret
```

بداية نقوم بتصفير eax,ebx لأننا سنيتخدمهم، ثم نحسب طول السريال، وننقله إلى edx

والآن تبدأ عملية التحقق ولا داعي لشرحها فهي - بالطبع - نفس المستخدمة في الضحية. في نهاية التحقق إذا كانت النتيجة مساوية لـ 7 هكس فسنذهب إلى good boy ومن ثم نعيد عملية البحث لعلنا نحصل على عدد آخر يحقق المطلوب. وستلاحظ وجود أعداد كبيرة جدا تحقق ما نريده.ذلك عائد إلى بساطة خوارزمية التحقق.كلما زاد تعقيدها قل عدد الأعداد التي تحققها. وبالطبع كلما زاد مجال البحث (أي جعلنا السريال من 10 خانات مثلا) فذلك يعني زيادة احتمال إيجاد أعداد أكثر.

هكذا نكون شرحنا الأجزاء الأساسية في الـ bruter. ستجد بالمرفقات مجلدين الأول اسمه BRUTE_ME

وهو الضحية والآخر BRUTER_AT4RE وهو الحل.

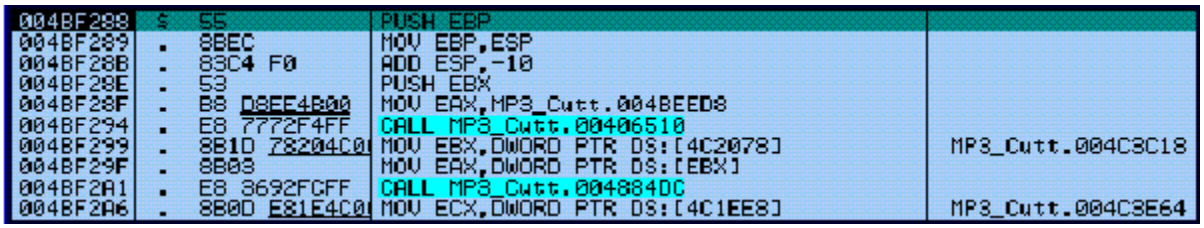
2.8 . المثال الثاني

دعونا نلقي نظرة على هذا البرنامج التجاري كي نختبر ما تعلمناه.

ستجد بالمرفقات الضحية واسمها mp3_cutter_joiner 2.12 قم بتثبيته وافحصه بـ PEiD.

Borland Delphi 6.0 - 7.0

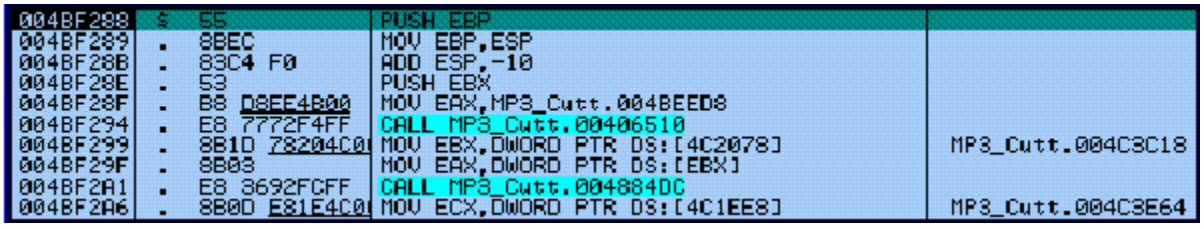
حسنا جرب البرنامج واحفظ رسالة الخطأ ثم افتح الضحية بOllyDbg وابحث عن رسالة الخطأ.



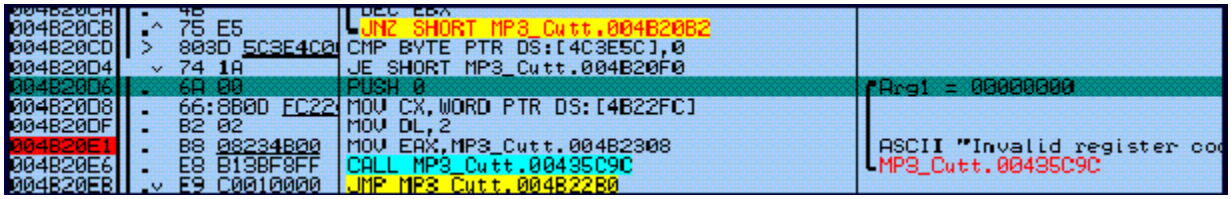
right click==>all referenced text strings

press Home==>right click==>search for text

وستصل إلى هنا



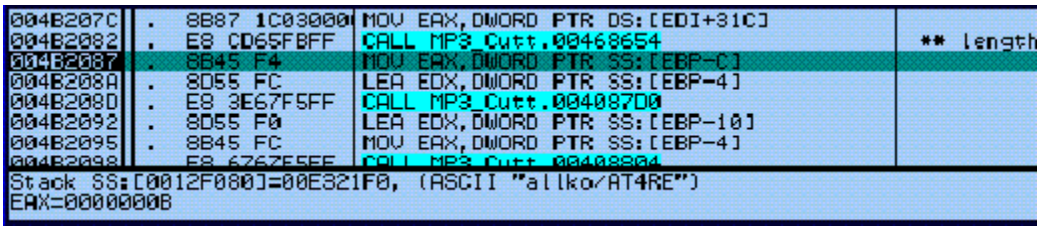
ضع نقطة توقف عليها ثم اضغط انتر



والآن اضعد للأعلى حتى تجد بداية هذه الـ procedure

ضح bp كما هو واضح. شغل الضحية واضغط على register وادخل اليوزرنيم allko/AT4RE و السريال 123456789 ثم register

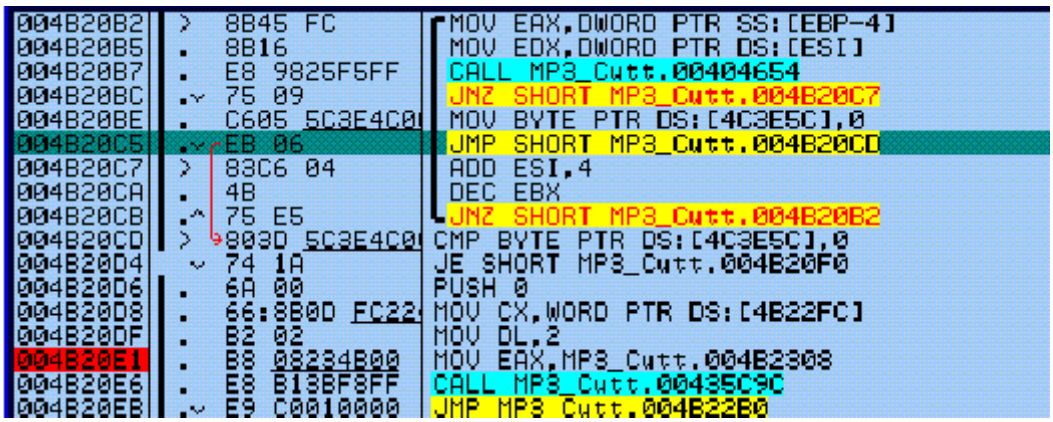
ها قد وصلنا إلى بداية الكود. تتبع حتى تصل إلى أول دالة وتجاوزها (F8) وانظر ما الذي حصل.



كما ترى بالصورة فقد أصبح eax=0B وهو طول اليوزرنيم. ويؤكد ذلك رؤيتنا لليوزر في الأسفل. إذن للتسهيل اضغط على زر "ك" على لوحة المفاتيح وقم بكتابة تعليق (comment). نحن كتبنا length وذلك سيسهل و على العمل فإذا ما أغلقت البرنامج وعدت إلى هنا سأرى التعليق وأعرف وظيفة هذه الدالة.

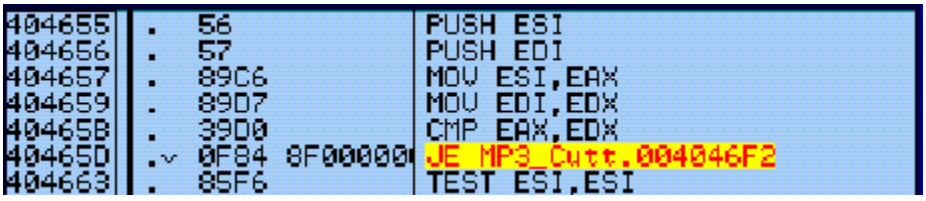
و الآن أكمل التتبع وتجاوز الدالة الثانية وانظر ما الذي سيحصل بعدها مباشرة.. لا شيء مهم. لا توجد قيم مميزة بالمسجلات. أذن تابع وتجاوز الدالة الثالثة. لا يبدو أن هناك شيئا مهما مع أنه قبل تنفيذ تلك الدالة كان eax يشير إلى اليوزرنيم. لا بأس فلنتابع. ألدالة الرابعة بعد تنفيذها نلاحظ أن eax=0.قمنا بكتابة تعليق بجانبها may be important.

دعك منها مؤقتا ولنتابع. وصلنا إلى حلقة :

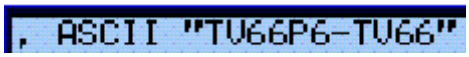


كما ترى يلي الحلقة أمر مقارنة ثم قفزة، والقفزة تتجاوز الـ bad_boy_msg إذن هذه الحلقة مهمة وعلينا أن نعرف ما بداخلها. تتبع حتى تصل إلى التعليمة الثانية في الحلقة وانظر ماذا يوجد في قسم التعليقات السفلي :

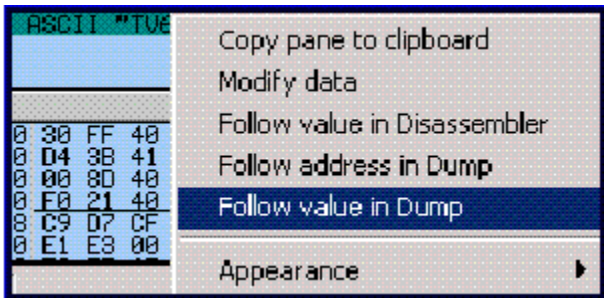
هذه الـ string غريبة. مممم. لا بأس تابع وادخل إلى داخل الدالة التي تلي هذه التعليمة مباشرة :



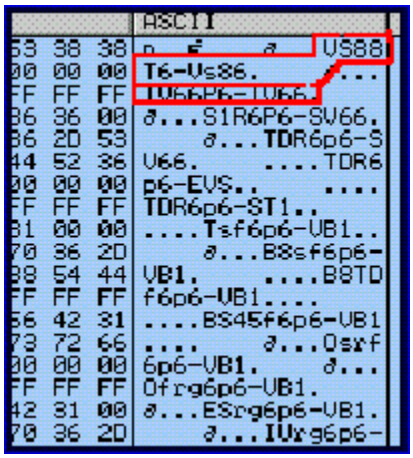
ها نحن بداخلها.. أعتقد أن الأمور واضحة فعند الوصول إلى **CMP** eax,edx ستلاحظ أن المسجلين بهما اليوزر المدخل والآخر به تلك السترنج الغريبة.. فلنتابع.. بالطبع ستخرج من الدالة لأنك لم تدخل اليوزر المطلوب. لكن بعد الخروج عدنا مجددا..تتبع حتى تصل إلى الأمر الثاني في التعليمة وللنظر إلى قسم التعليقات السفلي :



سترنج أخرى !! دعنا نضغط باليمين في قسم التعليقات :



ولنرى ماذا يوجد هناك :

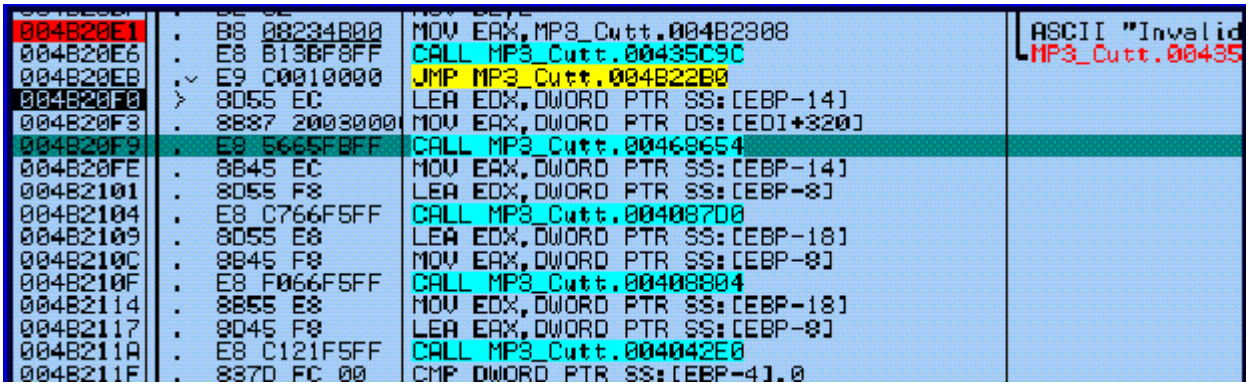


كما ترى بالصورة، في المربع الأحمر الأول هناك السترنج الأولى يليه الستنرج الثانية. لو أكمل تتبع الحلقة ستلاحظ أن البرنامج سيقارن اليوزرنيم المدخل مع 21 يوزرنيم مخزنين لديه.. لا بأس إذن علينا تغيير اليوزرنيم إلى أحد هذه الـ 21 اسما. اعد التشغيل وادخل TV66P6-TV66 مثلا والسريال 123456789. اضغط register وسنعود للحلقة تلك، لكن هذه المرة سننجح في تجاوزها

في الدورة الثانية (لأن اليوزر الذي أدخلناه ترتيبه 2 من بين الـ 21 يوزر).

لا مانع من وضع تعليق بجانب هذه الحلقة، كـ is ur name valid ? هكذا نعرف ان هذه الحلقة مسؤولة عن التحقق من اليوزرنيم.

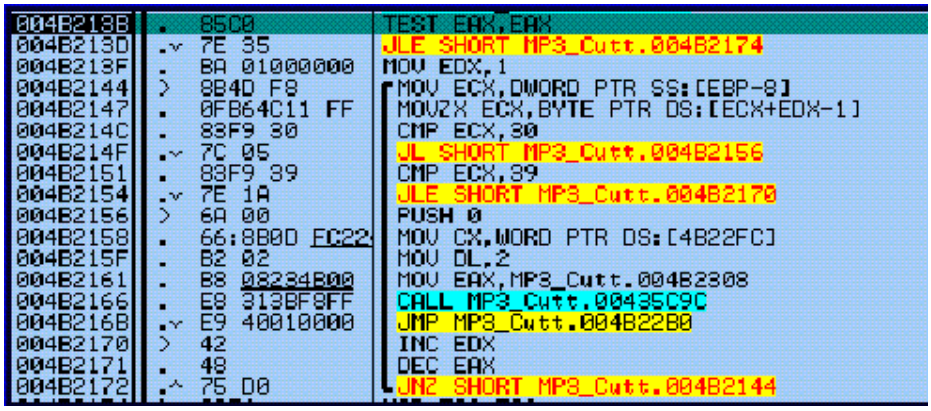
إذن نحن هنا :



كما ترى هناك 4 دوال. سنفعل نفس الشيء فلنتخطى الأول وننظر النتيجة.هل لاحظت؟ eax=9 وفي الأسفل مكتوب 123456789

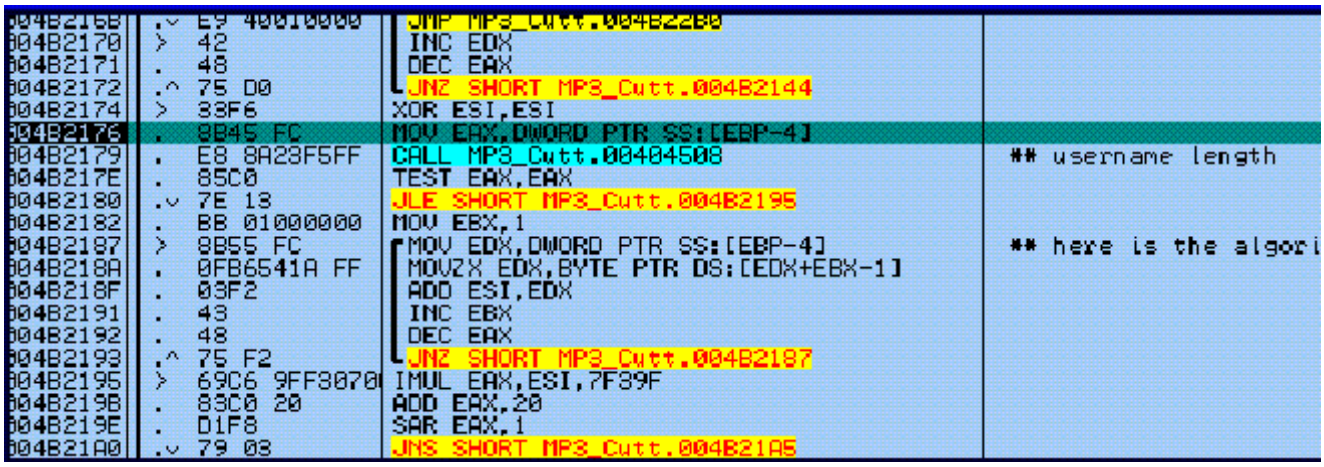
إذن هذه خاصة بحساب طول السريال. تجاوز الثانية والثالثة (غير مهمتين)..تجاوز الرابعة وسترى بعدها قفزتين. يبدوا أنهما تريدان التأكد أننا فعلا أدخلنا يوزرنيم وباسوورد.تتبع. ألدالة الخامسة تقوم بوضع طول السريال في eax.

نحن هنا :



القفزة العلوية لن تنفذ إلا في حال لم تدخل أي سريال. ألحلقة التي تليها تقوم بالتأكد من أن السريال مكون من أرقام فقط.. لا حاجة لتتبعها بالكامل فهي واضحة..هناك مقارنة مع 30هكس يعني الرقم 0 ومن ثم مع 39 هكس يعني الرقم 9.

نحن هنا :

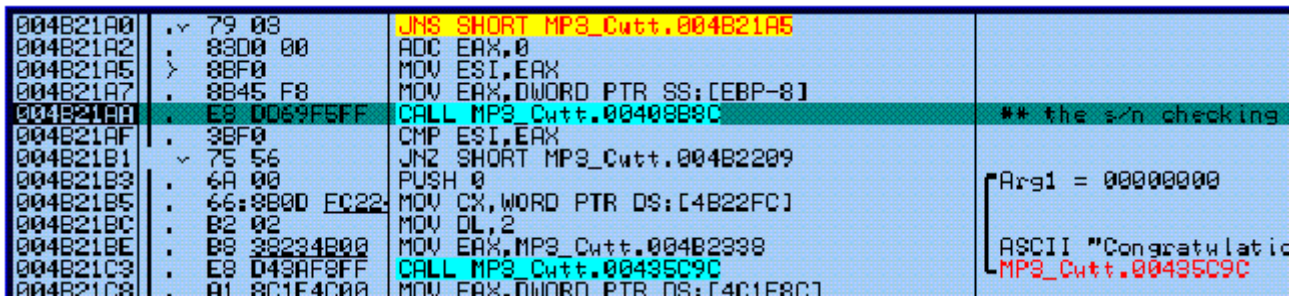


كما تلاحظ الدالة التي تظهر في الصورة هي لحساب طول اليوزرنيم.. ثم هناك حلقة..هذه الحلقة مهمة جدا فهي تقوم بتوليد قيمة من اليوزرنيم وآلية عملها كالتالي :

أول حرف هو T قيمته54 تنقل إلى ESI، ثاني حرف V قيمته 56 تضاف إلى ESI فيصبح المجموع AA0 وهكذا مع باقي أحرف الوزر.

يلي ذلك عملية ضرب IMUL EAX,ESI,7F39F حيث ستضرب القيمة الناتجة بـ F39F7 وتخزن النتيجة في EAX ثم في السطر التالي تضاف القيمة 20، ثم هناك SAR EX,1 (راجع دروس الأسمبلي لتفهمها) فنحصل على النتيجة النهائية B6A3AD00 ثم هناك قفزة JNS أي Jump if NOT Sign أي اقفز اذا كانت النتيجة موجبة. وواضح أنها موجبة. في الواقع في حالة ال 21 اسم دائما النتيجة موجبة، لذا فالتعليمة ADC EAX,0 يمكن تجاهلها.

سنصل إلى هنا :



كما ترى عند الوصول إلى الدالة التي عند 004B21AA لو نظرنا إلى قسم التعليقات السفلي ستجد السريال المعدل وستجد B6A3AD00 أي أن العملية النهائية تتم هنا. لذا دعونا نرى ما يوجد.

```
00408B8C  r$  53              PUSH EBX
00408B8D  .   56              PUSH ESI
00408B8E  .   83C4 F4         ADD ESP,-0C
00408B91  .   8BD8            MOV EBX,EAX
00408B93  .   8BD4            MOV EDX,ESP
00408B95  .   8BC3            MOV EAX,EBX
00408B97  .   E8 38A2FFFF     CALL MP3_Cutt.00402DD4
00408B9C  .   8BF0            MOV ESI,EAX
00408B9E  .   833C24 00       CMP DWORD PTR SS:[ESP],0
00408BA2  .v  74 19           JE SHORT MP3_Cutt.00408BBD
00408BA4  .   895C24 04       MOV DWORD PTR SS:[ESP+4],EBX
00408BA8  .   C64424 08 0B    MOV BYTE PTR SS:[ESP+8],0B
00408BAD  .   8D5424 04       LEA EDX,DWORD PTR SS:[ESP+4]
00408BB1  .   A1 B81E4C00     MOV EAX,DWORD PTR DS:[4C1EB8]
00408BB6  .   33C9            XOR ECX,ECX
00408BB8  .   E8 CBF8FFFF     CALL MP3_Cutt.00408488
00408BBD  >   8BC6            MOV EAX,ESI
00408BBF  .   83C4 0C         ADD ESP,0C
00408BC2  .   5E              POP ESI
00408BC3  .   5B              POP EBX
00408BC4  L.  C3              RETN
```

فلتتبع باستخدام F7 لأنه لا مجال لترك أي دالة فهنا تكمن الحماية. بعد الوصول إلى الدالة الأولى الظاهرة بالصورة والدخول إلى داخلها.

ستصل إلى هنا :

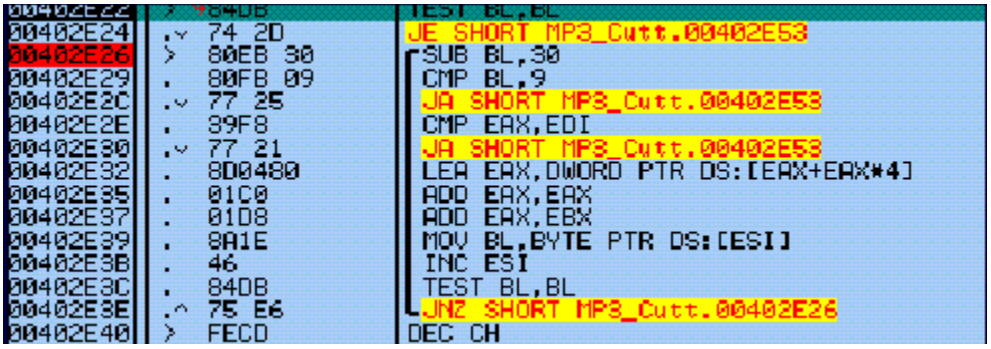
```
00402DD4  r$  53              PUSH EBX
00402DD5  .   56              PUSH ESI
00402DD6  .   57              PUSH EDI
00402DD7  .   89C6            MOV ESI,EAX
00402DD9  .   50              PUSH EAX
00402DDA  .   85C0            TEST EAX,EAX
00402DDC  .v  74 6C           JE SHORT MP3_Cutt.00402E4A
00402DDE  .   31C0            XOR EAX,EAX
00402DE0  .   31DB            XOR EBX,EBX
00402DE2  .   BF CCCCCC0C     MOV EDI,0CCCCCCC
00402DE7  >   8A1E            MOV BL,BYTE PTR DS:[ESI]
00402DE9  .   46              INC ESI
00402DEA  .   80FB 20         CMP BL,20
00402DED  .^  74 F8           JE SHORT MP3_Cutt.00402DE7
00402DEF  .   B5 00           MOV CH,0
00402DF1      80FB 2D         CMP BL,2D
```

إذن لنقم بالتتبع ببطء. الحلقة الظاهرة في الصورة لا يبدوا أنها ذات أهمية. فهي تقارن أول رقم في السريال مع القيمة 20 هكس. لا نجد معنى لذلك. لذا فلنتابع..

يلي الحلقة العديد من المقارنات والقفزات التي لا أهمية لها. فالسريال يفترض أن يكون أرقاما ولا فائدة من مقارنة الرقم الأول منه مع قيم مثلا 20 أو B2 أو 24.

المقارنة الأخيرة هي مع القيمة 30 أي صفر. وحيث أن السريال الذي أدخلناه يبدأ بـ 1 إذن فـ **JNZ** ستنفذ ونذهب إلى حلقة.

لو لاحظت، ففي حالة كان أول رقم هو 0 فسيتم عمل بعض المقارنات الغير مهمة ثم نذهب إلى الحلقة ذاتها. إذن دعنا نرى ما يوجد بهذه الحلقة :



أها. أرى بعض التعليمات المهمة,,,الأولى تطرح 30 من القيمة الأولى في السريال لنحصل على 31-30=1 ثم هناك مقارنة : هل النتيجة أكبر من 9؟ بالطبع لن تكون لأن السريال مكون من أرقام فقط. المقارنة التالية **CMP** eax,edi أيضا لا أهمية لها. (لأنك لو نظرت إلى العنوان 402DDE فستجد **XOR** eax, eax وبعده عند 402DE2 هناك **MOV** edi,0CCCCCCC وبالتالي القفزة التي تلي أمر المقارنة **JA**

من المستحيل أن تنفذ لأن eax لن يكون أكبر من edi لنتابع.

هناك أمر LEA. يقوم بنقل قيمة ما إلى eax.

هذا الامر بهذه الصيغة ليس كما تعودنا عليه. يمكنك فهمه على أنه سيأخذ القيمة التي تساوي EAX+EAX*4 وينقلها إلى EAX.

في الدورة الحالية من الحلقة، تلك القيمة تساوي صفر. فلنكمل. الأمر التالي add eax, eax يضاعف eax ثم هناك add eax,ebx. لاحظ أن ebx به قيمة الرقم (الخانة) الحالية بعدما طرحنا منه 30. الأمر التالي ينقل الرقم الثاني إلى bl أي أن bl=32. يتم زيادة العداد esi بمقدار واحد. بعدها هناك تعليمة **TEST** ebx, ebx والتي تعني : هل bl صفر؟ أي هل انتهينا من جميع الأرقام؟ لم ننته إذن سنعيد الحلقة من جديد.

الآن bl=32 ثم سيصبح bl=2. تعليمة [LEA EAX DWORD PTR DS:[EAX+EAX*4 في هذه الدورة ستقوم بنقل 4*1+1 إلى eax. لن نكمل فالمسألة أصبحت واضحة. في نهاية الحلقة ستكون قيمة eax=075BCD15.

بعد الحلقة هناك تعليمتان أو ثلاث، يتم تنفيذهم ثم نقفز إلى نهاية الدالة.حيث يتم استرجاع قيم المسجلات من المكدس إلى ما كانت عليه قبل تنفيذ الدالة.. وبعد العودة سنكون هنا :

```
00408B8C  /$  53            PUSH EBX
00408B8D  .   56            PUSH ESI
00408B8E  .   83C4 F4       ADD ESP,-0C
00408B91  .   8BD8          MOV EBX,EAX
00408B93  .   8BD4          MOV EDX,ESP
00408B95  .   8BC3          MOV EAX,EBX
00408B97  .   E8 38A2FFFF   CALL MP3_Cutt.00402DD4
00408B9C  .   8BF0          MOV ESI,EAX
00408B9E  .   833C24 00     CMP DWORD PTR SS:[ESP],0
00408BA2  .v  74 19         JE SHORT MP3_Cutt.00408BBD
00408BA4  .   895C24 04     MOV DWORD PTR SS:[ESP+4],EBX
00408BA8  .   C64424 08 0B  MOV BYTE PTR SS:[ESP+8],0B
00408BAD  .   8D5424 04     LEA EDX,DWORD PTR SS:[ESP+4]
00408BB1  .   A1 B81E4C00   MOV EAX,DWORD PTR DS:[4C1EB8]
00408BB6  .   33C9          XOR ECX,ECX
00408BB8  .   E8 CBF8FFFF   CALL MP3_Cutt.00408488
00408BBD  >   8BC6          MOV EAX,ESI
00408BBF  .   83C4 0C       ADD ESP,0C
00408BC2  .   5E            POP ESI
00408BC3  .   5B            POP EBX
00408BC4  L.  C3            RETN
```

كما ترى هناك عملية مقارنة مع 0 يليها قفزة، لكن لا تقلق ليس هذا ما نريده.تابع حتى تصل إلى **RETN**

```
04B21A7  .   8B45 F8        MOV EAX,DWORD PTR SS:[EBP-8]
04B21AA  .   E8 DD69F5FF    CALL MP3_Cutt.00408B8C          ** the s/n checking is
04B21AF  .   3BF0           CMP ESI,EAX
04B21B1  .v  75 56          JNZ SHORT MP3_Cutt.004B2209
04B21B3  .   6A 00          PUSH 0                          Arg1 = 00000000
04B21B5  .   66:8B0D EC22   MOV CX,WORD PTR DS:[4B22FC]
04B21BC  .   B2 02          MOV DL,2
04B21BE  .   B8 38234B00    MOV EAX,MP3_Cutt.004B2338       ASCII "Congratulation!
04B21C3  .   E8 D43AF8FF    CALL MP3_Cutt.00485C9C          MP3_Cutt.00485C9C
04B21C8  .   A1 8C1F4C00    MOV EAX,DWORD PTR DS:[4C1F8C]
04B21CD  .   C600 01        MOV BYTE PTR DS:[EAX],1
```

نحن أمام امر مقارنة ولو نظرت إلى قيمة esi لوجدتها B6A3AD00 أما eax=75BCD15 أعتقد أن كل شيء أصبح واضحا.

Username (TV66P6-TV66) _ algorithm 1 _ value1 (0B6A3AD0)
Serial (123456789) _ algorithm2 _ value2 (075BCD15)

ثم يتم المقارنة بين القيمتين.

كما تلاحظ فالضحية هذه لا تختلف كثيرا عن BRUTE_ME التي كسرناها قبل هذا المثال. الـ BRUTE_ME تقوم بتطبيق الخوارزمية على السريال ثم تقارن النتيجة مع 70هكس، أما هذه تقوم بتطبيق الخوارزمية على السريال لكنها تقارنها بقيمة متغيرة وليست ثابتة. هذه القيمة المتغيرة نحصل عليها بتطبيق خوارزمية أخرى على اليوزرنيم.

والآن سنقوم بعمل الكيجين. دعني أوضح لك الأجزاء التي يتكون منها :

1- الجزء الأول خاص باستقبال اليوزرنيم من المستخدم . حيث أن اليوزرنيم هو واحد من بين 21 يوزنيم مخزن... إذن لدينا خيارين :

- أن نقوم بعمل قائمة مندسلة (أم منسدلة ؟ :)) فيها الـ 21 يوزر وعلى المستخدم الاختيار فيما بينها.
- أن نقوم بعمل edit_box وبجانبه زر . في البداية يكون اليوزر رقم 1 موجود كقيمة افتراضية في الـ edit_box . عند الضغط على الزر هناك عداد سنزيد قيمته من 1 إلى 2 فيتم استدعاء دالة SetDlgItemTextA التي ستظهر اليوزر رقم 2 ، وعند الضغط مرة أخرى فسيزيد العداد ويصبح 3 ويتم استدعاء نفس الدالة لتظهر اليوزر 3 وهكذا.

وأنا أفضل الخيار الثاني لأن الأول لم يسبق لي أن جربته...

2- بعد الحصول على اليوزر سنطبق algorithm1 عليه لنحصل على value1

3- الجزء الثالث هو العداد لنحصل على سيريال ما . هذا الجزء لا يتغير من Bruter لآخر .

4- سنقوم بتطبيق algorithm2 على السيريال الذي حصلنا عليه من العداد لنحصل على value2 ثم نقارن value1 مع value2 فإن تطابقتا فقد انتهينا وإلا سنعود للعداد لنجرب سيريال آخر.

5- لاحظ أن أول سطر في البرنامج هو **include** gg.**INC** حيث أنه بسبب كثرة المتغيرات فقد قمنا بوضعها في ملف باسم gg.**INC** بدلا من وضع كل شيء في ملف asm.

6- لاحظ أن الدالة المستخدمة هنا algorithm2 أكثر تعقيدا من تلك المستخدمة في الضحية السابقة لذا ستحتاج إلى وقت أطول لإيجاد السيريال الصحيح. كما أن كل يوزر لن تجد إلا سيريال واحد فقط يناسبه لذا بعد إيجاد السريال الأول هناك تعليمة **ret** وليس jmp next_pass

الشكل النهائي :

ستجد الـ Bruter  هذا في المرفقات باسم Bruter-Mp3

وبهذا ينتهي الفصل الثامن بفضل الله.


الباب
الثاني
الفصل التاسع
Miscellaneous

9. الفصل التاسع : SERIAL SNIFFING & MUSIC EXTRACTING

سنتحدث في هذا الدرس عن فكرة الـ serial sniffing والتي تعني "قنص" السريال من البرنامج. أنا متأكد أنك قد حاولت أكثر من مرة كسر برنامج ما، وأثناء تنقيحك له رأيت أحد المسجلات registers يشير الى عنوان تخزين السريال الصحيح.حينها كنت تأخذ هذا السريال وينتهي الأمر. ألفكرة في serial sniffer هي أن نقوم بقنص ذلك السريال أي أن نشغل البرنامج حتى اذا ما وصل لتلك النقطة، أخذنا السريال وانتهى الأمر.

تجد الضحية في المرفقات باسم ch2_sec9_1

1.9 . الدوال المستخدمة : -

CreateProcess : لفتح البرنامج الضحية

ReadProcessMemory : لقراءة البايتات الأساسية و التي سنعدل عليها مؤقتاً لعمل دورة لا نهائية

WriteProcessMemory : لكتابة بايتات الدورة اللانهائية EBFE

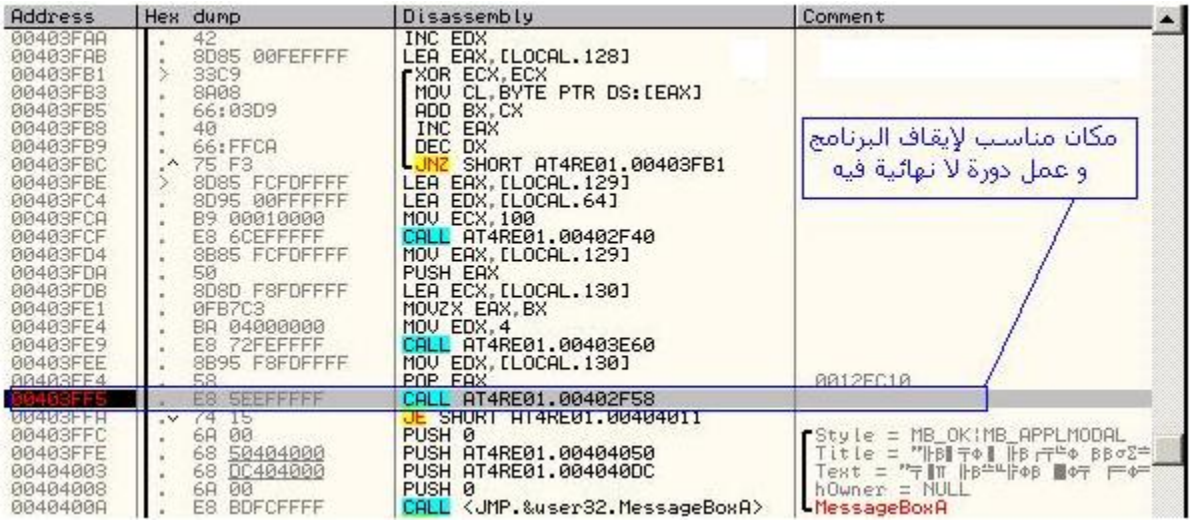
GetThreadContext : قلب البرنامج – دالة تجلب عناصر التنقيح من ... eax,ebx,eip

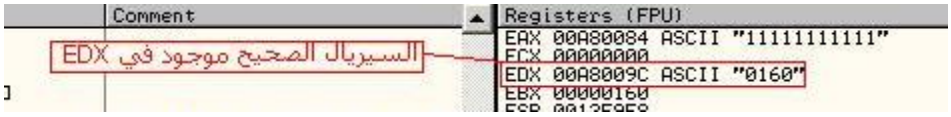
SuspendThread : سنستخدمها لتجميد البرنامج عند الوصول إلى السطر الصحيح

2.9 . دوال أضافية :-

MessageBox : بدلا من تخزين السيريال في ملف يمكنك إظهاره في رسالة

فكرة البرنامج. أنظر معي الى هذه الصورة للضحية في OllyDbg :





حسناً , الفكرة هي إيقاف البرنامج Suspend في سطر معين (00403FF5) يحمل احد مسجلاته (EDX) السيريال الصحيح أو الكود بعد ذلك يتم قنص السيريال من المسجل الذي يحتويه (EDX) , ثم إظهاره في MessageBox أو أي شكل أخر تريده.

لجعل البرنامج لا يتجاوز السطر 00403FF5 فإننا نقوم بتعديل أول 2 بايت فيه إلى EBFE و هي تعمل دوره لانهائية في نفس السطر.

شرح الكود :

.386 .model flat, stdcall option casemap :none include windows.INC include user32.INC include kernel32.INC includelib user32.lib includelib kernel32.lib	هذا الجزء لا يحتاج لشرح.
.data AppName db "AT4RE01.exe",0 PINFO PROCESS_INFORMATION<> SINFO STARTUPINFO<> cntx CONTEXT <> align dword	هنا نعرف اسم البرنامج الضحية، بالإضافة إلى بعض المتغيرات اللازمة للدوال المستخدمة.
.data? OriginalBytes dword ? BToWrite dword ? dup(?) Serial db 255	هنا نخزن البايتات الأصلية. هنا نخزن عدد البايتات الفعلي. نحجز 255 بايت لوضع السيريال الصحيح فيها
.const InfiniteLoop db 0EBh,0FEh	لبايتات التي سنقوم بوضعها في سطر ظهور السيريال الصحيح و هي تحدث دورة لانهائية في نفس المكان.
.code start:	لا تعليق.
invoke CreateProcess,addr AppName,0,0,0,0,CREATE_SUSPENDED,0,0,addr SINFO,addr PINFO	نقوم بتشغيل البرنامج في وضع توقف suspend
invoke ReadProcessMemory,PINFO.hProcess,403FF5h,addr OriginalBytes,2,0	نقرأ أول 2 بايت في عنوان ظهور السيريال الصحيح و نخزنه في أحد المتغيرات.
invoke WriteProcessMemory,PINFO.hProcess,403FF5h,addr	نقوم بعد ذلك بتغيير الـ2 بايت إلى EBFE

InfiniteLoop,2,0	لعمل دورة لانهائية Infinite Loop.
invoke ResumeThread,PINFO.hThread	نسمح للبرنامج بالمتابعة
MOV cntx.ContextFlags,CONTEXT_FULL	نعطي هنا تعريف للـ Context Flags كـ CONTEXT_FULL و ذلك كي يحتوي الـ cntx على جميع عناصر التنقيح
GetContextLoop:	*
invoke GetThreadContext,PINFO.hThread,addr cntx	نجلب عناصر التنقيح لـ thread معينة و نخزنها في Context Structure (cntx)
CMP cntx.regEip,403FF5h	يتم مقارنة الـ Eip مع العنوان المناسب
JE SniffSerial	إذا تساويا نقفز إلى SniffSerial
jmp GetContextLoop	إذا لم يتساويا نقفز إلى أول الدورة (GetContextLoop) مرة أخرى
SniffSerial:	عند هذه النقطة نكون في المكان الصحيح و يتم اقتناص السيريال الصحيح من EDX وتخزينه في متغير ثم عرضه في رسالة
invoke SuspendThread,PINFO.hThread	نجمد البرنامج الضحية مؤقتاً
invoke ReadProcessMemory,PINFO.hProcess, cntx.regEdx,addr Serial,4, addr BToWrite	نقرأ قيمة EDX من البرنامج الضحية و نخزنه في بفر(Serial)
invoke MessageBox,PINFO.hProcess,addr Serial,addr AppName,MB_ICONEXCLAMATION	نعرض السيريال الصحيح في رسالة
invoke WriteProcessMemory,PINFO.hProcess,403FF5h,addr OriginalBytes,2,0	نقوم بكتابة الـ2 بايت الأصلي الذي غيرناهما إلى EBFE
invoke ResumeThread,PINFO.hThread	نتابع عمل البرنامج الضحية
invoke ExitProcess,0	
end start	

* : في هذه الدورة المشروطة يتم جلب عناصر التنقيح , ثم التأكد إذا ما كان البرنامج قد وصل إلى السطر الذي نريده(403FF5) و ذلك عن طريق مقارنة EIP مع العنوان المناسب , فإذا تساوى العنوان مع EIP نكون في المكان الصحيح و يقفز الـSerial Sniffer إلى إجراءات عرض السيريال الصحيح (SniffSerial) , غير ذلك يتم القفز إلى أول الدورة من جديد.

3.9 . استخراج الموسيقى من الملفات التنفيذية (EXE)

سنتحدث اليوم عن كيفية استخراج الموسيقى من الملفات التنفيذية (.exe) وخاصة الـ keygens وذلك باستخدام LordPE بشكل أساسي. أتمنى أن يكون الدرس مفيدا وممتعا.

بداية أود أن أشير أن اغلب الـ keygens تحوي الموسيقى بصيغة xm. والبعض بصيغة mod وهاتان الصيغتان تتميزان بصغر حجمهما. وغالبا يكون الملف الصوتي مضافا بهيئة resource مثله مثل الـ icons,images,dialogs,etc وجميعها تضاف إلى قسم.rsrc من ملف الـ PE.

1.3.9 . لمحة تاريخية :

ملفات MOD وXM تم إنشاؤها بطريقة خاصة باستخدام مزيج من الـ samples و الـ patterns للـ note data(النوتة الموسيقية).

هذه الطريقة من توزيع الموسيقى تسمى tracker music. نسبة إلى البرامج التي تنشئ هذه الملفات الموسيقية مثل : (Soundtracker, Noisetracker, Protracker, Fasttracker, Impulsetracker,

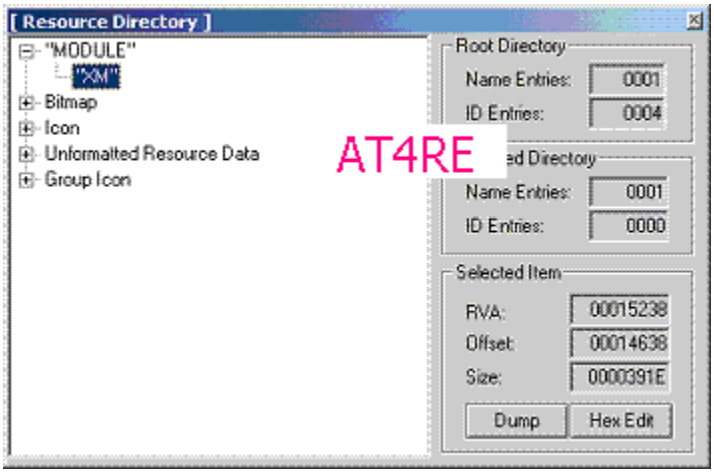
Modplug Tracker, etc) يرجع تاريخ هذه الموسيقى إلى حقبة الثمانينيات من القرن الماضي,,,واليوم فان الـ tracker music تستخدم غالبا في الألعاب وبعض تطبيقات الأجهزة الكفية المحمولة (وبالتأكيد الـ keygens :)). مما يميز هذه الملفات صغر حجمها وبالتالي فهي توفر في حجم الملف وفي الذاكرة المستخدمة من قبل البرنامج.

ملفات MOD تحوي 4 قنوات موسيقية audio channels من فئة 8-bit sound. بينما الـ XM (Fasttracker) يمكن أن تحتوي عددا اكبر من القنوات الموسيقية ومن فئة 16-bit، لذالك ملفات XM اكبر حجما من ملفات MOD.

يمكن تحويل كلا النوعين (XM و MOD) إلى صيغة.(Impulse Tracker) IT باستخدام برامج خاصة كـ ModPlug Tracker.

9.3.2 . الاستخراج:

بداية افحص الملف المرفق واسمه ch2_sec9_2 لترى انه مضغوط. أذن فك الضغط عنه مستخدما أي برنامج لفك الضغط. والآن افتح الملف الناتج باستخدام LordPE واختر Directories من القائمة اليمنى. الآن اضغط ... بجانب Resources لترى التالي

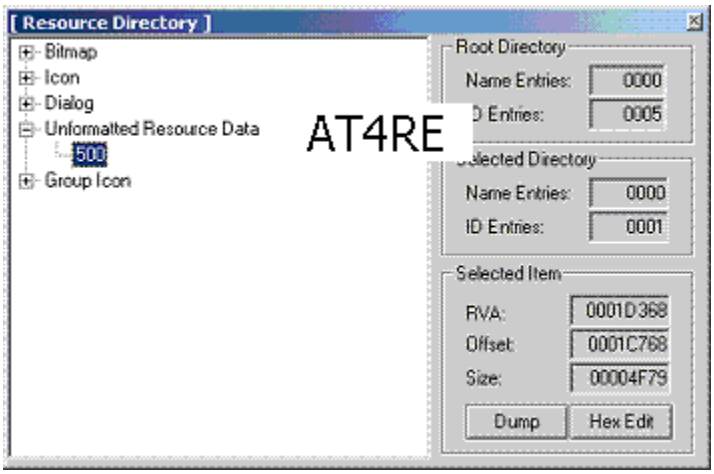


كما ترى فان أول قائمة بها الملف XM. اختر Hex Dump لترى التالي :

