An Exercise in Approaching a Target Differently

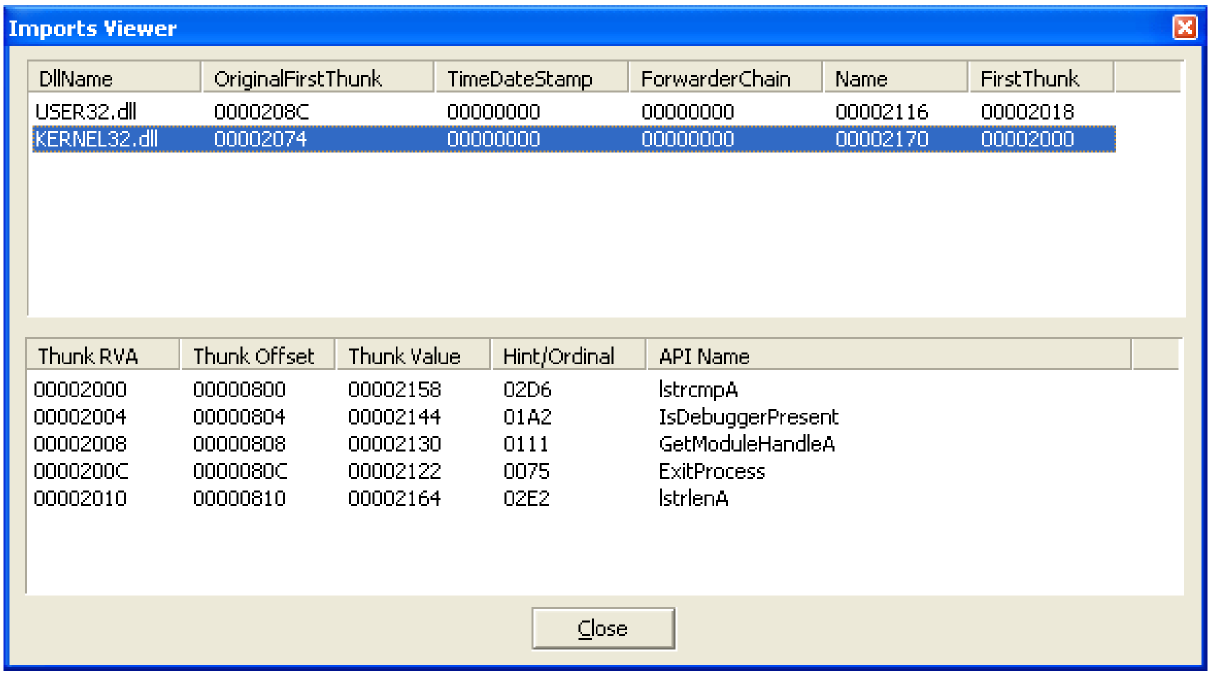
Required

Disassembler (I'm using IDA)
PEID
Nektra Spy Studio
Simple Crackme : Using ZeroZero's Miracle
Brain

Methodology

I wanted to experiment with API hooking and Nektra Spy Studio to examine how it could help me approach a target. So I grabbed a simple crackme and tried to experiment.