```
I.C.O.N.Extended
 Module: STRANGL
EHOLD (C)JT  .Fa
stTracker v2.00
```

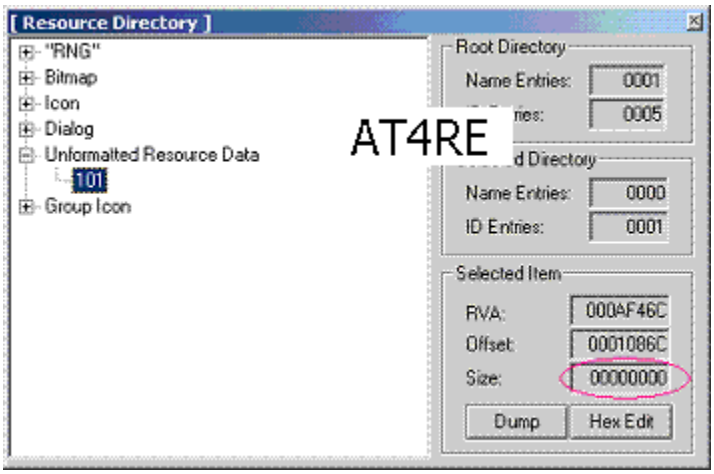
كما تلاحظ فإن جميع ملفات XM تبدأ بـ Extended Module. أغلق النافذة ثم اختر Dump والآن احفظ الملف بأي اسم بامتداد.xm شغل الملف مستخدما برنامج XMplay. ها قد انتهينا!!

يمكن تطبيق هذه الخطوات على الملفات المرفقة رقم 2 و 3 و 4 لكن لاحظ في 3 انك ستر التالي :

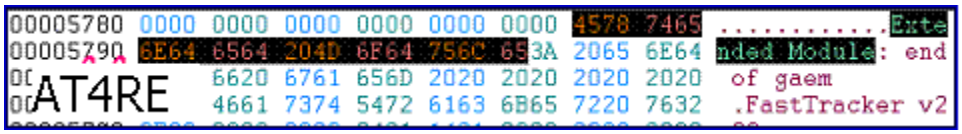


لاحظ عدم وجود قائمة Module بها ملف xm كالسابق. لا باس اختر الملف الموجود في قائمة Unformatted Resource Data ثم اختر HexDump. هل رأيت Extended Module؟ هذا معناه أن هذا الملف عبارة عن ملف موسيقي من نوع xm. الآن كرر نفس الخطوات السابقة.

والآن دعنا نرى الملف رقم 5. فك الضغط عنه ثم افتحه بـ LordPE لترى التالي :



Unformatted Resource Data مرة أخرى. لكن هذه المرة الملف حجمه صفر كما تلاحظ. أذن ما العمل؟ افتح البرنامج باستخدام HexWorkshop والآن اضغط باليمين في أي مكان واختر Find وفي نافذة البحث اكتب Extended Module ثم انتر لترى التالي



والآن عرفنا بداية الملف. أما نهايته فهذه غير واضحة تماما لكن ضع في ذهنك أن الملف الصوتي سوف يفصله عما بعده مجموعة من الأصفار... ألآن انظر إلى التالي:

انظر إلى ما بعد تلك الأصفار.هل يبدو لك وكأنه تابع للملف الصوتي؟ بالتأكيد لا!! يبدوا وكأنه تفاصيل خاصة بالملف نفسه(الكيجن). إذن فان الملف الصوتي ينتهي عند تلك النقطة.

إذن حدد من بداية حرف E إلى تلك النقطة و الآن اضغط باليمين على التحديد واختر copy. الآن من قائمة File اختر new ثم الآن اضغط باليمين واختر Paste ثم ok والآن احفظ الملف بأي اسم بامتداد xm واستمتع بالموسيقى.

وبهذا ينتهي الفصل التاسع.


الباب
الثاني
الفصل العاشر
Reversing 'not native' Languges

10. الفصل العاشر : REVERSING "NOT NATIVE" LANGUAGES

هذا الدرس خاص بطريقة كسر تطبيقات الـ NET وكما تعلمون فان أي برنامج مبرمج بأحد لغات الـ NET فأن طريقة كسرة تكون مختلفة عن البرامج الأخرى كما أن هذه النوعية لا يمكن كسرها ببرنامج Olly. لكن طريقة كسرها سهلة جدا وستري ذلك.

10.1 . المثال الاول : DOTNET REVERSING

الضحية التي سنتعامل معها عبارة عن Crack ME تجده بالمرفقات باسم ch2_sec10_1 .

طبعا يجب ان يكون لديك فريموورك الدوودنيت حتى تستطيع تشغيل تطبيقات dot net. dot Net Framework

على الجهاز. يمكن تحميله من موقع مايكروسوفت.

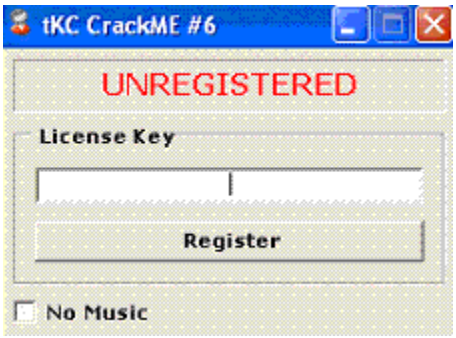
البرنامج الذي سوف نستخدمه هو برنامج Lutz Roeder's.NET Reflector v5.0.25.0

وهو برنامج مخصص للتعامل مع تطبيقات ال.NET بكل أنواعها ويمكن تحميله من موقع البرنامج الرسمي

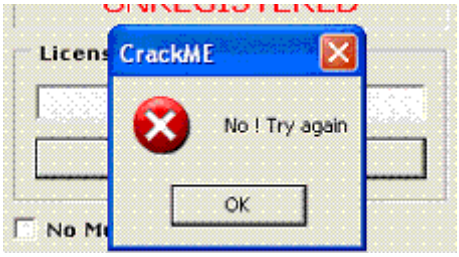
http://www.aisto.com/roeder/dotnet/

نبدأ بفحص البرنامج كالعادة بواسطة PEID

نقوم الآن باستكشاف البرنامج. شغل البرنامج الضحية

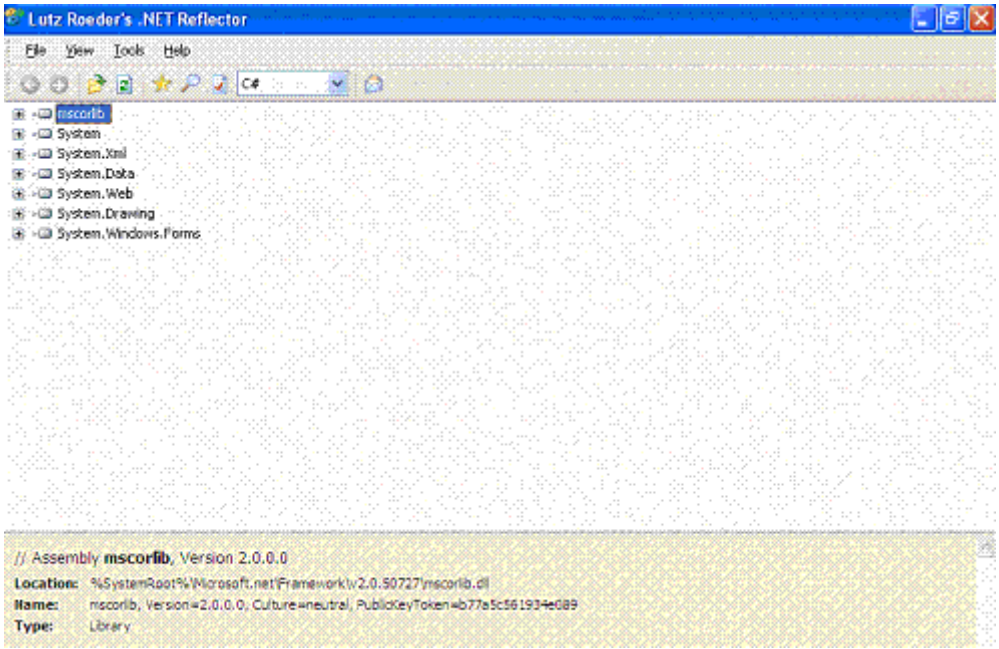


الآن ادخل أي سيريال نمبر لنري الرسالة التي سوف تظهر ادخل مثلا 1234567890

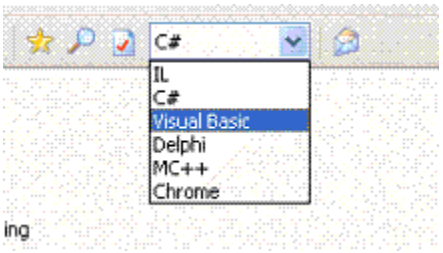


كسر البرنامج :

شغل برنامج الـ Reflector

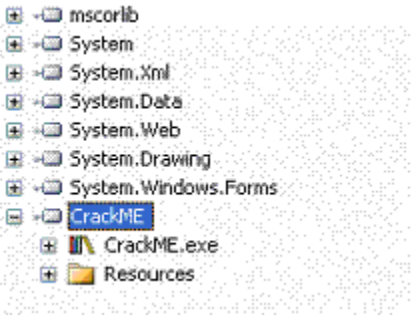


اختار اللغة Visual Basic

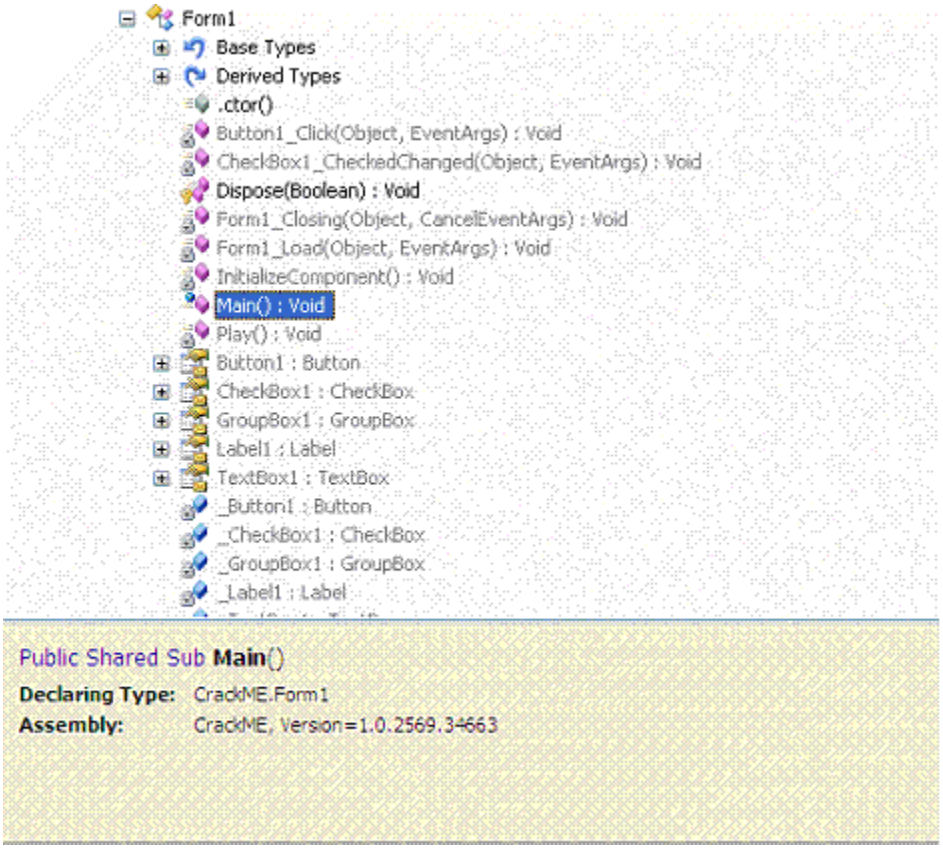


الآن حمل البرنامج الضحية. ستلاحظ ظهور البرنامج الضحية في القائمة باسم Crack ME

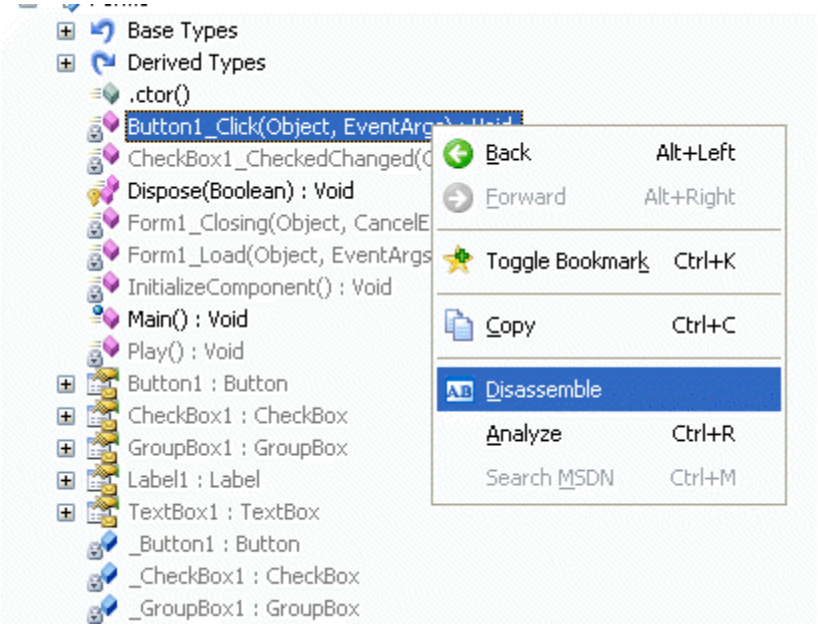
أثناء عمل هذا الدرس كانت الضحية مسماة باسم CrackMe وليس ch2_sec10_1 لكن لا فرق المهم الفكرة.



اضغط و على CrackMe ثم Right Click ثم اختار Go to Entry Point



اضغط و على Button1_Click(Object, EventArgs) ثم

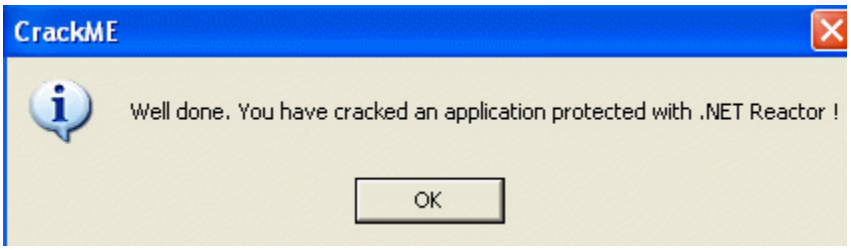


الآن ستظهر المفاجأة

```
Private Sub Button1_Click(ByVal sender As Object, ByVal e As EventArgs)
   If (StringType.StrCmp(Me.TextBox1.Text, "PPLKI965D2MH93KHOPLBT", False) = 0) Then
      Me.Label1.Text = "Registered"
      Interaction.MsgBox("Well done. You have cracked an application protected with .NET Reactor !", &H40, Nothing)
   Else
      Me.Label1.Text = "UNREGISTERED"
      Interaction.MsgBox("No ! Try again", &H10, Nothing)
   End If
End Sub
```

ستلاحظ أن كل شيء نحتاجه قد ظهر. رسالة الخطأ ورسالة الصواب و السريال نمبر الصحيح !!! إذا فان السريال نمبر الصحيح هو **PPLKI965D2MH93KHOPLBT**

لنجرب الآن هذا السريال لنتأكد:



مبارك لقد نجحنا.

10.2 . المثال الثاني: VISUAL BASIC REVERSING

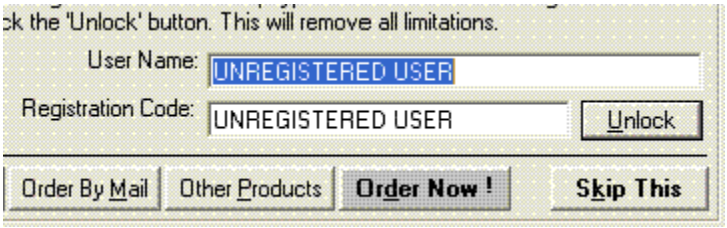
في هذا الدرس سنتحدث عن كسر برنامج تجاري مبرج بالفيجوال بيسك واسم البرنامج هو Talk 2Desktop V1.0 .
رابط تحميل البرنامج

 http://www.4developers.com/software/talk2d.exe 

وهو يستخدم في أعطاء اوامر للكمبيوتر عن طريق المايك.

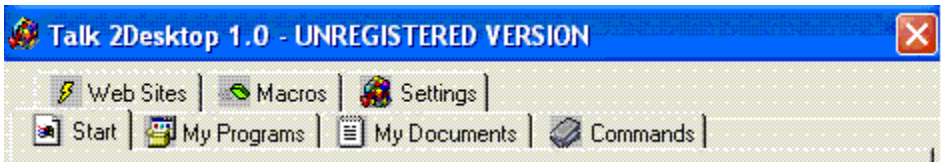
وسوف نقوم باستعمال طريقة الباتشينج مع هذا المثال غير ان عملية السريال فيشينج سهلة جدا الا اننا نريد تعلم طرق الباتشينج مع تطبيقات ال VB لانها مختلفة قليلا.

نقوم الآن بتشغيل البرنامج وسوف نلاحظ ظهور Nag Screen والتي تطلب تسجيل البرنامج

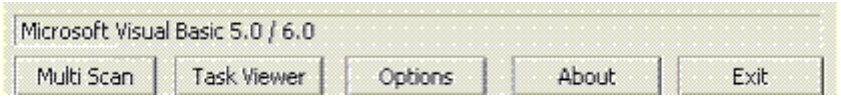


من الواضح ان البرنامج تجريبي لمدة 14 يوم فقط (وهذه هي الحماية الوحيدة الموجودة و على البرنامج اي حماية زمنية فقط يعني البرنامج سيعمل معك بشكل كامل لكن لمدة 14 يوم فقط)

اضغط الأن و على Skip This لتشغيل البرنامج ستلاحظ وجود هذه العبارة في الأعلي بجانب اسم البرنامج



الأن اغلق البرنامج ثم افحصة بواصطة PEID لتتأكد انة مبرمج بالفيجوال بيسك



نبدأ الكسر:

حمل البرنامج الي الاولي ثم ابدأ بالبحث داخل نصوص البرنامج عن اي شيء يدل علية انة غير مسجل مثلا كلمة unregistered الآن نبحث عن كلمة unregistered وهذه هي أول نتيجة

```
00427572 push    00410AF0       UNICODE "Exiting "
00427592 push    00410EA8       UNICODE ". Trial version has expired."
004277BE push    00410EE8       UNICODE "None"
004278AB mov     edx, 00410334  UNICODE "Overview"
00427918 push    0040ED38       UNICODE "Talk2D.INI"
00427A3C push    00410F00       UNICODE "The first time "
00427A53 push    00410F24       UNICODE " runs, you will go through some training. You can s
00427A70 push    0041102C       UNICODE " window. Press OK now, to start the training."
00427CB9 push    00411090       UNICODE "\Macros"
00427FCF push    004110AC       UNICODE " is already running. Please double click the icon i
0042811A push    004111BC       UNICODE " - UNREGISTERED VERSION"
00429667 push    00411200       UNICODE "\Pit="
004296AE push    00411210       UNICODE "\Spd="
```

ستلاحظ وجود عبارة Trial version has expired طبعا فمعناها واضح فهي التي سوف تظهر بعد مرور ال 14 يوم وفي الأسفل سوف تجد عبارة UNREGISTERED VERSION

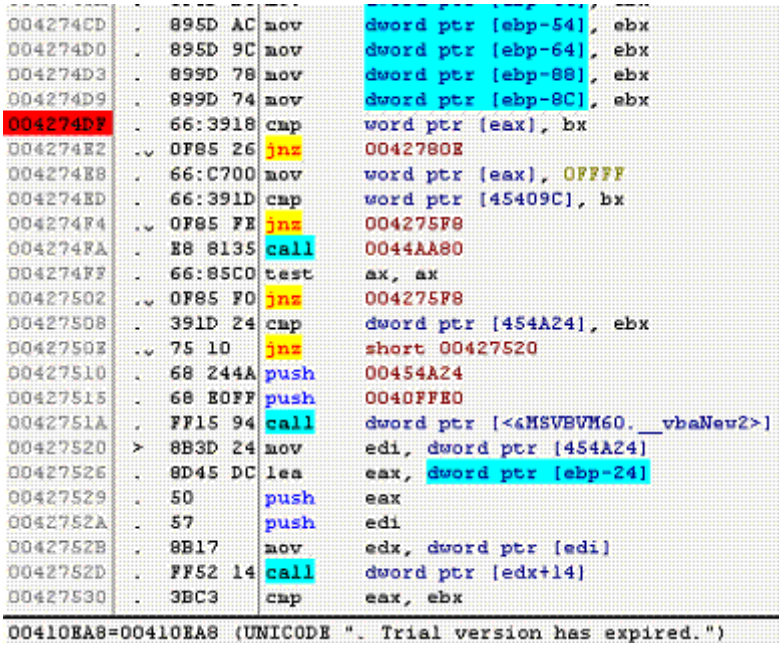
لكن يبقي شيء هام من الطبيعي ان عبارة Trial version has expired هناك امر قفز سوف يقفز اليها وكذلك عبارة UNREGISTERED VERSION هناك امر قفز سوف يقفز اليها لذلك يجب ان ندخل الي الكود ونجد هذ ه الأوامر او بعبارة اوضح نمسك الخيط من الأول :D

لذلك اضغط مرتين و على اول Trial version has expired حتي ندخل الي الكود ثم اصعد الي الأعلي حتي تجد بداية هذا الأجراء وسوف تجده هنا

```
00427470  >  55          push    ebp
00427471  .  8BEC        mov     ebp, esp
00427473  .  83EC 0C     sub     esp, 0C
00427476  .  68 462F     push    <jmp.&MSVBVM60.__vbaExceptHandler>  SE handler installation
0042747B  .  64:A1 0     mov     eax, dword ptr fs:[0]
00427481  .  50          push    eax
00427482  .  64:8925     mov     dword ptr fs:[0], esp
00427489  .  81EC 98     sub     esp, 98
0042748F  .  53          push    ebx
00427490  .  56          push    esi
00427491  .  57          push    edi
00427492  .  8965 F4     mov     dword ptr [ebp-C], esp
00427495  .  C745 F8     mov     dword ptr [ebp-8], 00401538
0042749C  .  8B75 08     mov     esi, dword ptr [ebp+8]
0042749F  .  8BC6        mov     eax, esi
004274A1  .  83E0 01     and     eax, 1
004274A4  .  8945 FC     mov     dword ptr [ebp-4], eax
004274A7  .  83E6 FE     and     esi, FFFFFFFE
00410EA8=00410EA8 (UNICODE ". Trial version has expired.")
```

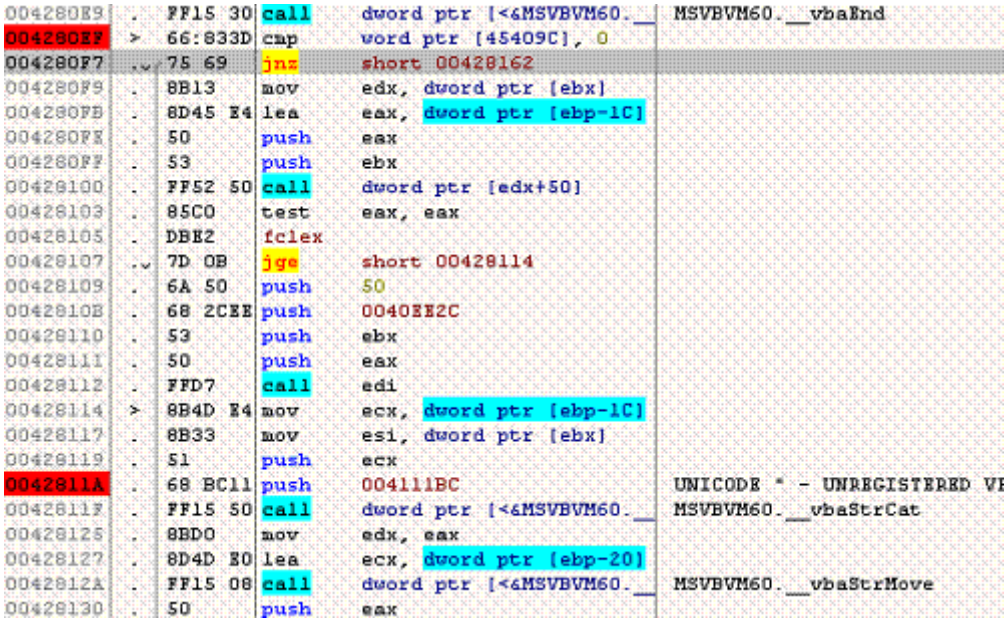
طبعا بداية الاجراء عند العنوان 00427470 الأن انزل الي السفل حتي تجد اول امر قفز من نوع JNZ او Je (وهذه هي القفزات الهامة في لغة ال VB يعني يوجد قفزات اخري كثيرة مثل JG لكنها غير هامة)

سوف تجد اول امر قفز هنا



ضع نقطة توقف عند العنوان 004274DF

الأن نعود الي عبارة ال UNREGISTERED VERSION ونفعل نفس الشيء لكن هذه المرة لن نبحث عن بداية الأجراء لانة كبير ويحتوي و على اوامر خاصة ببداية تشغيل البرنامج ككل بل سنبحث عن اول امر قفز يمكنه تجاوز هذه العبارة وسوف تجدة هنا عند العنوان 004280F7



الآن نقوم بتشغيل البرنامج بالضغط و على F9

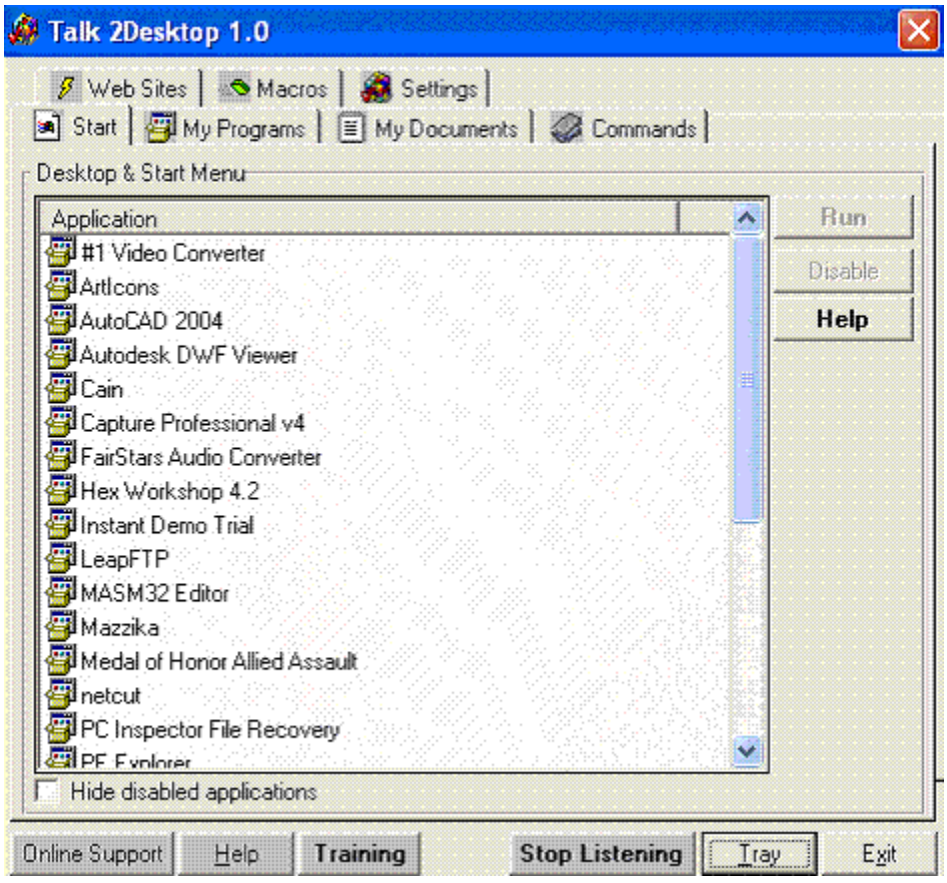
ستلاحظ ان البرنامج توقف عند العنوان 004280F7 الذي وضعنا عنده bp . الآن تتبع F8 ستجد أن أمر القفز JNZ لن يتم وبذلك سوف تظهر عبارة UNREGISTERED VERSION ومعها سوف تظهر الناج سكرين التي تطلب تسجيل البرنامج لذلك سنقوم بتحويل هذا الأمر من JNZ الي JMP لكي يقفز دائما

```
004280E9  .  FF15 30  call   dword ptr [<&MSVBVM60.__  MSVBVM60.__vbaEnd
004280EF  >  66:833D  cmp    word ptr [45409C], 0
004280F7  .  EB 69    jmp    short 00428162
004280F9  .  8B13     mov    edx, dword ptr [ebx]
004280FB  .  8D45 E4  lea    eax, dword ptr [ebp-1C]
004280FE  .  50       push   eax
004280FF  .  53       push   ebx
00428100  .  FF52 50  call   dword ptr [edx+50]
00428103     85C0     test   eax, eax
```

الآن F9 ...

ستجد أن البرنامج توقف عند العنوان 004274DF الذي وضعنا عنده نقطة توقف سابقا بالتتبع ب F8 حتي العنوان 00427502 ستجد ان امر القفز JNZ سيقفز بنا بعيدا عن Trial version has expired وطبعا ذلك لاننا مازلنا في الفترة التجريبية ال 14 يوم اي انة بعد مرور ال 14 يوم فلن يقفز ابدا لذلك سوف نقوم بعكسة ال JMP حتي يقفز دائما. الآن F9

رائع..... لقد اختفت الشاشة المزعجة التي تطلب التسجيل كما اختفت عبارة Unregistered version



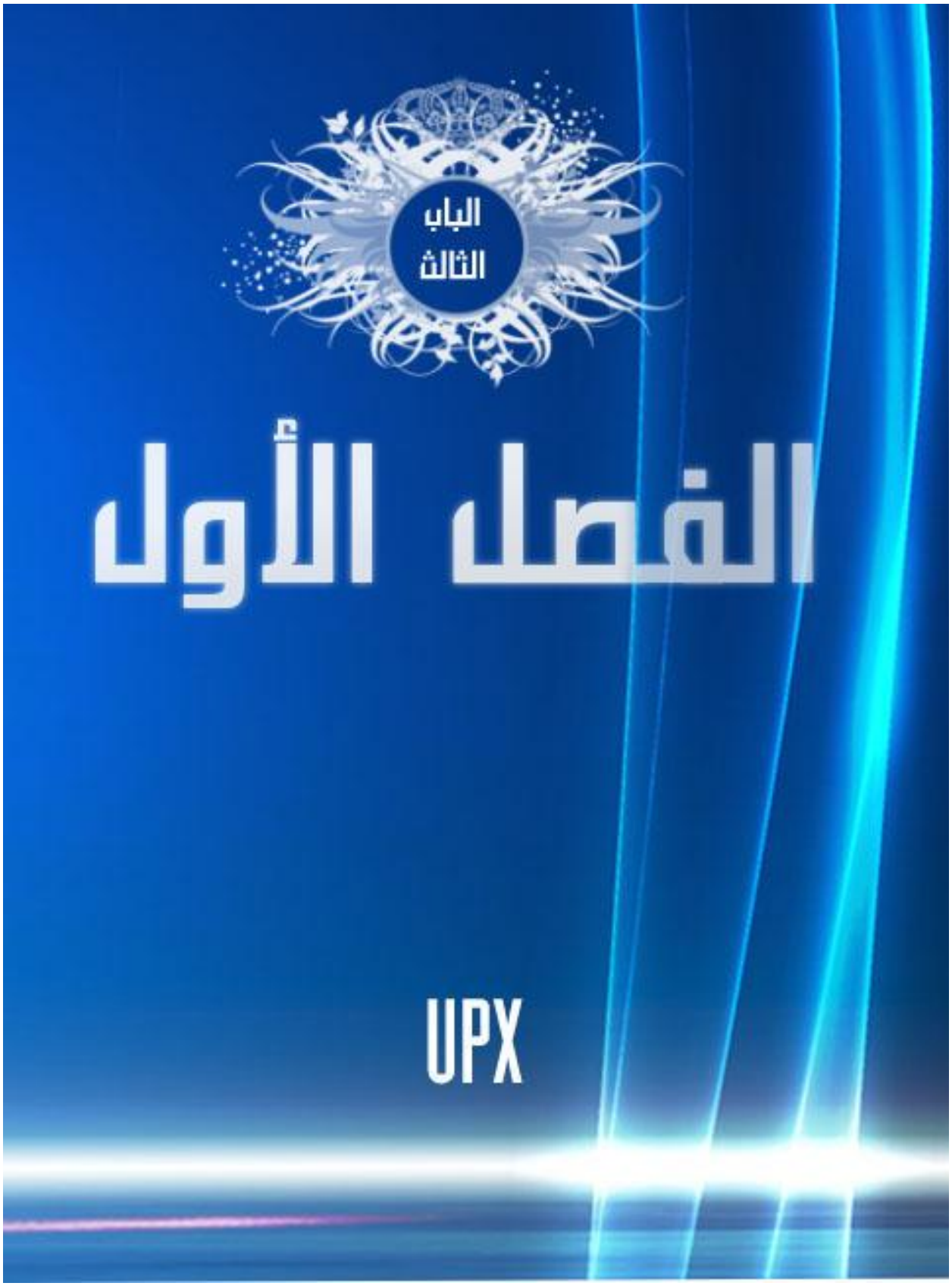
الآن احفظ التغيرات واحفظ النسخة المكسورة بأسم جديد وليكن Cracked ثم اغلق الأولي وزود التاريخ عندك شهر الي الأمام وشغل النسخة المكسورة.

مبروك تعمل بنجاح تام .

وبهذا ينتهي الفصل العاشر

وبانتهاء الفصل العاشر ينتهي الباب الثاني.


الباب الثالث
الفصل الأول : UPX
الفصل الثاني : FSG
الفصل الثالث : ASPack
الفصل الرابع : tElock
الفصل الخامس : exeShield
الفصل السادس : Scripting


الباب
الثالث
الفصل الأول
UPX

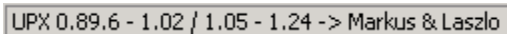
1. الفصل الاول : UPX AND CLONE

سنبدأ اليوم مع برنامج الضغط الأسهل ألا وهو UPX.هذه الأحرف هي اختصار لـ Ultimate Packer for eXecutable

1.1 . UPX X.XX :

بداية نشير إلى أننا أرفقنا UPX packer مع الدرس، حتى يمكنك استخدامه لضغط البرامج وبالتالي التدرب على الدرس بنفسك.

الآن قم بفتح الضحية المسماة UPX 1.25  مستخدما PEiD لترى التالي :

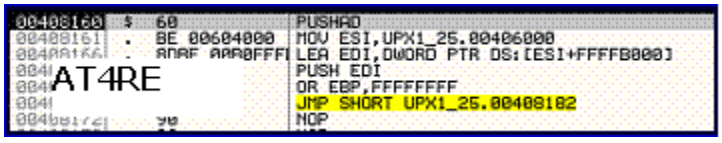


افتح الملف مستخدما LordPE ثم اختر directories ثم اضغط زر ... بجانب Import Table

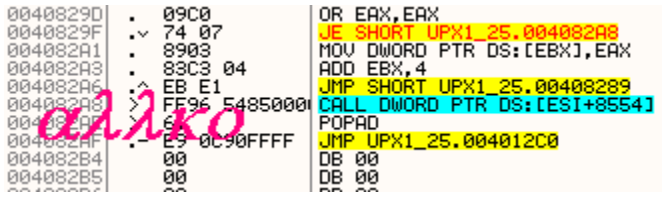
والآن اختر ملف KERNEL32.dll وانظر إلى الدوال المستوردة منه بالأسفل.كما ترى لا يوجد إلا 3 دوال فقط. وهي LoadLibraryA و GetProcAddress و ExitProcess. إن بقية الدوال اختفت بسبب "بعثرة" جدول الموارد من قبل UPX لكن بقية هذه الدوال لان UPX يحتاجها كي يتم تشغيل الملف بصورة سليمة. أيضا لو اخترت الملف الثاني ستجد انه يستورد دالة واحدة فقط.

شغل الملف مستخدما Olly

كما ترى، أول تعليمة هي تعليمة **PUSHAD** والتي تقوم بعمل **PUSH** لجميع المسجلات. لاحظ انه في اغلب – اغلب وليس كل – إصدارات UPX تكون هذه التعليمة هي أول ما تراه عند فتح الملف وآخر تعليمتان هما **POPAD** يتبعها jmp إلى الـ OEP.

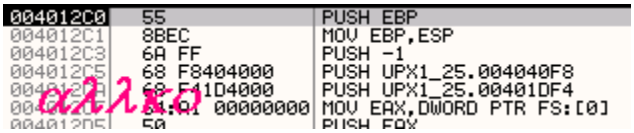


نحن عند أول تعليمة. لنر ماذا يوجد في نهاية هذه التعليمات.. أنزل للاسفل حتى تصل الى آخر تعليمة كما في الصورة التالية :



كما ترى فان آخر تعليمتان هما **POPAD** يليها jmp. ضع bp عند القفزة(4082AF) ثم F9. الآن اضغط F8 لتصل

إلى هنا :

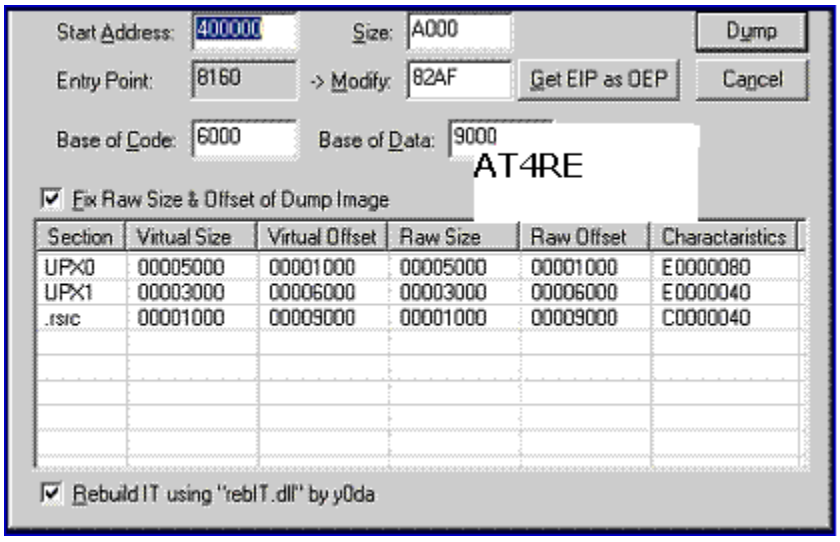


هذه بداية البرنامج الأصلية.. لاحظ أن الـ OEP تساوي 4012C0(كما ترى في الصورة) أما الـ EP

فتحسب كالتالي :

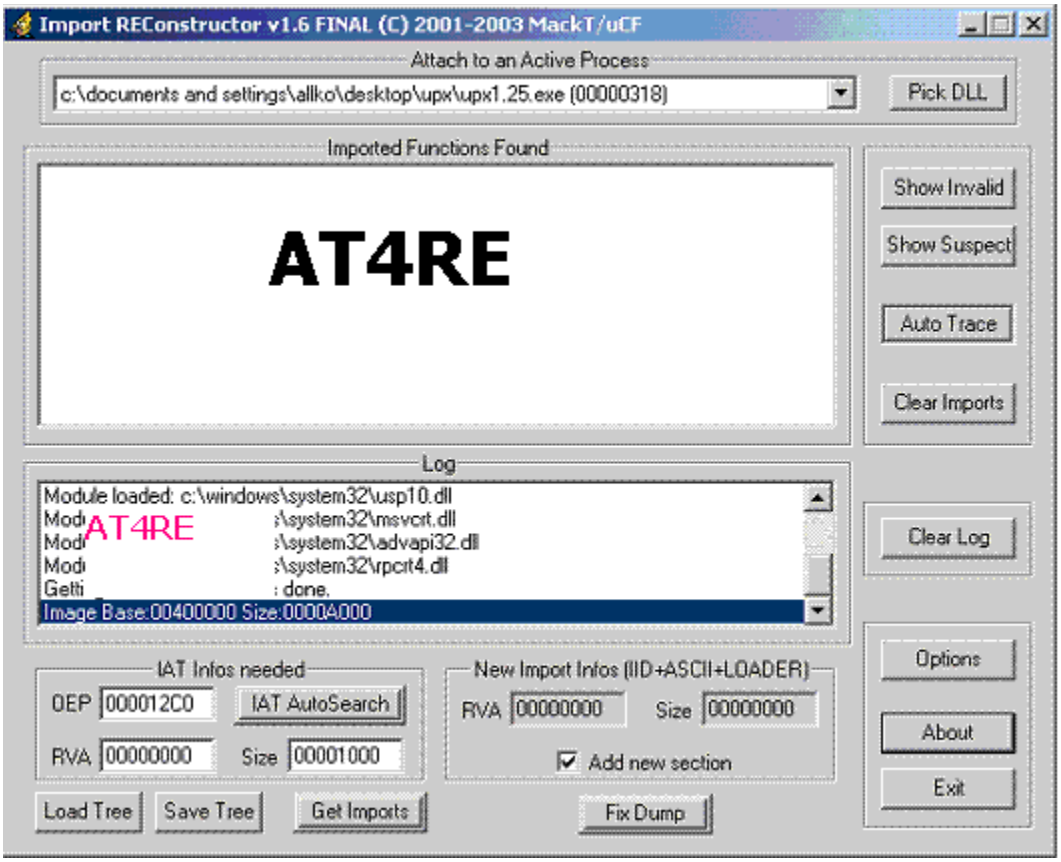
EP=OEP-ImageBase =4012C0 – 400000 = 12C0

الآن اختر Plug-in ثم OllyDump ثم اختر Dump (يفضل إزالة علامة الصح من الخيار الذي في الأسفل. على الرغم من أننا لم نزله وتم كل شيء بنجاح.)

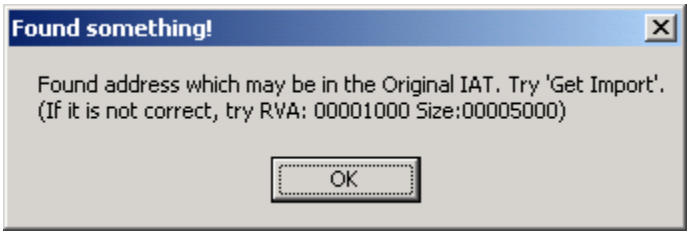


احفظ الملف الناتج بأي اسم.

قم بتشغيل ImpREC (ابق olly مفتوحا ولا تغلقه). من القائمة بالأعلى اختر ملفنا. أي upx 1.25 ثم في خانة OEP اكتب الـ EP.(الـ EP وليس الـ OEP) كما بالصورة



اختر IAT AutoSearch لتخرج لك الشاشة التالية :



حسنا، اضغط على GetImports لترى في المستطيل الأبيض الفارغ، مجموعة من الدوال المستوردة وبجانبها كلمة YES دلالة على نجاح العملية. في النهاية اختر FixDump ثم اختر الملف الذي كنا قد خزناه من Olly.

افحص الملف الناتج بـ PEiD

Microsoft Visual C++ 6.0

ها قد انتهينا. بالمرفقات مع هذا الدرس تجد ملفا باسم UPX1.91 وآخر باسم UPX 2.00 وثالث باسم UPX-Scrambler RC1.x حاول التدرب عليهم فهم مطابقون تماما لما قمنا بعمله للتو.

2.1. UPX $HIT 0.0.1

الآن دعنا نتفحص UPX $hit 0.0.1. بداية افحصه بـ PEid لترى التالي :

UPX$HiT 0.0.1 -> dj-siba

كما ترى فهو من برمجة dj-siba أحد أعضاء فريقنا ☺.

حمله إلى olly لترى التالي.

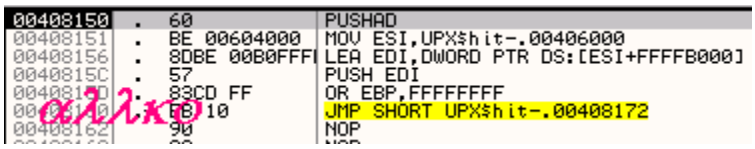
```
004095E8   94              XCHG EAX,ESP
004095E9   BC D1954000     MOV ESP,UPX$hit-.004095D1
004095EE   B9 17000000     MOV ECX,17
004095F3   80340C DB       XOR BYTE PTR SS:[ESP+ECX],0DB
004095F7 ^ E2 FA           LOOPD SHORT UPX$hit-.004095F3
004095F9   94              XCHG EAX,ESP
           AT4RE           NOP
                           NOP
                           NOP
                           JMP EAX
004095FF   FF00            INC DWORD PTR DS:[EAX]
00409601   0000            ADD BYTE PTR DS:[EAX],AL
00409603   0000            ADD BYTE PTR DS:[EAX],AL
```

والآن لنضع نقطة توقف عند تلك القفزة ثم اضغط F9. ها قد تغير الكود. يجب أن ترى التالي :

```
004095CE   0000            ADD BYTE PTR DS:[EAX],AL
004095D0   0000 50814000   ADD BYTE PTR DS:[EAX+408150],BH
004095D6   B9 54010000     MOV ECX,154
004095DB   803408 08       XOR BYTE PTR DS:[EAX+ECX],8
004095DF ^ E2 FA           LOOPD SHORT UPX$hit-.004095DB
           AT4RE           JMP EAX
                    08F83  MOV ECX,838F508B
                           DEC EDI
004095E9   BC D1954000     MOV ESP,UPX$hit-.004095D1
004095EE   B9 17000000     MOV ECX,17
004095F3   80340C DB       XOR BYTE PTR SS:[ESP+ECX],0DB
004095F7 ^ E2 FA           LOOPD SHORT UPX$hit-.004095F3
004095F9   94              XCHG EAX,ESP
004095FA   90              NOP
004095FB   90              NOP
004095FC   90              NOP
004095FD ^ FFE0            JMP EAX
004095FF   FF00            INC DWORD PTR DS:[EAX]
```

ضع نقطة توقف عند القفزة العلوية. ثم run (F9) الآن إذا تتبعت باستخدام step over أي F8 فسوف يشتغل البرنامج. لذا تتبع مستخدما step into أي F7 لتصل إلى التالي :

إن ما نراه هو أشبه بالـ junk code.. لا يوجد أي تعليمات هنا. لذا اضغط باليمين ثم اختر analysis ثم Analyze code (ستظهر لك رسالة.. أضغط ok). أو ببساطة يمكنك استخدام الاختصار وهو ctrl+A. والآن لنر ما الذي حصلنا عليه :



رائع.ها هي تعليمة **PUSHAD** المألوفة. أذن انزل للأسفل (آخر البرنامج) حتى ترى **POPAD** يليها jmp. ضع bp عند تلك الـ jmp ثم F9 ومن ثم F8. نحن الآن عند الـ OEP. أعتقد أنك تعرف البقية.

وبهذا يكون الفصل الأول قد انتهى.


الباب
الثالث
الفصل الثاني
FSG

2. الفصل الثاني : FSG

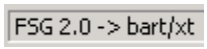
تكلمنا في الدرس الماضي عن كيفية فك ضغط UPX يدويا. أليوم سنقوم بفك ضغط برنامج FSG.

FSG هو ضاغط مخصص للبرامج الصغيرة الحجم (أقل من 100 كيلوبايت)، تحديدا البرامج المبرمجة بالاسمبلي. وهو يستخدم خوارزمية aPLib للضغط. طبعا هذا لا يعني أنه لن يستخدم للبرامج الأكبر حجما أو المبرمجة بلغات أخرى.

تجد في المرفقات برنامج FSG بمختلف إصداراته، حتى تتمكن من ضغط الملفات لتختبر نفسك.

1.2. FSG 2.0

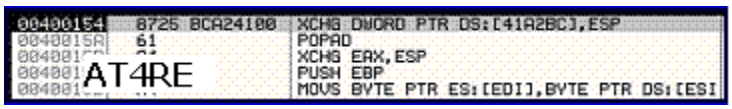
قم بفحص الضحية المسماة PackedFSG_2.0 باستخدام PEiD.



هناك طريقتان للحصول على الـ OEP وكلاهما تؤديان إلى نفس النتيجة.

2.1.1. سنبدأ بالطريقة الأولى.

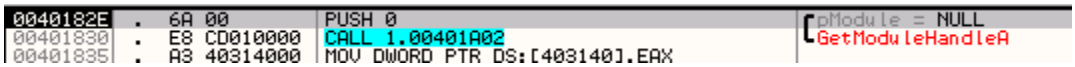
أفتح الملف المسمى 2.0 باستخدام olly. ها نحن هنا :



انزل قليلا للأسفل.ستجد التالي :

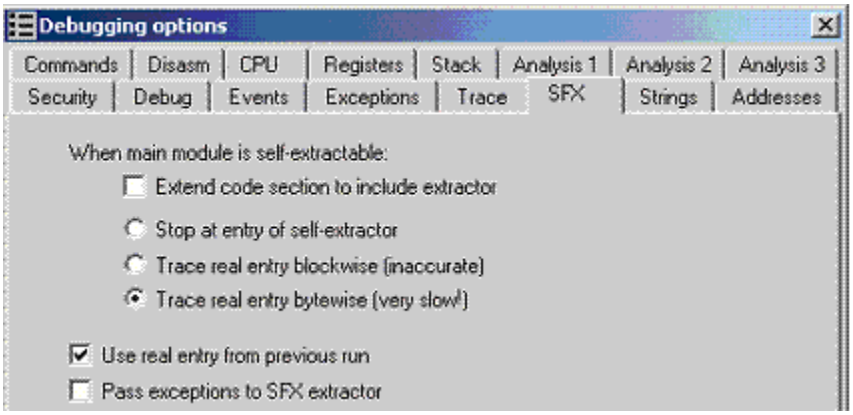
هل ترى هذه القفزات الثلاث المتتابعة؟ هذه هي العلامة المميزة لـ FSG 2.0. ضع نقطة توقف عند القفزة الثالثة ثم F9. والآن F7.

ها قد وصلنا، إذن الـ OEP هي 0040182E. هل ترى كم العملية سهلة. يمكنك (ليس ضروريا) أن تحلل الكود لتزيل الإبهام عن ما تراه. أضغط ctrl+A لتحصل على :

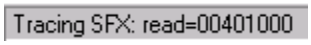


(ملاحظة : في بعض الحالات عندما تصل إلى هذه المرحلة ستجد أن التحليل لن يجدي نفعا وسيبقى الكود مبهما. لا بأس عندها أكمل العملية كالمعتاد ولا تهتم كثيرا)

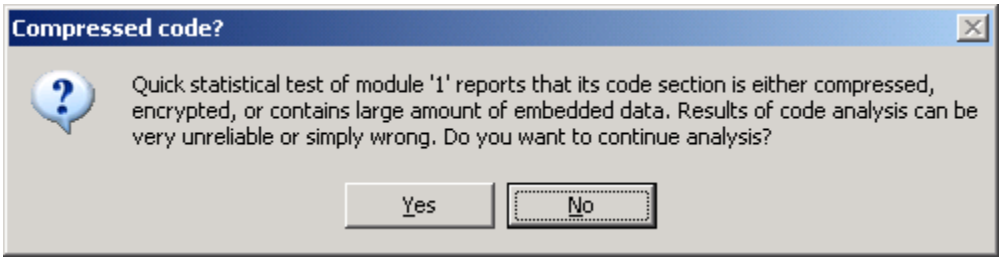
نأتي الآن إلى الطريقة الثانية. فلنقم بتغيير بعض الإعدادات. من قائمة options اختر debugging options ثم من النافذة التي ستظهر اختر لسان التبويب SFX وغير الإعدادات إلى ما تراه بالصورة :



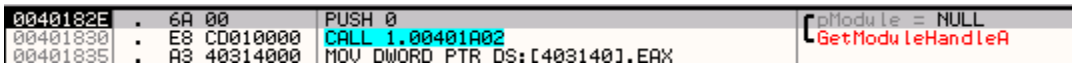
كما ترى، اخترنا الخيار الثالث الذي سيوفر علينا عناء البحث عن الـ oep حيث سيتكفل olly بالمهمة. والآن افتح الملف.سترى في الأسفل ما يلي :



هذا يعني أن olly يقوم بتتبع البرنامج لذا عليك الانتظار قليلا (يختلف الوقت باختلاف سرعة المعالج لديك). بعدما ينتهي ستخرج لك الرسالة التي تعودنا عليها :

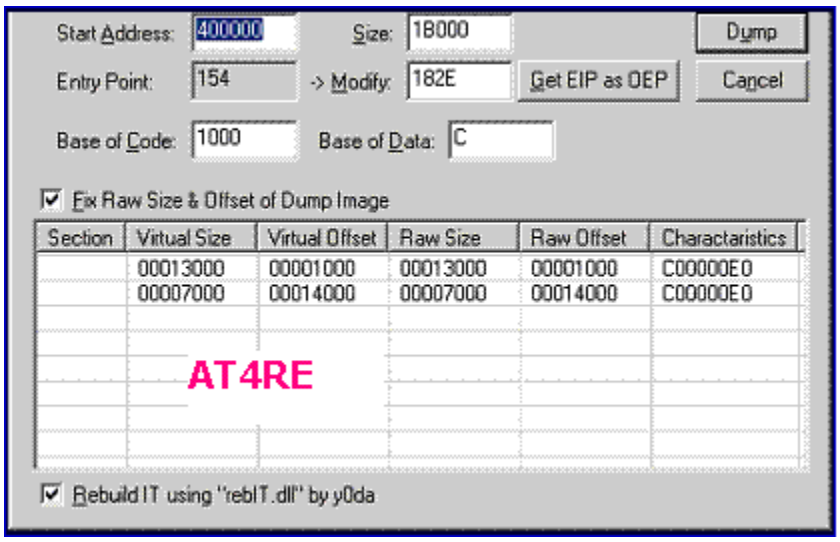


بالطبع اختر موافق. والآن سترى التالي :



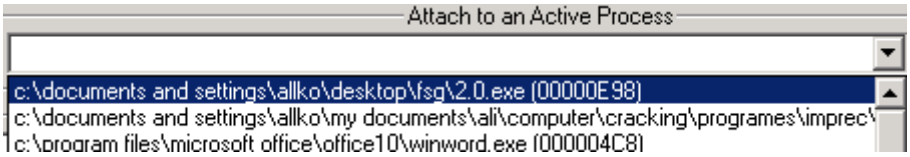
هذه هي الـ OEP. لاحظ أننا حصلنا على نفس النتيجة باستخدام الطريقتين.

والآن من قائمة plug in اختر olly Dump ثم Dump debugged process. ستخرج لك النافذة التالية :

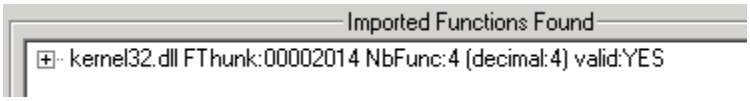


اختر Dump. بخصوص الخيار في الأسفل، فبالتجربة ثبت لي أن وضع علامة صح أو عدم وضعها لن يؤثر.(يفترض إذا وضعنا علامة الصح أن يغنينا ذلك عن استخدام imprec.)

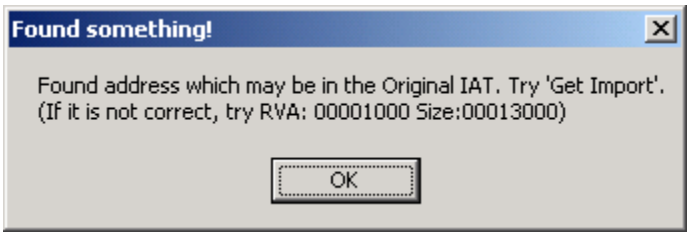
والآن شغل IMPrec. من القائمة المنسدلة (أم المنسدلة؟ :)) في الأعلى والمسماة Attach to an active process ابحث حتى تجد برنامجنا أي 2.0.exe.



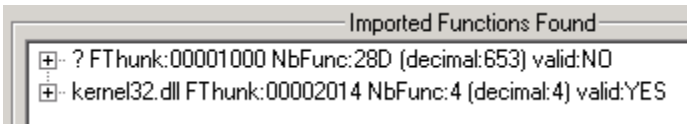
الآن في خانة OEP اكتب العنوان الذي حصلنا عليه، أي 182E ثم اضغط IATauto search، ستخرج لك رسالة. أضغط موافق.. ألآن اضغط على زر GetImports. يفترض أن يظهر لديك الصورة التالية :



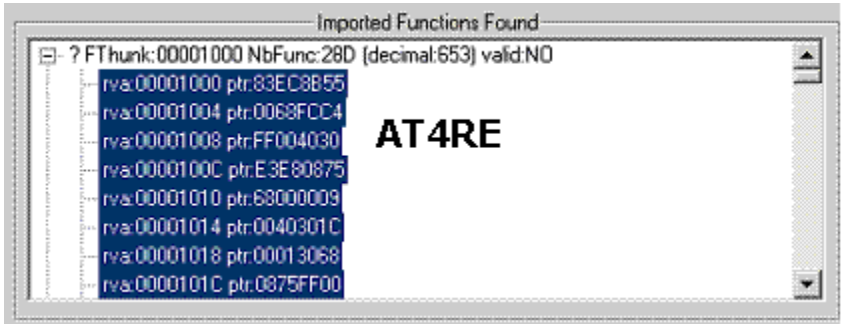
لكن وجود thunk واحد مثير للاستغراب. دعني أذكرك بالرسالة التي ظهرت عند الضغط على IAT SEARCH :



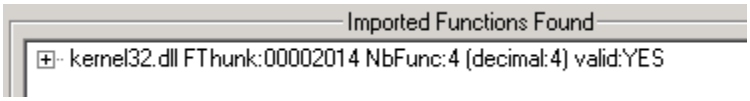
يقول ImpREC : If it is NOT correct , try RVA:1000 Size: 13000. حسنا شكرا على النصيحة :) اذهب غلى مربع RVA وضع القيمة 1000 ثم إلى مربع Size وضع القيمة 1000 (القيمة 13000 كبيرة جدا.سنبدأ بالقيمة 1000 لأنها اصغر و أخف على الجهاز). الآن اضغط GetImports لترى التالي :



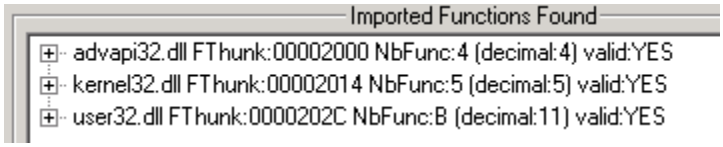
ها قد وجدنا ملف dll مستورد آخر. لكن يبدوا أنه معطوب لاحظ كلمة valid : NO. لا بأس دعنا نجرب. والآن اضغط على زر Show Invalids.



اضغط على العناوين المظللة بالأزرق بالزر اليمين واختر Cut Thunks. هذا ما ستحصل عليه :



هذا يعني أن جميع الدوال في ذلك الملف (ملف الـ dll) كانت معطوبة ولا يوجد بها أي ملف صالح. أذن فلنقم بتوسيع نطاق البحث.تذكر أن ImpRec اقترح علينا القيمة 13000 في خانة الحجم. لذلك قم بتكبير القيمة التي استخدمناها من 1000 إلى 2000. اضغط GetImports.(ستحصل على عدد كبير من الملفات.) ثم ShowInvalid ثم اضغط على العناوين المظللة بالأزرق بالزر اليمين واختر Cut Thunks. النتيجة هي :



رائع !! حصلنا على ملفي dll مستوردين صحيحين آخرين. إذن اختر FixDump ثم اختر الملف الذي حفظناه من olly. الآن قم بتشغيل الملف.رائع!! إنه يعمل بنجاح.

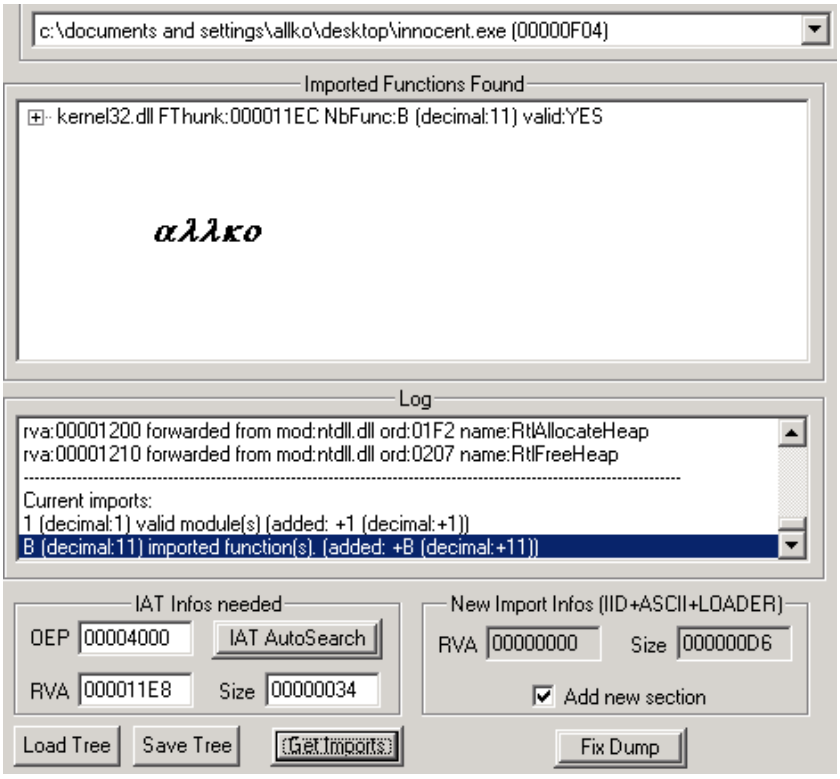
في بعض الأحيان تضطر إلى توسيع نطاق البحث عدة مرات وقد تصل إلى القيمة 5000. لذا، مستقبلا، لديك الخيار إما أن تقوم بتوسيع نطاق البحث بمقدار 1000 بالمرة الواحدة أو أن تقفز إلى القيمة 5000 فورا. نحن اخترنا الطريقة الأولى لسبب بسيط. عندما تصل إلى 5000، فعملية الـ cutthunks ستكون مرهقة للجهاز كثيرا، وحيث أن الجهاز الذي استخدمناه يعمل على معالج من العصور الوسطى (1800 mhz) لذلك لم نشأ أن نغامر من البداية فقد نحصل على النتيجة الصحيحة مع القيمة 2000 أو 3000. وإذا اضطررنا فسنذهب إلى الـ 5000.

2.1.2 . أسلوب آخر :

سنتعلم أسلوبا آخر في عملية إصلاح الـ IAT (Import Address Table) للملفات المضغوطة بـ FSG

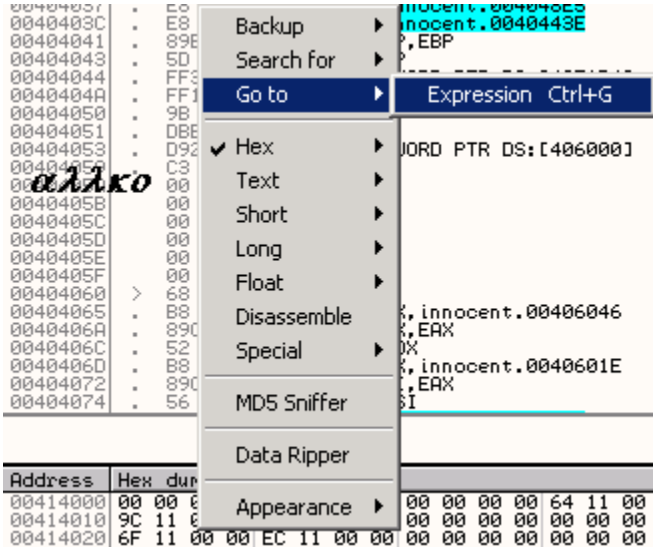
كما رأينا فآخر خطوة كانت توسيع نطاق البحث في ImpREC، عندما غيرنا قيمة RVA إلى 1000 وقيمة size إلى 1000، وبدئنا بزيادة قيمة الـ size إلى أن حصلنا على نتيجة صحيحة وتم إصلاح الملف (البرنامج).

ما سنقوم به اليوم هو نفس الشيء. توسيع نطاق البحث. لكن يدويا، وبدقة أكبر. حسنا افتح الملف المسمى innocent بـ olly. ستصل إلى الـ OEP (404000). dump. افتح imprec ثم IATautosearch. ثم.GetImports ثم Fix Dump. وعند تجربة الملف الناتج لن يعمل.



لكن لاحظ أن imprec قد وضع قيمة في قسم RVA. في الواقع هذه القيمة هي التي وجد عندها الـ IAT الخاص بملف الـ dll الذي استطاع إيجاده(تحديدا kernel32.dll). من الواضح أن بقية الـ IATs الخاصة بباقي ملفات الـ dl ستكون بالقرب من من ذلك العنوان (11E8). ما كنا نفعله هو الإبقاء على نفس العنوان لكن زيادة مدى الرؤية (أي زيادة size) وبالتالي اذا كان هناك ملفات dll أعلى أو أسفل 11E8 فسوف نجدها.

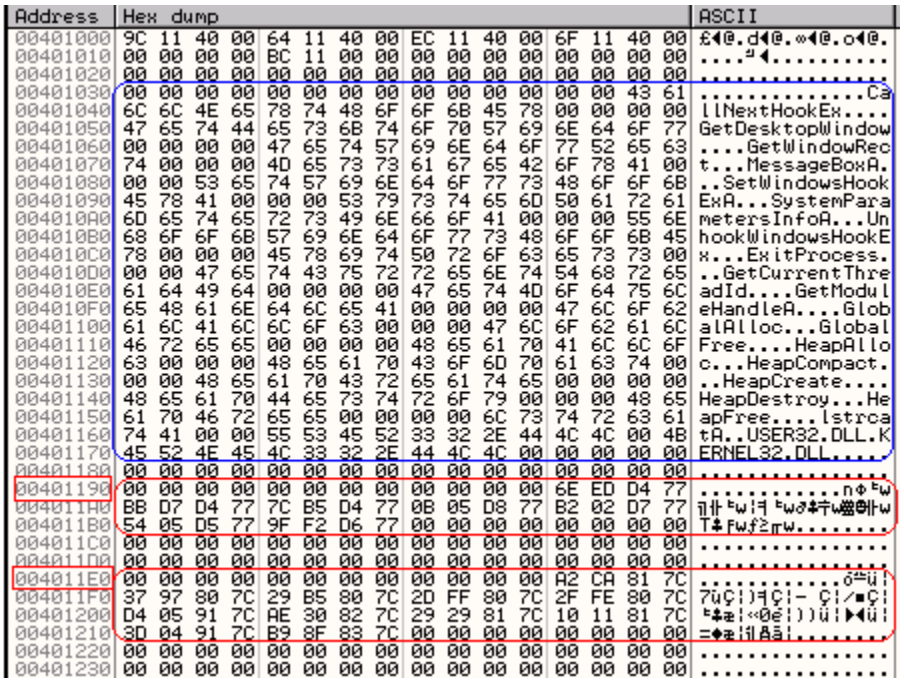
لكن اليوم سنقوم يدويا بالبحث عن تلك الملفات. الأمر بسيط، في olly في قسم Dumped Data بالأسفل اضغط باليمين واختر Go TO ثم Expression :



سيظهر المربع التالي.. أكتب عنوان الـ VA الذي حصلنا عليه

(VA=ImageBase + RVA = 400000 + 11E8 = 4011E8)

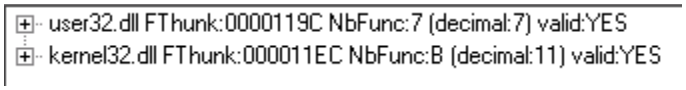
والآن قم بتكبير القسم السفلي حتى يتسنى لنا البحث بشكل جيد.



Address	Hex dump	ASCII
00401000	9C 11 40 00 64 11 40 00 EC 11 40 00 6F 11 40 00	£♦@.d♦@.∞♦@.o♦@.
00401010	00 00 00 00 BC 11 00 00 00 00 00 00 00 00 00 00	....╝♦..........
00401020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
00401030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 43 61	..............Ca
00401040	6C 6C 4E 65 78 74 48 6F 6F 6B 45 78 00 00 00 00	llNextHookEx....
00401050	47 65 74 44 65 73 6B 74 6F 70 57 69 6E 64 6F 77	GetDesktopWindow
00401060	00 00 00 00 47 65 74 57 69 6E 64 6F 77 52 65 63	....GetWindowRec
00401070	74 00 00 00 4D 65 73 73 61 67 65 42 6F 78 41 00	t...MessageBoxA.
00401080	00 00 53 65 74 57 69 6E 64 6F 77 73 48 6F 6F 6B	..SetWindowsHook
00401090	45 78 41 00 00 00 53 79 73 74 65 6D 50 61 72 61	ExA...SystemPara
004010A0	6D 65 74 65 72 73 49 6E 66 6F 41 00 00 00 55 6E	metersInfoA...Un
004010B0	68 6F 6F 6B 57 69 6E 64 6F 77 73 48 6F 6F 6B 45	hookWindowsHookE
004010C0	78 00 00 00 45 78 69 74 50 72 6F 63 65 73 73 00	x...ExitProcess.
004010D0	00 00 47 65 74 43 75 72 72 65 6E 74 54 68 72 65	..GetCurrentThre
004010E0	61 64 49 64 00 00 00 00 47 65 74 4D 6F 64 75 6C	adId....GetModul
004010F0	65 48 61 6E 64 6C 65 41 00 00 00 00 47 6C 6F 62	eHandleA....Glob
00401100	61 6C 41 6C 6C 6F 63 00 00 00 47 6C 6F 62 61 6C	alAlloc...Global
00401110	46 72 65 65 00 00 00 00 48 65 61 70 41 6C 6C 6F	Free....HeapAllo
00401120	63 00 00 00 48 65 61 70 43 6F 6D 70 61 63 74 00	c...HeapCompact.
00401130	00 00 48 65 61 70 43 72 65 61 74 65 00 00 00 00	..HeapCreate....
00401140	48 65 61 70 44 65 73 74 72 6F 79 00 00 00 48 65	HeapDestroy...He
00401150	61 70 46 72 65 65 00 00 00 00 6C 73 74 72 63 61	apFree....lstrca
00401160	74 41 00 00 55 53 45 52 33 32 2E 44 4C 4C 00 4B	tA..USER32.DLL.K
00401170	45 52 4E 45 4C 33 32 2E 44 4C 4C 00 00 00 00 00	ERNEL32.DLL.....
00401180	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
00401190	00 00 00 00 00 00 00 00 00 00 00 00 6E ED D4 77	............nΦ╘w
004011A0	BB D7 D4 77 7C B5 D4 77 0B 05 D8 77 B2 02 D7 77	╗╫╘w|╡╘w♂♣╪w▒☻╫w
004011B0	54 05 D5 77 9F F2 D6 77 00 00 00 00 00 00 00 00	T♣╒w ≥╓w........
004011C0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
004011D0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
004011E0	00 00 00 00 00 00 00 00 00 00 00 00 A2 CA 81 7C	............ó╩ü|
004011F0	37 97 80 7C 29 B5 80 7C 2D FF 80 7C 2F FE 80 7C	7ù|)╡|- |/■|
00401200	D4 05 91 7C AE 30 82 7C 29 29 81 7C 10 11 81 7C	╘♣æ|«0é|))ü|►◄ü|
00401210	3D 04 91 7C B9 8F 83 7C 00 00 00 00 00 00 00 00	=♦æ|╣Å|........
00401220	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
00401230	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....

كما ترى، في المربع الأحمر السفلي هناك الـ IAT الخاص بملف الـ dll الذي وجده imprec عند عنوان 4011E8 أي Kernel32.dll. إذا نظرت للأعلى ستتنبه إلى وجود IAT آخر خاص بملف dll آخر. لاحظ أنه يبدأ عند 401198. إذن الذي حصل هو أن FSG قام بالتلاعب بالـ IATs فجعل imprec يظن أنها تبدأ عند 4011E8 وبالتالي لم يرى ما يوجد قبل هذا العنوان. وعندما نكبر قيمة size فإن Imprec يزيد مدى الرؤية حول القيمة 4011E8 وبالتالي سيجد الملف الذي بالأعلى. أعتقد أن الأمور أصبحت واضحة الآن.

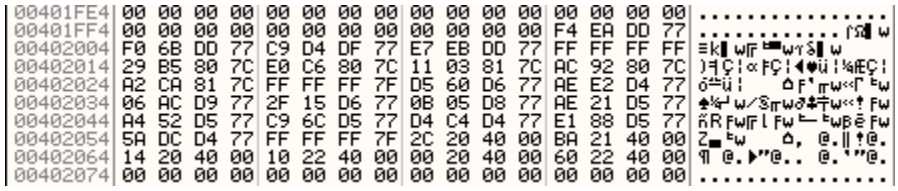
والآن كل ما علينا فعله هو العودة إلى imprec وتغيير قيمة RVA إلى 1198. أما الـ size فلم يعد مهما كثيرا لكن من الجيد تغييره إلى قيمة مناسبة ولتكن 100.. ثم الضغط على GetImports.



كما ترى فقد حصلنا على ملف الـ dll الثاني. وكما ترى فهو user32.dll. ألآن كخطوة احتياطية اضغط ShowInvalid. لحسن الحظ لا يوجد شيء خطأ. أذن اختر Fix Dump وستلاحظ أن الملف الناتج يعمل بشكل صحيح.

خلاصة : بالنسبة لـ FSG. عندما لا يعمل الملف الناتج فعليك توسيع نطاق بحثك عن الـ IAT إما أوتوماتيكيا بتغيير قيمة size أو يدويا.

لكن، ومع ذلك فالعمل اليدوي قد يكون أصعب. فمثلا ألق نظرة على ملف 2.0 (لن تجده مرفقا، فهذا هو الضحية التي استخدمناها في الدرس السابق.).قم بنفس الخطوات، لن يعمل الملف وسنزيد نطاق البحث. يخبرنا ImpRec أنه قد وجد ملف dll (تحديدا kernel32.dll) عند عنوان 2014. ألق نظرة :



في الواقع ما تراه في الصورة هو ثلاث IATs لثلاث ملفات dll. لن تستطيع تحديد أين ينتهي كل واحد منهم لذا ببساطة عليك العودة الى الطريقة الاوتوماتيكية لان العمل اليدوي صعب جدا في هذه الحالة.

2.2. FSG 1.1 , 1.2 , 1.3

بالنسبة لهذا الإصدارات فانه ما من علامة مميزة كما في الإصدار 2.0، لكن ببساطة استخدم الطريقة الثانية لإيجاد الـ OEP (أقصد طريقة تغيير الإعدادات) ومن ثم أكمل كالمعتاد. هذه الطريقة تنجح مع جميع الإصدارات.

وبهذا ينتهي الفصل الثاني بحمد الله.

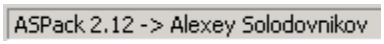

الباب
الثالث
الفصل الثالث
ASPack

3. الفصل الثالث : ASPACK 2.XX

يتحدث درس اليوم عن كيفية فك ضغط البرامج المضغوطة بـ ASPack 2.12 يدويا.

برنامج ASPack هو ضاغط ملفات متقدم، قادر على تصغير حجم الملف لغاية 70%. إن ASPack يجعل برامج ومكتبات Windows 95/98/NT أصغر، ويقلل زمن التحميل(download) من الانترنت أو عبر الشبكات, إضافة إلى ذلك فهو يحمي البرامج من الهندسة العكسية من قبل المبتدئين.

تجد في المرفقات برنامج ASPack. يمكنك استخدامه لضغط الملفات كي تتدرب عليها بنفسك. أيضا هناك ضحية أخرى للتدرب عليها. أفحص الضحية target1  بـ PEiD :

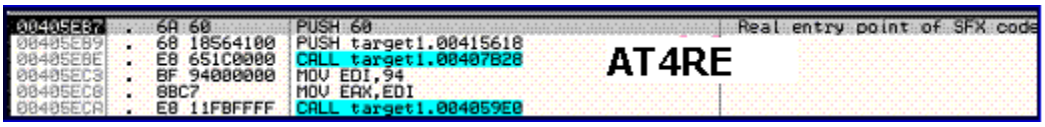


للحصول على الـ OEP هناك طريقتان.

3.1. الطريقة الأولى

هي تغيير إعدادات SFX كما فعلنا في الدرس الماضي مع FSG 2.0. إذن اذهب إلى options ثم debugging options ثم اختر لسانا لتبويب SFX وغير الإعدادات إلى التالي :

والآن قم بتشغيل الملف.كما تلاحظ فـ olly مازال يتتبع البرنامج وتختلف مدة التتبع باختلاف سرعة المعالج. عندما ينتهي من التتبع سنكون هنا :

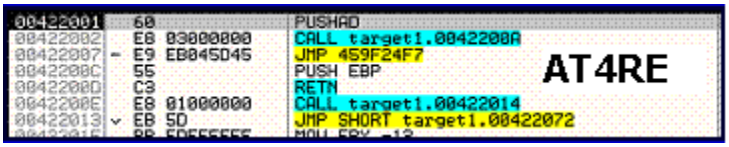


إذن الـ OEP هي 405EB7.

3.2. الآن دعنا نرى الطريقة الثانية

. أعد الإعدادات إلى ما كانت عليه (أي اختر الخيار الأول بدلا من الثالث) وقم بفتح الملف.

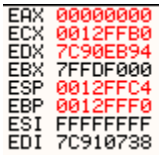
سنكون بداية هنا :



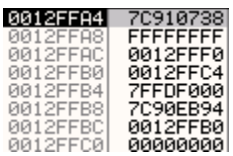
أول تعليمة هي **PUSHAD** وبالتالي نتوقع أن تكون آخر تعليمتان **POPAD** ثم jmp إلى الـ OEP. لكن الحال يختلف قليلا عما قابلناه مع UPX فهناك أكثر من تعليمة **POPAD** وليس تعليمة واحدة. لا بأس فتعليمة **PUSHAD** الموجودة بالبداية لها تعليمة**POPAD** بالنهاية ولا يهم سواء كان هناك تعليمات من هذا النوع بين البداية والنهاية أم لا.

ما أريدك أن تعرفه هو أنه وبعد تنفيذ تعليمة **PUSHAD** فإن المسجلات الثمانية (بدءا بـ EDI) سوف تدفع إلى المكدس. ما سنفعله هو وضع نقطة توقف على احد تلك القيم الثمانية، وبالتالي في نهاية الروتين الخاص بالـ packer وعندما يصل إلى تعليمة **POPAD** سيصل إلى تلك القيمة ويتوقف لان عليها نقطة توقف. عندها نكون قد وصلنا إلى نهاية روتين الـ packerو نعرف أين يبدأ كود البرنامج أي نعرف الـ OEP.

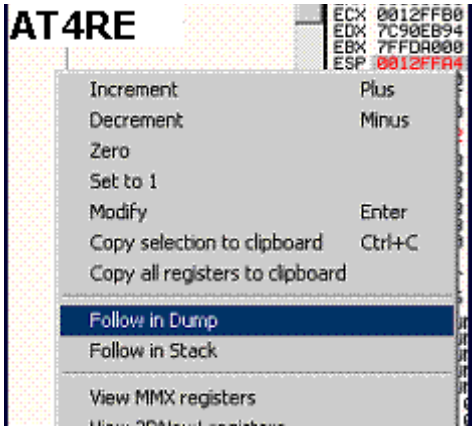
والآن قبل تنفيذ تعليمة **PUSHAD** لاحظ المسجلات الثمانية :



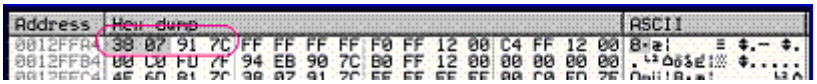
إذن اضغط F8 ولاحظ الـ stack : (لاحظ أن أول قيمة هي قيمة EDI آخر قيمة هي قيمة EAX)



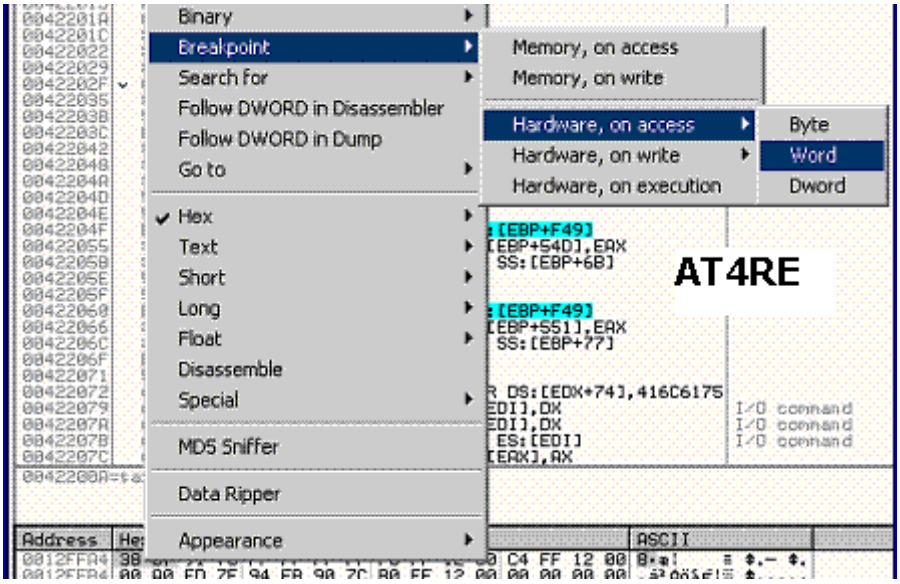
سنقوم بوضع نقطة توقف على أول قيمة وهي قيمة EDI (كما قلنا فـ EDI يدفع إلى المكدس أولا). بجانب ESP ستجد القيمة 0012FFA4. اضغط عليها (باليسار) كي يتم تحديدها، والآن اضغط باليمين واختر follow in dump :



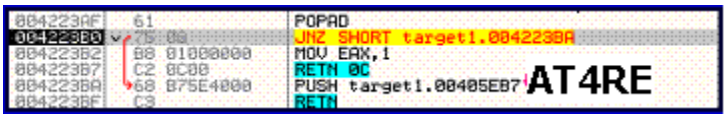
ستصل إلى هنا :



لاحظ أول سطر.ها هي قيمة EDI (وضعت مستطيلا حولها) لكن معكوسة.حدد أول بايتين منها (لا داعي لتحديدها كلها.) والآن اضغط على ما حددته باليمين، ثم اختر Breakpoint ثم Hardware , on access ثم word

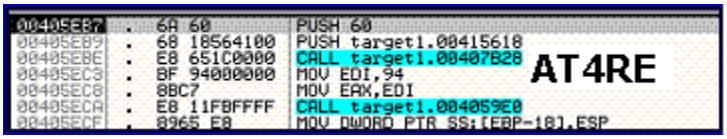


والآن شغل البرنامج (F9). ها قد وصلنا إلى هنا :

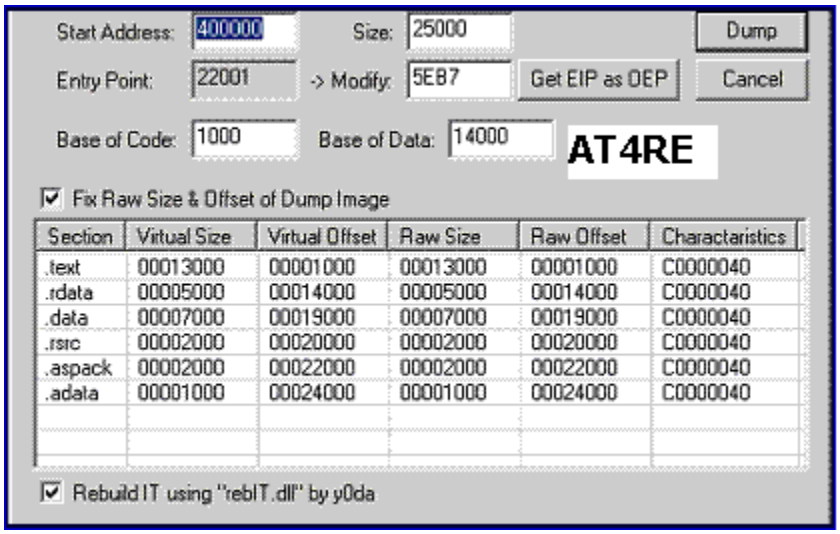


ها هي تعليمة **POPAD**. تتبع (اضغط F8 ثلاث مرات) وستصل إلى هنا :

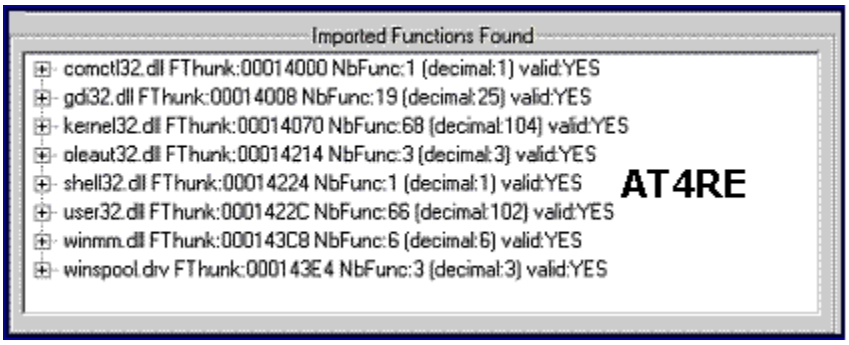
لاحظ أننا حصلنا على نفس الـ OEP التي حصلنا عليها بالطريقة السابقة. و كما ترى فالكود "مبهم" لأن olly لم يقم بتحليله بعد.. يمكنك – إن أردت – أن تزيل الإبهام بالضغط على CTRL+A لتحليله وستحصل على :



والآن من قائمة Plug ins اختر Olly Dump ثم Dump Debugged Process ليظهر لك :



والآن اختر Dump واحفظه بأي اسم. أبق olly مفتوحا وشغل ImpREC. من القائمة العلوية اختر الملف الضحية وفي مربع OEP اكتب 5EB7 ثم IATauto search ثم GetImports.



لحسن الحظ فجميع الدوال كما ترى صحيحة (انظر إلى كلمة yes بجانب كل منهم).

أعتقد انك تعرف ما يجب فعله في حال كان هناك دالة أو أكثر معطوبة (NO). ببساطة اختر show invalid ثم اضغط باليمين على ما تم تحديده بالأزرق واختر cut Thunks.

والآن اختر Fix Dump ثم اختر الملف الذي قمنا توا بحفظه.شغل الملف.رائع أنه يعمل بنجاح !!

تجد بالمرفقات ملفا آخر (رقمه 2) يمكنك التدرب عليه فهو بنفس الفكرة تماما.

وبهذا ينتهي الفصل الثالث.


الباب
الثالث
الفصل الرابع
tELock

4. الفصل الرابع : TELOCK 0.98

درس اليوم يتحدث عن كيفية فك ضغط وكسر حماية tElock 0.98 يدويا.

مرة أخرى أقوم بتأجيل الدرس المخصص لـ exeshield 3.X لأسباب ذكرتها سابقا. درس اليوم ليس صعبا، يمكنني تصنيفه على أنه " سهل ممتنع " :) على كل حال دعنا نبدأ وكفى إضاعة للوقت.

4.1. TELOCK 0.98 : A CLOSER LOCK

tElock logo : 1 Figure

tElock هو برنامج يقوم بتشفير وضغط (Encryptor/-Compressor) ملفات PE-Files من نوع exe , dll and ocx، وفي نفس الوقت يبقي الملفات المشفرة/ المضغوطة قابلة للتنفيذ (أي executable) ويحميها من عمليات الـ patching والـ disassembling.

هذا البرنامج هو من برمجة tE، أول إصدار صدر في عام 2001، لا تتوقع أن تجد العديد من البرامج المحمية به. لكن هذا لا يمنع من تعلم كيفية حمايته.

يحوي البرنامج قائمة طويلة من المميزات. أهمها ما يلي :

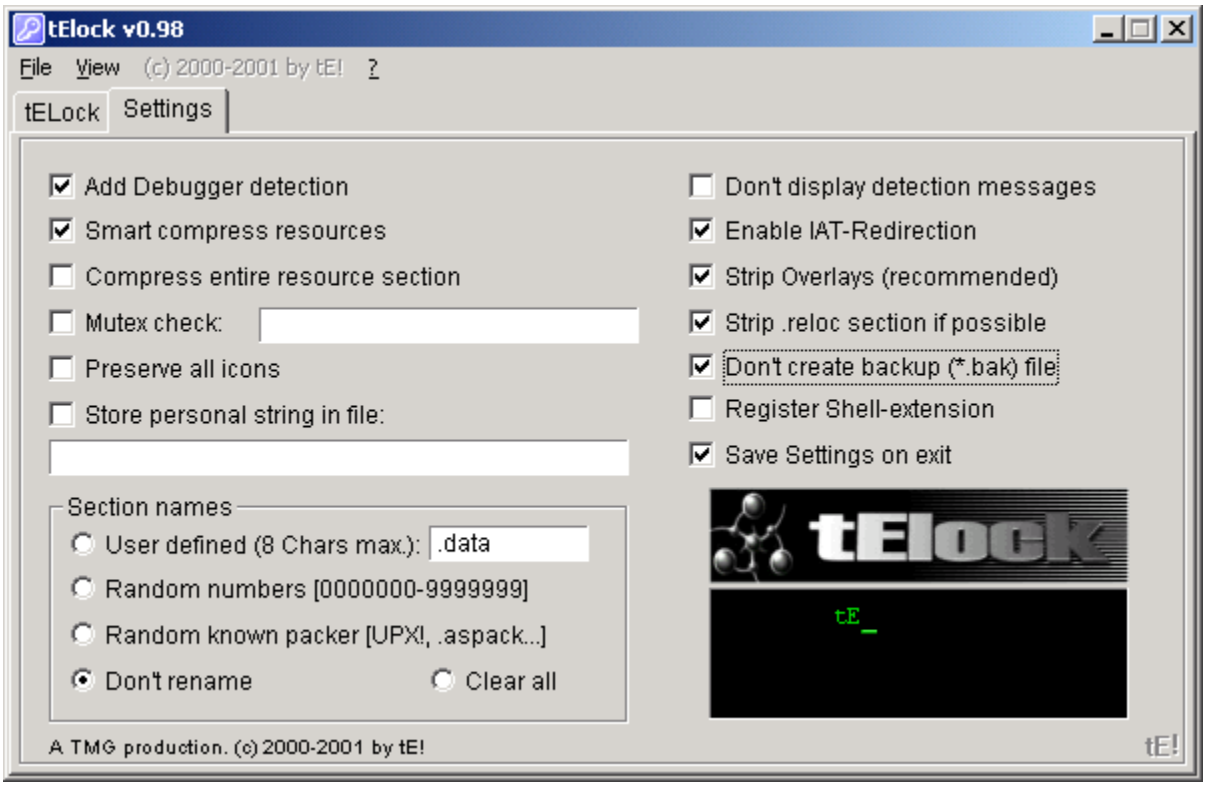
- يقوم بضغط الملفات ويصغر حجمها بنسبة كبيرة.
- يحمي برنامجك من عمليات الـ patching/modifications and disassembling
- يستطيع اكتشاف الـ breakpoints وعمليات الـ Debugger trace و يستطيع اكتشاف وجود Softice
- Anti-(Proc)Dump code
- IAT(WinAPI calls)-redirection وربما هذه أهم نقطة فيه.
- Resource, Relocations, simple tls and exports-handling/redirection
- Importable encryption

كالعادة، تجد في المرفقات برنامج tElock الإصدار 0.98 ، وهو مجاني.حاول تجربته قليلا وتمعن في الخيارات الموجودة وإن أردت المزيد من التفاصيل يمكنك رؤية ملف المساعدة التابع للبرنامج.

أيضا هناك مجلدان مرفقان، الأول باسم targets وبه الملفات المضغوطة الأربعة (من كل لغة برمجة اخترت برنامجا).

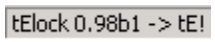
LET'S BEGIN , SHALL WE ? .4.2

نود أن نشير إلى أن الإعدادات التي استخدمناها لضغط الملفات الأربعة هي كالتالي :



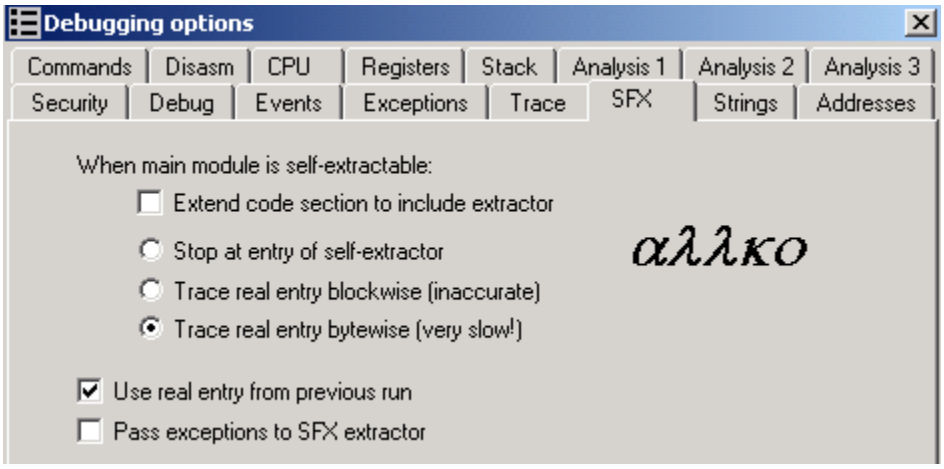
tElock settings window , : 2 Figure

بعد ضغط الملف asm وفحصه باستخدام PEiD :

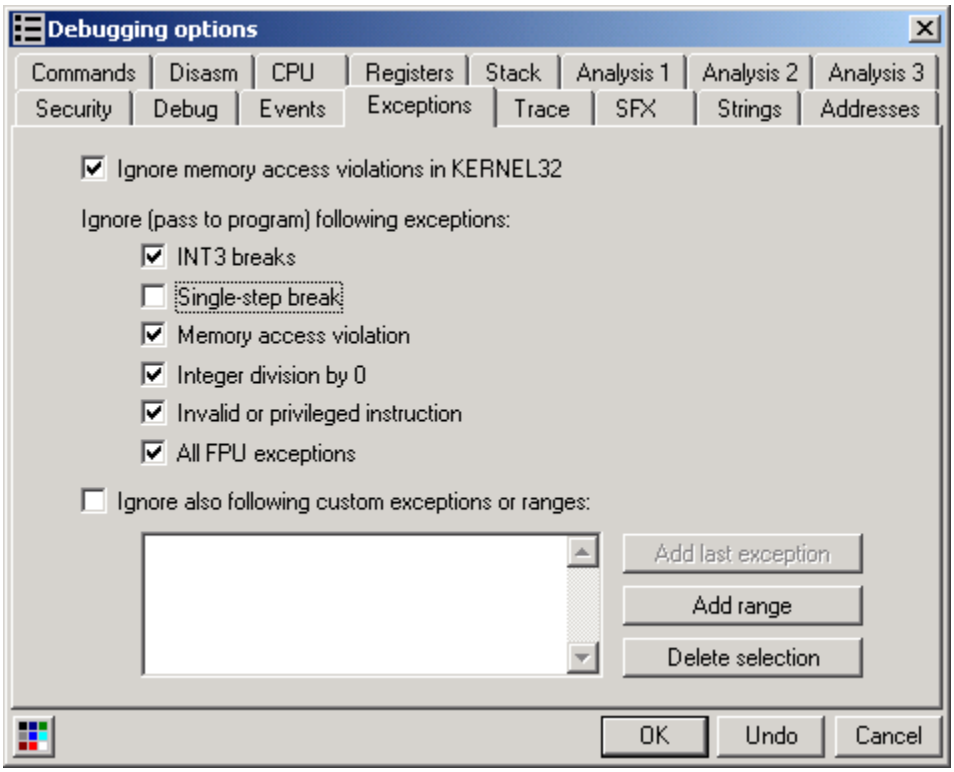


سنقوم بفتح الملف asm (إما الموجود في مجلد targets أو أن تقوم بضغط الملف الموجود في مجلد original لكني أفضل الموجود في targets)

لكن أولا سنغير بعض الإعدادات. أول الإعدادات التي سنغيرها هي ما يتعلق بـ SFX و أعتقد إن كنت متابعا للدروس السابقة أنك تعرف ما أتحدث عنه جيدا. بأي حال من قائمة SFX ➔ debugging options ➔ options ستظهر لك النافذة التالية، غير الإعدادات كالتالي :



هذا سيسهل عملية إيجاد الـ OEP، أي سيجعل Olly يجدها بدلا منا :). والآن اختر لسان التبويب Exceptions ثم :



ستلاحظ أثناء تتبعك للبرامج المحمية بـ tElock وجود العديد من الاستثناءات exceptions ولذلك سنقوم هنا بتجاهلها جميعا عدا الخيار الثاني، حيث سنعلّم عليه لكن بعد تحميل البرنامج في olly.

هناك طريقتان لكسر الحماية :

الطريقة الأولى تتلخص في ثلاث نقاط :

- إيجاد الـ OEP
- استعادة الـ imports التي قام tElock بعمل redirection لها
- عمل dump ثم fix dump باستخدام imprec.

الطريقة الثانية تعتمد على فكرة بسيطة. تعلم أن هناك خيار في tElock هو IAT(WinAPI calls)-redirection.

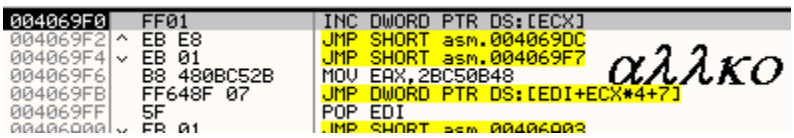
حسنا. أن اخترنا هذا الخيار سيقوم البرنامج بعمل redirection و إن لم نختر فلن يقوم بعمل ذلك. الإجرائية الخاصة بعمل الـ redirection تكون موجودة بغض النظر عما إذا اخترت ذلك الخيار أم لا. ما يحدث هو أن هناك jump في البرنامج (أو لنقل هناك عدة تعليمات) تحدد هل سيتم الذهاب إلى إجرائية الـ redirection أم سيتم تجاوزها. في حالة اخترنا نحن (أو بالأحرى المبرمج) ذلك الخيار، فبكل تأكيد سيتم الذهاب إلى الإجرائية، هنا نأتي نحن – وبكل بساطة – بعكس تلك القفزة. فنكون قد أوهمنا البرنامج بأن ذلك الخيار لم يتم تحديده !!! وبذلك لا يقوم بعمل redirection.

أي يمكن تلخيص الخطوات كالتالي :

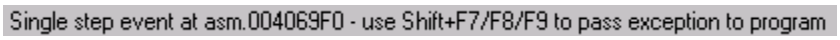
- إيجاد الـ OEP
- عكس أمر القفز المسؤول عن الذهاب إلى إجرائية الـ redirection.
- عمل dump ثم fix dump باستخدام imprec.

4.2.1 . الطريقة الأولى :

إذن قم بفتح البرنامج الضحية والمسمى asm من مجلد targets باستخدام olly لتر التالي :

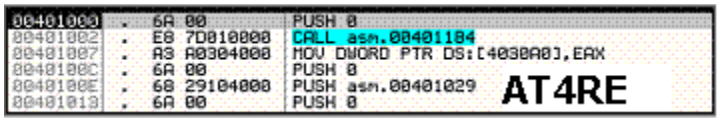


لكن انظر إلى الأسفل :



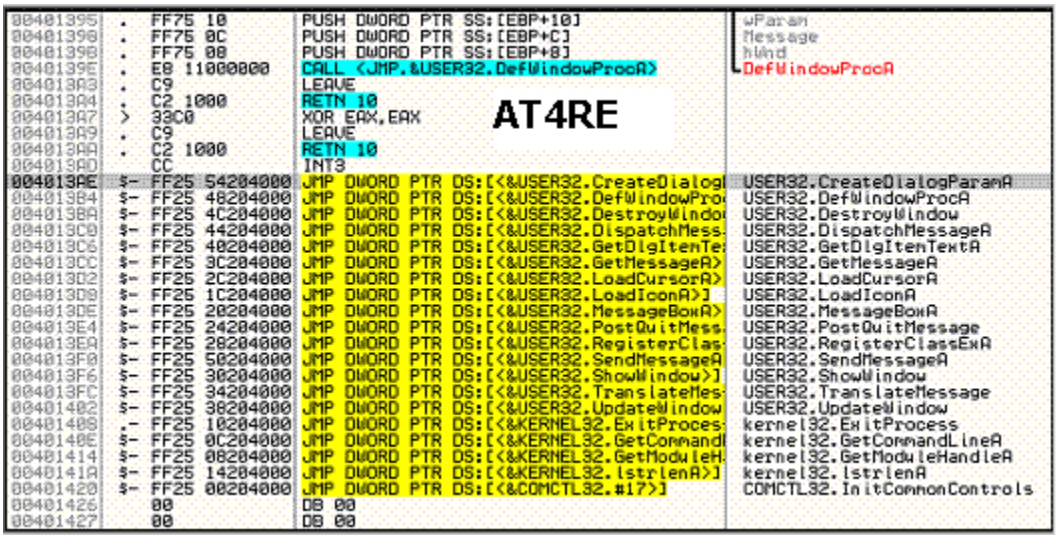
هناك exception أوقف عملية الـ tracing. إذن يفترض أن نضغط على shift+F9 لتشغيل البرنامج، لكن كما قلت هناك العديد من الـ exceptions لذا سنعود إلى Fig 5 ونقوم بوضع علامة صح على الخيار الثاني الذي تركناه.

والآن اضغط shift + F9، ستلاحظ بالأسفل أن olly وجود كلمة tracing لكنها ستختفي بسرعة فعملية الـ tracing لهذا الملف لا تتطلب وقتا كبيرا. ستجد أنك توقفت هنا :

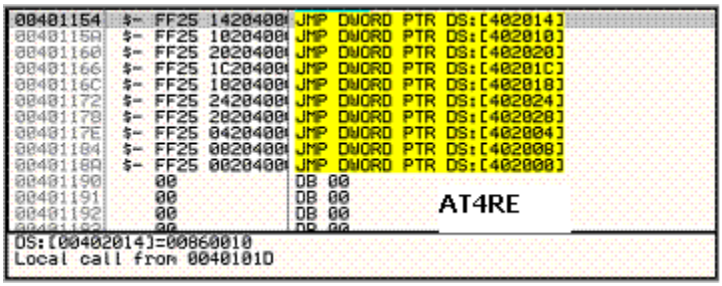


ها قد وصلنا إلى الـ OEP. سننتقل الآن إلى الخطوة الثانية.

الخطوة الثانية سنقوم فيها بإعادة الـ IAT إلى مكانه الأصلي يدويا. أفضل أن تقوم بفتح الملف testIAT، وهو ملف غير مضغوط.طبعا قم بفتحه باستخدام نسخة أخرى من olly غير التي نعمل عليها حاليا. ابحث فيه عن الـ IAT (لعلك كنت تراها دائما لكن دون أن تدرك أن هذه هي الـ IATs) ستجدها على شكل jumps متتابعة بدءا من العنوان 4013AE كما في الشكل التالي:

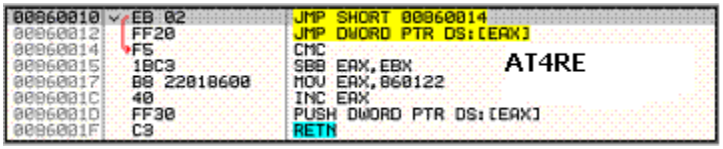


جيد والآن أغلق تلك النسخة من olly ولنعد إلى ضحيتنا asm. نريد أن نبحث عن الـ jmps IATs'. لا يوجد مكان محدد لكن غالبا تجدها في أول أو آخر البرنامج. في حالتنا هذه وجدناه في النهاية تماما، عند العنوان 401154.



لعلك لاحظت أنه هناك شيء ناقص، فكما ترى فآن olly لم يظهر أسماء دوال الـ API على اليمين كما في الملف الغير مضغوط السابق (Fig. 9). إذن دعنا نقوم بها يدويا. كما ترى عند تحديد القفزة الأولى يظهر في الأسفل 00860010. لنر ماذا يوجد هناك، اضغط زر انتر لنتتبع القفزة. أو يمكنك أن تضغط على CTRL + G وتدخل العنوان 860010 كما في الصورة التالية

بعد ذلك سنصل إلى هنا :

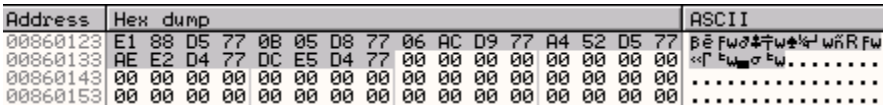


دعنا نرى.كما ترى فالقفزة الأولى تتجاوز القفزة التالية، وتأخذك إلى 860014. ثم يتم نقل القيمة 860122 إلى EAX ثم يتم تزويده بمقدار واحد، أي أن قيمته النهائية هي 860123. ثم في النهاية هناك أمر **RETN** للعودة حيث كنا.

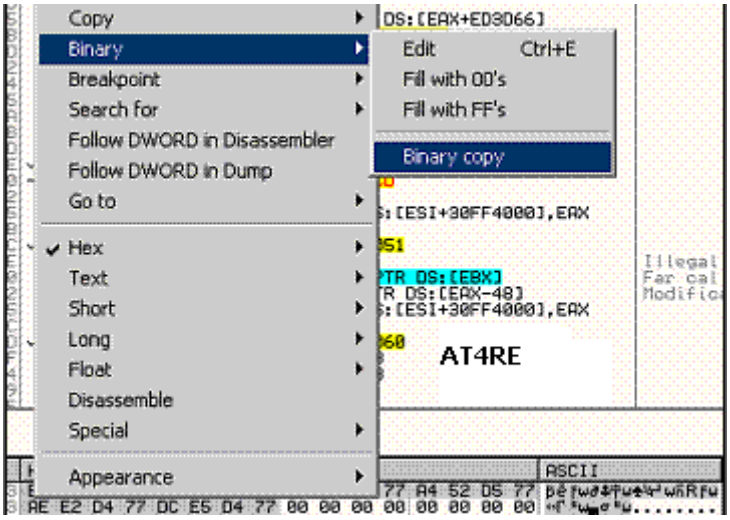
إذن نستنتج أن الـ IAT الـ "مرافق" للقفزة الأولى قد تم نقله إلى 860123. لا بأس دعنا نذهب إلى هناك. في قسم الـ data (أي القسم السفلي إلى اليسار) اضغط باليسار على أي مكان، ثم اضغط ctrl + G أو، اضغط باليمين واختر go to ثم expression كما في الصورة التالية :

والآن ستظهر لك النافذة التالية :

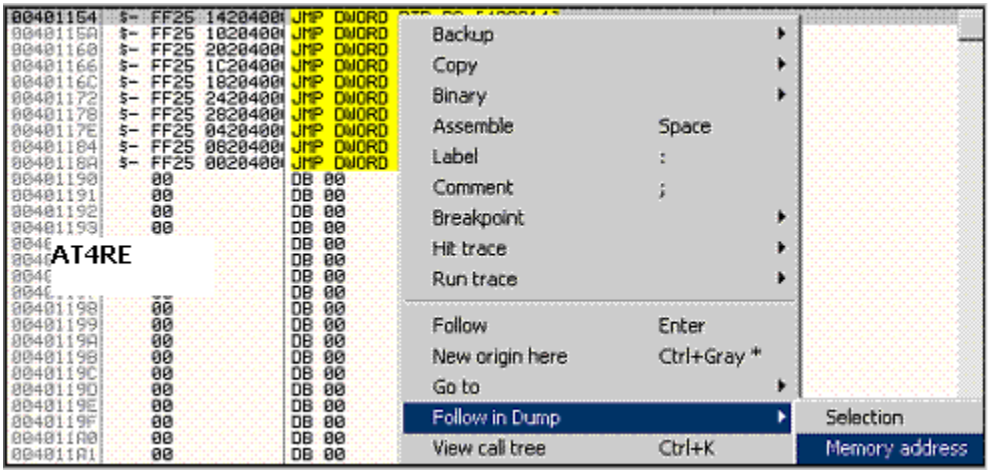
ها قد وصلنا :



إذن هذا هو الـ IAT (في الواقع هذا جزء منه.ستعرف المزيد لاحقا) قم بتحديده من البداية حتى آخره أي حتى تصل إلى الأصفار المتتابعة. والآن اضغط على التحديد باليمين واختر binary ثم binary copy.



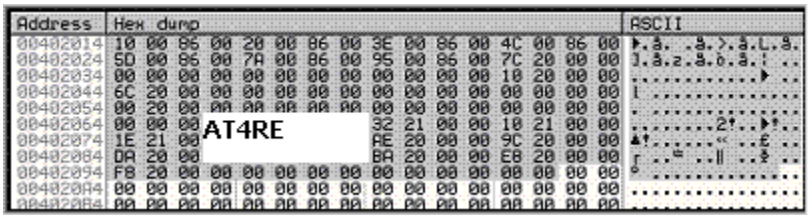
والآن اضغط باليسار على أي مكان في قسم الـ CPU، ثم اضغط علامة الناقص (-) كي نعد من حيث أتينا. الآن بعد أن عدنا لأول قفزة عند العنوان 401154 اضغط عليها باليمين ثم اختر follow in dump ثم memory address.



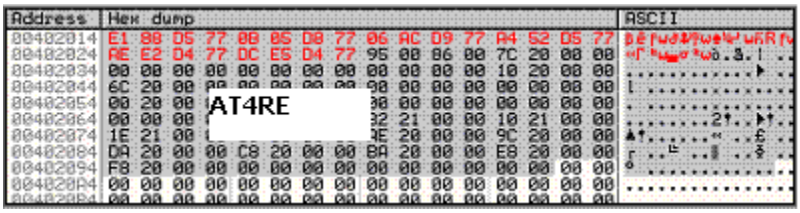
ستلاحظ أن قسم الـ data بالأسف تغير و أننا قد انتقلنا إلى مكان آخر (مكان الـ IAT الأصلي قبل نقله).

والآن نريد أن نلصق ما نسخناه إلى هنا. دعني أخبرك شيئا، عند اللصق أهم شيء أن تنتبه من أين تبدأ. فمثلا لو كنت قد نسخت 100 بايت وتريد لصقها عند العنوان 400000 مثلا، وقمت بتحديد 200 بايت وليس 100، بدءا من 400000 وحتى 400200 و ألصقت ما نسخته هناك، فلا مشكلة سيتم لصق المئة بايت من 400000 إلى 400100 وكأنك قمت بتحديد 100 بايت فقط.قد لا تتقن هذه العملية من المرة الأولى. لا تيأس فهذا ما حصل معي أيضا.كرر المحاولة إلى أن تتقن كل شيء جيدا.

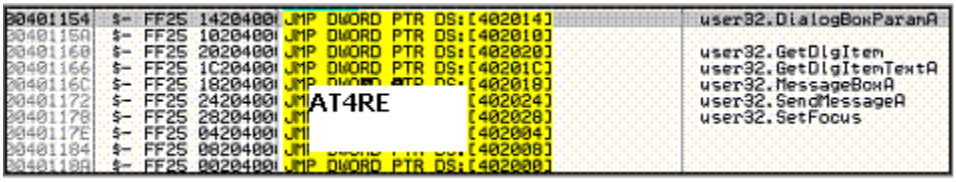
إذن ما سنقوم به هو تحديد كمية من البايتات (المهم أن تكون أكبر مما نسخناه) كالتالي :



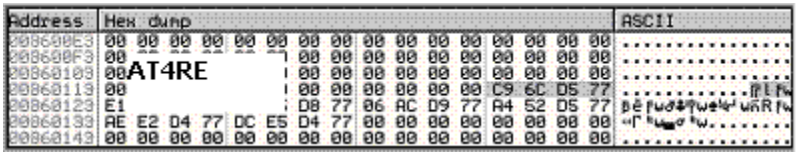
كما تلاحظ فما قمنا بتحديده يتجاوز حجم ما قمنا بنسخه. ألآن اضغط باليمين على التحديد واختر binary ثم binary paste. لا داعي لتوضيح ذلك بصورة فالأمر واضح. بعد اللصق سترى التالي :



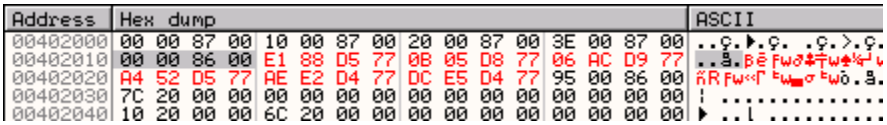
كما ترى تم لصق البايتات بالحجم الذي نسخنا ولم تتأثر بقية البايتات التي تم تحديدها. لكن ما الذي حصل بالأعلى ؟ دعنا نرى :



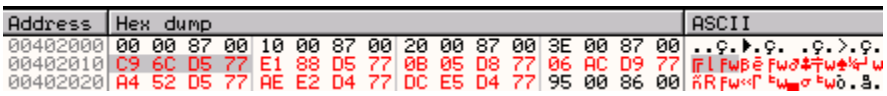
كما ترى فأول 7 دوال API تتبع ملف user32.dll لكن انتظر الدالة الثانية غير موجودة.حسنا لنعد إلى 860123، أي في قسم Data اضغط ctrl + G ثم 860123. والآن بعد أن وصلنا اصعد بضعة أسطر للأعلى :



أها. هذا ما كان ينقصنا. كما ترى هناك جزء ناقص (4 بايتات)، ولعلك لاحظت وجود 28 بايت أي 7 دوال وهذا فعلا ما وجدناه قبل قليل. إذن حدد الـ 4 بايتات الباقية علينا، قم بعمل binary copy ثم حدد أول قفزة (على عنوان 401154) على عنوان 40117E. والآن اصعد سطرا للأعلى. وحدد الأربع بايتات التي تسبق ما قمنا بلصقه في الخطوة السابقة :



كل ما عليك فعله الآن هو الضغط باليمين على هذه البايتات الأربعة ثم binary ثم binary paste.



لنر ما الذي حصل بالأعلى :