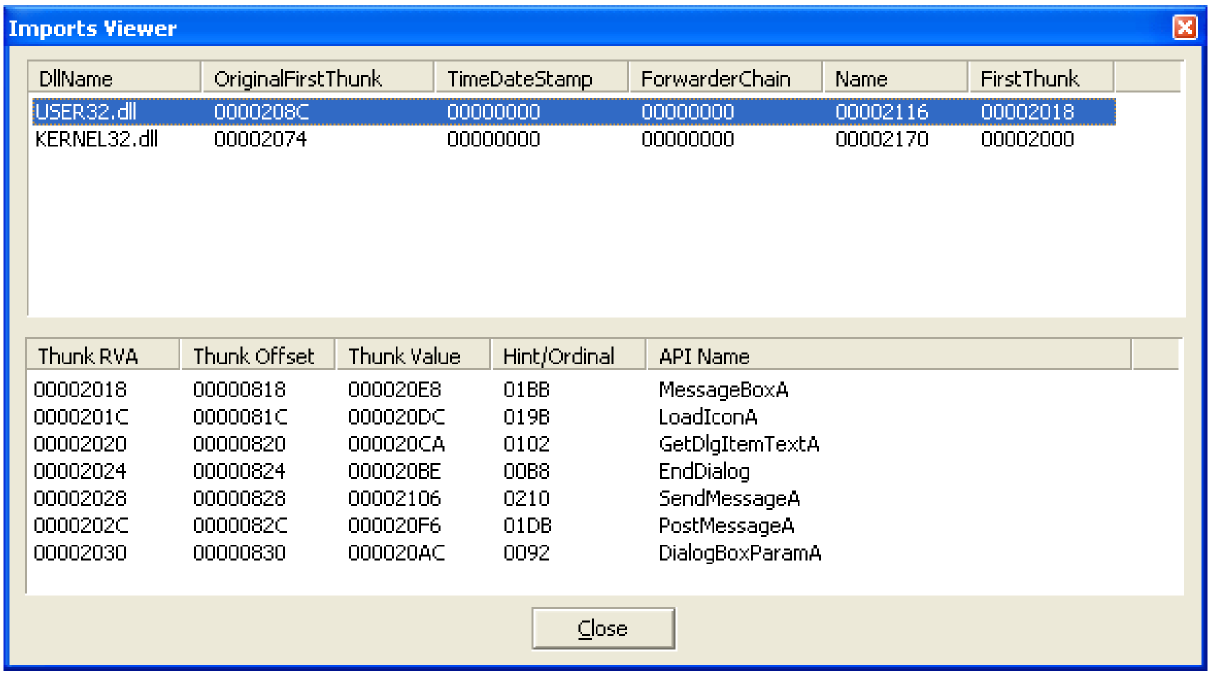
I had a quick look at the imports for the crackme via PEiD :



DllName	OriginalFirstThunk	TimeDateStamp	ForwarderChain	Name	FirstThunk
USER32.dll	0000208C	00000000	00000000	00002116	00002018
KERNEL32.dll	00002074	00000000	00000000	00002170	00002000

Thunk RVA	Thunk Offset	Thunk Value	Hint/Ordinal	API Name
00002000	00000800	00002158	02D6	lstrcmpA
00002004	00000804	00002144	01A2	IsDebuggerPresent
00002008	00000808	00002130	0111	GetModuleHandleA
0000200C	0000080C	00002122	0075	ExitProcess
00002010	00000810	00002164	02E2	lstrlenA

lstrcmpA looks like a good one to start with! Looks like a small amount of AntiDebugging as well – should be no problem. Anything in User32 for us to look at?

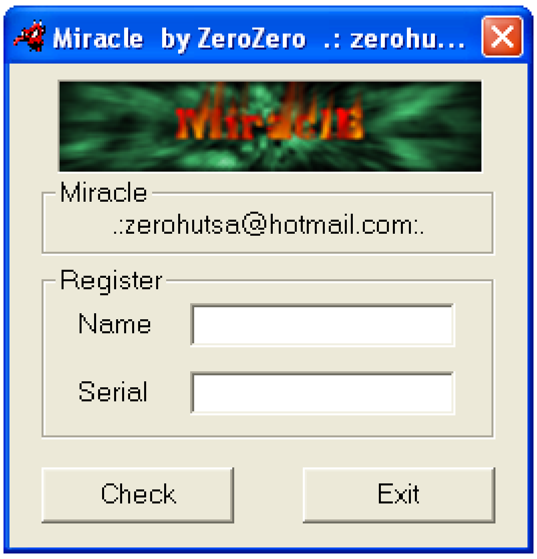


DllName	OriginalFirstThunk	TimeDateStamp	ForwarderChain	Name	FirstThunk
USER32.dll	0000208C	00000000	00000000	00002116	00002018
KERNEL32.dll	00002074	00000000	00000000	00002170	00002000