```
00401154  $- FF25 1420400( JMP DWORD PTR DS:[402014]    user32.DialogBoxParamA
0040115A  $- FF25 1020400( JMP DWORD PTR DS:[402010]    user32.EndDialog
00401160  $- FF25 2020400( JMP DWORD PTR DS:[402020]    user32.GetDlgItem
00401166  $- FF25 1C20400( JMP DWORD PTR DS:[40201C]    user32.GetDlgItemTextA
0040116C  $- FF25 1820400( JMP DWORD PTR DS:[402018]    user32.MessageBoxA
00401172  $- FF25 2420400( JMP DWORD PTR DS:[402024]    user32.SendMessageA
00401178  $- FF25 2820400( JMP DWORD PTR DS:[402028]    user32.SetFocus
0040117E  $- FF25 0420400( JMP DWORD PTR DS:[402004]
00401184  $- FF25 0820400( JMP DWORD PTR DS:[402008]
0040118A  $- FF25 0020400( JMP DWORD PTR DS:[402000]
```

هذا رائع. والآن سنكرر نفس العملية بالنسبة للجزء الثاني. حدد القفزة التالية أي التي على عنوان 40117E

سترى في الأسفل 870010، اذهب إلى هناك، سترى أن قيمة EAX النهائية هي 87007F، تتبع هذه القيمة في قسم Data، ستجد جزء من الـ IAT انسخه، ثم حدد القفزة على عنوان 40117E ثم اختر follow in dump ثم memory address والآن في مكان الـ IAT الأصلي قبل التغيير ألصق ما قمت بنسخه.ستلاحظ أن الـ IAT هذا يسبق مباشرة الـ IAT الخاص بملف user32.dll.

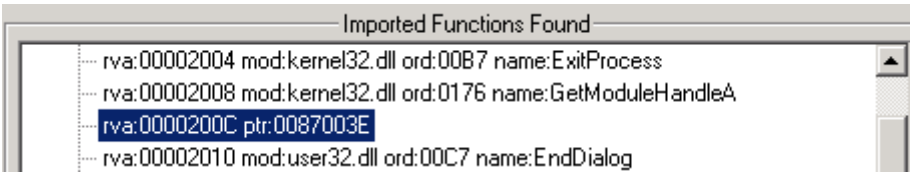
والآن سنحصل على دالتي API جديدتين خاصتين بملف kernel32.dll كما ترى :

```
00401154  $- FF25 1420400( JMP DWORD PTR DS:[402014]    user32.DialogBoxParamA
0040115A  $- FF25 1020400( JMP DWORD PTR DS:[402010]    user32.EndDialog
00401160  $- FF25 2020400( JMP DWORD PTR DS:[402020]    user32.GetDlgItem
00401166  $- FF25 1C20400( JMP DWORD PTR DS:[40201C]    user32.GetDlgItemTextA
0040116C  $- FF25 1820400( JMP DWORD PTR DS:[402018]    user32.MessageBoxA
00401172  $- FF25 2420400( JMP DWORD PTR DS:[402024]    user32.SendMessageA
00401178  $- FF25 2820400( JMP DWORD PTR DS:[402028]    user32.SetFocus
0040117E  $- FF25 0420400( JMP DWORD PTR DS:[402004]    kernel32.ExitProcess
00401184  $- FF25 0820400( JMP DWORD PTR DS:[402008]    kernel32.GetModuleHandleA
0040118A  $- FF25 0020400( JMP DWORD PTR DS:[402000]
```

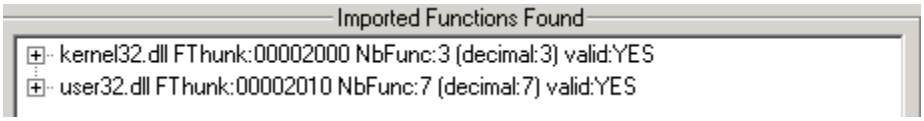
كما ترى لم يبق إلا دالة API واحدة فقط. أعتقد أنك تعرف ما يجب فعله. ألنتيجة النهائية ستكون كالتالي :

```
00401154  $- FF25 1420400( JMP DWORD PTR DS:[402014]    user32.DialogBoxParamA
0040115A  $- FF25 1020400( JMP DWORD PTR DS:[402010]    user32.EndDialog
00401160  $- FF25 2020400( JMP DWORD PTR DS:[402020]    user32.GetDlgItem
00401166  $- FF25 1C20400( JMP DWORD PTR DS:[40201C]    user32.GetDlgItemTextA
0040116C  $- FF25 1820400( JMP DWORD PTR DS:[402018]    user32.MessageBoxA
00401172  $- FF25 2420400( JMP DWORD PTR DS:[402024]    user32.SendMessageA
00401178  $- FF25 2820400( JMP DWORD PTR DS:[402028]    user32.SetFocus
0040117E  $- FF25 0420400( JMP DWORD PTR DS:[402004]    kernel32.ExitProcess
00401184  $- FF25 0820400( JMP DWORD PTR DS:[402008]    kernel32.GetModuleHandleA
0040118A  $- FF25 0020400( JMP DWORD PTR DS:[402000]    kernel32.lstrcmpA
```

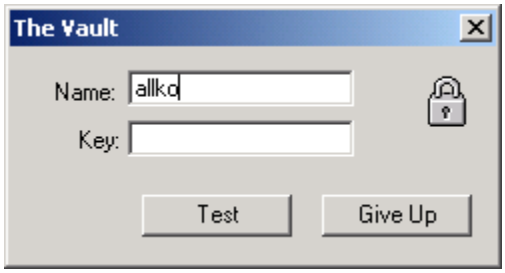
والآن دعنا ننتقل إلى الخطوة الثالثة و الأخيرة. من قائمة الـ plug ins اختر olly dump ثم dump (تأكد من إزالة علامة الصح في الأسفل من خيار IAT auto build). احفظ ملف الـ dump بأي اسم، أبق olly مفتوحا وشغل imprec. من القائمة العلوية اختر الملف الضحية (asm) ثم في خانة OEP ضع ما حصلنا عليه (أي 1000) ثم IAT autosearch ثم get imports، والآن اختر show invalid :



تعرف ما يجب فعله.حددها باليمين ثم cut thunks :



الخطوة الأخيرة هي fix dump والآن اختر الملف الذي قمنا بحفظه توا، شغل الملف الناتج.



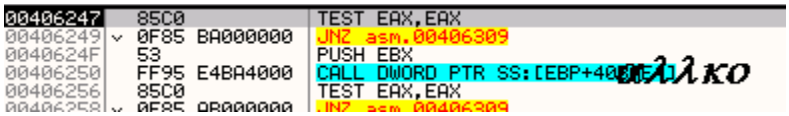
رائع لقد نجحنا !

4.2.2 . الطريقة الثانية :

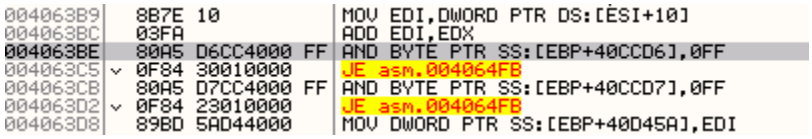
الخطوة الأولى لم تتغير. أيجاد الـ OEP. والآن تأكد أن الخيار الثاني ضمن exceptions غير معلّم، شغل الضحية asm ثم قم بتحديده. الآن نريد التوقف عند القفزة التي تحدد ما إذا كان البرنامج سيقوم بعمل الـ redirection أم لا، هذه القفزة نجدها بالقرب من دالة GetModuleHandleA لذا في الـ command bar ضع نقطة توقف كالتالي :

والآن shift + F9، ستتوقف هنا :

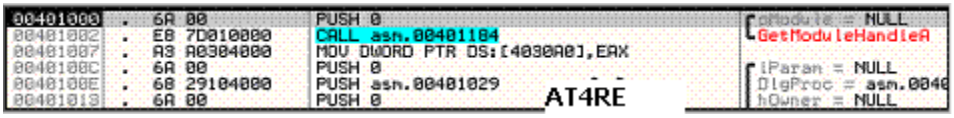
قم بإزالة الـ bp. نحن لا يهمنا دالة GetModuleHandleA بحد ذاتها. بل ما يأتي بعدها.تتبع حتى تصل إلى تعليمة **ret**، أو ببساطة اضغط على ctrl + F9، بعد أن وصلنا اضغط F8 لتصل إلى هنا :



انزل للأسفل حتى تجد تعليمة **AND** يليها **JE** يليها **AND** يليها **JE**. ستجدهم عند 4063BE



سنغير القفزة الأولى إلى jmp. إذن حددها ثم اضغط زر space على لوحة المفاتيح، وفي مربع التحرير الذي سيظهر قم بتغييرها. الآن كل ما عليك فعله هو الضغط على shift + F9.



هل ترى ذلك، إلى اليمين هناك تحليل يظهر دوال الـ API المستخدمة مما يعني أن كل شيء على ما يرام. وإذا نزلت للأسفل سترى أن كل الدوال موجودة.

الخطوة الثالثة لن نشرحها. فأنت تعرفها. عملية الـ dump المعتادة. أكمل بنفسك !!

<u>ننتقل الآن إلى ملف Delphi.</u>

البرنامج السابق كان مبرمجا بالاسمبلي. في الواقع الطريقتين اللتان سبق ذكرهما تنطبقان هنا دون أية مشاكل. فقط لاحظ في الطريقة الأولى اليدوية، أنك ستجد قفزات الـ IAT في عدة أماكن موزعة على البرنامج.جزء عند 401060 وهذه تحتاج إلى تصحيح يدوي، جزء آخر من القفزات موجود عند 404568، هذه لا تحتاج إلى تصحيح فيبدو أننا توقفنا عند نقطة بحيث كان الـ packer stub قد بدأ بتصحيحها، لكن ستلاحظ أن هناك دالة واحدة فقط من بينهم تحتاج إلى إعادة تصحيح (إعادة نقل).

بالنسبة لملف c++.

لن تستطيع تحديد مكان قفزات الـ IAT بسهولة. لكن إن أردت أن تصبح reverse engineer حقيقي يجب عليك أن تفك ضغطه. حاول مرة واثنتين وثلاثة. وستنجح بالنهاية.

أما عن ملف vb

فلا يوجد طريقة يدوية !! ما أقصده هو بعد فتح الملف بـ olly ثم وضع علامة صح على جميع الخيارات في exceptions ثم shift + F9 فإنك ستصل إلى OEP وتجد أن جميع الـ IATs قد عادت إلى مكانها الأصلي. يبدوا أنه في برامج الـ vb فإننا نتوقف عند نقطة يكون عندها الـ Packer قد أعاد كل شيء إلى مكانه الطبيعي.

بالطبع فالطريقة الثانية (تغيير JE إلى jmp) أيضا تنجح لكن لا داعي لها طالما أن الموضوع أسهل من ذلك بكثير.

وبهذا ينتهي الفصل الرابع.


الباب
الثالث
الفصل الخامس
exeshield

5. الفصل الخامس : EXESHIELD 3.6X

درس اليوم يتحدث عن packer/protector فريد من نوعه.

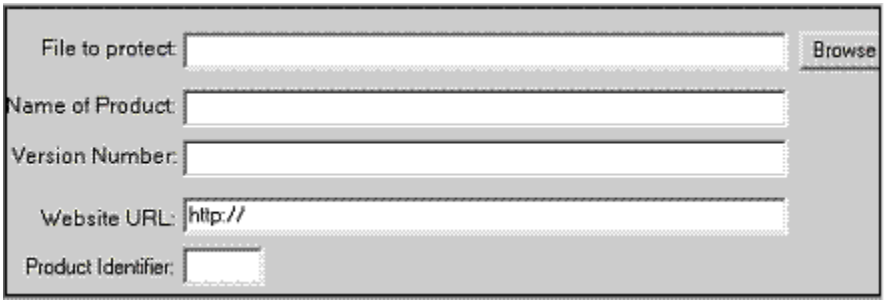
5.1. EXESHIELD...A CLOSER VIEW:

إن exe shield يعتبر من الـ packers الجيدة نسبيا التي تقدم لك وسيلة عملية لتحويل برنامجك إلى try before buy

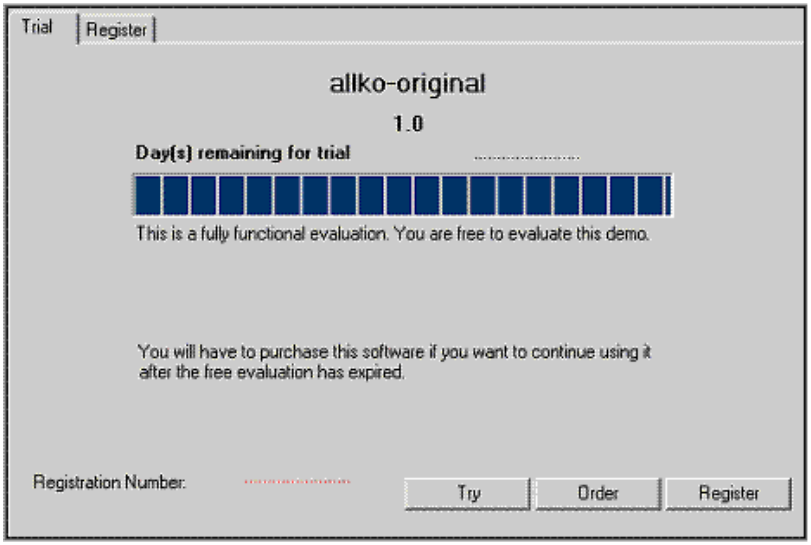
عند الحديث عن مميزات exe shield فإننا نذكر :

- Anti-debugging & Anti-monitoring Protection
- Sophisticated Encryption Technology
- Detects backdating or reinstallation to gain additional usage
- Secure Encryption Protection
- Reliable Machine Locking Protection
- Invisible software-based protection requiring no dongles
- Create evaluation copies by days or uses
- No source code modification
- No dependency files to distribute
- Lease your software
- Protect your application in less than 5 minutes

وحيث أنه من المهم أن تقوم بحماية بعض البرامج الصغيرة بنفسك وتتدرب عليها، لذا سأعلمك كيفية استخدام البرنامج لحماية برامجك الخاصة. عند فتح البرنامج ترى التالي :



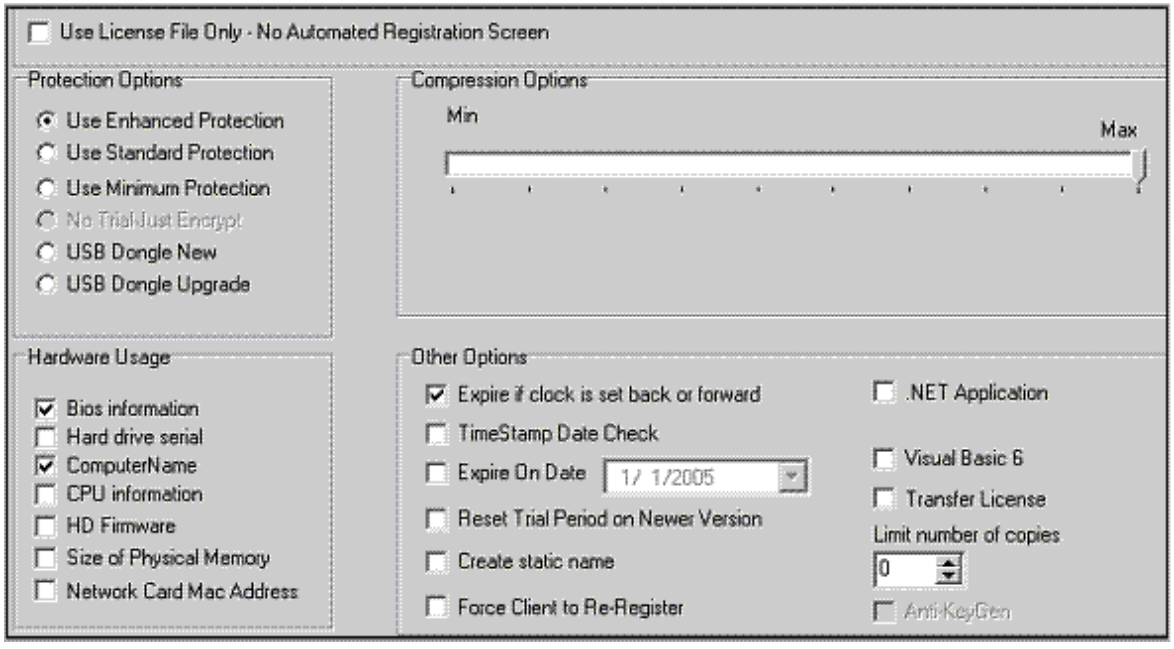
في الخانة الأولى. أضغط browse وابحث عن البرنامج الذي تريد حمايته. وليكن الملف الذي أرفقته بالحزمة والمسمى allko-original. في الخانات الباقية أكتب أي شيء أو اتركها فارغة. والآن لننتقل للصفحة التالية :



هذه الصفحة تخيرك بين عمل nag أو لا.

كل العبارات التي تراها يمكنك تغييرها. فقط اضغط على أي منها باليمين. ينطبق هذا على الـ trail و register

الصفحة التالية :

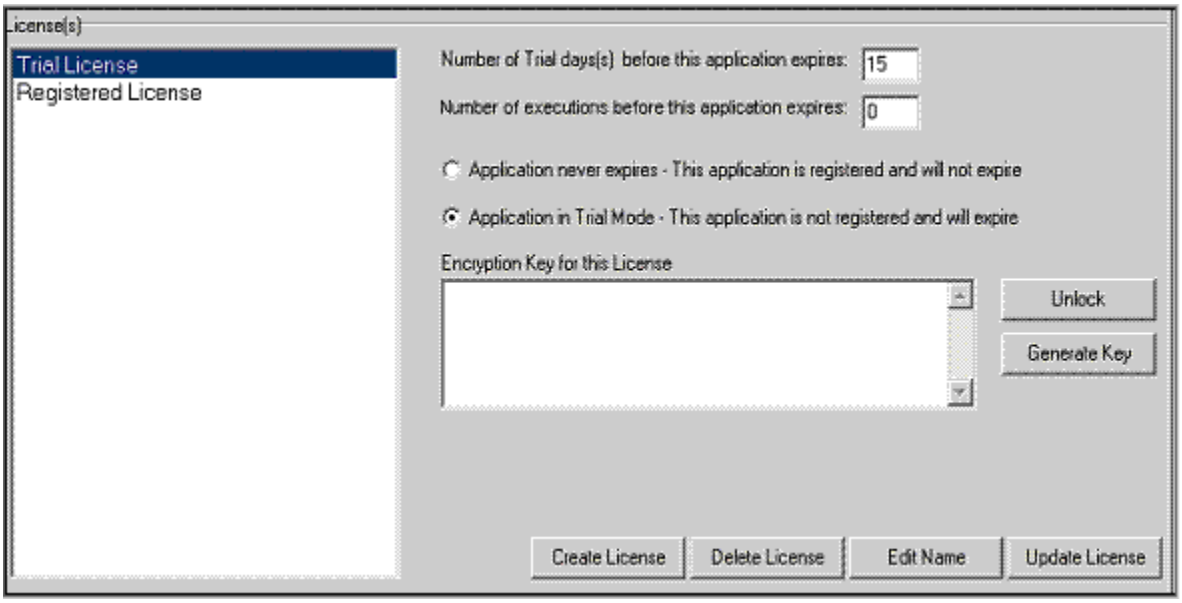


في الجزء العلوي الأيسر.. أختر أول خيار لحماية أكبر. في الجزء السفلي الأيسر هناك بعض الخيارات التي تتعلق بقراءة بعض المعلومات من الجهاز.. عموما هذا ليس موضوع حديثنا لذا اتركها كما هي.

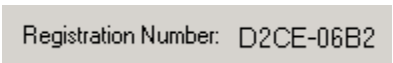
في الجزء السفلي الأيمن.تجد هناك خيارا معلما وهو ...Expired if clock is هذا يعني أنك إذا حاولت تغيير الوقت كي يستمر الجهاز بالعمل فإن محاولتك ستبوء بالفشل وسوف ينتهي البرنامج(expired)

باقي الخيارات ليست ذات أهمية كبيرة. لمزيد من المعلومات راجع ملفات المساعدة التابعة للبرنامج.

الصفحة ما بعد التالية :

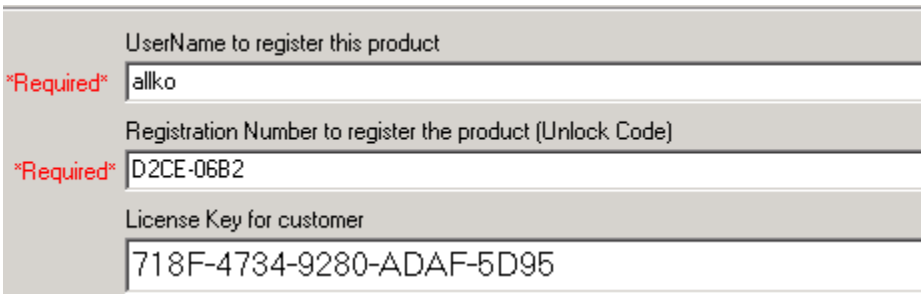


إلى اليسار تجد خيارين.هناك modes 2 الأول trial. اختره وعلّم على الخيار السفلي كما في الصورة. ثم اضغط على Generate Key. أيضا يمكنك تحديد عدد الأيام التي يبقى فيها البرنامج قابلا للتشغيل تحت وضع trial (افتراضيا، 15 يوم). والآن اختر الـ mode الثاني وهو registered واختر الخيار الأول أي أن البرنامج في هذا الوضع لن ينتهي عمله بعد فترة زمنية محددة. والآن اضغط على Generate Key. سيقوم exeShield بإنشاء "مفتاح"، يقوم على أساسه بتوليد الـ serial numbers. والآن من أعلى احفظ المشروع ثم اضغط protect. والآن جرب الملف الناتج.ستلاحظ خروج الـ nag وفي الأسفل ستلاحظ التالي :



هذا الرقم مميز لكل مستخدم أي أنه سيختلف في حالتك. والآن لتسجيل البرنامج اذهب إلى صفحة keys

من اليسار اختر Registered license وفي اليمين املأ البيانات. ثم اضغط على زر Generate Ulock code :



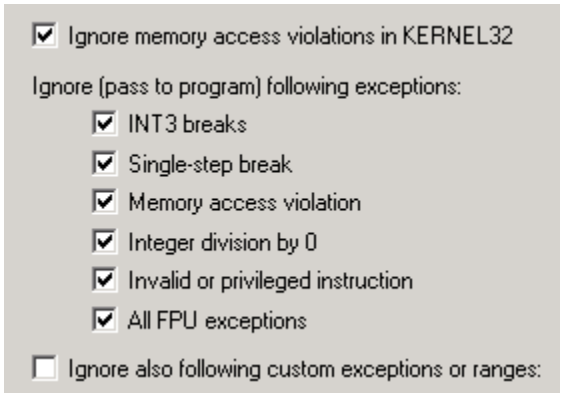
هذا هو السيريال الخاص باسم allko على جهازي..سيختلف بالنسبة لك. عموما هذا غير مهم بالنسبة لنا لكن أحببت أن تكون على دراية في كيفية استخدام البرنامج.

5.2. STEP #1 : PASSING THE PROTECTION LAYERS

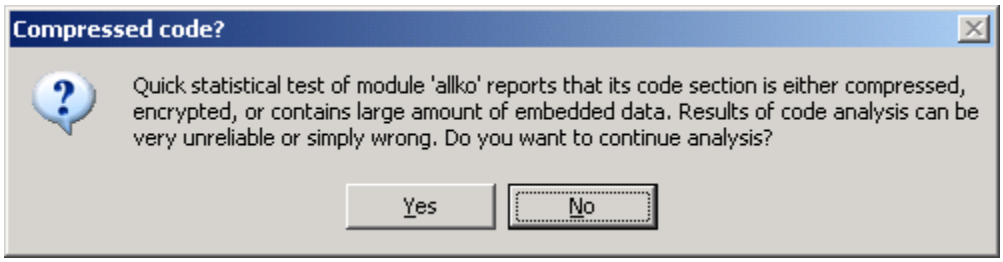

سنبدأ الآن بالعمل.كي لا يكون هناك اختلاف. أرفقت الملف بعد حمايته.تجده مع المرفقات باسم allko .

افحص البرنامج بـ PEiD 0.94 وأشدد على استخدام الإصدار الأحدث وهو 0.94 فالإصدار 0.93 لا يكشف exeshield.

جيد. ألآن افتح البرنامج باستخدام olly، مع ملاحظة أن تكون خيارات الـ exceptions كالمعتاد. أي هكذا :

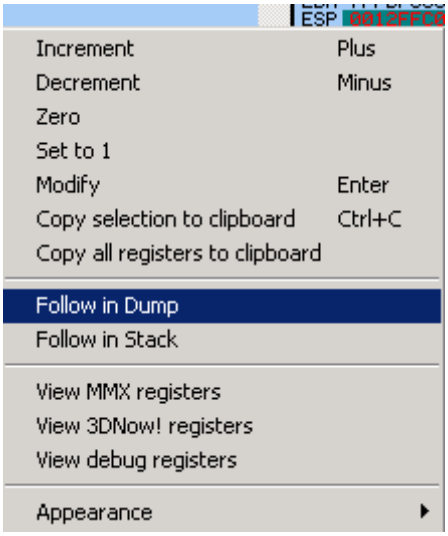


والآن شغل البرنامج :

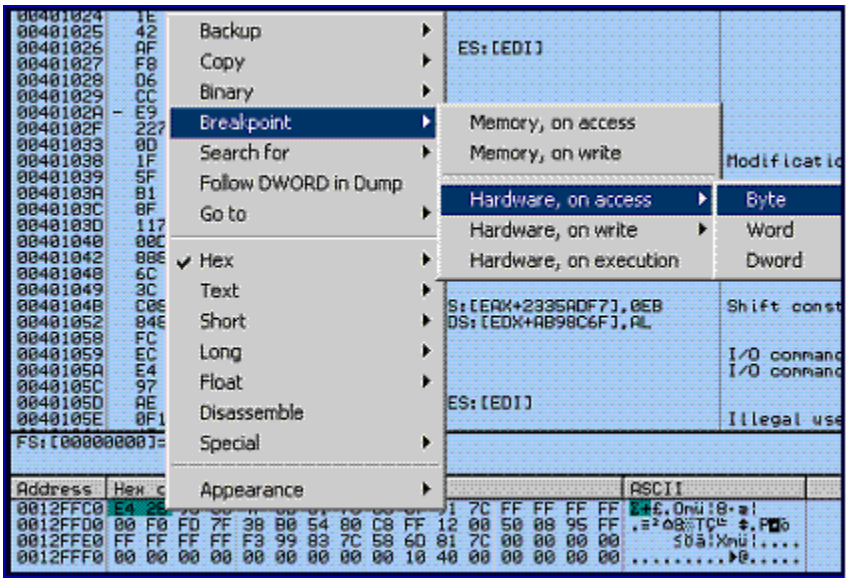


اختر NO.(أما إذا اخترت yes فسترى أن الكود مبهم، وستضطر للضغط باليمين ثم اختيار analysis ثم remove analysis from module.)

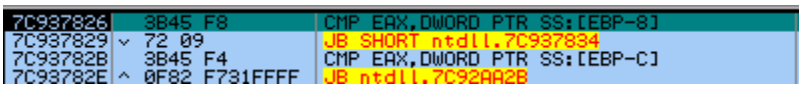
والآن اضغط F8 مرتين. والآن بعد تنفيذ **PUSH** EAX، اذهب إلى قسم registers - وكما كنا نفعل في المرات السابقة – اضغط على ESP لتحديده ثم اضغط باليمين واختر Follow in Dump :



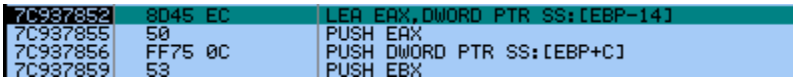
والآن في قسم Hex Dump بالأسفل، حدد أول بايتين ثم اضغط عليهما باليمين واختر Break Point ثم hardware on access ثم word (أو Dword. لن تؤثر كثيرا..)



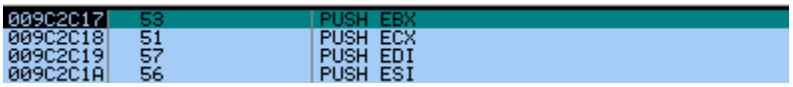
والآن اضغط F9 لتصل إلى هنا :



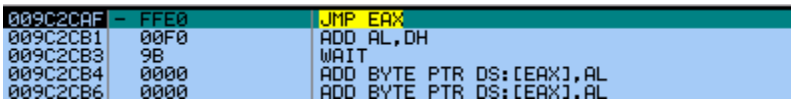
F9 مرة أخرى :



مرة ثالثة :

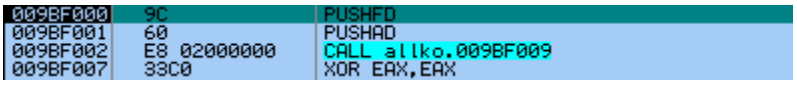


المرة الأخيرة :

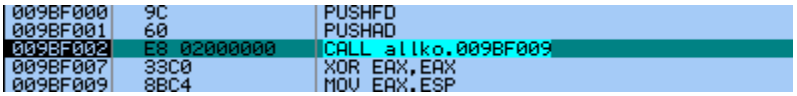


والآن اضغط F7 (أو F8) لتنفيذ هذه القفزة.

ستصل إلى هنا :



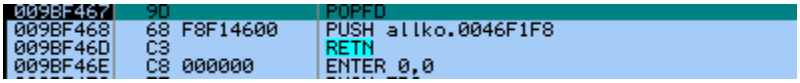
والآن قم بالضغط على F8 مرتين، حتى تنفذ أمر **PUSHAD**.



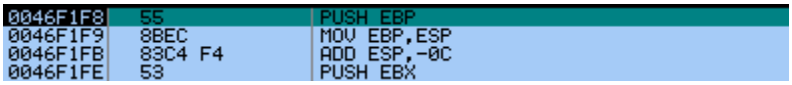
في قسم Registers، ستلاحظ أن قيمة ESP قد تغيرت وأصبحت ملونة باللون الأحمر. ألآن قم بوضع نقطة توقف كالتي فعلناها سابقا. لا حاجة لأن نعيد نفس الخطوات من جديد!

إذا كنت lamer ولم تفهم، فسنعيد باختصار : اضغط على ESP باليمين ثم follow in dump ثم في قسم hex dump حدد أول بايتين واضغط عليهم باليمين، اختر breakpoint ثم hardware on access ثم word.

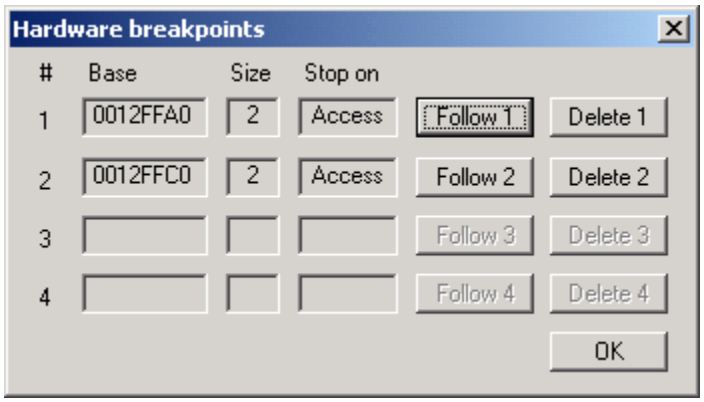
والآن اضغط F9 :



تتبع (F8) حتى تصل إلى هنا :



بهذا تكون الطبقة الثانية من الحماية قد انتهت. دعنا نتخلص من الـ HWBP فلسنا بحاجة لها. من قائمة debug اختر HardWare BreakPointsليخرج لك الشكل التالي :



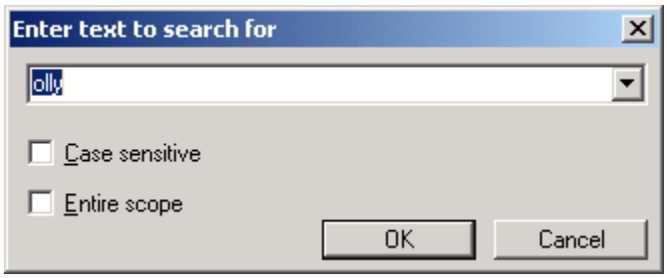
اضغط زر delete بجانب كل نقطة توقف لحذفها.

STEP #2 : PASSING ANTI DEBUGGING TRICKS .5.3

والآن نريد التخلص من تلك الخدع السخيفة. أستخدام الـ plug in وحده لن يكفي. لذا دعك منها.

الطريقة التي يستخدمها هذا الـ packer طريقة بدائية.هم في الواقع 2 tricks الأولى تعتمد على دالة IsDEbuggerPresent وسنتخلص منها ببساطة باستخدام الـ plug in المعروفة. من قائمة Plug ins اختر IsDebugPresent ثم اختر hide. والثانية تعتمد على دالة FindWindowA، هذه الدالة تبحث على جميع البرامج المفتوحة من خلال string معين يعطى لها كبارامتر. مثلا كي تكتشف وجود olly لاحظ الجملة التي توجد بالأعلى : OllyDbg. إذن تكون هذه هي الـ string التي تأخذها دالة FindWindowA وتبدأ البحث، فإذا وجدت برنامجا بهذه المعطيات فإنها تتصرف على ذلك الأساس.(مثلا تخرج لك nag.)

والآن اضغط باليمين واختر search for ➔all referenced text strings ثم – بعد خروج النافذة الجديدة - اضغط زر home على لوحة المفاتيح أو اصعد بالماوس للأعلى (كي يشمل البحث كل الـ strings الموجودة) ثم ابحث عن OllyDBG.هكذا :



تأكد من إزالة علامة الصح بجانب أي من الخيارين. هذا أمر ضروري كي نحصل على نتائج صحيحة. والآن تنقّـل من string لأخرى بواسطة ctrl+L..

ستكون النتيجة 6 عبارات كالتالي :

ASCII "**OLLYDBG**"	46428C
ASCII "**OllyDbg - [CPU]**"	46429A
ASCII "**OLLYDBG**"	4644C1
ASCII "**OllyDbg - [CPU]**"	46A208
ASCII "**OLLYDBG**"	46A21C
ASCII "**OLLYDBG**"	46ABCD

يبدوا أن المبرمجين كانوا حذرين. فقد أخذوا الحالتين بعين الاعتبار. ألحالة الأولى تكون الجملة الموجودة بأعلى نافذة OLLY هي OLLYDBG" والحالة الثانية تكون" [CPU] - OllyDbg" (يبدوا أن هذه كانت في الإصدارات القديمة.) أي أنهم يريدون اكتشافنا بأي وسيلة ممكنة !!

حسنا بالنسبة لـ "[CPU] - OllyDbg" دعك منها. لأنها ليست هي التي ستفضح أمرنا p:، إذن لدينا 4 أماكن يوجد فيها "OLLYDBG".

اضغط double click على الجملة الأولى عند عنوان 46428C كي تأخذنا إلى هنا :

```
0046428A   6A 00          PUSH 0
0046428C   68 68434600    PUSH allko.00464368          ASCII "OLLYDBG"
00464291   E8 EA27FAFF    CALL allko.00406A80          JMP to user32.FindWindowA
00464296   85C0          TEST EAX,EAX
00464298   75 0C          JNZ SHORT allko.004642A6
0046429A   68 70434600    PUSH allko.00464370          ASCII "OllyDbg - [CPU]"
0046429F   6A 00          PUSH 0
004642A1   E8 DA27FAFF    CALL allko.00406A80          JMP to user32.FindWindowA
004642A6   85C0          TEST EAX,EAX
004642A8   75 0C          JNZ SHORT allko.004642B6
```

كما ترى، هناك بالأعلى تجد "OLLYDBG" وفي الأسفل هناك "[CPU] - OllyDbg". دعك من الثانية. بالنسبة للأولى كما تلاحظ هناك قفزة عند العنوان 464298، إذن اعكسها من **JNZ** إلى **JZ** لتحصل على :

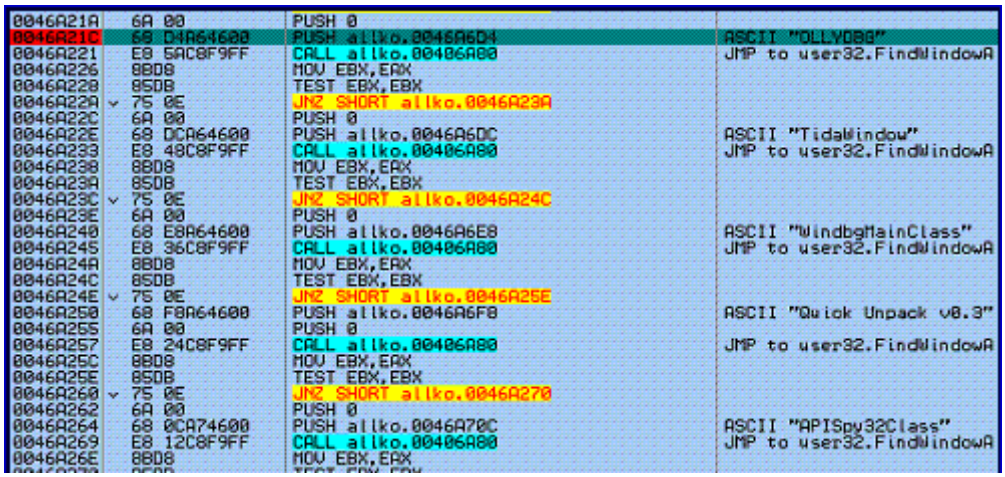
```
00464296   85C0          TEST EAX,EAX
00464298   74 0C          JE SHORT allko.004642A6
0046429A   68 70434600    PUSH allko.00464370          ASCII "OllyDbg - [CPU]"
0046429F   6A 00          PUSH 0
```

جيد. والآن لنتابع. أفتح نافذة الـ text strings مرة أخرى وانتقل للـ string التالية عند عنوان 4644C1. أضغط عليها مرتين بالزر اليسار لنذهب إلى هنا :

```
004644BF   6A 00          PUSH 0
004644C1   68 4C454600    PUSH allko.0046454C          ASCII "OLLYDBG"
004644C6   E8 B525FAFF    CALL allko.00406A80          JMP to user32.FindWindowA
004644CB   85C0          TEST EAX,EAX
004644CD   7E 13          JLE SHORT allko.004644E2
```

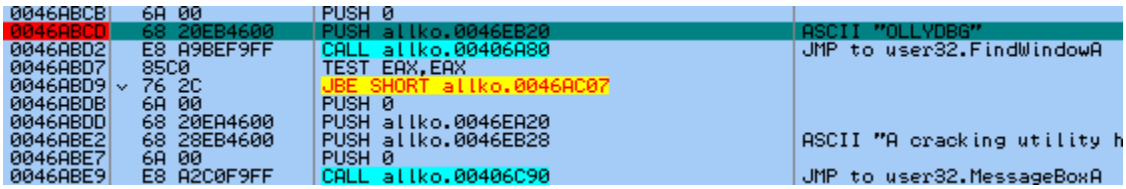
أعتقد أن كل شيء واضح. أعكس JLE إلى JNLE (أو إلى JG.).

جيد. ألى العنوان التالي. 46A21C :



هل ترى ما أرى !!! يبدو أن المبرمج على دراية بجميع برامج الكراك !!! في الصورة يظهر جزء من البرامج التي يحاول التصدي لها (للأسف محاولة فاشلة :)) هناك olly و TideWindow و exescope وغيره. ما نريده هو تغيير القفزة عند العنوان 46A22A من **JNZ** إلى **JZ**.

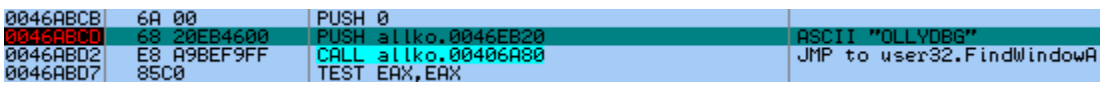
والآن سننتقل للعنوان التالي والأخير.46ABCD .



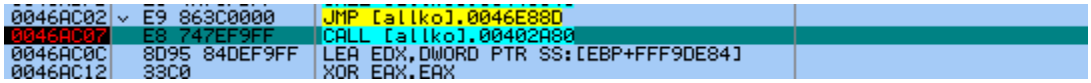
هل ترى تلك الجملة بالأسفل ؟ وهل ترى تلك الـ **CALL** إلى دالة MessageBoxA؟ يبدوا أنها nag تخرج لنا إذا تم اكتشاف أي أداة للكراك. لا بأس. ما نريده هو القفزة عند العنوان 46ABD9. أعكسها من JBE إلى JNBE..

والآن لنضع نقطة توقف بعد هذه الإجرائية. ولتكن عند وجهة القفزة. أي عند 46AC07. أذن انزل للأسفل 5 أو 6 أسطر تقريبا وضع نقطة التوقف. والآن شغل البرنامج F9.

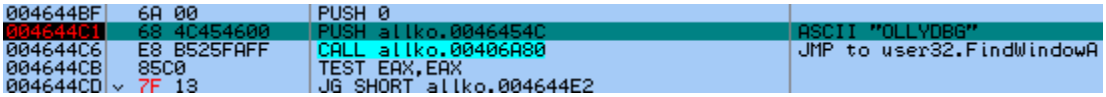
ها قد توقف هنا :



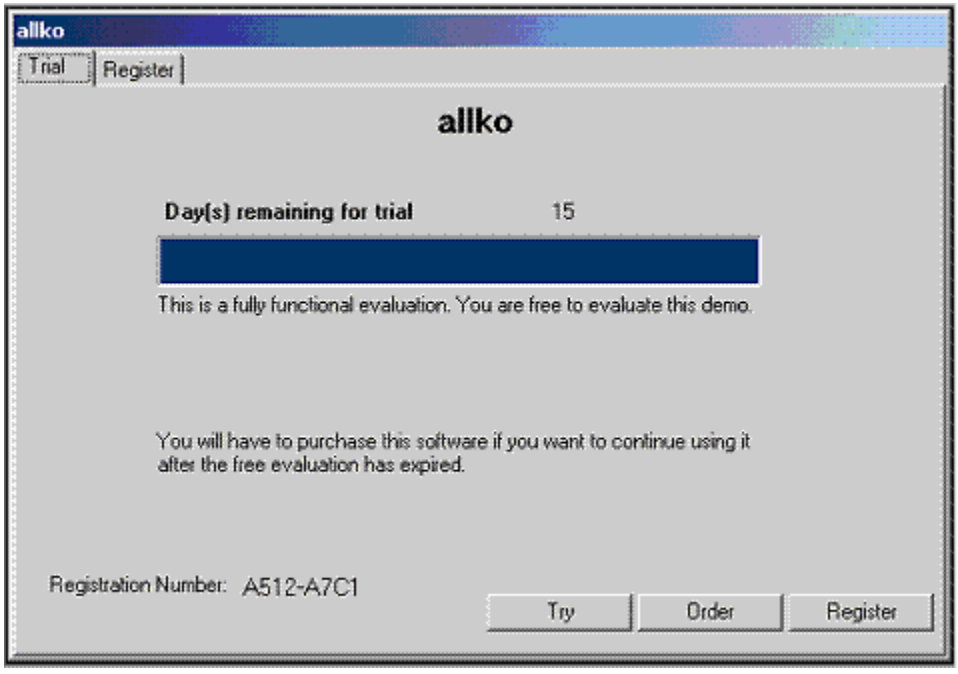
F9 مرة أخرى :



مرة أخرى :



مرة أخرى وستلاحظ خروج الـ nag :



والآن قبل الضغط على أي شيء لنفكر قليلا. من بين الأماكن الأربعة التي وضعنا نقاط توقف عندها.. اثنان فقط توقفنا عندهما.. أي أن اختباري فقط تم العمل بهما. و السبب في هذا قد يكون وجود خاصية معينة في الـ Packer تسمح بجعل الحماية أقوى. أو ربما يكون سببا آخر. عموما هذا لن يهمنا كثيرا.

STEP #3 : RESTORING STOLEN BYTES . 5.4

والآن وقبل الضغط على try هناك دالتان نريد وضع نقاط توقف عليهما. ألأولى هي VirtualProtectEx

لنر تعريفها في msdn :

VirtualProtectEx (	
HANDLE hProcess,	مقبض (handle) للـ process التي نريد تغيير نوع الحماية على الذاكرة الخاصة بها.
LPVOID lpAddress,	مؤشر (pointer) للـ Base Address للـ region of pages التي نريد تغيير نوع الحماية عليها.
SIZE_T dwSize,	حجم المنطقة التي نريد تغيير نوع الحماية عليها. (بالبايت)
DWORD flNewProtect,	خيارات الـ The memory protection
PDWORD lpflOldProtect);	مؤشر إلى متغير يستقبل نوع الحماية السابقة.

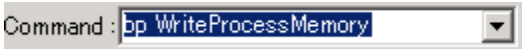
والثانية هي WriteProcessMemory.

لنلق نظرة عليها :

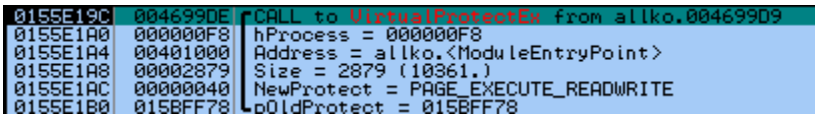
WriteProcessMemory(	
HANDLE hProcess,	مقبض (handle) للـ process التي نريد تغيير نوع الحماية على الذاكرة الخاصة بها.
LPVOID lpBaseAddress,	مؤشر إلى الـ base address حيث سيتم كتابة البايتات. قبل بدء عملية الكتابة، يقوم النظام بالتحقق من أن البيانات في الوجهة قابلة لإعادة الكتابة.
LPCVOID lpBuffer,	مؤشر إلى العنوان الذي يحتوي على البايتات التي سيتم نسخها.
SIZE_T nSize,	عدد البايتات التي سيتم نسخها.
SIZE_T* lpNumberOfBytesWritten);	مؤشر إلى المتغير الذي سيستقبل عدد البايتات التي تم نقلها إلى العنوان المحدد. هذا البارامتر اختياري.

والآن ضع نقطتي التوقف :

و



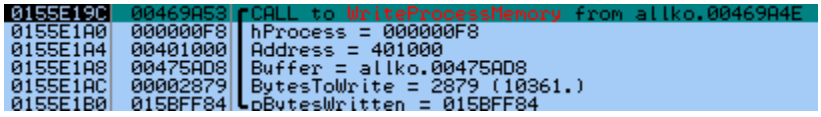
والآن نحن جاهزون للضغط على try :



كما ترى فقد توقفنا عند دالة VirtualProtectEx.تمعن في البارامترات جيدا.

ما يقوم به exe shield هو إنشاء نسخة مؤقتة. tmp من البرنامج الأصلي، وينشئها في نفس المجلد الذي توجد فيه النسخة الأصلية، لكن هذه النسخة المؤقتة لا تكون كاملة، بل هناك جزء من البايتات مفقود – وهي ما نسميه بـ stolen bytes – لذا فإنه يستخدم دالة WriteProcessMemory لنسخ تلك البايتات من النسخة الأصلية إلى المؤقتة. لكن لنسخ تلك البايتات يجب أن يغير "حماياتها" أو لنقل يغير الـ attribute، وذلك يتم بواسطة دالة VirtualProtectEx.قد تتساءل. لم لا ننتظره حتى يقوم بنسخ كل البايتات ثم نأخذ الملف الناتج جاهزا ؟ لن ينفع ذلك جرب بنفسك. لأنك لو فتحت الملف المؤقت بعد تشغيل البرنامج لن تجد به أية بايتات (أقصد لن تجد البايتات المسروقة به) فتلك البايتات لا تنسخ مباشرة إلى الملف المؤقت بل يمكنني أن أشبه لك الفكرة – مجرد تشبيه – بأن exeshield يأخذ البايتات من البرنامج الأصلي ويضعها بالذاكرة وكلما احتجناها يتوجه بنا إلى هناك. أي أنها غير موجودة في البرنامج نفسه.هذا طبعا مجرد تشبيه قد يكون خطأ. في الواقع لقد تساءلت كثيرا حول هذه النقطة لكن لم أجد جوابا شافيا رغم أنني وجهت السؤال في العديد من المنتديات الأجنبية):

في بعض البرامج تجد الـ OEP للنسخة المؤقتة في بارامتر Address كما في الصورة.. لكن هنا لا يوجد.. لا بأس نستطيع إيجاده من مكان آخر. والآن ركز على البارامتر الثالث Size.هذا هو حجم البايتات المسروقة. 2879 بالهكس.جيد الآن اضغط F9 :



ها قد توقفنا عند دالة WriteProcessMemory. ألبارامتر الثاني هو الـ OEP للملف المؤقت.كما ترى فهو 401000

والآن هل ترى البارامتر الثالث Buffer؟ هذا عنوان الذي تبدأ منه البايتات المسروقة في الملف الأصلي. أنه 475AD8

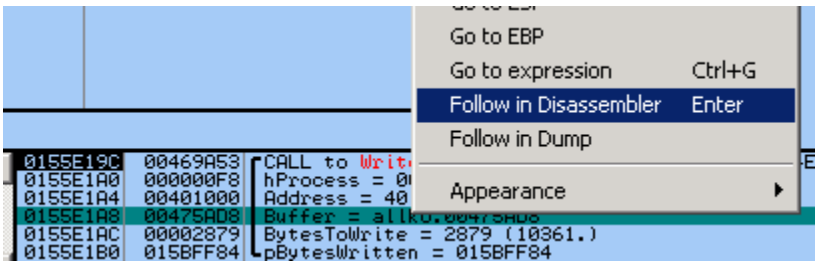
أما البارامتر الرابع BytestoWrite فهو يؤكد لنا ما رأيناه في الدالة السابقة.حجم البايتات المسروقة يساوي 2879 هكس.

قبل أن نكمل هناك خطوة يجب القيام بها. لعلك مللت من سماعي أقول "الملف المؤقت". أذن اذهب إلى المجلد الذي وضعت به البرنامج (أقصد البرنامج المحمي، المسمى allko). تأكد أن خيار إظهار الملفات المخفية مفعل (إن لم تكن تعرف كيف فراجع الملاحظات بنهاية الدرس.) سترى التالي :

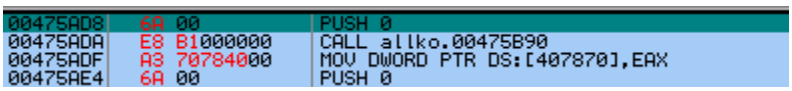
هل رأيت الملف المؤقت الذي تحدثت عنه سابقا.ها هو. لاحظ أن اسمه يبدأ بـ xshld و يحتوي على رقم، وهذا الرقم قد يختلف لديك.. ألآن أنشه مجلدا جديدا. وضع الملف المؤقت فيه، وقم بتغيير اسم الملف المؤقت إلى allko.exe أي إلى اسم الملف الأصلي وستلاحظ تغير أيقونته إلى الأيقونة الأصلية.

جيد. لنتابع العمل.

اضغط باليمين على البارامتر Buffer واختر Follow In Disassembler :

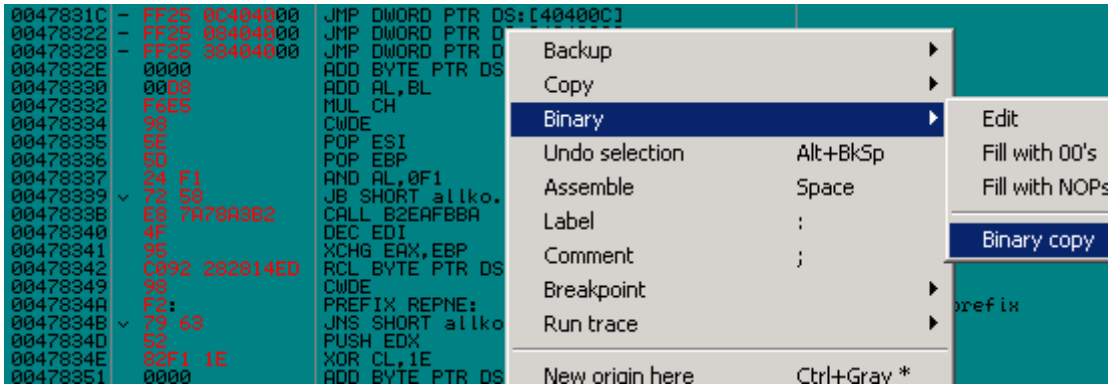


سنصل إلى هنا :

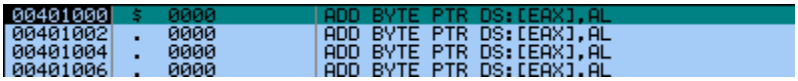


نحن نعرف أن الحجم 2879، إذن 475AD8 + 2879 = 478351

أي أن البايتات المسروقة تبدأ من 475AD8 وحتى 478351.كل ما عليك فعله هو تحديد تلك البايتات من 475AD8 وحتى 478351، والآن اضغط باليمين واختر Binary ثم binary copy



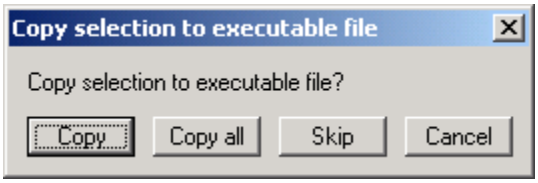
الآن افتح نسخة ثانية من olly وبالطبع يا عزيزي إياك أن تغلق النسخة الأولى !!! الآن في النسخة الثانية من olly افتح الملف المؤقت.



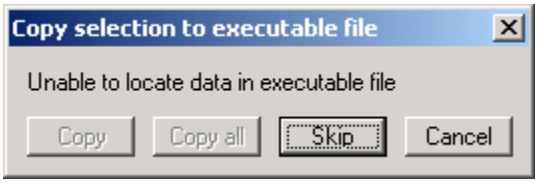
كما تلاحظ توقفنا عند الـ OEP. والآن نريد أن نلصق ما نسخناه. 403879 = 2879 + 401000 أي : قم بالتحديد من 401000 وحتى 403879. والآن اضغط باليمين على التحديد واختر binary ثم binary paste.

والآن علينا أن نحفظ التغييرات. أضغط باليمين على التحديد واختر copy to executable ثم all modifications

ستخرج لك هذه النافذة :



اختر copy all ستخرج لك هذه :



يبدوا أن olly يواجه مشكلة. لا تقلق فقط اضغط cancel. والآن في النافذة الجديدة التي ظهرت، اضغط باليمين ثم اختر save file واحفظه بأي اسم (وأفضل أي اسم غير اسم الملف الأصلي كي نميز بين الملفات الثلاثة.سمه مثلا allko_un.

ها قد انتهينا. بخصوص الـ IAT فـ exe shield لا يقوم بعمل أي تغيير عليه. أي لا حاجة لأي تغييرات كما فعلنا في الدرس السابق مع tElock 0.98. والآن جرب الملف الناتج.رائع، نجحنا !

وبهذا ينتهي الفصل الخامس.


الباب
الثالث
الفصل السادس
Scripting

6. الفصل السادس : SCRIPTING

6.1. مقدمة:

في هذا الدرس سنشرح كيفية كتابة ما يسمي ب Scripts في برنامج ollydbg

فقد أصبحت الـ Scripts طريقة قوية للإنجاز بعض المهام و الأعمال في olly بشكل أوتوماتيكي و هذه الأعمال كانت تأخذ الكثير من الجهد و الوقت.

ولغة ال Script و التي تعرف ب scripting Language هي شبيهة جدا بالأسمبلي حيث أنها تحتوي و على 97 أمر يمكن من خلالهم عمل كل شيء في OllyDbg.

6.2. البداية:

أول شيء في ال Scripts هي المتغيرات Variables فمثلا إذا أردت استرجاع حجم كود البرنامج لتستخدمه لاحقا فإن عليك في البداية ان تعرف المتغير و تجعل له اسما مناسبا:

Var codebase

إن الأمر var يعرف المتغير كما انه من الممكن ان تسمي المتغير باي اسم يخطر ببالك و لكن من المستحب أن تكتب أسماء لها دلالة حتى تسهل عليك الأمر و تتجنب الأخطاء.

ولكي تقوم بحفظ شيء فانك سوف تستخدم الدالة $Result وهي تقوم بإعادة القيم إلي باقي الدوال وبذلك يكون الشكل العام لل Script كما يلي

- Declare variable.
- Do some action and store result in $RESULT
- Transfer $RESULT to a variable.

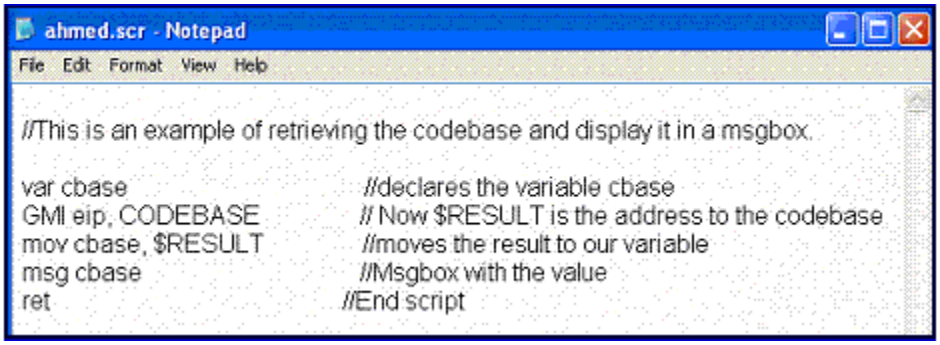
6.3. مثال:

هذا المثال لعرض Code BASE و إظهارة في رسالة msgbox

```
//This is an example of retrieving the codebase and display it in a msgbox.

var cbase                //declares the variable cbase
GMI eip, CODEBASE        // Now $RESULT is the address to the codebase
MOV cbase, $RESULT       //moves the result to our variable
```

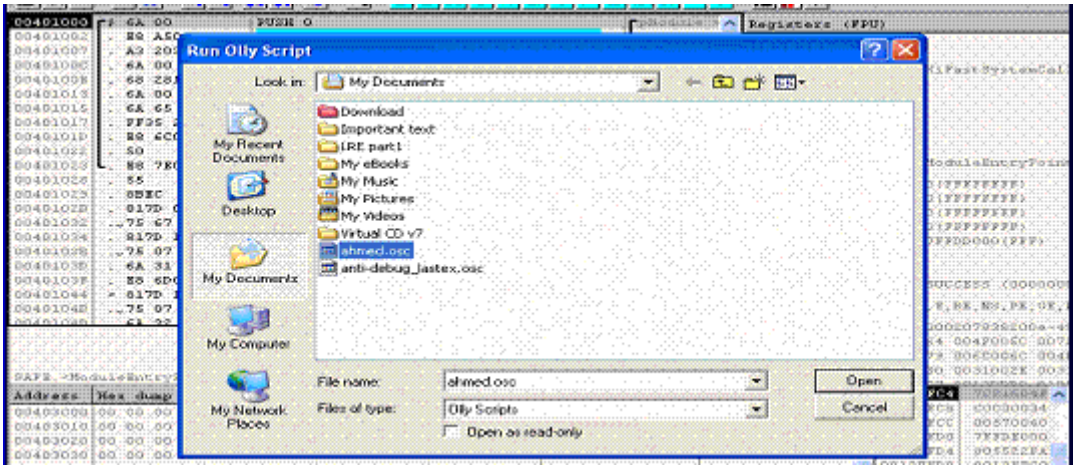
```
msg cbase          //Msgbox with the value
ret                //End script
```

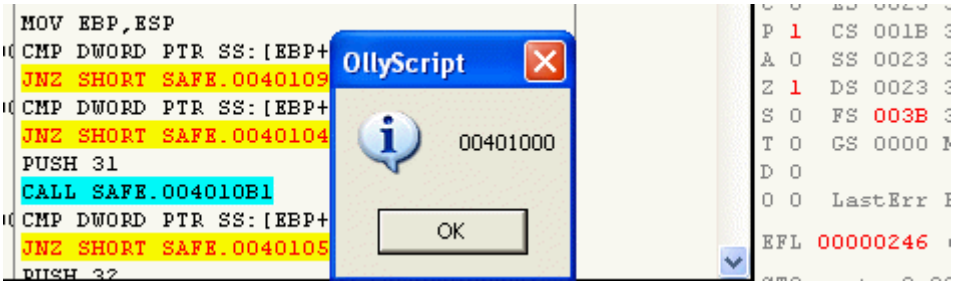
أول شيء قم بفتح المفكرة واكتب هذا الكود كما هو فيها بهذا الشكل



احفظ الكود بهذا الامتداد.OSC

قم بفتح OllyDbg وشغل ال Script وسوف تري النتيجة





وهذه بعض أوامر التشغيل التي يمكن استخدامها في ال script

RUN	Execute F9 in Ollydbg
STO	Execute F8 in OllyDbg.
STI	Execute F7 in OllyDbg.
RTR	Executes "Run to return" in OllyDbg

و يكمن للـ Script تنفيذ أمر معين حتى يتحقق شرط من الشروط و يتم ذلك عن طريق أمر المقارنة **CMP** و بعد المقارنة نستخدم الـ conditional jumps المعروفة

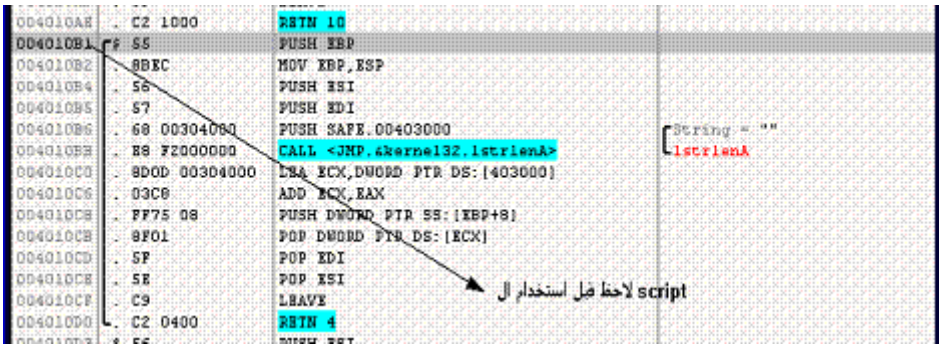
JNE	Jump not equal
JMP	Jump
JE	Jump equal
JBE	Jump If Below or Equal
JB	Jump below
JA	Jump above

6.4 . مثال 2

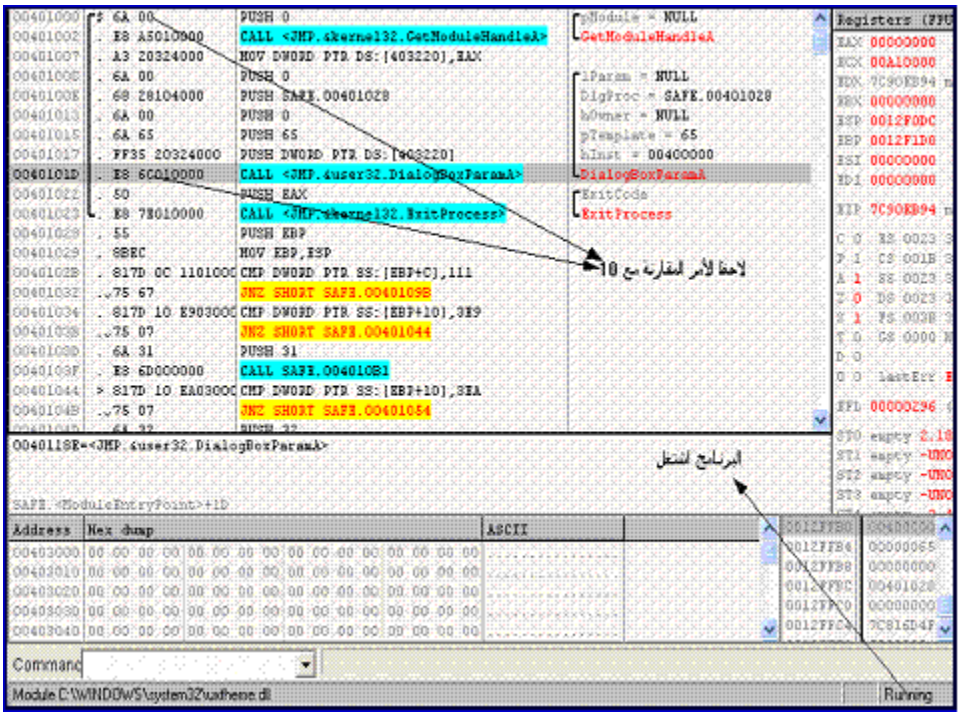
هذا مثال على استخدام ال conditional jumps

```
var counter      //declares the variable counter
start:           //Putting a ':' after the text transform the text into a name of a label This is useful so we
                 can make jumps to specific parts of the code.
CMP counter,10 //Compares the variable counter with 10
ja finish        //jump if above 10.
sto              //executes F8 in OllyDbg
INC counter      //Increase our counter by 1, otherwise endless loop
jmp start        //If the counter is lower than 10 we jump back.
finish:
msg counter
```

نكتب الكود في المفكرة كما فعلنا سابقا ثم نشغل ال script في OllyDbg و نرى النتيجة



النتيجة



كتابة script لفك تشفير upx

المهمة هي البحث عن سلسلة متعاقبة من ال bytes و ال bytes في أوامر الاسمبلي تظهر و على يسار olly.

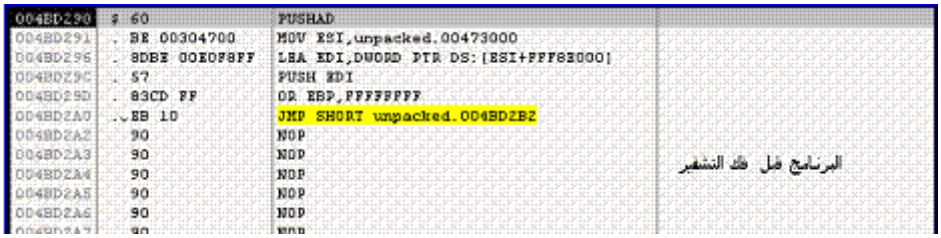
```
sti                    // Executes F7 (step into)
```

```
findop eip, #60#        // Searches code starting at addr //for an instruction, (Find command PUSHAD)
// Wildcards can be used.

bphws $RESULT,"x"       // Set hardware breakpoint. Available modes are //"r" -read, "w" - write or "x" -
// execute.

run                     // Executes F9

sti                     // Executes F7 (step into)