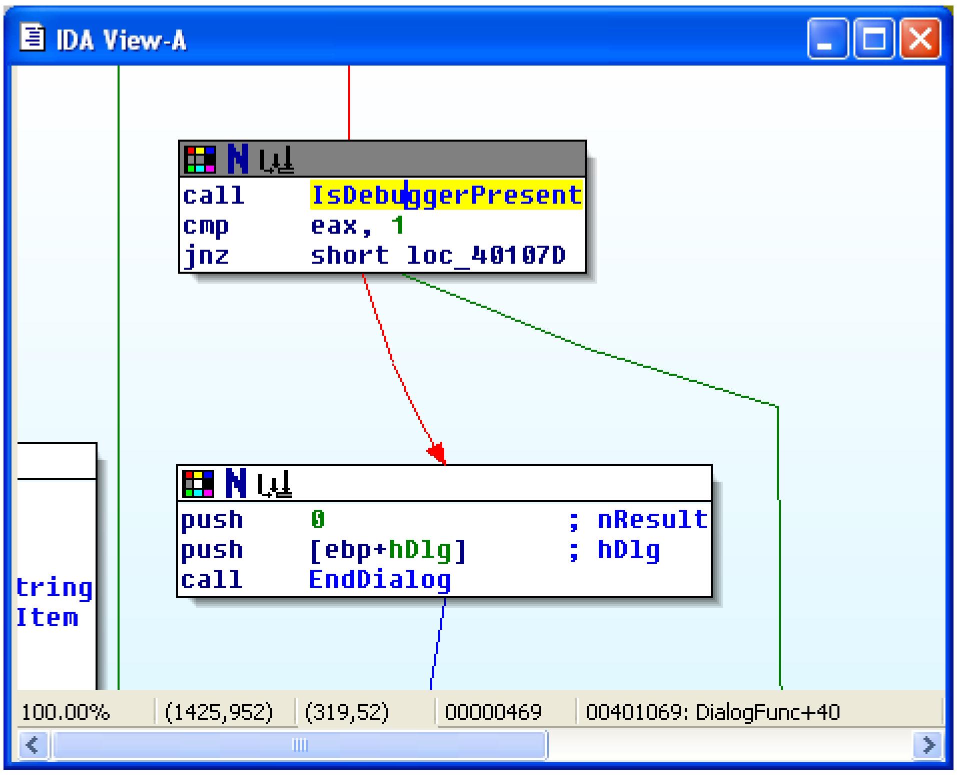
Thunk RVA	Thunk Offset	Thunk Value	Hint/Ordinal	API Name
00002018	00000818	000020E8	01BB	MessageBoxA
0000201C	0000081C	000020DC	019B	LoadIconA
00002020	00000820	000020CA	0102	GetDlgItemTextA
00002024	00000824	000020BE	00B8	EndDialog
00002028	00000828	00002106	0210	SendMessageA
0000202C	0000082C	000020F6	01DB	PostMessageA
00002030	00000830	000020AC	0092	DialogBoxParamA

GetDlgItemTextA is an interesting one too.

Fire up miracle.exe and you should see this :



Looks like it is running just fine, no anti-debug has fired yet by the looks of it. A quick check on the calls to IsDebuggerPresent brings us here in IDA :



```
call    IsDebuggerPresent
cmp     eax, 1
jnz     short loc_40107D
```

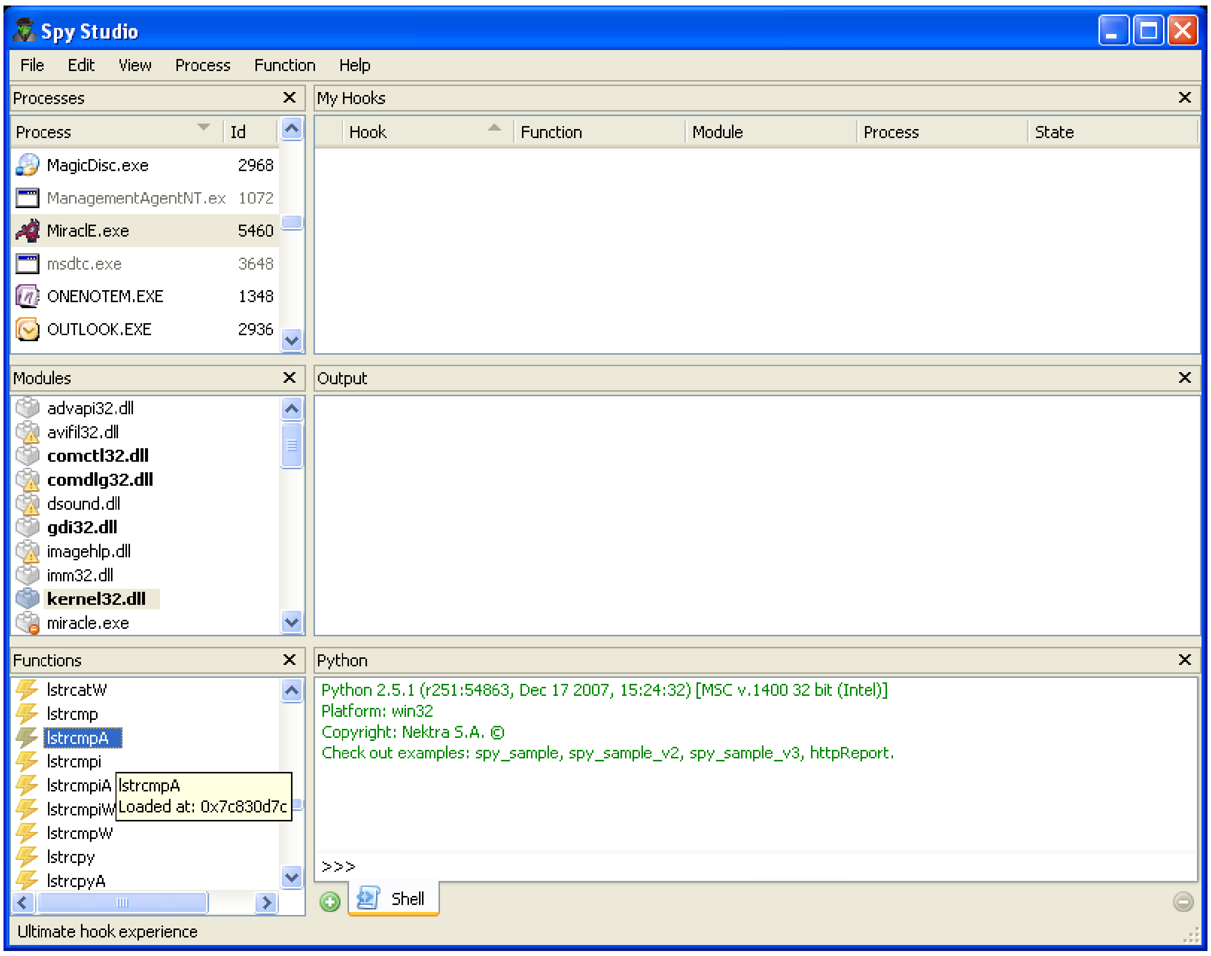
```
push    0                   ; nResult
push    [ebp+hDlg]          ; hDlg
call    EndDialog
```

```
100.00%   (1425,952)   (319,52)   00000469   00401069: DialogFunc+40
```

Looks like it will only fire when the program is started so the fact that it is running is a good sign!