ret                     // Exit script
```

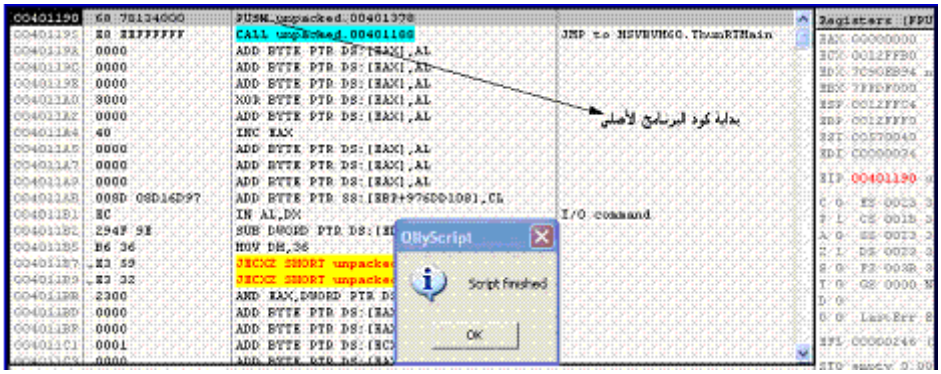
شرح الكود:

- هذا ينفذ امر ال step into وهو للدخول الى **CALL**
- و هذا الأمر هو للبحث عن تعليمة ال **PUSHAD**
- وضع نقطة توقف و على تعليمة ال **PUSHAD**
- تشغيل البرنامج
- و هذا مثل أول أمر
- الخروج من الكود أو نهاية الكود

و هذه هي الطريقة العامة لفك تشفير upx حيث نقوم بالبحث عن تعليمة ال **PUSHAD** وهي تكون أول تعليمة في البرنامج في معظم إصدارات upx وتكون آخر تعليمة هي **popad** يليها jmp نضع نقطة توقف و على هذه ال jmp ثم نشغل البرنامج سوف نجد هذه ال jmp تأخذنا إلي بداية كود البرنامج.



و عند استخدام ال script



وبهذا ينتهي الفصل السادس

وبانتهاء الفصل السادس ينتهي الباب الثالث

وبانتهاء الباب الثالث ينتهي الكتاب.


الملاحق
الملحق الأول : شرح بسيط لـ Boolean Algebra
الملحق الثاني : شرح عن الـ PE Files
الملحق الثالث : بعض تعليمات الأسمبلي غير الشائعة
الملحق الرابع : تثبيت و تركيب WinAsm & RadAsm
الملحق الخامس : حماية ملف بالـ CRC
الملحق السادس : درس تطبيقي شامل
الملحق السابع : هدية الكتاب


الملاحق
الملحق الأول
شرح بسيط لـ
Boolean Algebra

1. الملحق 1 : BOOLEAN ALGEBRA

ليس الغرض من هذا الملحق أن نشرح الـ Boolean Algebra بأكمله، بل سنقوم بشرح جزء صغير من هذا العلم الواسع.

و لمساعدتك في هذا الباب, اخترنا لك هذه الآلة الحاسبة CybultCalV1.7

نريد التعرف على أهم العمليات المنطقية Logic Operations التي تهمنا كـ reverse engineers.

قبل البدء تذكر أن

- 1 byte ➔ 8 bit
- 1 word ➔ 16 bit
- 1 DoubleWord ➔ 32 bit
- 1 nibble ➔ 4 bit

1.6. أولا – عملية OR

عملية OR والتي تسمى في اللغة العربية "أو" هي عملية لها جدول الحقيقة (Truth Table) التالي :

First Operand	Second Operand	Result
0	0	0
0	1	1
1	0	1
1	1	1

الفكرة كالتالي : إذا كان أي معامل من المعاملات قيمته 1، فالنتيجة النهائية هي 1، بغض النظر عن باقي المعاملات.

فمثلا افرض أننا نريد تطبيق هذه العملية على 4 معاملات : 1 = 0 OR 1 OR 1 OR 0.

كما تلاحظ فإن أحد هذه المعاملات قيمته 1، وبالتالي الجواب النهائي هي 1 ولا يهمنا ما هي قيمة باقي المعاملات.

2.6. ثانيا – عملية AND

هذه العملية هي كعملية الضرب المعتادة تماما. أي أننا سنضرب المعاملات ببعضها البعض.

First Operand	Second Operand	Result
0	0	**0**
0	1	**0**
1	0	**0**
1	1	1

الفكرة هي كالتالي : اذا كان أي معامل من المعاملات قيمته 0، فالنتيجة النهائية هي 0، بغض النظر عن باقي المعاملات.

فمثلا افرض أننا نريد تطبيق هذه العملية على 3 معاملات : 1 **AND** 1 **AND** 0 = 1

كما تلاحظ فإن أحد هذه المعاملات قيمته 0، وبالتالي الجواب النهائي هي 0 ولا يهمنا ما هي قيمة باقي المعاملات.

3.6. ثالثا – عملية NOT

هذه العملية تعني أن تعكس القيمة التي لديك.

First Operand	Result
0	1
1	0

بالطبع ليس شرطا أن يكون المعامل مكونا من بت واحد. فمثلا NOT 0110 = 1001.

المثال السابق ينطبق لو كنا نتكلم بشكل عام. لكن في كثير من الأحيان – خاصة في البرمجة – نتعامل مع الأعداد التي نود عمل NOT لها بحسب حجم المتغير المستخدم. فمثلا لنفرض أننا نتعامل مع أعداد بحجم بايت. عندها فإن 1001 هي بالأصل 00001001 لأنه كما قلنا فنحن نتعامل مع بايت والذي بدوره يتكون من 8 بت. في هذه الحالة فإن NOT 0110 = 11110110. وهذا ما يتم تطبيقه في المعالج وفي البرمجة وفي التنقيح.

في حالة كنا نتعامل مع نظام عد آخر كنظام hexadecimal مثلا، فالفكرة هي ذاتها. حيث أننا في البداية نحول العدد من هكس إلى بايناري ثم نجري العملية كالمعتاد. فمثلا لو كان لدينا العدد التالي : 0A1150E

في الأصل القيمة هي A1150E لكن أثناء كتابتك لهذه القيمة في OllyDBG أو winasm أو أي برنامج آخر، سيظهر لك خطأ، حيث أنه في حالة الأعداد المكتوبة بالهكس، إذا كان الـ MSB (القيمة الأكبر.. أي التي في اليسار) وفي حالتنا هي A، إذا كانت حرفا فيجب أن تضع 0 قبلها. إذن A1150E نكتبها 0A1150E، لكن مثلا 1BC78A تبقى كما هي لأن الـ MSB رقم وليس حرف.

في البداية نحوله إلى Binary فيصبح 1110 0000 0101 0001 0001 1010 0000 0000. الآن إن كنا نتعامل مع العدد بشكل عام فسنعكسه مباشرة، لكن دعك من ذلك فهذا الكتاب مخصص للهندسة العكسية لذا لا نريد تلك الحالة العامة، فدائما سنتعامل مع أعداد بأحجام محددة وغالبا 32 بت. إذن علينا أن نكمل العدد 1110 0000 0101 0001 0001 1010 إلى 32 بت فيصبح

1110 0000 0101 0001 0001 1010 0000 0000 ثم نقوم بعكسه فيصبح 0001 1111 1010 1110 1110 0101 1111 1111

هناك طريقة أخرى، حيث يمكن عمل NOT مباشرة دون التحويل إلى binary. حيث نقوم أولا بإكمال العدد الى 32 بت أي 4 بايت (كل حرف أو رقم عبارة عن نصف بايت أي nibble) وبالتالي كي نكمل 0A1150E سيصبح لدينا 00A1150E (دعك من الصفر الذي يوضع قبل الأحرف. أنساه مؤقتا). والآن اطرح كل حرف أو رقم من F. فيصبح لدينا FF5EEAF1 وهذا العدد لا يمكن أن نكتبه كما هو بل يجب أن نضع صفرا قبله لأنه يبدأ بحرف، فنحصل على 0FF5EEAF.

4.6. رابعا – عملية XOR

الفكرة هي كالتالي : إذا كانت جميع المعاملات متشابهة، فالنتيجة 0، و إن كانت مختلفة فالنتيجة 1.

First Operand	Second Operand	Result
0	0	0
0	1	1
1	0	1
1	1	0

افرض أن لدينا العملية التالية 1 = 1 **XOR** 0 **XOR** 0 **XOR** 0 **XOR** 0 كما تلاحظ فهناك معامل يختلف عن الباقي لذا النتيجة 1.

أما 0 = 1 **XOR** 1 **XOR** 1 فالأمر واضح.

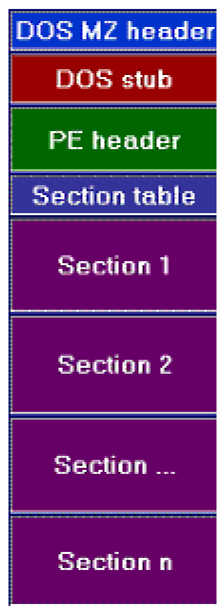

الملاحق
الملحق الثاني
شرح بسيط لـ
PE Files

2. ملحق 2 : PE-FILE FORMAT

في هذا الملحق ستجد شرحا عن ملفات PE لما لها من أهمية كبيرة في عمليات الـ reverse engineering.

الجزء الأكبر من هذا الملحق هو عبارة عن ترجمة – بتصرف- لمقالات الكاتب Iczelion.

إن PE ترمز إلى Portable Executable. وهي صيغة " طبيعية" (native) لملفات windows32. إن معنى التنقل(portable) هو أن أي PE loader على أي جهاز win32 سوف يقوم بتشغيل هذا الملف بغض النظر عن نوع المعالج المستخدم. إن هذا النوع من الملفات شائع جدا على كثير من الأنظمة. يتكون أي ملف PE من الأقسام العامة التالية كما يتضح بالصورة:



1.2. DOS MZ HEADER & DOS STUB

إن أول قسمين غير مهمين لنا لكن يمكننا القول أنهما مسؤولان عن جملة this program connot run in dos mod ولعلك لاحظت هذه الجملة كثيرا عند محاولتك لتشغيل ملف باستخدام الدوس.

سنتحدث قليلا عن القسم الأول. يسمى هذا القسم بـ IMAGE_DOS_HEADER. هذا القسم يحتوي قيمتين فقط مهمتين لنا هما e_lfanew و e_magic. أن e_magic تحتوي على الـ MZ "string" وعند قراءة ملف الـ PE من قبل الـ loader فانه يبحث عن هذه القيمة ويقارنها مع IMAGE_DOS_SIGNATURE فإذا تطابقا فالـ DOS HEADER سيعتبر صالح (valid). أما e_lfanew. فهذه الـ structure تحتوي على الـ offset لـ PE Header

PE HEADER .2.2

القسم الثالث هو الـ PE Header. عندما يتم تحميل الملف بواسطة PE loader من على نظام يتعرف على PE files كـ win32 فان هذا الـ loader يقرا الـ offset لـ PE Header من القسم الأول ثم "يقفز"إن صح التعبير عن القسم الثاني ويتوجه إلى الثالث مباشرة.

ما يجب عليك معرفته هو أن تنظيم الـ data إلى sections يتم على أساس الـ attribute، فبيانات الread-only لوحدها وبيانات system لوحدها وهكذا مع hidden, archive الخ. إياك أن تظن إن التقسيم يتم على أساس منطقي أي الـ data و الـ code وغيرها كل لوحده.

جدير بالذكر أن عملية التقسيم يقوم بها الـ loader وتسمى هذه العملية عادة بـ mapping the sections into memory(لعلك لاحظت الـ memory map فيolly) إن هذا القسم يكون الـ structure له كالتالي:

IMAGE_NT_HEADERS STRUCT	**هو اسم PE header structure**
Signature *dd* **?**	هي dword تحوي القيم التالية h, 45h, 00h, 00h50 أي "PE" متبوعة بصفرين.
FileHeader *IMAGE_FILE_HEADER* **<>**	هو structure تحتوي على معلومات كعدد الأقسام، الـ machine التي يعمل عليها الملف وهكذا.
OptionalHeader *IMAGE_OPTIONAL_HEADER32* **<>**	لا تنخدع بالاسم فهذا الجزء ليس "اختياريا" بل هو أساسي وبه عدة معلومات ستذكر لاحقا.
IMAGE_NT_HEADERS ENDS	

عند قراءة الملف، إذا كانت قيمة Signature تساوي "PE" متبوعة بصفرين فيتم تصنيف الملف على انه ملف PE صالح(valid)

: FILEHEADER .3.2

والآن دعنا نتحدث عن ثاني جزء في الـ PE header وهو FileHeader الذي يظهر بالشكل التالي:

IMAGE_FILE_HEADER STRUCT	
Machine *WORD* **?**	نوع الـ platform أي الـ CPU التي يعمل عليها الملف. لمعالجات انتل تكون كالتالي IMAGE_FILE_MACHINE_I386
NumberOfSections *WORD* **?**	كما هو واضح من الاسم يحتوي على عدد الأقسام.
TimeDatestamp *dd* **?**	غير مهم في هذا المستوى
PointerToSymbolTable *dd* **?**	غير مهم في هذا المستوى
NumberOfSymbols *dd* **?**	غير مهم في هذا المستوى
SizeOfOptionalHeader *WORD* **?**	غير مهم في هذا المستوى
Characteristics *WORD* **?**	بعض المعلومات، كنوع الملف(exe أم dll)
IMAGE_FILE_HEADER ENDS	

الجزء الأخير هو OptionalHeader. و قبل الحديث عن أجزائه دعنا نشرح الـ RVA

هذه الحروف الثلاثة ترمز لـ relative virtual address، ببساطة الـ RVA هو المسافة من نقطة مرجعية.

على سبيل المثال إذا تم تحميل ملف الـ PE عن العنوان 400000h في الفراغ الوهمي virtual address (حاول أن تتخيله!!)

وبدأ البرنامج بالتنفيذ(execution) عند العنوان الوهمي 401000h، عندها يمكننا القول إن البرنامج قد بدا عند RVA=1000h

وفي هذا المثال فان RVA قيمته 1000 نسبة للنقطة المرجعية وهي بداية الـ virtual address (الفراغ الوهمي)

لكن لماذا يستخدم ملف PE الـ RVAs؟ إن هذا يؤدي إلى تخفيف الحمل load عن الـ loader.(إذا أردت إن تعرف كيف يتم ذلك فراجع المقال الأصلي ل iczelion)

4.2. OPTIONALHEADER :

والآن نأتي لمكونات OptionalHeader : (لم أجد الشكل العام له. لكنه مشابه لشكل الـ FileHeader مع اختلاف العناصر)

IMAGE_OPTIONAL_HEADER STRUCT	
Magic dw ?	
MajorLinkerVersion db	نسخة اللينكر
MinorLinkerVersion db	نسخة اللينكر
SizeOfCode dd ?	مجموع احجام السكشنز التي تحتوي على كود
SizeOfInitializedData dd ?	مجموع احجام السكشنز التي تحتوي على Init data
SizeOfUninitializedData dd ?	مجموع احجام السكشنز التي ينبغي على اللودر حجزها
AddressOfEntryPoint dd ?	هذا هو قيمة RVA التي تشير إلى عنوان أول instruction سوف يتم تنفيذها.
BaseOfCode dd ?	هو RVA التي تبتدأ منها السكشن التي تحتوي على الكود
BaseOfData dd ?	هو RVA التي تبتدأ منها السكشن التي تحتوي على Data
ImageBase dd ?	هذا هو القيمة "المفضلة" التي يستخدمها الـ loader. مثلا لو كانت هذه القيمة h400000 فان الـ loader سوف يحاول تحميل(loading) الملف عند هذا العنوان. لكن لو كانت هذه المنطقة محجوزة من قبل تطبيق أخر فسيضطر الـ loader للبحث عن منطقة خالية أخرى.
SectionAlignment dd ?	لو كانت قيمته h1000 فسيبدأ كل section من عنوان عند مضاعفات h1000 و هذا في الذاكرة طبعا ما دمنا نتحدث عن السكشنز.
FileAlignment dd ?	نفس فكرة السابق لكن على القوص الصلب و ليس الذاكرة.
MajorOperatingSystemVersion dw ?	نسخة السيستم OS الاقل التي ينبغي التوفر عليها لتشغيل الملف
MinorOperatingSystemVersion dw ?	دائما 1.
MajorImageVersion dw ?	نسخة الملف التنفيذي التي تحددها انت
MinorImageVersion dw ?	نسخة الملف التنفيذي التي تحددها انت
MajorSubsystemVersion dw ?	نسخة الساب سيستم OS التي ينبغي التوفر عليها لتشغيل الملف
MinorSubsystemVersion dw ?	و تكون عادة 3.1 للدلالة على وجوب تواجد الوندوز 3.1
Win32VersionValue dd ?	دائما 0
SizeOfImage dd ?	الحجم الكلي لـ PE image في الذاكرة.

SizeOfHeaders dd ?	حجم الـ headers+section table أي انه يساوي حجم الملف ناقص حجم جميع الـ sections
CheckSum dd ?	CRC الخاص بالملف التنفيذي
Subsystem dw ?	NATIVE=1, WINDOWS_GUI=2, WINDOWS_CUI=3, OS2_CUI=5, POSIX_CUI=7
DllCharacteristics dw ?	لن تحتاجها في هذا المستوى
SizeOfStackReserve dd ?	لن تحتاجها في هذا المستوى
SizeOfStackCommit dd ?	
SizeOfHeapReserve dd ?	
SizeOfHeapCommit dd ?	
LoaderFlags dd ?	
NumberOfRvaAndSizes dd ?	
DataDirectory IMAGE_NUMBEROF_DIRECTORY_ENTRIES * IMAGE_DATA_DIRECTORY<>	هذه عبارة عن مصفوفة لـ IMAGE_DATA_DIRECTORY.
IMAGE_OPTIONAL_HEADER ENDS	

5.2. SECTION TABLE :

إن section table هو array of structures كل structure تحوي معلومات عن كل قسم في PE file كـ attribute, the file offset, virtual offset. لو كان هناك 3 أقسام فان الـ section table يحوي 3 قواعد (structures)، وهو يسمى بـ IMAGE_SECTION_HEADER ويكون كالتالي :

IMAGE_SECTION_HEADER STRUCT	
Name1 db IMAGE_SIZEOF_SHORT_NAME dup **(?)**	IMAGE_SIZEOF_SHORT_NAME equ 8
union Misc PhysicalAddress dd **?** VirtualSize dd **?** ends	
VirtualAddress dd **?**	قيم RVA لهذا القسم. فان كانت قيمته h3000 وكان الـ loader قد بدا عند h400000 فان هذا القسم (section table)سيتم تحميله عند h4003000.
SizeOfRawData dd **?**	حجم الـ section data أي قسم البيانات. (rounded up to the next multiple of file alignment)
PointerToRawData dd **?**	يحتوي على offset لبداية هذا القسم(section table)
PointerToRelocations dd **?**	
PointerToLinenumbers dd **?**	
NumberOfRelocations dw **?**	
NumberOfLinenumbers dw **?**	
Characteristics dd **?**	بعض الرايات (flags). مثلا هل يحتوي القسم على executable code, initialized data, uninitialized data؟هل يمكن القراءة منه أو الكتابة عليه؟
IMAGE_SECTION_HEADER ENDS	

6.2. IMPORT TABLE :

دعنا بداية نتعرف على import function.. إنها دالة غير موجودة في البرنامج.كل ما هو موجود هو معلومات عنها أما الدالة نفسها فموجودة في احد ملفات الـ DLL. (بالنسبة لدوال API فملفات الـ DLL موجودة في مجلد system32 في مجلد ويندوز الرئيسي) هل تذكر آخر عنصر في الـ optional header؟ نعم انه DataDirectory.كما قلنا فهي عبارة عن مصفوفة تسمى IMAGE_DATA_DIRECTORY بها 16 عنصر إن DataDirectory يحتوي على الـ locations and sizes للبيانات الهامة في الملف. كما هو واضح في الصورة..(يظهر 14 عنصر فقط)

Member	Info inside
0	Export symbols
1	Import symbols
2	Resources
3	Exception
4	Security
5	Base relocation
6	Debug
7	Copyright string
8	Unknown
9	Thread local storage (TLS)
10	Load configuration
11	Bound Import
12	Import Address Table
13	Delay Import
14	COM descriptor

العناصر الملونة بالأصفر هي ذات اهمية بالنسبة لك في هذا المستوى المهمة.كل عنصر من العناصر ال 16 له الاسم IMAGE_DATA_DIRECTORY (متبوعا باسم العنصر) وله الشكل العام التالي:

IMAGE_DATA_DIRECTORY STRUCT	
VirtualAddress dd **?**	DataDirectory لـ RVAهو ال
isize dd **?**	يحوي الحجم بالبايت.
IMAGE_DATA_DIRECTORY ENDS	

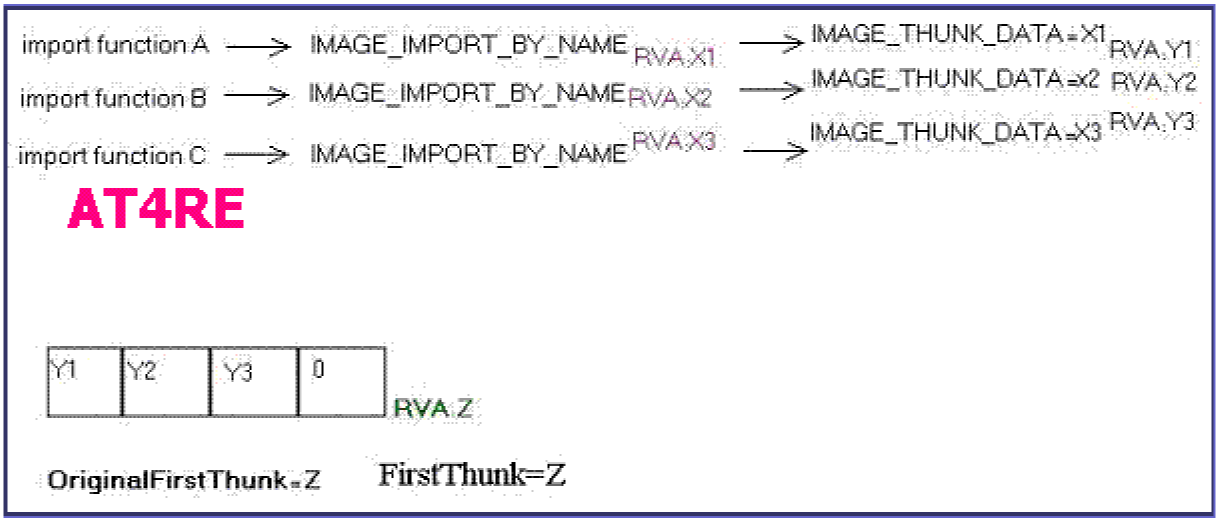
إذا كنت تريد أن تعرف أين يقع Base relocation (مثلا) فعليك ضرب قيمة حجم IMAGE_DATA_DIRECTORY أي 8 بايت ب 5 ثم تضيف الناتج إلى عنوان IMAGE_DATA_DIRECTORY. مثال : عنوان IMAGE_DATA_DIRECTORY هو 4001000 وحجمه 8 بايت. إذن Base relocation يقع عند عنوان 40+4001000 أي 4001040.

والآن دعنا نتحدث عن import table. عنوان هذا القسم موجود في خانة VirtualAddress في العنصر الثاني من الـ DataDirectory (أي في import symbols). إن الـ import table هو مصفوفة IMAGE_IMPORT_DESCRIPTOR. لو كان الـ PE File يستورد دوالا من 10 ملفات DLL مختلفة فان هذه المصفوفة تحوي 10 IMAGE_IMPORT_DESCRIPTOR. هذه هو الشكل العام لها :

IMAGE_IMPORT_DESCRIPTOR STRUCT
union
Characteristics dd ?
OriginalFirstThunk dd ?
ends
TimeDatestamp dd ?
ForwarderChain dd ?
Name1 dd ?
FirstThunk dd ?
IMAGE_IMPORT_DESCRIPTOR ENDS

- أول عنصر هو الـ union وهو يقدم اسما مستعارا (alias)للـ OriginalFirstThunk. لذا يمكن اعتبار الـ Union بمثابة Characteristics. بعبارة أخرى، هذا العنصر يحوي عنوان RVA للـ IMAGE_THUNK_DATA والتي بدورها تشير لأحد الـIMAGE_IMPORT_BY_NAME . دعنا نفسر ما سبق ذكره.
- هناك عدة دوال مستوردة import function
- كل import function لها IMAGE_IMPORT_BY_NAME
- كل IMAGE_IMPORT_BY_NAME لها RVA (فلنسمه RVA 1)
- كل RVA 1 يتم تخزينه في IMAGE_THUNK_DATA
- كل IMAGE_THUNK_DATA بدورها لها RVA (فلنسمه RVA 2)

يتم ترتيب جميع الـ2 RVAs في مصفوفة تنتهي بالقيمة 0. ألـ RVA للمصفوفة الناتجة (فلنسمه RVA 3) يتم وضعه في OriginalFirstThunk. أن لم تستوعب ذلك فانظر إلى الصورة التالية.



إن كل IMAGE_IMPORT_BY_NAME تبدوا بالشكل التالي:

IMAGE_IMPORT_BY_NAME STRUCT
Hint dw ?
Name1 db ?
IMAGE_IMPORT_BY_NAME ENDS

Name1 : غير مهم في الوقت الحالي

Hint: غير مهم في الوقت الحالي

نعود إلى IMAGE_IMPORT_DESCRIPTOR STRUCT لشرح العناصر:

TimeDatestamp : في المرحلة الحالية، غير مهم

ForwarderChain : في المرحلة الحالية، غير مهم

Name1: يحوي على RVA لاسم الـ DLL

FirstThunk : مشابه لـ OriginalFirstThunk.(هل ترى الشرح في الصورة؟ انه يطبق أيضا على الـ FirstThunk وبالتالي في النهاية سنحصل على مصفوفتين بدل من واحدة. ألأولى OriginalFirstThunk والثانية FirstThunk.

على سبيل المثال لو كان PE file يستورد 10 دوال من kernel32.dll فان Name1 في IMAGE_IMPORT_DESCRIPTOR سيحوي على RVA للـ string المسماة "kernel32.dll". وسوف يكون هنالك 10 IMAGE_THUNK_DATAs في الـ OriginalFirstThunk ومثلهم في FirstThunk

لكن السؤال هو. لم نحتاج إلى مصفوفتين متماثلتين (OriginalFirstThunk و FirstThunk)؟؟

عند تحميل الملف فان الـ loader ينظر إلى IMAGE_THUNK_DATAs and IMAGE_IMPORT_BY_NAMEs ليجدد عناوين الدوال المستوردة. ومن ثم يقوم باستبدال IMAGE_THUNK_DATAs (الخاصة بـ FirstThunk) ، يستبدلها بالعناوين الحقيقية لتلك الدوال المستوردة. أي إن الدالة المنتهية ب 0 والتي يشير إليها OriginalFirstThunk لا تتغير بينما مثيلتها المنتهية ب 0 والتي يشير إليها FirstThunk تتغير محتوياتها (قيم Y بالصورة) إلى العناوين الحقيقية للدوال المستوردة.

وما الهدف من ذلك؟

قلنا بان الدوال المستوردة يتم استدعائها باسمها. لكن هناك بعض الدوال لا تستدعى بالاسم بل بالترتيب (oridinal). في هذه الحالة لن يكون هناك IMAGE_IMPORT_BY_NAME لأنه ببساطة لا يوجد اسم للدالة.. عوضا عن ذلك فان IMAGE_THUNK_DATA لتلك الدالة المستوردة بالموضع سيحتوي على ترتيب (موضع) هذه الدالة في الجزء الأصغر منه، و على 1 في موضع الـ MSB. مثلا إذا كانت الدالة تستورد بترتيبها، وكان ترتيبها h123 فان IMAGE_THUNK_DATA لهذه الدالة سيكون 80001234h (عند التحويل إلى بالنظام الثنائي ستصبح رقم طويل وفي أقصى اليسار (MSB) الرقم 1).

أي ان الـ IMAGE_THUNK_DATA او ما يعرف بالـ IAT له احتمالان كالتالي :

8000XXXX حيث XXXX هي قيمة الـ oridinal

0YYYYYYY حيث YYYYYYY هي عبارة عن RVA يشير إلى IMAGE_THUNK_DATA (أي الى اسم دالة مستوردة)

والآن لنفترض أننا نريد إن نحصل على قائمة بأسماء جميع الدوال المستوردة في ملف PE ما. ألخطوات هي:

- تأكد إن الملف هو ملف PE صالحvalid PE
- من الـ DOS header اذهب إلى PE header
- احصل على عنوان data directory من OptionalHeader
- اذهب إلى العنصر الثاني في data directory (أي : Import symbols) واستخرج VirtualAddress
- استخدم تلك القيمة للذهاب إلى IMAGE_IMPORT_DESCRIPTOR
- تحقق من قيمة OriginalFirstThunk. إذا كانت لم تكن صفرا، فاتبع عنوان RVA للذهاب إلى تلك المصفوفة(المنتهية بـ0) أما إذا كانت صفرا فاستخدم القيمة في FirstThunk بدلا من OriginalFirstThunk
- (السبب في ذلك هو إن بعض الـ linkers تضع 0 دائما في OriginalFirstThunk رغم إن هذا يعتبر bug)
- لكل عنصر في المصفوفة(المنتهية ب0) تحقق من قيمة MSB : إذا كانت 1 إذن هذه الدالة مستوردة بالترتيب. ونحصل على الترتيب من الجزء الأصغر (80001234h ==> 1234h)

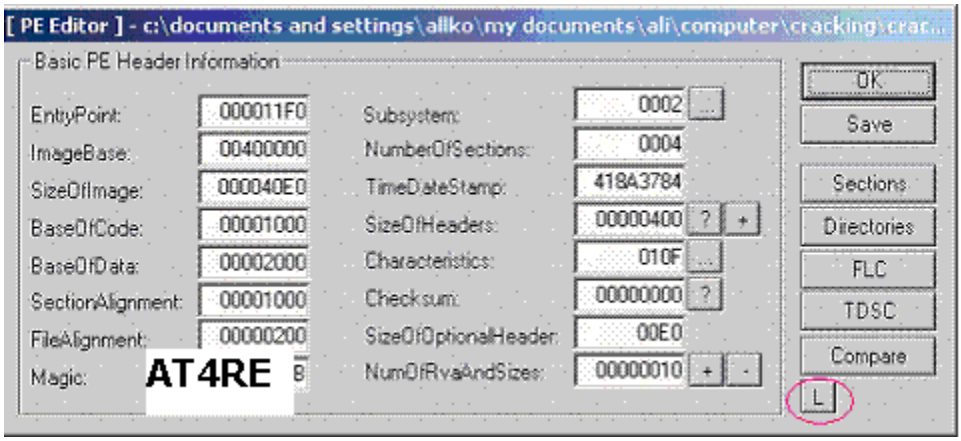
إذا كان الـ MSB يساوي 0 فاذهب إلى تلك المصفوفة(المنتهية ب0) والقيمة التي ستجدها هي عبارة عن RVA لـ IMAGE_IMPORT_BY_NAME اذهب إليها لتحصل على اسم الدالة.

الآن اذهب إلى العنصر التالي في المصفوفة (المنتهية ب0) وكرر الخطوات لتحصل على اسم للدالة.تابع إلى إن تجد الصفر(مللت وأنا أقول : المنتهية ب0، أخيرا وصلنا إلى هذا الـ0 :)ووصولنا إلى الصفر يعني انتهاء المصفوفة أي أننا حصلنا على أسماء جميع الدوال المستوردة من ذلك الـ DLL.

اذهب إلى IMAGE_IMPORT_DESCRIPTOR التالي وكرر نفس الخطوات لاستخراج الأسماء لملف DLL التالي

7.2. تطبيق

والآن كتطبيق، شغل LordPE ثم اختر PE Editor واختر أي ملف exe لتظهر شاشة مشابهة للتالي:

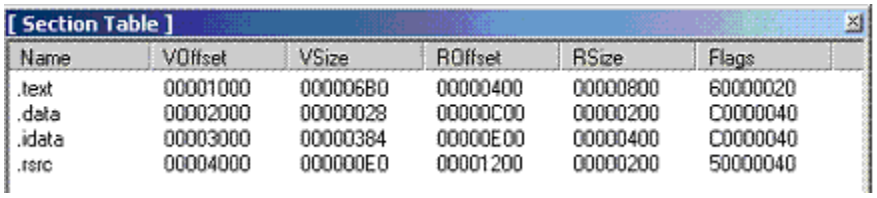


لاحظ أن معظم هذه الخصائص قمنا بشرحها والباقي ليس ذو أهمية كبيرة.

اضغط على الزر L الموضح.. والآن ستظهر لك list بها مكونات هذا الـ PE بدءا بـ DOS Header وانتهائا بـ DataDirectory..

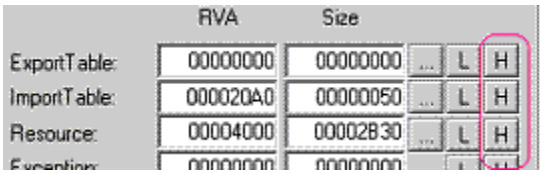
تمعنه جيدا فهو يقوم بربط أفكارك معا.

حسنا لقد قلنا أن كل ملف PE مكون من عدة sections. أل sections التي تلي ال 4 الأولى (أي بعد DOS MZ header و DOS stub و PE Header و section table) نستطيع أن نجدهم بالضغط على زر sections لتظهر نافذة مشابهة للتالي :

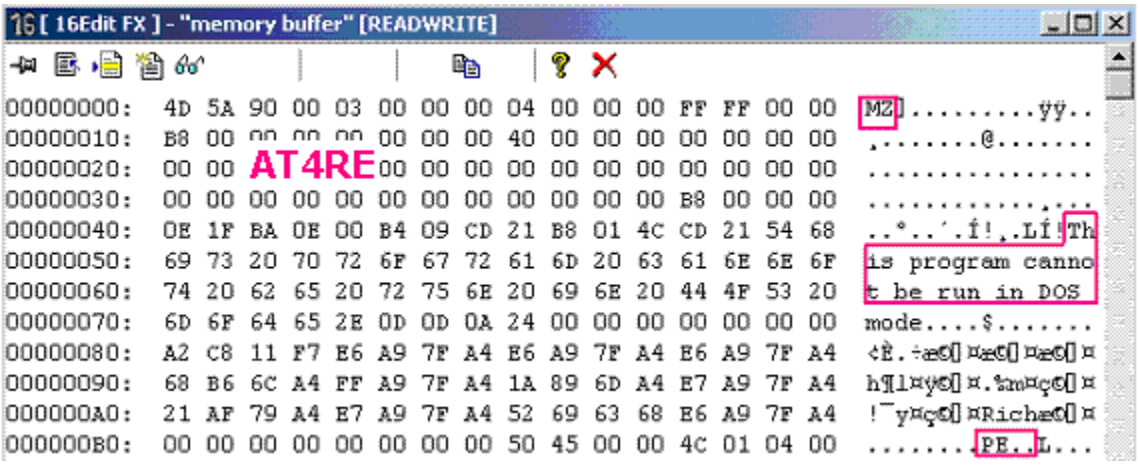


كما ترى هناك 4 أقسام هي.text و.data و .idata و الأخير.rsrc للـ resources.

بقي الـ Import Table. أختر Directories ليظهر الـ IT بعناصره الـ 16. اضغط على حرف L بجانب أي عنصر لتلقي عليه نظرة عن قرب.حسنا اضغط على H بجانب أي منها كما بالصورة (زر H الأول لن يظهر لك شيئا.حسنا لا بأس جرب غيره. ما نريده هو عرض الملف ولا يهم أي زر تضغط..)



سيظهر لك الملف بهيئة hex. أصعد للأعلى إلى بداية الملف لتر التالي :



هل تلاحظ حرفي MZ؟ هذا هو قسم DOS MZ Header. ثم هل تلاحظ تلك الجملة التي تحدثنا عنها؟ هذا هو قسم DOS stub. ثم هلا تلاحظ حرفي PE المتبوعين بصفين؟ هذا هو قسم PE Header. وهكذا مع البقية.

إذا اردت خطاطة شاملة كاملة ل PE 32-bit FORMAT فما عليك الا تحميل الملف التالي


الملاحق
الملحق الثالث
بعض تعليمات الأسمبلي غير الشائعة

3. ملحق 3 : بعض تعليمات الأسمبلي الأقل أهمية

التعليمات الاكثر أهمية تم شرحها في الفصل الأول، هذه التعليمات أقل أهمية لكنك قد تواجهها أثناء تنقيحك لأحد البرامج.

سنبدأ بتعليمات تغيير حالة الرايات (Flags)

1.3. CLC

هذه التعليمة تضع 0 في Cartage Flag

2.3. CLD

هذه التعليمة تضع 0 في Address Flag

3.3. CLI

تضع 0 في interruption flag، وبالتالي فإنها تعطل (disable) نوع محدد من الـ interrupts يدعى maskarable interruptions.

4.3. CMC

تقوم بعكس قيمة Cartage Flag، فإن كانت قيمته 1 يصبح 0 والعكس صحيح.

5.3. STC

تضع القيمة 1 في Cartage Flag

6.3. STD

تضع القيمة 1 في Address Flag

7.3. STI

تضع القيمة 1 في Interruption Flag.

لاحظ أن ST اختصار لـ set، بينما CL اختصار لـ clear، أما CM اختصار لـ complement.


الملاحق
الملحق الرابع
تثبيت و تركيب
WinAsm & RadAsm

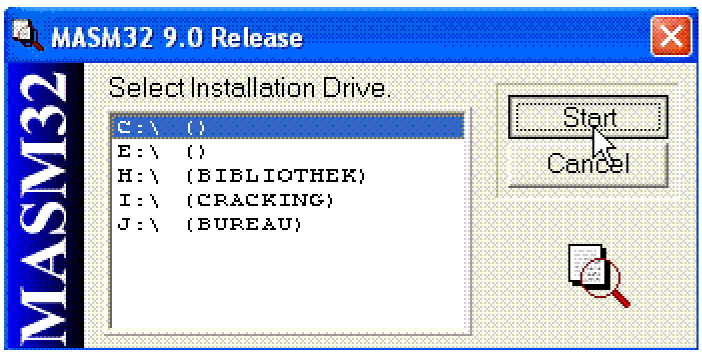
.4 ملحق 4 : كيفية تثبيت MASM & WINASM & RADASM

.1.4 MASM32 V 9.0

انقر مرتين على أيقونة البرنامج (المضغوط) والمسماة

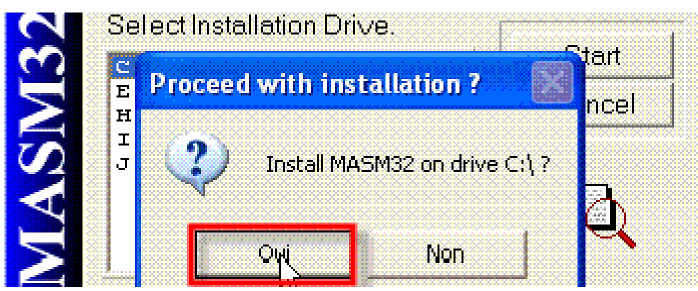
MASM32 V9.0

عندها ستظهر النافذ التالية :

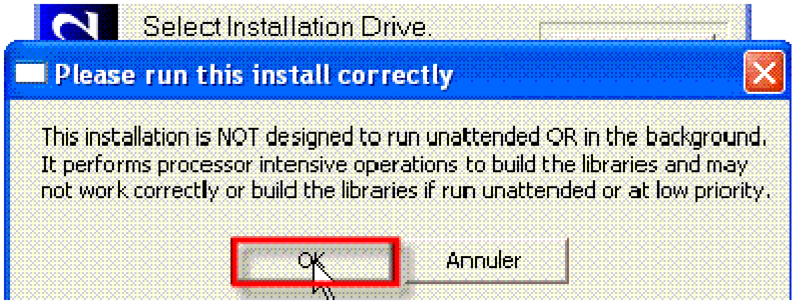


اختر مكان التثبيت وليكن \:C. يمكنك اختيار أي درايف آخر لكن لا تختر مسارات طويلة، أي مجلدات فرعية.

الآن اضغط على زر START :

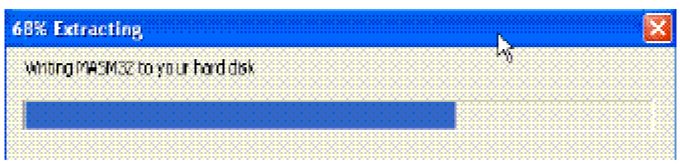


اضغط OK، ثم :

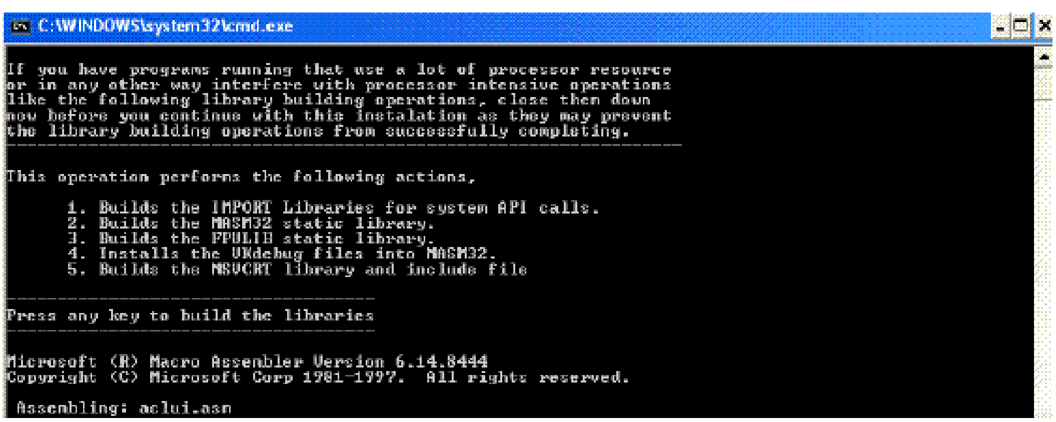


OK، ثم :

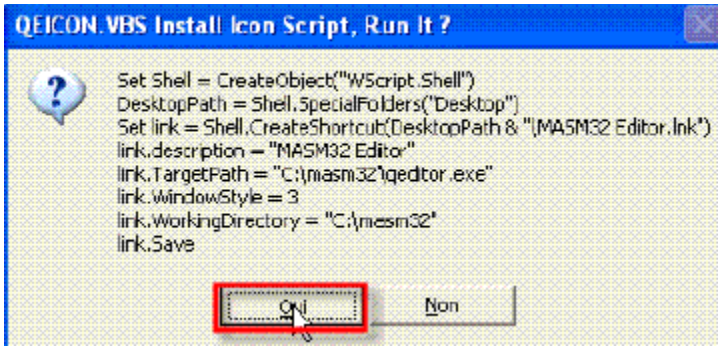
Extract، ثم :



عند انتهاء فك الضغط، ستظهر شاشة الـ DOS،



اضغط على أي زر للمتابعة.سيبدأ تثبيت البرنامج. وفي النهاية سيطلب منك مجددا أن تضغط أي زر لإغلاق شاشة DOS.



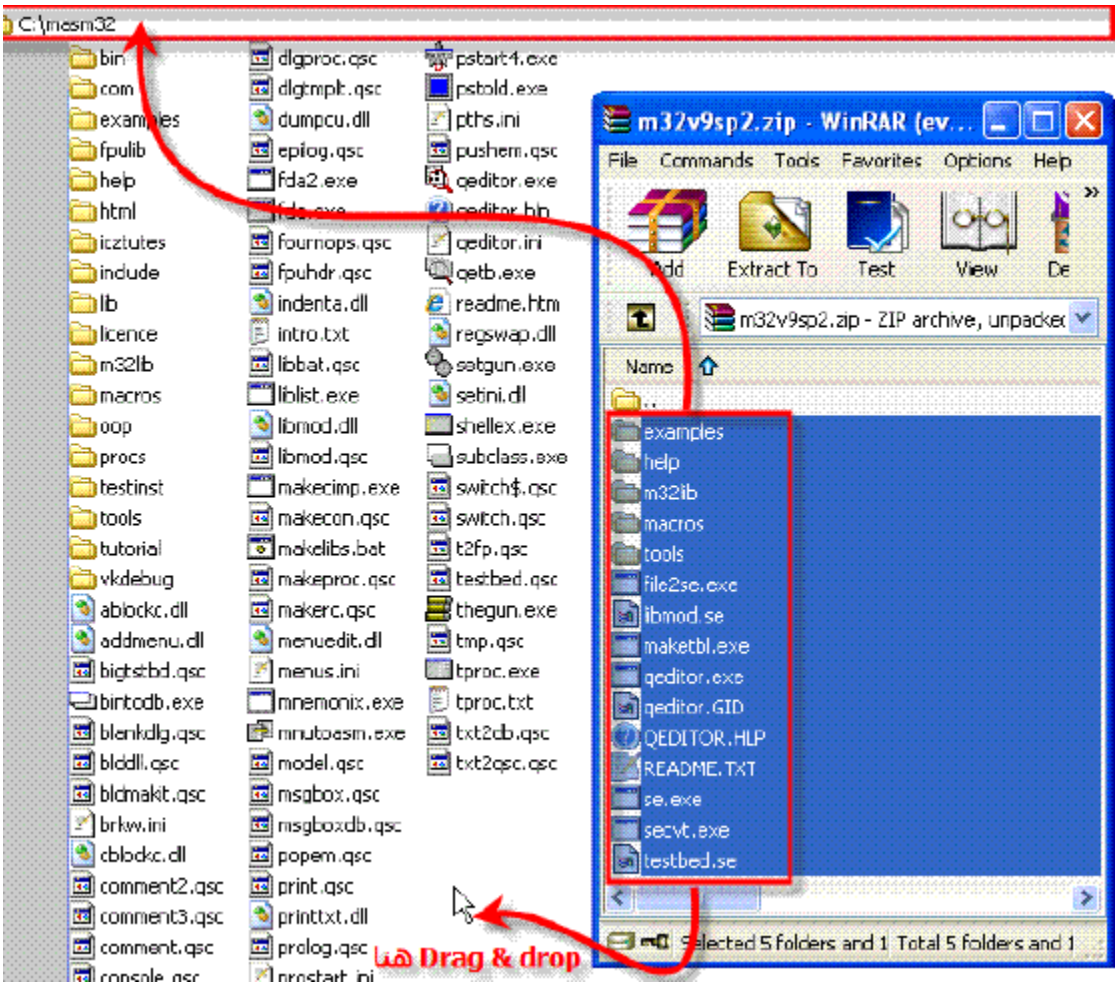
هذه الشاشة تعرض عليك وضع اختصار في سطح المكتب، اختر ما تشاء (موافق او غير موافق) وستظهر النافذة التالية :

انتهينا.

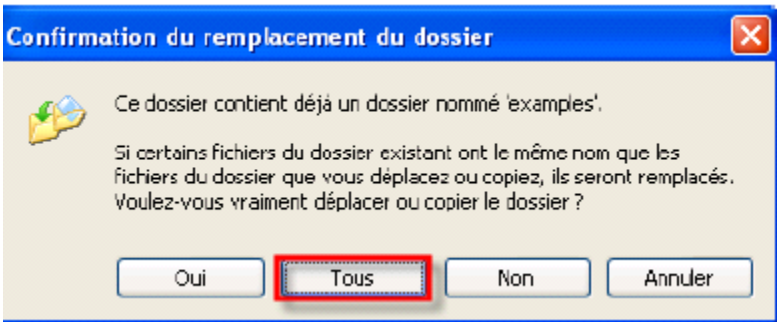
والآن لتحديث هذه النسخة، قم بتحميل الملف المسمى

 m32v9sp2.zip 

وقم بفك ضغطه في المكان الذي ثبت فيه البرنامج.



ستظهر النافذة التالية :



اضغط Yes for All. بعد الانتهاء ستجد في مجلد البرنامج مجلدا فرعيا اسمه icztutes وبداخله مجلدا فرعيا آخر اسمه m32lib وستجد فيه ملفا اسمه make.bat اضغط عليه. بعد انتهاء العملية ستخرج النافذة التالية:

انتهينا.

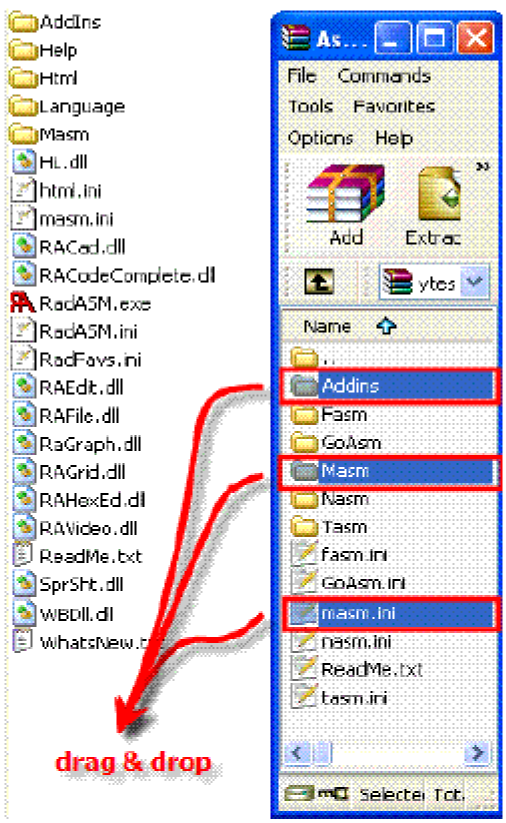
RADASM .2.4

هذا الـ IDE يعتبر من أفضل المحررات للأسمبلي وهو منافس قوي لـ WinAsm

ستجد الملف RadAdm.zip قم بفك الضغط عنه في أي مسار ترغب فيه و ليكن مثلا :

E:\Cracking\RadAsm

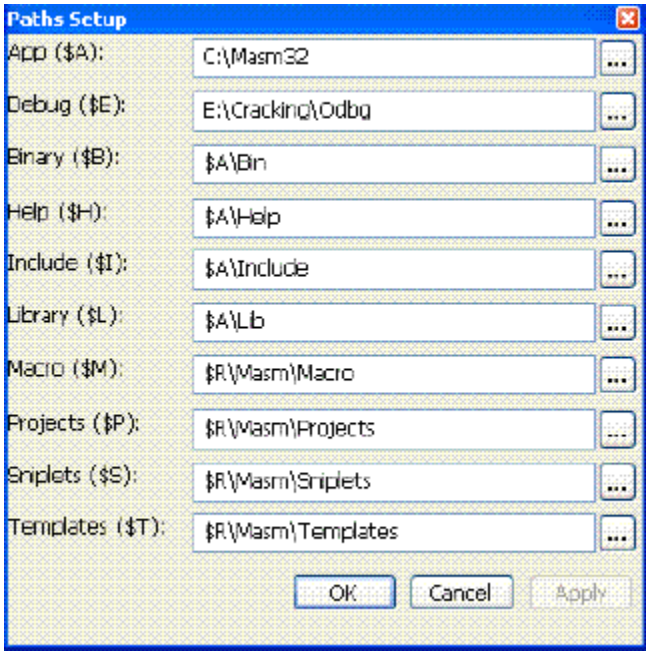
بعد فك الضغط هناك، ستجد في المجلد ملفا اسمه assembly.zip يحتوي على عدة مجمعات.سنختار الملفات المتعلقة ب masm فقط :



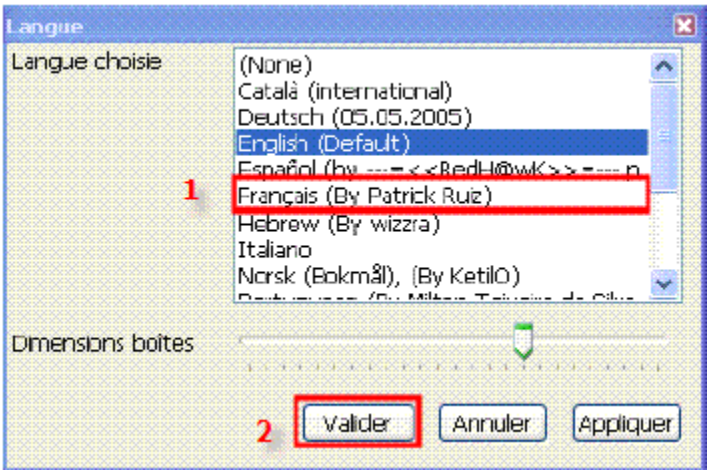
اذا أردت استخدام لغة أخرى غير الانجليزية، فقم باستخراج محتويات الملف RadLng.zip (ستجد أن ملف RadLng.zip يحتوي عدة لغات. مثلا لتثبيت الفرنسية فك ضغط الملف RadFRA.lng)

الآن هناك إعدادات مهمة يجب عملها.

أولا شغل RadAsm ومن قائمة options اختر setPaths

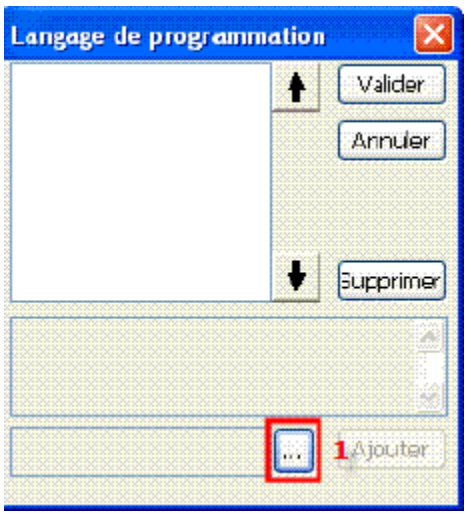


الآن إن كنت قد قررت تغيير اللغة، فستجد في قائمة options خيارا اسمه Language :

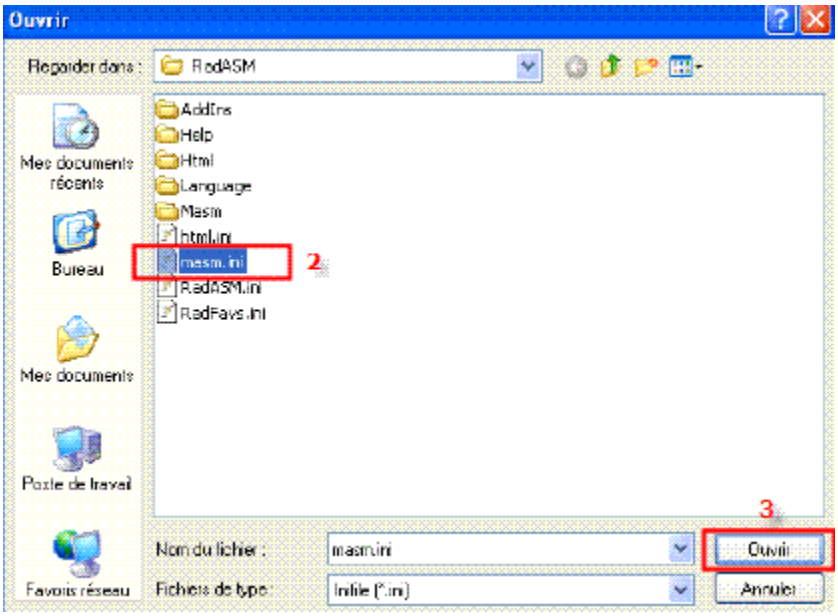


اختر اللغة التي تريدها.

الآن هناك خطوة مهمة جدا. في قائمة options هناك خيار اسمه Programming Language اختره :

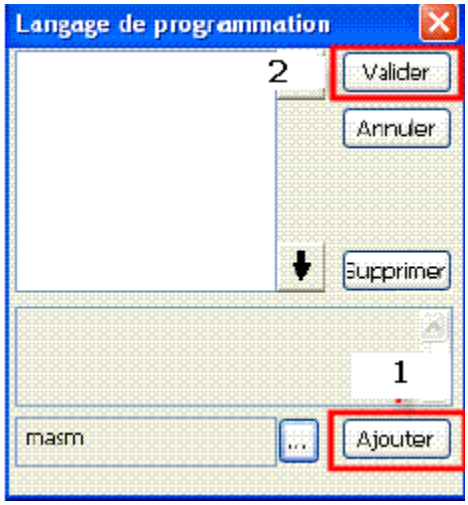


وبعد ضغط الزر الموضح :



اختر ملف masm كما هو واضح.

الآن ننتقل إلى إعدادات الألوان. في نفس النافذة السابقة اضغط ما هو موضح في الصورة التالية :



ستظهر لك شاشة بها عدة ألوان. اختر الألوان والـ themes التي تعجبك.

WINASM .3.4

المسألة مع winasm أسهل بكثير. فبعد أن قمنا بتثبيت masm32 v9.0 كل ما يجب فعله الآن هو فك الضغط عن حزمة winasm في أي مكان يحلو لك. اذا أردت التحديث يمكنك فك الضغط عن حزمة

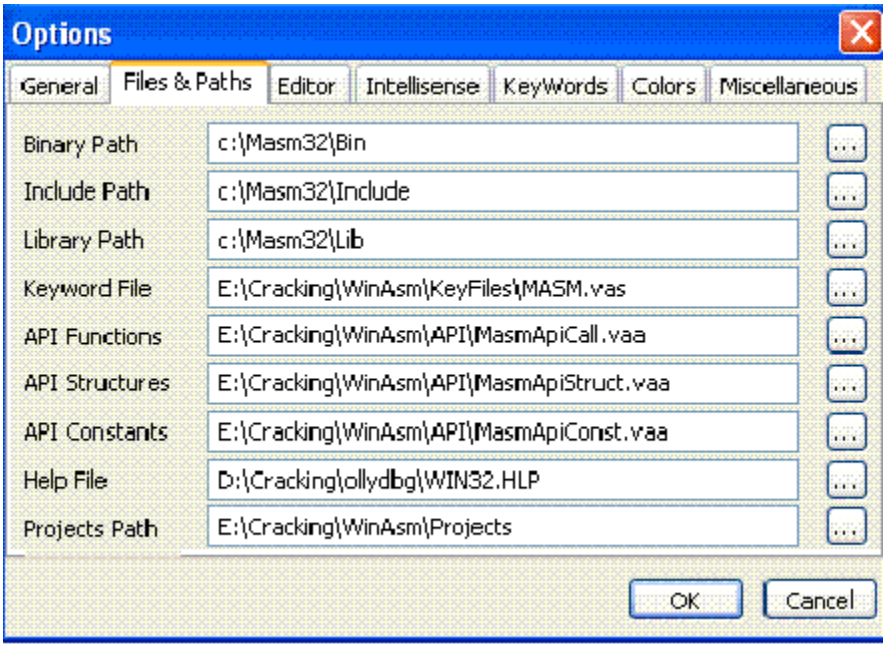
 WinAsm515 

في المكان الذي ثبتت البرنامج فيه.

الإعدادات المطلوبة هي :

أولا اذهب الى tools➔options

غير المسارات كما هو واضح (لاحظ أنك ان ثبتت برنامج في مكان آخر غير E:\Cracking\WinAsm فسيختلف المسار بالنسبة لك)



أما الألوان فيمكن تغييرها من لسان التبويب colors الظاهر في الصورة السابقة.

بالنسبة للغة، من قائمة tools➔interface واختر إما الإنجليزية أو الفرنسية.


الملاحق
الملحق الخامس
حماية ملف بالـ
CRC

5. ملحق 5 : حماية ملف بال CRC

اليوم سنتكلم عن كيفية حماية البرامج بواسطة خوارزمية crc32 . لن نتكلم هنا عن شرح لهذه الخوارزمية فهذا سيأخذ وقتا طويلا ، كما أن هذه المعلومات هي رياضية بحتة ، ويمكنك البحث في الانترنت فستجد كل ما تريد من معلومات.

تستخدم هذه الخوارزمية للتأكد من أن برنامجك لم يتم العبث به ، أو بعبارة أدق لم يتم عمل patching له .

حسنا فلنبدأ ، في المرفقات تجد برنامجا اسمه ch2_sec9_2 قم بفتح السورس كود بواسطة برنامج winasm .

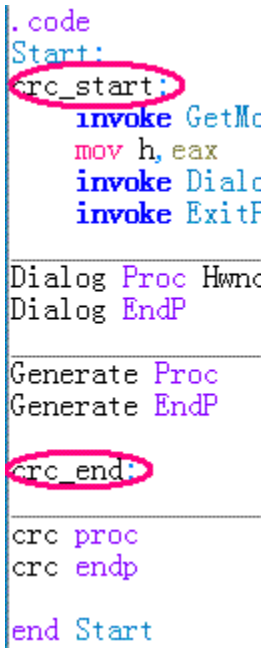
الكود المطلوب إضافته هو كالتالي :