Now let’s have a look and see how we can API hook this and what we can do from there.

Loading up Spy Studio we select the process we want to install a hook into and then the correct module (lstrcmpA is part of Kernel32). Once we select the module, a list of all the available exports are shown in the Functions window.



Process	Id
MagicDisc.exe	2968
ManagementAgentNT.ex	1072
MiraclE.exe	5460
msdtc.exe	3648
ONENOTEM.EXE	1348
OUTLOOK.EXE	2936

Hook	Function	Module	Process	State

- advapi32.dll
- avifil32.dll
- **comctl32.dll**
- **comdlg32.dll**
- dsound.dll
- **gdi32.dll**
- imagehlp.dll
- imm32.dll
- **kernel32.dll**
- miracle.exe

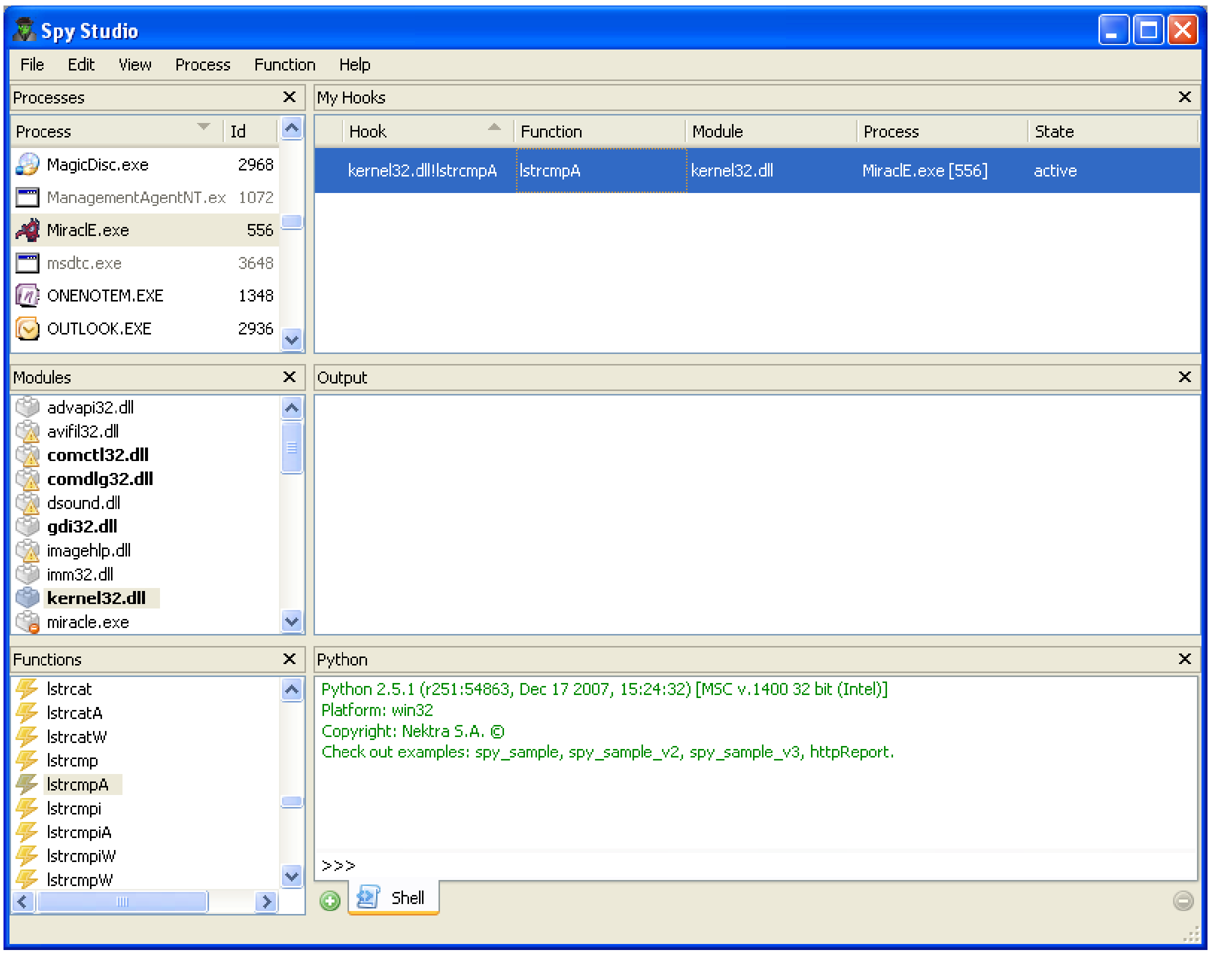
- lstrcatW
- lstrcmp
- lstrcmpA
- lstrcmpi
- lstrcmpiA
- lstrcmpiW
- lstrcmpW
- lstrcpy
- lstrcpyA

```
lstrcmpA
Loaded at: 0x7c830d7c
```

```
Python 2.5.1 (r251:54863, Dec 17 2007, 15:24:32) [MSC v.1400 32 bit (Intel)]
Platform: win32
Copyright: Nektra S.A. ©
Check out examples: spy_sample, spy_sample_v2, spy_sample_v3, httpReport.

>>>
```

Let's install the hook. Once installed it should look like this :



Process	Id
MagicDisc.exe	2968
ManagementAgentNT.ex	1072
MiraclE.exe	556
msdtc.exe	3648
ONENOTEM.EXE	1348
OUTLOOK.EXE	2936

Hook	Function	Module	Process	State
kernel32.dll!lstrcmpA	lstrcmpA	kernel32.dll	MiraclE.exe [556]	active

- advapi32.dll
- avifil32.dll
- **comctl32.dll**
- **comdlg32.dll**
- dsound.dll
- **gdi32.dll**
- imagehlp.dll
- imm32.dll
- **kernel32.dll**
- miracle.exe

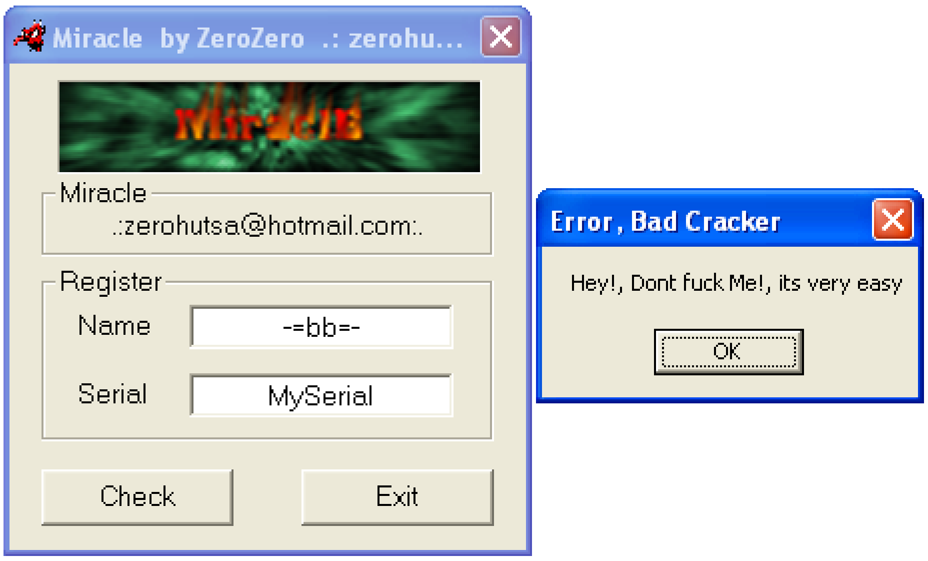
- lstrcat
- lstrcatA
- lstrcatW
- lstrcmp
- lstrcmpA
- lstrcmpi
- lstrcmpiA
- lstrcmpiW
- lstrcmpW

```
Python 2.5.1 (r251:54863, Dec 17 2007, 15:24:32) [MSC v.1400 32 bit (Intel)]
Platform: win32
Copyright: Nektra S.A. ©
Check out examples: spy_sample, spy_sample_v2, spy_sample_v3, httpReport.

>>>
```