```
crc proc
        pusha
        mov     esi,  crc_start         ; Address To Start CRC Check
        xor     edx, edx
        or      eax, -1
        mov     ecx, crc_size           ; No. of bytes to CRC check

CRC32_loop:
        mov     dl, byte ptr [esi]
        xor     dl, al
        shr     eax, 8
        xor     eax, dword ptr [crc32_table+ 4*edx]
        inc     esi
        dec     ecx
        jnz     CRC32_loop
        not     eax
        popa
        cmp eax,  07DD07FEh                 ; The Real CRC checksum for the segment
        jne patched
        ret
crc endp
```

هل ترى كم هو بسيط...والآن ما يجب معرفته قبل إضافة هذا الكود هو :

1 -عنوان بداية التحقق ، حيث يجب وضع label يشير إلى المكان الذي سنبدأ من عنده بالحساب ، بالإضافة إلى label يشير إلى نهاية منطقة التحقق ...أي أن الكود الموجود خارج هذين "المؤشرين" لن تمم حمايتهما ويمكن للكراكر إضافة أو تغيير أية تعليمة أو تعليمات كما يشاء.

لكن أين سنضع هذين الـ two labels ؟ ببساطة واحدة في بداية البرنامج ، أسفل كلمة start مباشرة ، والأخرى بعد Generate Proc مباشرة . كما هو واضح في الصورة التالية :



كما تلاحظ فقد وضعنا crc procedure في النهاية ، ولم نشملها في منطقة الحماية .

2 -الأمر الثاني الواجب معرفته ، **هو حجم هذه المنطقة**...أي منطقة الحماية ، لذالك أسهل طريقة هي أن نذهب إلى قسم .data? ونعرف المتغير التالي :

crc_size db crc_end - crc_start

3 -الأمر الثالث الواجب معرفته هو **بداية المنطقة**.. ببساطة إنها crc_start

4 -علينا أن نعرف **قيمة crc** لهذه المنطقة أي لهذه البايتات المحصورة بين crc_start و crc_end .

ولعمل ذلك فأسهل طريقة هي أن تضع أي قيمة عشوائية بداية ، ثم تقوم بتنقيح البرنامج وتتبعه ، وعند تجاوز crc procedure فإن قيمة eax ستكون هي القيمة الصحيحة . انتبه فهذه الخطوة تكون في النهاية ، لأنك مثلا لو أحببت عمل تغيير ما في البرنامج ، وكان التغيير موجود في منطقة الحماية ، فإن قيمة crc سوف تتغير ، عندها ستضطر لإعادة التنقيح من جديد...لذا من الأفضل ترك هذه الخطوة للآخر .

5 - سنضيف **جدولا بقيم crc الثابتة** والتي تستخدم في هذه الخوارزمية . هذه القيمة هي ذاتها في جميع البرامج وجميع التطبيقات. تجد هذا الجدول في السورس كود .

6 - بالطبع يلزمك **وضع label باسم patched** ، ولا داعي لأن نخبرك بأنه بإمكانك اختيار أي اسم آخر . ما سيوضع أسفل هذا الـ label ليس بحاجة لشرح... ببساطة نضع رسالة خطأ ثم دالة exit process مثلا...

ولا تنسى أن استدعاء هذه الخوارزمية (crc) يجب أن يكون في بداية البرنامج...هكذا مثلا : call crc

يمكنك الإطلاع على السورسكود المرفق لمزيد من التوضيح .

والآن لنعد للنقطة الرابعة...بعد وضع قيمة عشوائية لها ، قم بعمل build all في winasm أو أي IDE آخر ، وقم بتنقيح النسخة الناتجة في olly .

سنضع نقطة توقف عن أمر المقارنة في الخوارزمية كما يلي :

```
00401184  $  60              PUSHAD
00401185  .  BE 00104000     MOV ESI,ee.<ModuleEntryPoint>
0040118A  .  33D2            XOR EDX,EDX
0040118C  .  83C8 FF         OR EAX,FFFFFFFF
0040118F  .  B9 25000000     MOV ECX,25
00401194  >  8A16            MOV DL,BYTE PTR DS:[ESI]
00401196  .  32D0            XOR DL,AL
00401198  .  C1E8 08         SHR EAX,8
0040119B  .  330495 7F304(   XOR EAX,DWORD PTR DS:[EDX*4+40307F]
004011A2  .  46              INC ESI
004011A3  .  49              DEC ECX
004011A4  .^ 75 EE           JNZ SHORT ee.00401194
004011A6  .  F7D0            NOT EAX
004011A8  .  3D FE07DD07     CMP EAX,7DD07FE
004011AD  .  61              POPAD
004011AE  .v 75 01           JNZ SHORT ee.004011B1
004011B0  .  C3              RETN
004011B1  >  6A 00           PUSH 0
004011B3  .  E8 00000000     CALL <JMP.&kernel32.ExitProcess>
```

والآن شغل الضحية F9 . ستجد أن قيمة EAX=F65CBA67 إذن افتح winasm وغير القيمة العشوائية إلى 0F65CBA67h

السورس كود: ASM

وهكذا نكون قد انتهينا.


الملاحق
الملحق السادس
درس تطبيقي شامل

6. ملحق 6 : تمرين شامل

بسم الله الرحمن الرحيم

كسر برنامج بمختلف الطرق المارة في هذا الكتاب

البرنامج : **Easy Video to iPod MP4 PSP 3GP Converter**

الرابط

سأشرح طرق كسر البرنامج وصولا للكيجن

وفي كل مرحلة يمكن استنتاج طريقة كسر مختلفة

أولا:

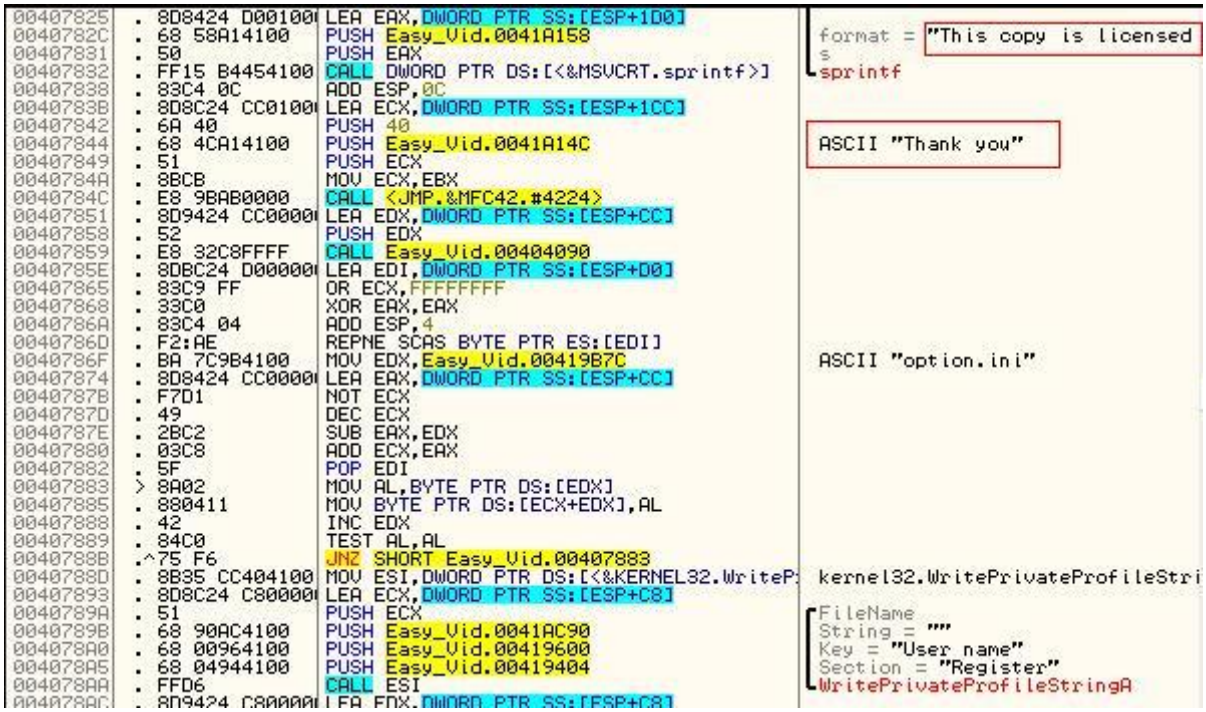
- نشغل البرنامج
- ادخل إلى حيث يطلب التسجيل
- اكتب أي اسم
- اكتب أي رقم

1.6 . الوصول إلى النقطة التي يتحقق فيها البرنامج من رقم التسجيل

1.1.6 . طريقة إيقاف البرنامج :

إضغط على زر ok لتظهر لك رسالة تقول أن السيريال خاطئ

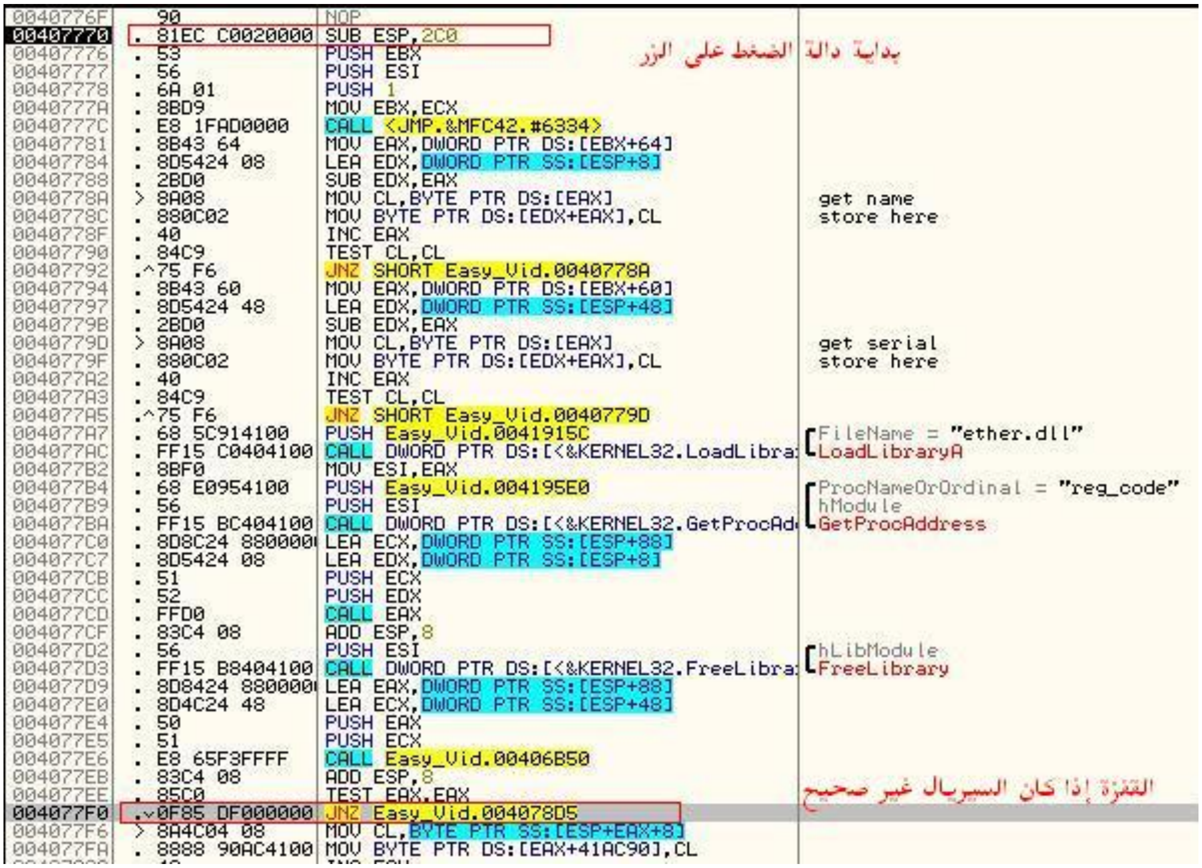
الآن إضغط على زر ⏸ في olly ← ثم إكبس ALT+F9 ← ثم إضغط على زر ok في رسالة البرنامج و سيقف البرنامج هنا



```
00407825  . 8D8424 D00100( LEA EAX,DWORD PTR SS:[ESP+1D0]
0040782C  . 68 58A14100    PUSH Easy_Vid.0041A158                       format = "This copy is licensed
00407831  . 50             PUSH EAX                                     s
00407832  . FF15 B4454100  CALL DWORD PTR DS:[<&MSVCRT.sprintf>]        sprintf
00407838  . 83C4 0C        ADD ESP,0C
0040783B  . 8D8C24 CC0100( LEA ECX,DWORD PTR SS:[ESP+1CC]
00407842  . 6A 40          PUSH 40
00407844  . 68 4CA14100    PUSH Easy_Vid.0041A14C                       ASCII "Thank you"
00407849  . 51             PUSH ECX
0040784A  . 8BCB           MOV ECX,EBX
0040784C  . E8 9BAB0000    CALL <JMP.&MFC42.#4224>
00407851  . 8D9424 CC0000( LEA EDX,DWORD PTR SS:[ESP+CC]
00407858  . 52             PUSH EDX
00407859  . E8 32C8FFFF    CALL Easy_Vid.00404090
0040785E  . 8DBC24 D00000( LEA EDI,DWORD PTR SS:[ESP+D0]
00407865  . 83C9 FF        OR ECX,FFFFFFFF
00407868  . 33C0           XOR EAX,EAX
0040786A  . 83C4 04        ADD ESP,4
0040786D  . F2:AE          REPNE SCAS BYTE PTR ES:[EDI]
0040786F  . BA 7C9B4100    MOV EDX,Easy_Vid.00419B7C                    ASCII "option.ini"
00407874  . 8D8424 CC0000( LEA EAX,DWORD PTR SS:[ESP+CC]
0040787B  . F7D1           NOT ECX
0040787D  . 49             DEC ECX
0040787E  . 2BC2           SUB EAX,EDX
00407880  . 03C8           ADD ECX,EAX
00407882  . 5F             POP EDI
00407883  > 8A02           MOV AL,BYTE PTR DS:[EDX]
00407885  . 880411         MOV BYTE PTR DS:[ECX+EDX],AL
00407888  . 42             INC EDX
00407889  . 84C0           TEST AL,AL
0040788B  .^75 F6          JNZ SHORT Easy_Vid.00407883
0040788D  . 8B35 CC404100  MOV ESI,DWORD PTR DS:[<&KERNEL32.WriteP:  kernel32.WritePrivateProfileStri
00407893  . 8D8C24 C80000( LEA ECX,DWORD PTR SS:[ESP+C8]
0040789A  . 51             PUSH ECX                                     FileName
0040789B  . 68 90AC4100    PUSH Easy_Vid.0041AC90                       String = ""
004078A0  . 68 00964100    PUSH Easy_Vid.00419600                       Key = "User name"
004078A5  . 68 04944100    PUSH Easy_Vid.00419404                       Section = "Register"
004078AA  . FFD6           CALL ESI                                     WritePrivateProfileStringA
```

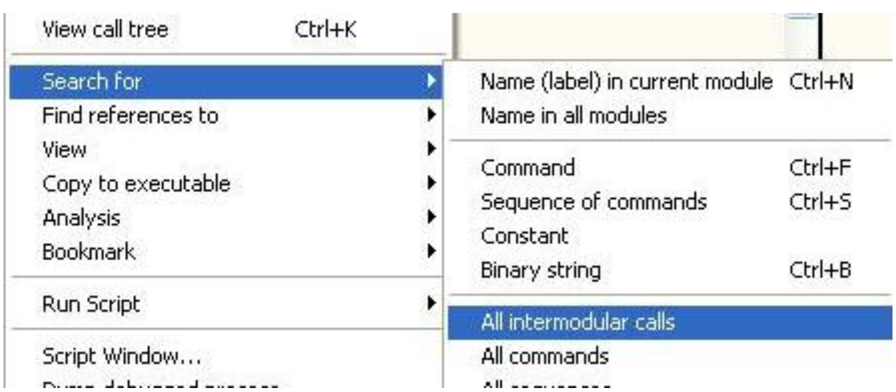
```
004078AC  . 8D9424 C80000( LEA EDX,DWORD PTR SS:[ESP+C8]
004078B3  . 52             PUSH EDX                                     FileName
004078B4  . 68 40A84100    PUSH Easy_Vid.0041A840                       String = ""
004078B9  . 68 EC954100    PUSH Easy_Vid.004195EC                       Key = "Registration code"
004078BE  . 68 04944100    PUSH Easy_Vid.00419404                       Section = "Register"
004078C3  . FFD6           CALL ESI                                     WritePrivateProfileStringA
004078C5  . 8BCB           MOV ECX,EBX
004078C7  . E8 6AA90000    CALL <JMP.&MFC42.#4853>
004078CC  . 5E             POP ESI
004078CD  . 5B             POP EBX
004078CE  . 81C4 C0020000  ADD ESP,2C0
004078D4  . C3             RETN
004078D5  > 6A 40          PUSH 40
004078D7  . 68 44A14100    PUSH Easy_Vid.0041A144                       ASCII "Sorry"
004078DC  . 68 1CA14100    PUSH Easy_Vid.0041A11C                       ASCII "Invalid user name or regi
004078E1  . 8BCB           MOV ECX,EBX
004078E3  . C705 2CB04100  MOV DWORD PTR DS:[41B02C],0
004078ED  . E8 FAAA0000    CALL <JMP.&MFC42.#4224>
004078F2  . 5E             POP ESI                                      MFC42.#4234
004078F3  . 5B             POP EBX
004078F4  . 81C4 C0020000  ADD ESP,2C0
004078FA  . C3             RETN
```

لا حظ يمكن الوصول للعنوان 004078F2 من القفزة في العنوان



2.1.6 . طريقة البحث عن دالة :

أعد تشغيل البرنامج بواسطة olly ثم ← ضغطة يمين للفأرة ثم اختر



للبحث عن أي دالة فقط ابدأ بالكتابة

طبعا بالضغط على أعلى حقل وصف الدالة يتم ترتيب الدوال حسب الاسم

سنبحث عن احدى الدالتين : GetWindowText و GetDlgItemText

وهنا في مثالنا لن نجد أي منها

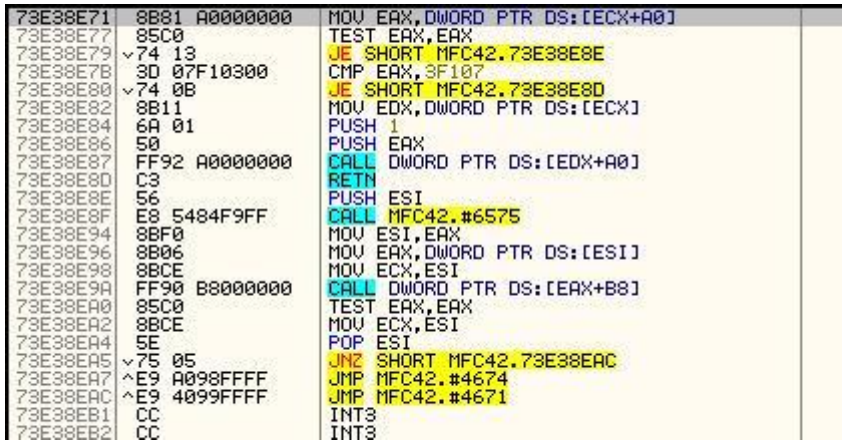
ولكن انظر

أترى MFC42 في البداية :

هذا دليل أن البرنامج استخدم مكاتب ال MFC42 وهى مكاتب ل C++. وبدل أن يستخدم المبرمج الدالتين GetWindowText و GetDlgItemText

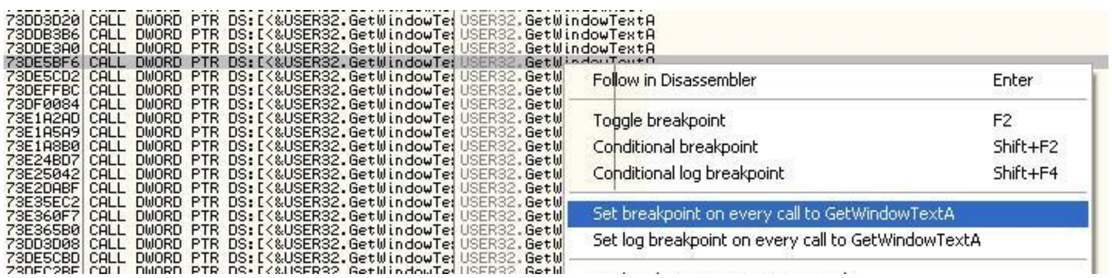
بصراحة استخدم دوال mfc ولكن هذه المكتبة ستستدعي هاتين الدالتين حتما لذا اذهب إلى بداية البرنامج للأعلى قليلا لترى

لاحظ جدول الدوال أختر أول دالة ثم اضغط Enter

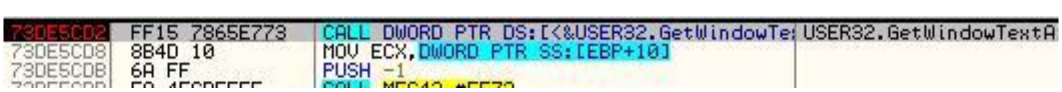


أنت في مكان يشابه هذا

الآن ابحث عن الدالة : GetWindowText



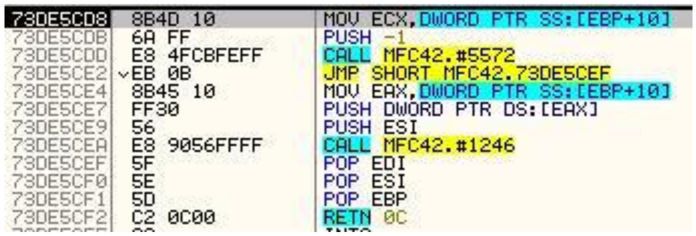
- تبدأ بكتابة اسمها وهو يبحث عنها
- الآن اختر ما تراه في الأعلى
- شغل البرنامج
- قد يتوقف olly الكثير من المرات
- كل دالة تجد نفسك وقفت عليها كثيرا أزل نقطة التوقف من أمامها
- يعني مرتين ثلاثة لا يوجد مشكلة حتى يعمل البرنامج بالكامل
- الآن ادخل حيث التسجيل
- اكتب الاسم و الرقم و إضغط الزر
- سيقف البرنامج هنا



اضغط على ثم F8

وهذا يعني نفذ حتى نهاية الروتين الحالي و الهدف هو الخروج من دوال ال MFC إلى البرنامج

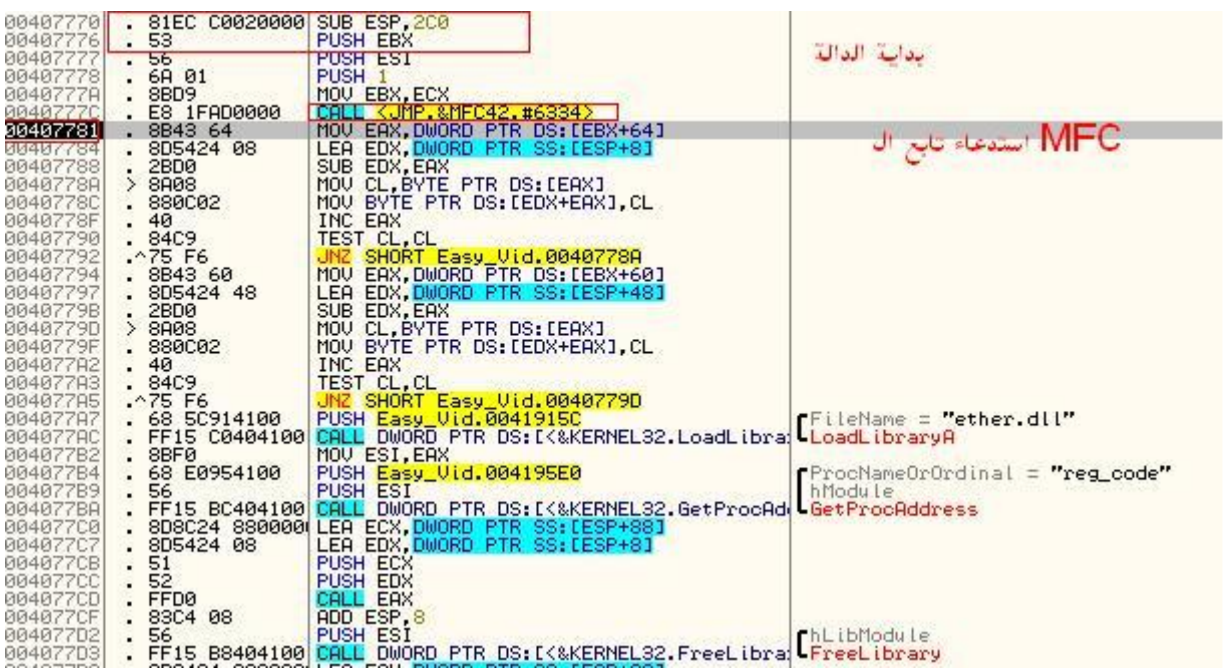
ستصل إلى هنا



مرة أخرى

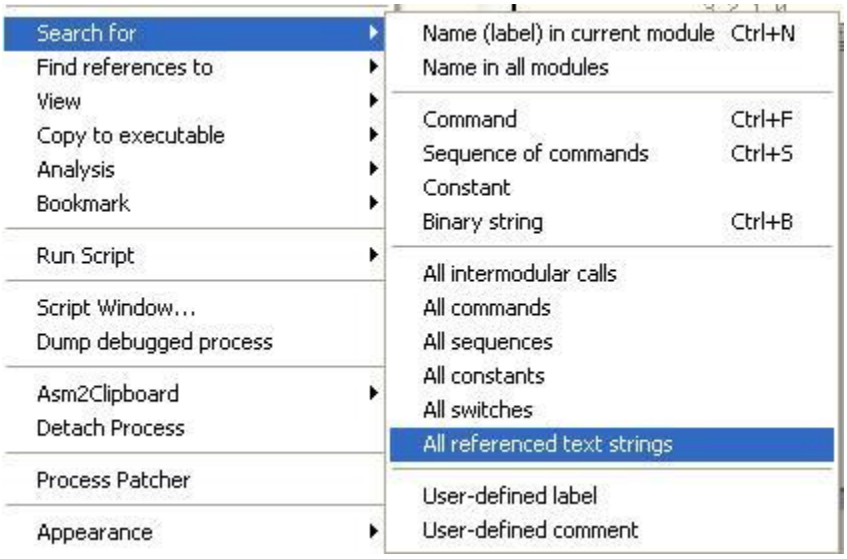
مرة أخرى

وأخرى حتى تصل إلى عنوان داخل البرنامج مشبوه مثل هنا

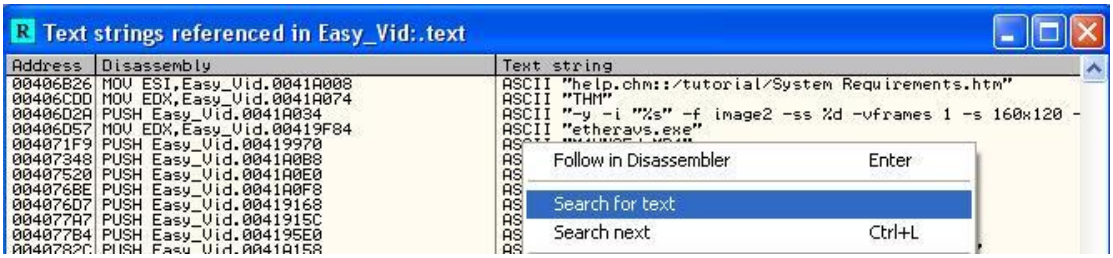


الآن تعرف كيف تجد القفزة

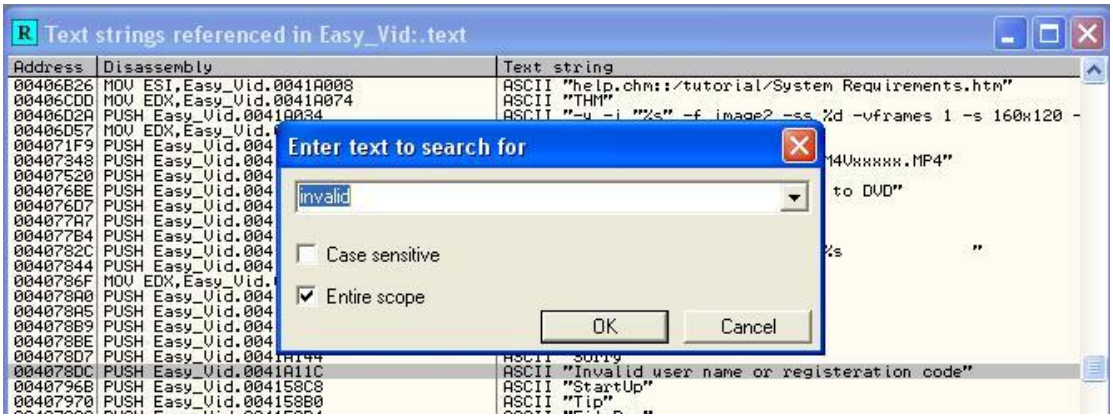
3.1.6 . طريقة البحث عن المحارف :



اختر الخيار الموجود فوق



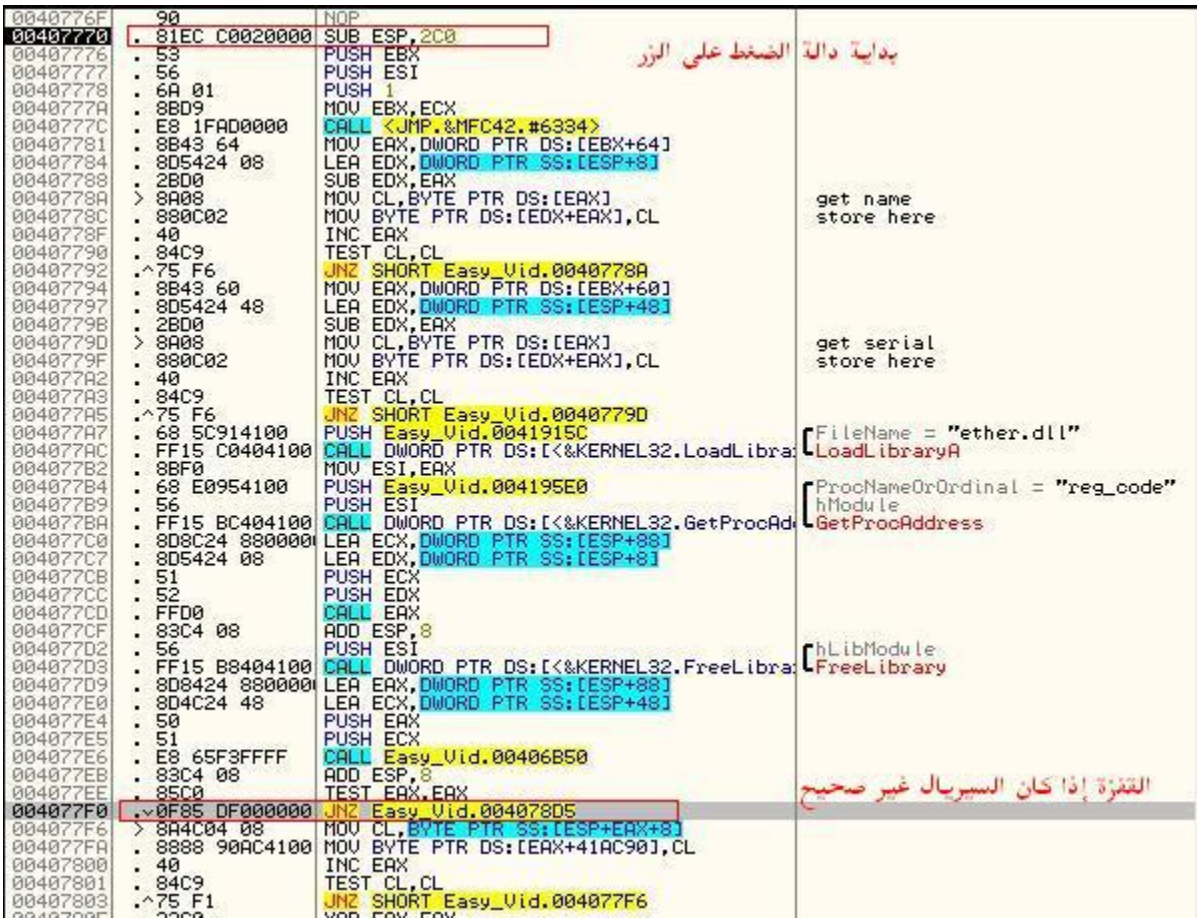
اختر



اكتب نص من الرسالة وتأكد من الخيارات فوق ثم اضغط ok

الآن ضغطتين على النص سيأخذك للمكان المطلوب

و بعد أن وجدنا القفزة و التي هي هنا



يمكن عمل

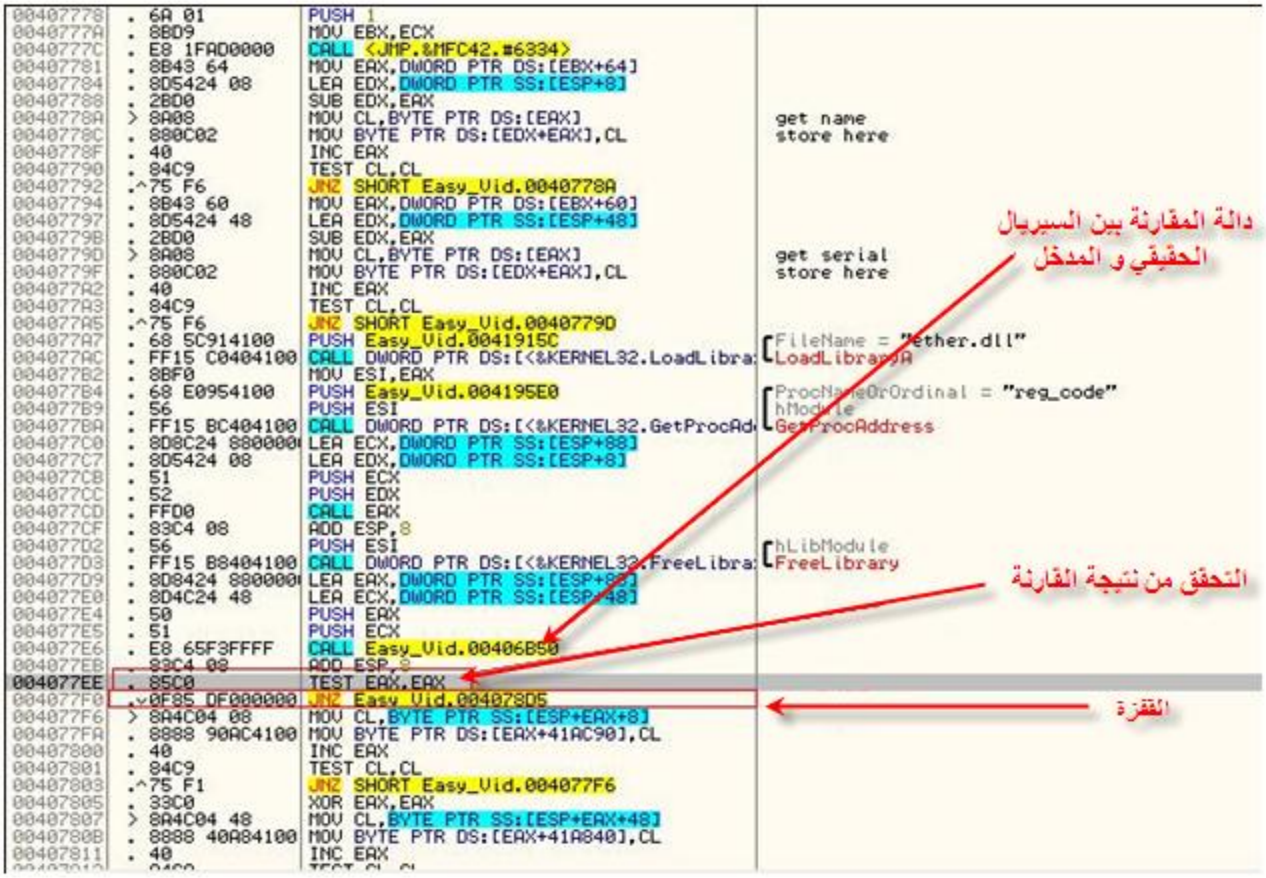
PATCH : لعمل باتش اذهب إلى الفقرة 4

LOADER : للعمل لودر اذهب إلى الفقرة 5

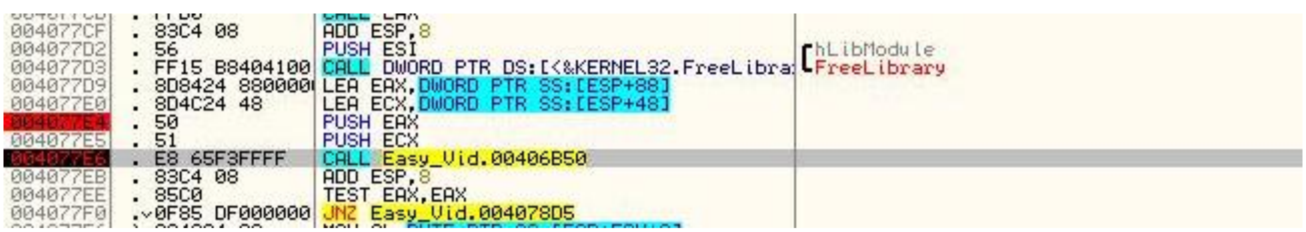
2.6 . إيجاد رقم التسجيل :

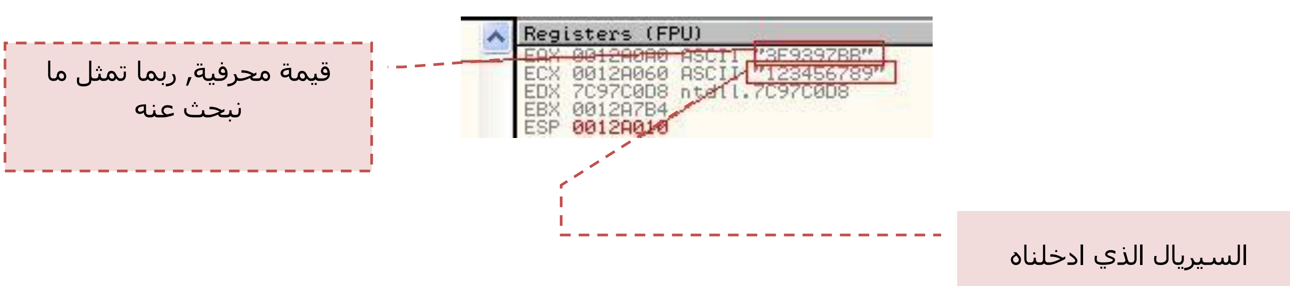
اكتب في حقل الاسم : catena_man/at4re (حروف صغيرة LowerCase). ثم اكتب أي رقم تسجيل

اذهب لمكان القفزة وهي عند العنوان : 004077F0 ولاحظ:



قبل هذه الدالة يوجد دفع لقيمتين لنر ما هما





لنجرب القيمة نعم هي السيريال

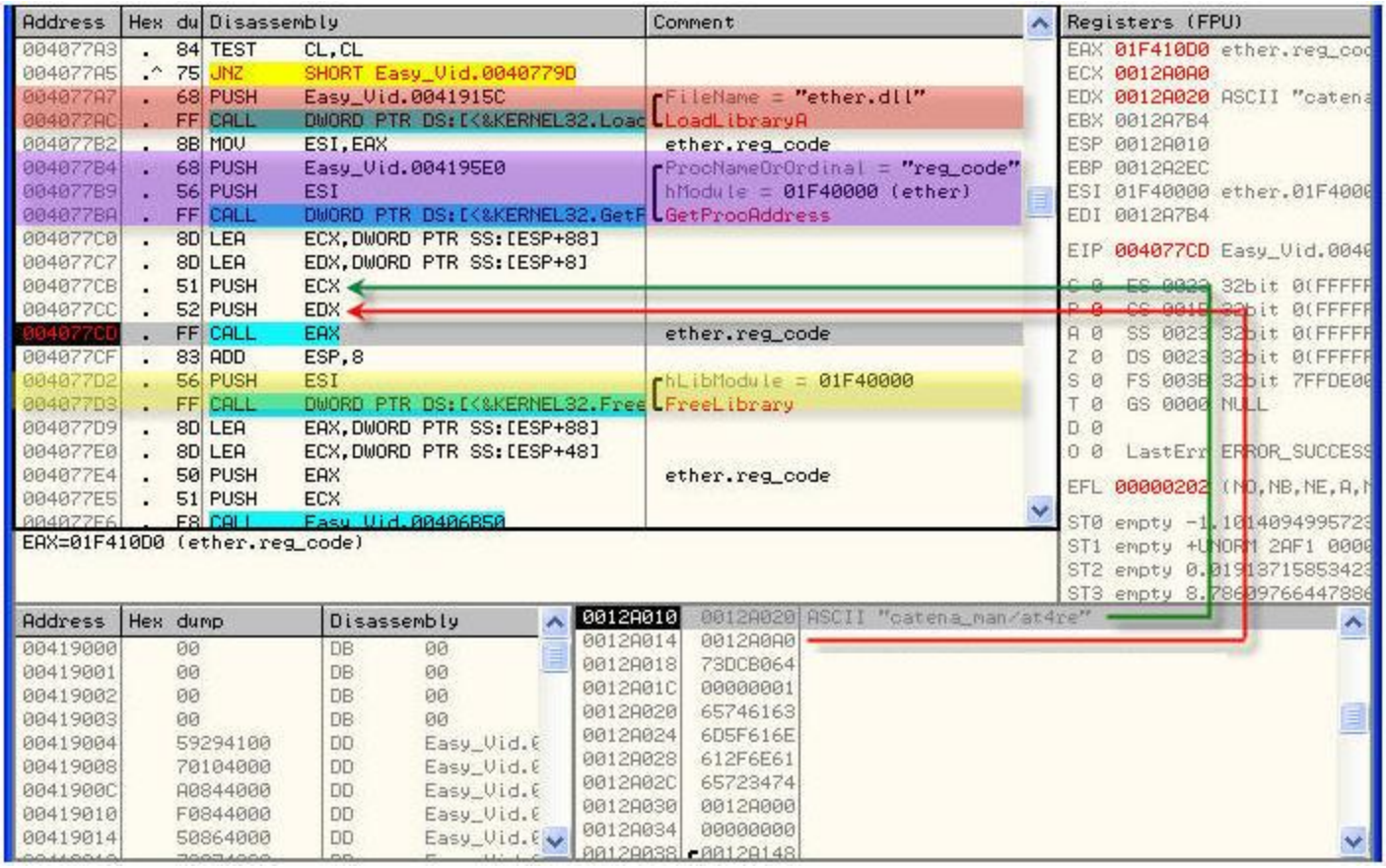
مبروك

3.6. صنع الكيجن :

كنا نريد التطرق الى صنع الكيجين بعد عكس الخوارزمية, لكن تبين ان هذا لن يفيد المبتدئين في شئ لصعوبة فهمها أولا ثم لصعوبة كتابتها من ناحية اخرى.

لكن لا تحزن, فحتى المبتدئ يستطيع عمل كيجين اذا اراد, بكفي ان يمتلك حسا من الدقة في الملاحظة و قليل من الفهم و لنبدأ باسم الله:

دقق جيدا في قطعة الكود هذه و حاول ان تجد الثغرة و الخطأ القاتل الذي قام به صاحب البرنامج لحمايته:



```
Address   Hex du Disassembly                              Comment
004077A3  .   84 TEST    CL,CL
004077A5  .^  75 JNZ     SHORT Easy_Vid.0040779D
004077A7  .   68 PUSH    Easy_Vid.0041915C                ┌FileName = "ether.dll"
004077AC  .   FF CALL    DWORD PTR DS:[<&KERNEL32.Loac    └LoadLibraryA
004077B2  .   8B MOV     ESI,EAX                           ether.reg_code
004077B4  .   68 PUSH    Easy_Vid.004195E0                ┌ProcNameOrOrdinal = "reg_code"
004077B9  .   56 PUSH    ESI                              │hModule = 01F40000 (ether)
004077BA  .   FF CALL    DWORD PTR DS:[<&KERNEL32.GetP    └GetProcAddress
004077C0  .   8D LEA     ECX,DWORD PTR SS:[ESP+88]
004077C7  .   8D LEA     EDX,DWORD PTR SS:[ESP+8]
004077CB  .   51 PUSH    ECX
004077CC  .   52 PUSH    EDX
004077CD  .   FF CALL    EAX                               ether.reg_code
004077CF  .   83 ADD     ESP,8
004077D2  .   56 PUSH    ESI                              ┌hLibModule = 01F40000
004077D3  .   FF CALL    DWORD PTR DS:[<&KERNEL32.Free    └FreeLibrary
004077D9  .   8D LEA     EAX,DWORD PTR SS:[ESP+88]
004077E0  .   8D LEA     ECX,DWORD PTR SS:[ESP+48]
004077E4  .   50 PUSH    EAX                               ether.reg_code
004077E5  .   51 PUSH    ECX
004077E6  .   E8 CALL    Easy_Vid.00406B50
EAX=01F410D0 (ether.reg_code)
```

```
Registers (FPU)
EAX 01F410D0 ether.reg_coc
ECX 0012A0A0
EDX 0012A020 ASCII "catena
EBX 0012A7B4
ESP 0012A010
EBP 0012A2EC
ESI 01F40000 ether.01F4000
EDI 0012A7B4

EIP 004077CD Easy_Vid.0040

C 0  ES 0023 32bit 0(FFFFF
P 0  CS 001B 32bit 0(FFFFF
A 0  SS 0023 32bit 0(FFFFF
Z 0  DS 0023 32bit 0(FFFFF
S 0  FS 003B 32bit 7FFDE00
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS

EFL 00000202 (NO,NB,NE,A,N

ST0 empty -1.1014094995723
ST1 empty +UNORM 2AF1 0000
ST2 empty 0.01913715853423
ST3 empty 8.7860976644788
```

```
Address   Hex dump   Disassembly
00419000  00         DB      00
00419001  00         DB      00
00419002  00         DB      00
00419003  00         DB      00
00419004  59294100   DD      Easy_Vid.€
00419008  70104000   DD      Easy_Vid.€
0041900C  A0844000   DD      Easy_Vid.€
00419010  F0844000   DD      Easy_Vid.€
00419014  50864000   DD      Easy_Vid.€
```

```
0012A010  0012A020  ASCII "catena_man/at4re"
0012A014  0012A0A0
0012A018  73DCB064
0012A01C  00000001
0012A020  65746163
0012A024  6D5F616E
0012A028  612F6E61
0012A02C  65723474
0012A030  0012A000
0012A034  00000000
```

- **في الاطار الاحمر** : دالة LoadLibrary تجد معلومات عنها في الجدول اسفله, تحمل ملفا تنفيذيا الى ذاكرة الملف المنَقَّح. و الملف المحمل هنا هو DLL اسمه « ether.dll »

- **في الاطار البنفسجي** : دالة اخرى تجدها تحت, و هي للبحث عن عنوان دالة معينة في ذاكرة الملف المحمل بالدالة السابقة. و تذكر ايضا ان العنوان الذي تحسبه هذه الدالة, ترجعه في المسجل EAX.
  - هنا البرنامج يبحث عن دالة اسمها : « reg_code » اذن EAX هو عنوان الدالة reg_code
  - الم يثر انتباهك هذا الاسم, كانه اسم لدالة تتحقق او تحسب السيريال الصحيح.
  - لاحظ شيئا آخر, الذالة لها بارامتران : الاول في EDX و سنرى ما هو و الثاني في ECX و هو مؤشر نحو الاسم المدخل

- **في الاطار الاصفر**: دالة تقوم بعكس ما قامت به الدالة LoadLibrary

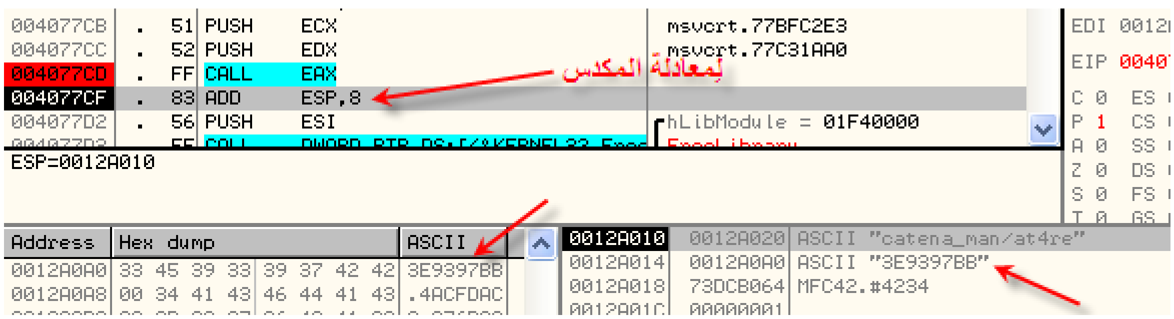
السهم الاخضر: يشير الى البارامتر الثاني للدالة reg_code و هو الاسم المدخل.

السهم الاحمر: يشير الى البارامتر الأول للدالة reg_code

LoadLibrary Function	GetProcAddress Function
Maps the specified executable module into the address space of the calling process. HMODULE WINAPI LoadLibrary(__in LPCTSTR *lpFileName*); **Parameters** *lpFileName* The name of the executable module (either a .dll or .exe file). The name specified is the file name of the module and is NOT related to the name stored in the library module itself, as specified by the LIBRARY keyword in the module-definition (.def) file. Return Value If the function succeeds, the return value is a handle to the module. If the function fails, the return value is NULL.	Retrieves the address of an exported function or variable from the specified dynamic-link library (DLL). FARPROC WINAPI GetProcAddress(__in HMODULE hModule, __in LPCSTR lpProcName); **Parameters** *hModule* A handle to the DLL module that contains the function or variable. The LoadLibrary or GetModuleHandle function returns this handle. *lpProcName* The function or variable name, or the function's ordinal value. If this parameter is an ordinal value, it must be in the low-order word; the high-order word must be zero. Return Value If the function succeeds, the return value is the address of the exported function or variable. If the function fails, the return value is NULL.

الآن لمعرفة ما يشير اليه البارامتر الاول, اعمل له دامب في نافذة الدامب

ثم اضغط F8 لتنفيذ خطوة فوق ال CALL و انتبه الى القيمة التي ستسجل في نافذة الدامب:



اذن الامور توضحت, البارامتر الأول للدالة reg_code هو البافر الذي سيستقبل السيريال الصحيح ☺ و هذا كل ما نحتاجه لكيجين المبتدئين الذي سنصنعه في هذا القسم من الدرس باذن الله.

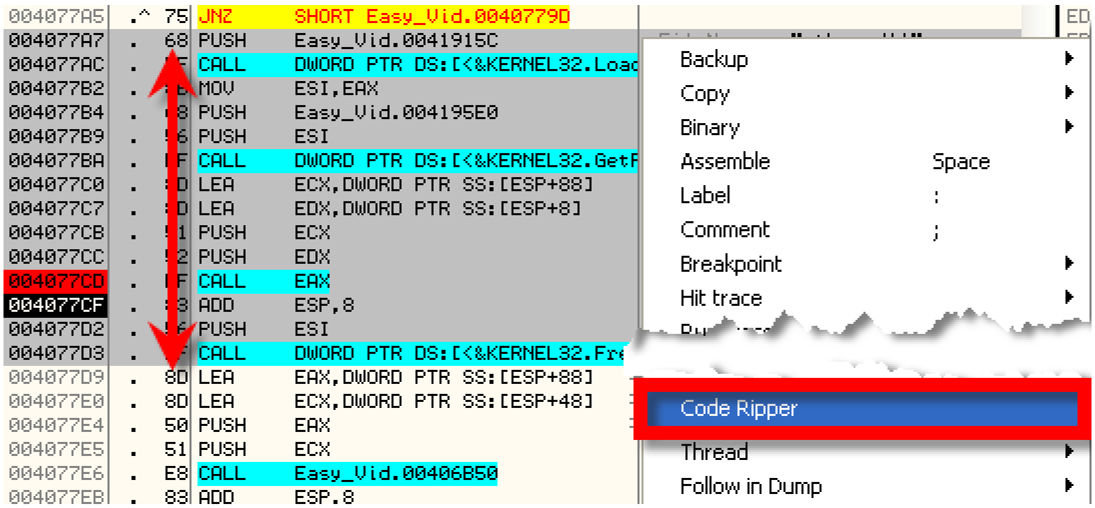
افتح كالمعتاد ال IDE المفضل عندك لكتابة الكيجين واليك الخطوات التي ستتبعها :

- نقرأ الإسم المدخل من Edit box الاسم
- تحميل المكتبة ether.dll
- البحث عن الدالة reg_code
- المناداة على الدالة reg_code بالبارامترين : مؤشر الى البافر الذي سيتلقى السيريال و مؤشر الى البافر الذي يحتوي الاسم
- نطبع في Edit box الخاصة بالسيريال, السيريال المحسوب.

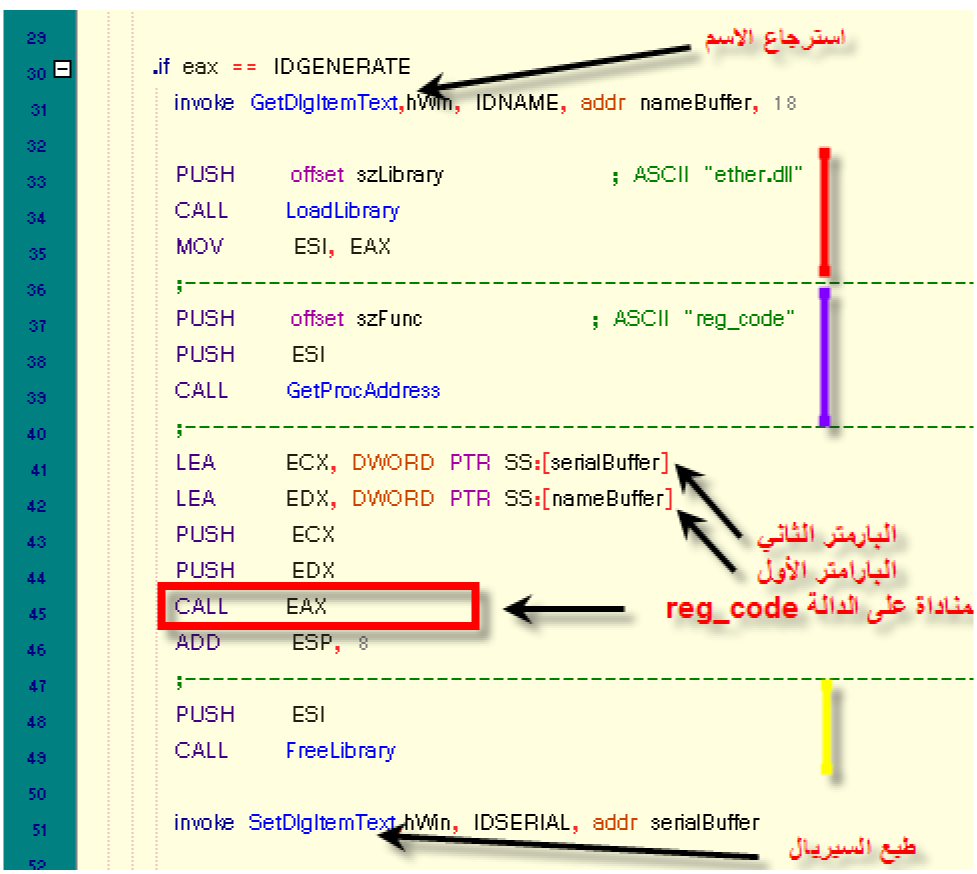
6.3.1 . الكيجين بالأسمبلي:

سنستعمل RADASM و MASM32:

الآن ظلل على الاوامر التي سنحتاجها لعمل الكيجين ثم اختر Code Ripper (اذا لم تثبت البلاغين فعليك الرجوع الى الفقرة التي تتحدث عن اعدادات OLLYDBG.



ثم نسخ التعليمات من نافذة البلاغين و الصقها في المكان الذس خصصته لحساب السيرال في الكيجين لديك. ز هذا مثال على الشكل النهائي بعد التعديلات الضرورية:



السورس كاملا هنا

تذكر ان تنسخ المكتبة ether.dll الى نفس مسار الكيجين.

6.3.2 . الكيجين ب C

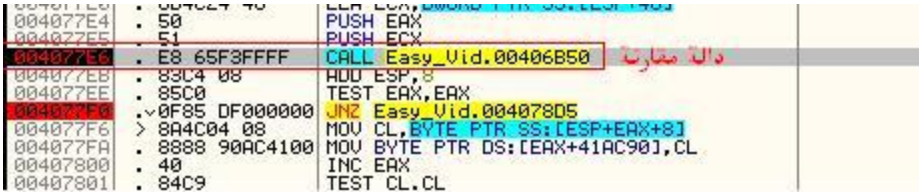
في هذا المرفق

4.6 . صناعة PATCH :

1.4.6 . البداية :

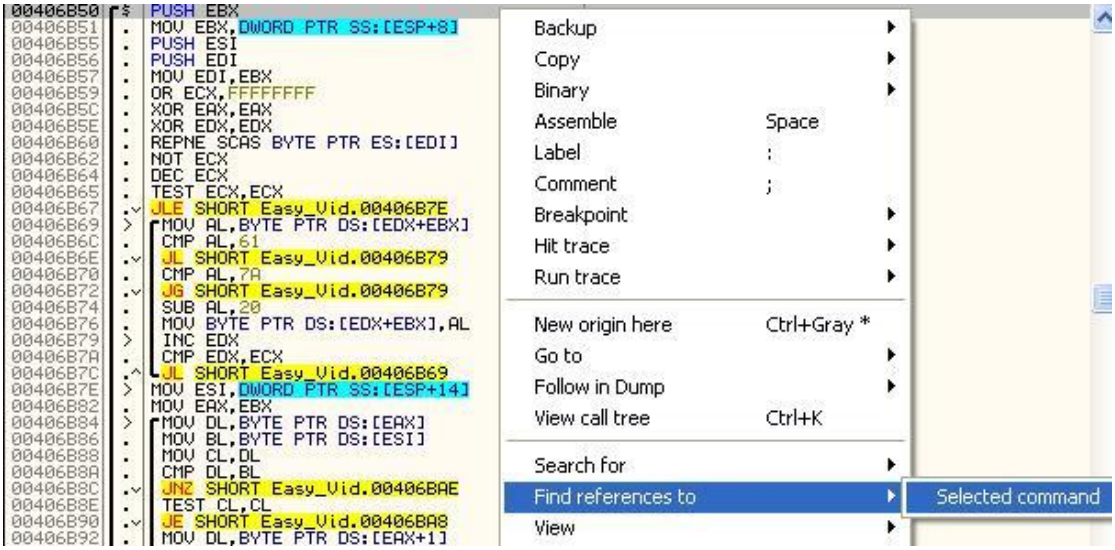
لاحظ إذا تم تعديل هذه القفزة فقط عند إعادة تشغيل البرنامج يكتشف الأمر

لنتأمل قليلا



لندخل لدالة المقارنة

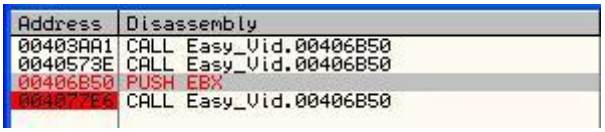
سترى



```
00406B50 r$ PUSH EBX
00406B51 .  MOV EBX,DWORD PTR SS:[ESP+8]
00406B55 .  PUSH ESI
00406B56 .  PUSH EDI
00406B57 .  MOV EDI,EBX
00406B59 .  OR ECX,FFFFFFFF
00406B5C .  XOR EAX,EAX
00406B5E .  XOR EDX,EDX
00406B60 .  REPNE SCAS BYTE PTR ES:[EDI]
00406B62 .  NOT ECX
00406B64 .  DEC ECX
00406B65 .  TEST ECX,ECX
00406B67 .v JLE SHORT Easy_Vid.00406B7E
00406B69 >  rMOV AL,BYTE PTR DS:[EDX+EBX]
00406B6C .  CMP AL,61
00406B6E .v JL SHORT Easy_Vid.00406B79
00406B70 .  CMP AL,7A
00406B72 .v JG SHORT Easy_Vid.00406B79
00406B74 .  SUB AL,20
00406B76 .  MOV BYTE PTR DS:[EDX+EBX],AL
00406B79 >  INC EDX
00406B7A .  CMP EDX,ECX
00406B7C .^ LJL SHORT Easy_Vid.00406B69
00406B7E >  MOV ESI,DWORD PTR SS:[ESP+14]
00406B82 .  MOV EAX,EBX
00406B84 >  rMOV DL,BYTE PTR DS:[EAX]
00406B86 .  MOV BL,BYTE PTR DS:[ESI]
00406B88 .  MOV CL,DL
00406B8A .  CMP DL,BL
00406B8C .v JNZ SHORT Easy_Vid.00406BAE
00406B8E .  TEST CL,CL
00406B90 .v JE SHORT Easy_Vid.00406BA8
00406B92 .  MOV DL,BYTE PTR DS:[EAX+1]
```

في نهاية دالة المقارنة إما eax=0 أو eax=-1

اختر الخيار فوق لنعرف المواضع التي استدعيت منها الدالة

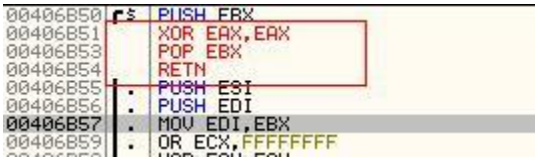


```
Address   Disassembly
00403AA1  CALL Easy_Vid.00406B50
0040573E  CALL Easy_Vid.00406B50
00406B50  PUSH EBX
004077E6  CALL Easy_Vid.00406B50
```

هناك **ثلاثة مواضع**

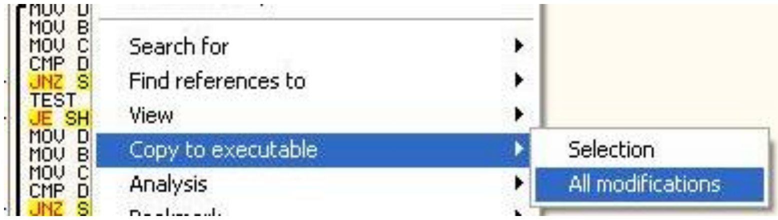
- وهذا طبيعي عند بداية البرنامج و عند التسجيل وعند الضغط على About
- والتعديل في هذه الدالة أسهل من التعديل في كل المواضع
- والتعديل المطلوب هو أن نجعل الدالة تعيد الصفر لأن القفزة jne

لنعدل هنا



```
00406B50 r$ PUSH EBX
00406B51    XOR EAX,EAX
00406B53    POP EBX
00406B54    RETN
00406B55 .  PUSH ESI
00406B56 .  PUSH EDI
00406B57 .  MOV EDI,EBX
00406B59 .  OR ECX,FFFFFFFF
```

الآن احفظ



والآن copy all ← ثم ضغطه يمين save file ← وجرب : رائع يعمل

الآن علينا تحديد أين يسجل البرنامج بياناته كي يتم إدخالها مع الباتش

لاحظ

لاحظ هذا option.ini و WritePrivateProfileString

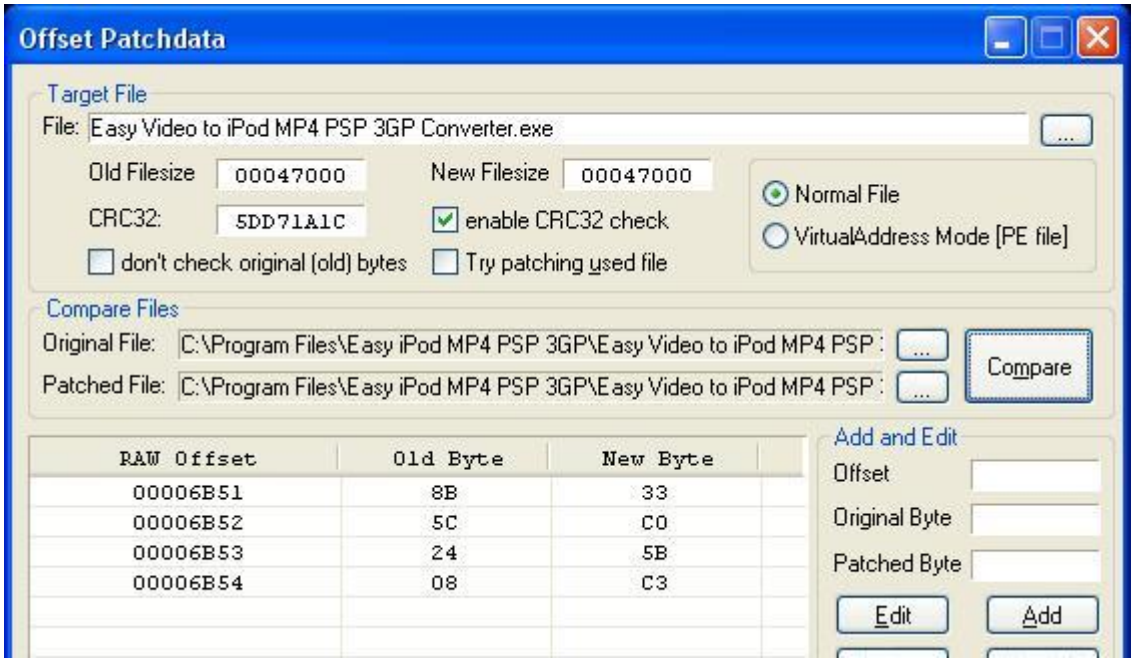
هذه الدالة تكتب بالملف عند key معين و باختصار البرنامج يكتب البيانات في الملف الموجود في نفس المسار ويحمل اسم option.ini

اذهب وتأكد

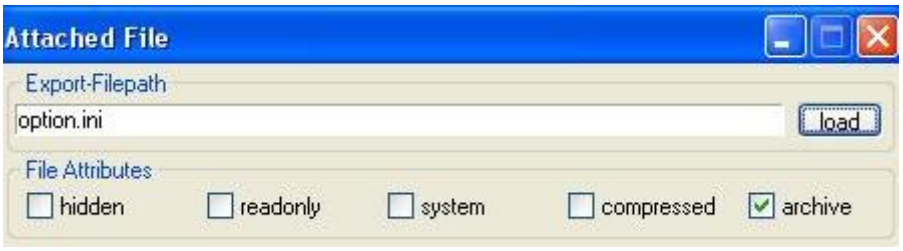
الآن كل شيء تمام, عرفنا ماذا نعدل و أين البيانات

2.4.6 . باستخدام برنامج DUP :

1. شغل dup
2. New
3. Add
4. Offset patch
5. حدد الملفين الأصلي و الكسور



6. ثم Add
7. Attach file
8. حدد الملف option.ini
9. بعد تعديل ال username فيه للاسم الذي تريد



01.ثم create Patch

3.4.6 . برمجيا :

سنكتب كودا برمجيا يعبر عما قمنا به (البرنامج في المرفقات)

6.5 . صناعة LOADER :

(في المرفقات)


الباب
الثالث
الملحق السابع
هديه الكتاب

7. ملحق 7: هدية الكتاب (ساخنة من الفرن)

في هذا الملحق سنرى ان شاء الله درسا حول كيفية صنع Template للبرنامج الجديد و الرائع uPPP الذي يمكن اعتباره و بحق خلفا جيدا ورائعا لبرنامج رائع ايضا, و الذي هو DuP "صانع الباتشات الشهير".

سيكون درسا شاملا لجميع الحالات الممكن الوقوع فيها بعد كسر برنامج معين.

و دون اطالة, حمل البرنامج من هنــا. و للإشارة فقط, لابد من توفر جهازك على Dotnet Framework ليعمل صانع الباتشات الجديد, لكن الباتش الناتج مصنوع ب MASM و سيكون Standalone.

و لنبدأ بسم الله

7.1. معلومات عامة مهمة:

1.1.1. تذكير ببعض المصطلحات:

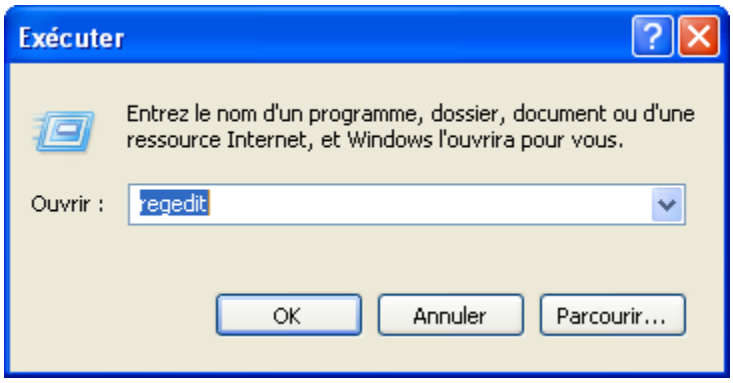
- **كراك (CRACK):** لفظ يجمع كل انواع و امكانيات و طرق كسر حماية ما (يعني شمل ما سيأتي ذكره تحت و دون استثناء)
- **باتش (PATCH):** هو الطريقة الأكثر شيوعا و اللفظ يطلق على الطريقة التي نغير فيها احد أو مجموعة من بايتات (Bytes) الملف الاصلي على القرص الصلب.
- **لودر (LOADER):** سبق التطرق الى هذا اللفظ. لكن نذكر انه تغيير بايت أو عدة بايتات في الملف الأصلي لكن اثناء تحميله الى الذاكرة دون حفظ للتغييرات على القرص الصلب و الا فسيصير PATCH.
- **كيجين (KEYGENERATOR):** سبق التطرق اليه ايضا, و هذه اصعب و اكثر الطرق استهلاكا للوقت, لأنها تتطلب فهما مدققا و كاملا لخوارزمية انشاء السيريال بلغة الأسمبلي, و من تم عمل واحدة مشابهة لها بأي لغة يجيدها الكراكر.
- **الكيفايل مايكر (KEYFILEMAKER):** هو كذلك من اصعب الطرق و يتطلب نفس متطلبات الكيجين, لكن بدلا من اعطاء سيريال على الشاشة تنسخه, يعطيك ملفا تنسخه الى مكان معين (غالبا ما يكون نفس مجلد الضحية).
  و قد يحمل الملف أي لاحقة (و اشهر اللواحق : reg. و key. و ini. و txt.)
- جميع الطرق السابقة, قد تستعمل فيها اضافات لابد منها لعمل الضخية, كاضافة مفتاح الى الريجستري, او حذف ملف زائد يحدد وقت الانتهاء ...

1.1.2. طريقة اخذ المعلومات من ال REGISTRY

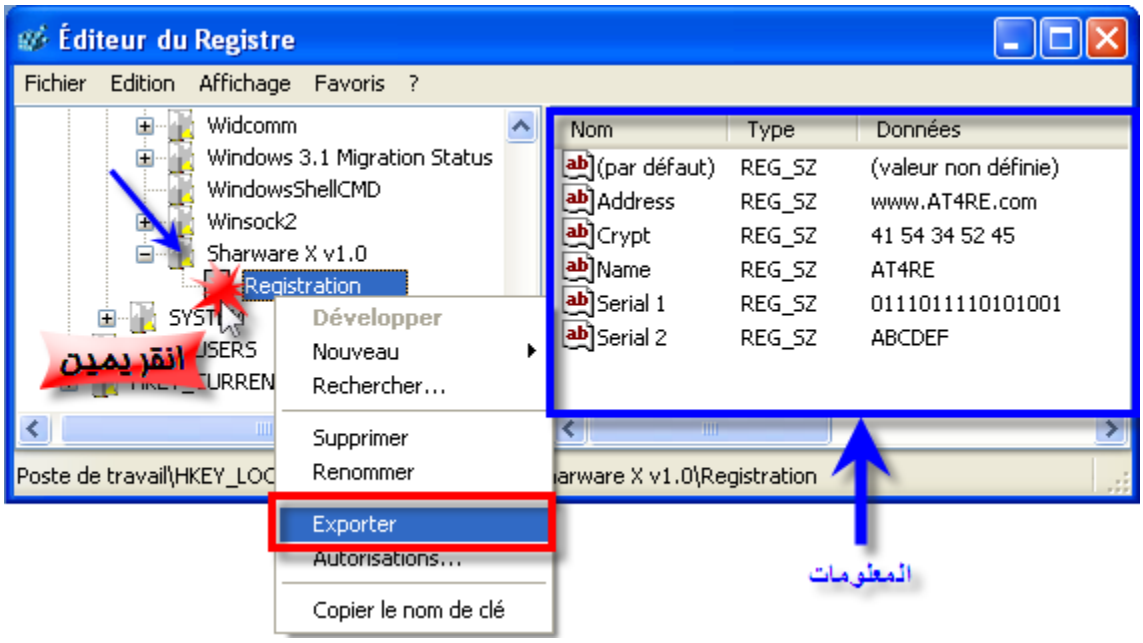
1. اضغط على التوالي : 

ثم 

ثم ادخل كلمة Regedit و اضغط OK كما في الصورة:



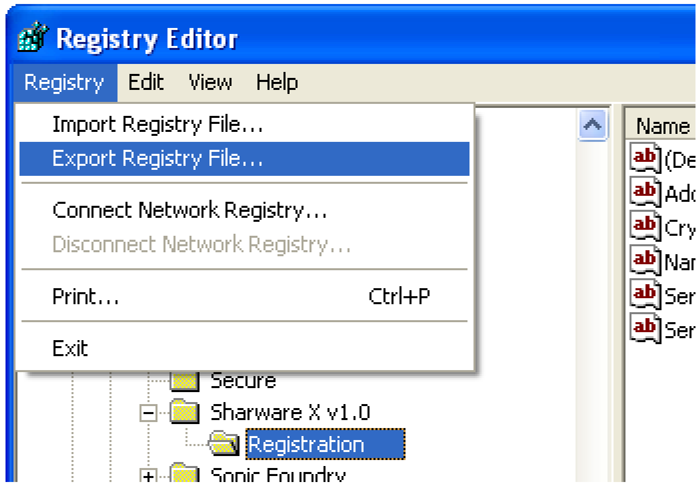
2. انقر يمين على المفتاح (Registration) الذي يضم القيم المحتوية على المعلومات الضرورية (الاطار الزرق)
3. نقر فوق Export و اختر مكانا تحفظ فيه الملف الناتج. (الصورة تحت)



ستحصل على:

صانع الباتشات uPPP لا يتعامل مع الملفات الناتجة عن REGEDIT 5.0 فما فوق, بل فقط مع 4.0 (و لا ادري التوافق مع النسخ تحت 4.0) لذا كنت تعمل على الوندوز XP فلا بد من عمل اضافي:

- فاما ان تفتح الملف بالنوتباد و تغير السطر الأول من Windows Registry Editor Version 5.00 الى « REGEDIT4 »
- أو ان تستعمل نسخة قديمة من Regedit (مرفقة هنا) مع الملاحظة ان زر الImport موجود في App Menu و ليس في ال Popup Menu الذي يطلع عند النقر يمين على المفتاح (انظر الصورة).



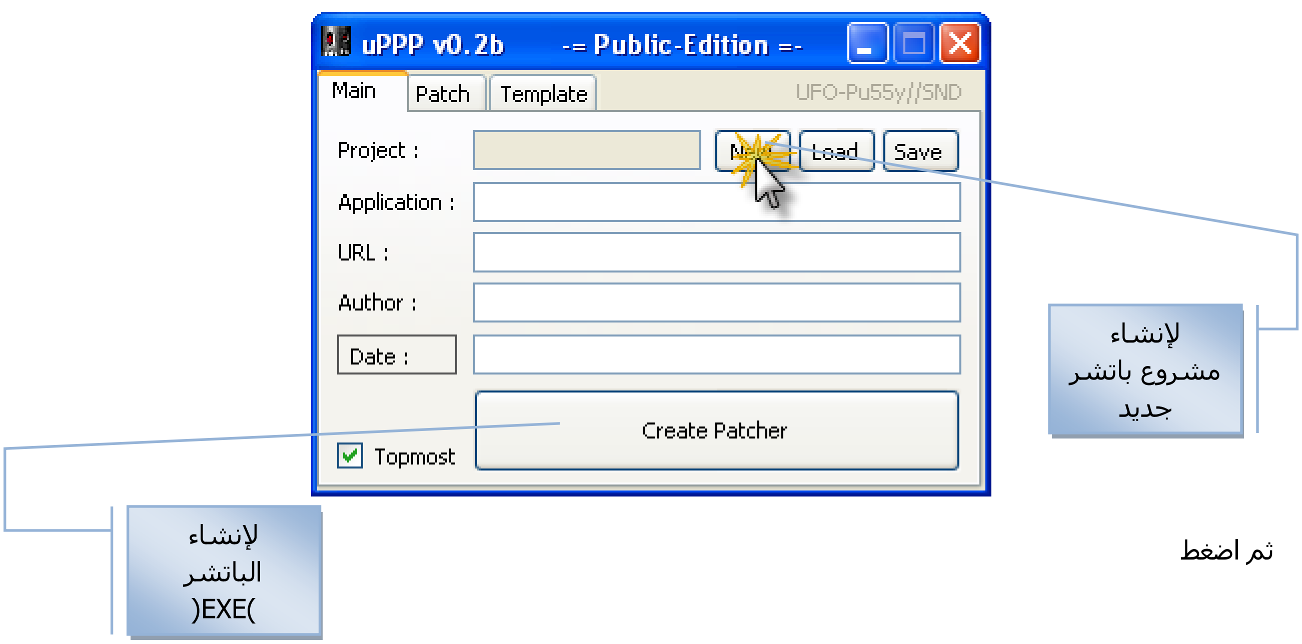
7.2. عمل الباتش متعدد الخصائص:

قبل البدأ سنفترض ما يلي

- ان البرنامج المكسور اسمه Shareware X v1.0
- انه يحفظ معلومات التسجيل في الريجسترى في المفتاح Registration

7.2.1. ادخال المعلومات اللازمة:

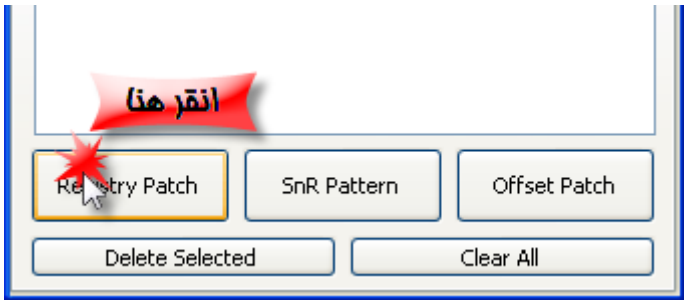
افتح البرنامج ثم اضغط New:



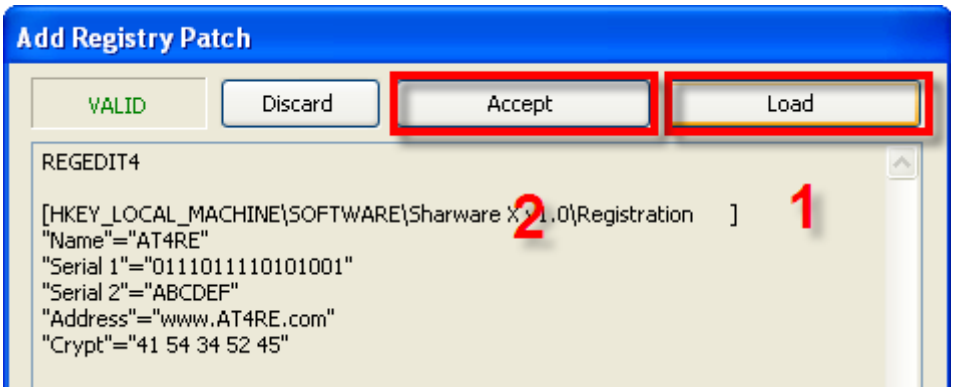
ثم اضغط

7.2.2 . اضافة REGISTRY PATCH:

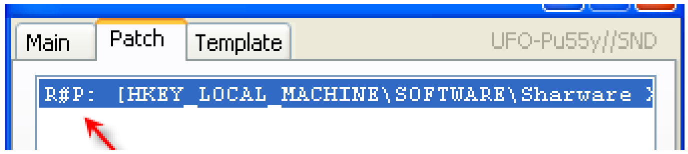
الآن, اذهب الى اللسان الثاني: Patch ثم لننقر على Registry Patch لاضافة معلومات الريجستري:



ثم



سيظهر الباتش المضاف في List:



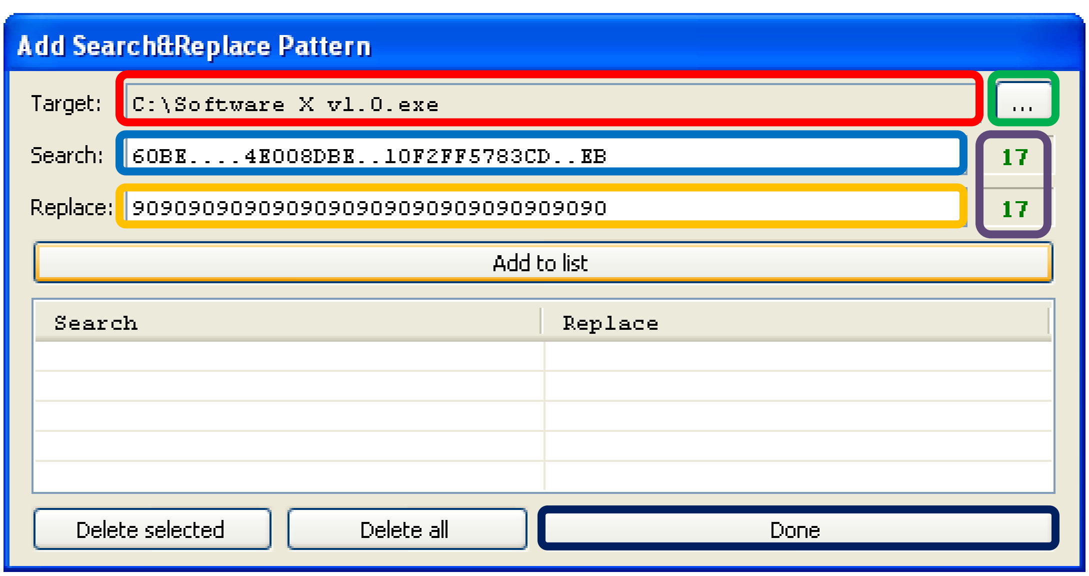
7.2.3 . إضافة SEARCH AND REPLACE PATTERN:

كل ما ينبغي معرفته هو ان:

- المَحرَفَ السحري هو النقطة (.) و تضعها مكان المَحرَفِ المتغير
- كل نقطتين تمثلان بايتا واحدا
- ان عدد محارف الباترن يجب ان يكون عددا زوجيا
- اذا لم تعرف اهمية ال Search&Replace Patch فلا مشكل, انتظر صدور النسخة الثانية من الكتاب ☺

لنبدأ الانشاء:

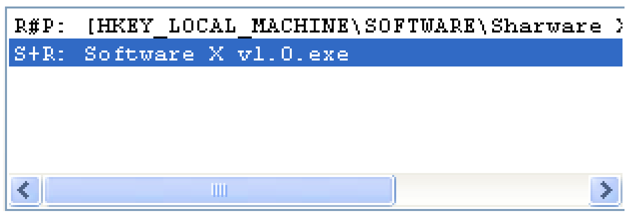
1 - اضغط الزر بالأخضر لفتح الملف المراد عمل باتش له (الاخضر) أو اعمل Drag and drop للملف في الخانة العليا (الاحمر)
2 - ادخل الباترن المراد البحث عنه في الخانة العليا الثانية (الأزرق)
3 - ادخل الباترن الذي تريد التعويض به في الخانة الثالثة (البرتقالي)
4 - انتبه الى لون الأعداد في المربع البنفسجي, يجب ان تكون خضراء (في حالة خطأ ستصبح حمراء)
5 - و أخيرا اضغط الزر الطويـــــــــــــــــــــــــــل Add to list لإضافة الباترن الأول الى اللائحة



كما يمكنك اضافة اكثر من باترن

Search	Replace
60BE....4E008DBE..10F2FF5783CD..EB	9090909090909090909090909090909090
EA0646......01DB75078B1E	A064688074701DB75078B1EA

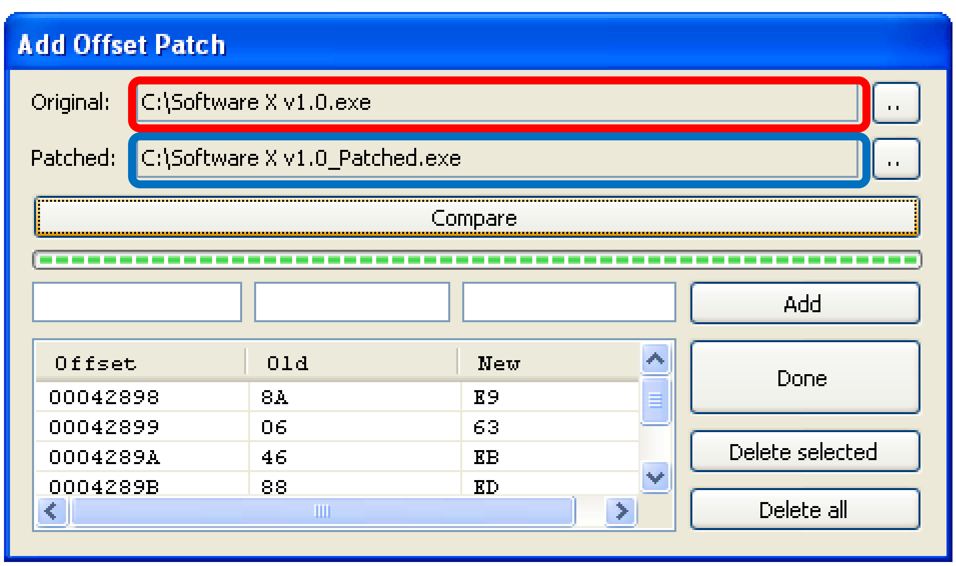
عند الإنتهاء, اضغط الزر Done (الاطار الأسود فوق) و سضاف الباترن باتشر الى اللائحة:



7.2.4 . اضافة أوفسيت باتش (باتش علدي)

لا شيء اسهل:

1 - اعمل Drag and Drop للملف الاصلي في الخانة العليا (الأحمر)
2 - اعمل Drag and Drop للملف المعدَّل في الخانة العليا (الأزرق)
3 - اضغط الزر الطـــــــــــــــــــــويل Compare



4 - لتغيير معلومات باتش-بايت, يمكنك النقر مرتين على السطر الذي يحتويه.
5 - يمكنك ايضا دخال معلومات الباتش Byte after byte, فقط ادخل في الخانة الأوى يسار العنوان (الأوفست و ليس ال VA), ثم ادخل في خانة الوسط القيمة الأصلية للبايت, ثم في خانة اليمين القيمة المعدَّلة.

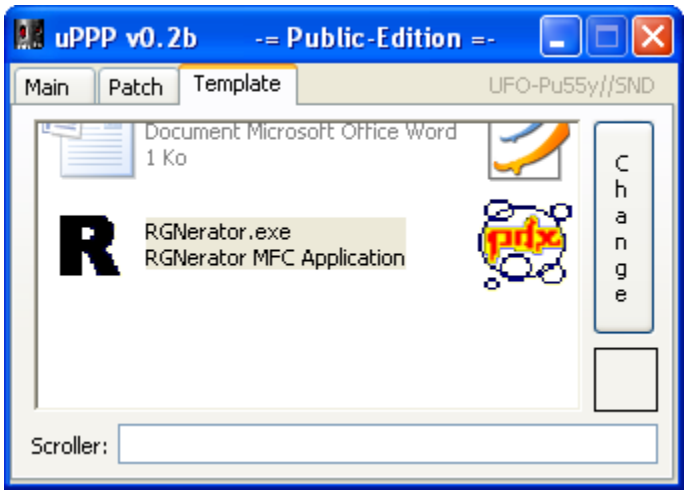
6 - ثم أخيرا الزر Done

```
R#P: [HKEY_LOCAL_MACHINE\SOFTWARE\Sharware ]
S+R: Software X v1.0.exe
O-P: Software X v1.0.exe
```

للتعديل على احد الباتشات, يكفي النقر على سطره مرتين بالفأرة.

7.2.5 . اختيار اللباس ☺

انقر على اللسان Template و سترى ان داخل نافذة الباتشر اصبح شفافا:



اضغط على الزر العمودي Change لتختار السكين الذي تريد. هناك 4 سكينات مرفقة: 3 من صانع البرنامج و واحدة من الأخ RobenHoodArab بارك الله فيه و هذه Preview لها:

ثم لإضافة Text متحرك, اكتب ما تشاء في الخانة السفلى (الأحمر):

الآن يمكنك حفظ الباتش بالرجوع الى اللسان Main و النقر على الزر الطويل للحصول على الباتش الجميل و الذي تدل على نجاح انشائه, الرسالة:

و قبل مغادرتك, لا تنس حفظ المشروع فقد تحتاج اليه مستقبلا بالضغط على 

و هذا هة الشكل النهائي:

هناك اعدادات متقدمة قليلة اخرى لكن لن نتطرق اليها في هذا الكتاب, ربما في نسخته الثانية بإذن الله.

انتهى.

منكم و اليكم و السلام عليكم و رحمة الله