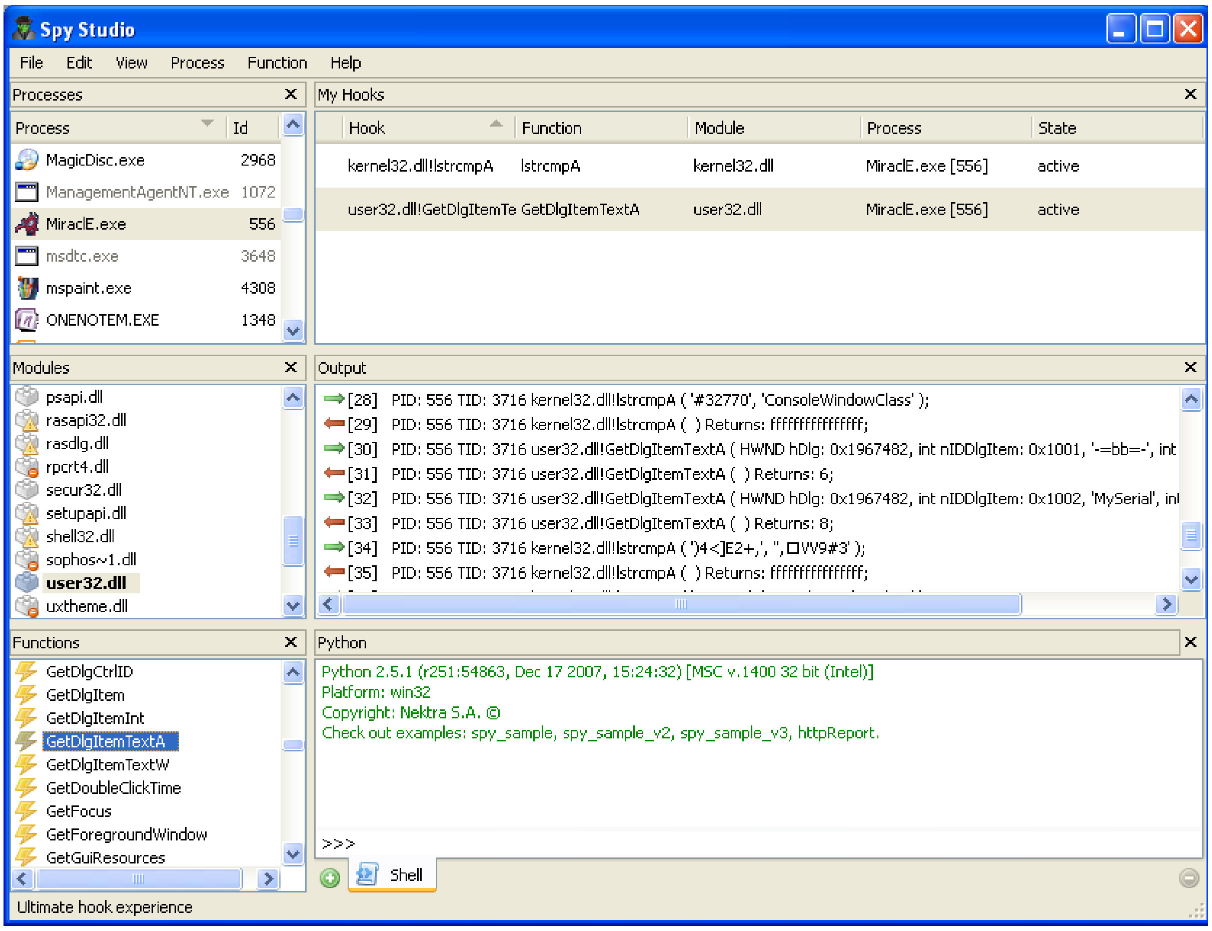
Perfect. Let's add one to GetDlgItemTextA as well whilst we are here.

Now – let's run through a quick test of the hook and click on a button in the crackme.



Hey!, Dont fuck Me!, its very easy

Well, that’s not the right serial. Let’s see what Nektra says :



Process	Id
MagicDisc.exe	2968
ManagementAgentNT.exe	1072
MiraclE.exe	556
msdtc.exe	3648
mspaint.exe	4308
ONENOTEM.EXE	1348

Hook	Function	Module	Process	State
kernel32.dll!lstrcmpA	lstrcmpA	kernel32.dll	MiraclE.exe [556]	active
user32.dll!GetDlgItemTe	GetDlgItemTextA	user32.dll	MiraclE.exe [556]	active

```
psapi.dll
rasapi32.dll
rasdlg.dll
rpcrt4.dll
secur32.dll
setupapi.dll
shell32.dll
sophos~1.dll
user32.dll
uxtheme.dll
```

```
GetDlgCtrlID
GetDlgItem
GetDlgItemInt
GetDlgItemTextA
GetDlgItemTextW
GetDoubleClickTime
GetFocus
GetForegroundWindow
GetGuiResources
```

```
[28]  PID: 556 TID: 3716 kernel32.dll!lstrcmpA ( '#32770', 'ConsoleWindowClass' );
[29]  PID: 556 TID: 3716 kernel32.dll!lstrcmpA (  ) Returns: ffffffffffffffff;
[30]  PID: 556 TID: 3716 user32.dll!GetDlgItemTextA ( HWND hDlg: 0x1967482, int nIDDlgItem: 0x1001, '-=bb=-', int
[31]  PID: 556 TID: 3716 user32.dll!GetDlgItemTextA (  ) Returns: 6;
[32]  PID: 556 TID: 3716 user32.dll!GetDlgItemTextA ( HWND hDlg: 0x1967482, int nIDDlgItem: 0x1002, 'MySerial', in
[33]  PID: 556 TID: 3716 user32.dll!GetDlgItemTextA (  ) Returns: 8;
[34]  PID: 556 TID: 3716 kernel32.dll!lstrcmpA ( ')4<]E2+,', '',□VV9#3' );
[35]  PID: 556 TID: 3716 kernel32.dll!lstrcmpA (  ) Returns: ffffffffffffffff;
```

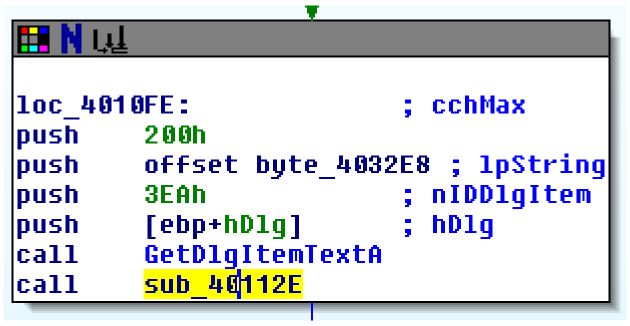
```
Python 2.5.1 (r251:54863, Dec 17 2007, 15:24:32) [MSC v.1400 32 bit (Intel)]
Platform: win32
Copyright: Nektra S.A. ©
Check out examples: spy_sample, spy_sample_v2, spy_sample_v3, httpReport.

>>>
```

The program grabs our input through GetDlgItemTextA and the proceeds to manipulate our input to generate a value to compare to a known value. We can tell it is a fixed value as changing our username or password does not alter the value that it is compared to.

>> PID: 556 TID: 3716 kernel32.dll!lstrcmpA (')4<]E2+,', '',□VV9#3');

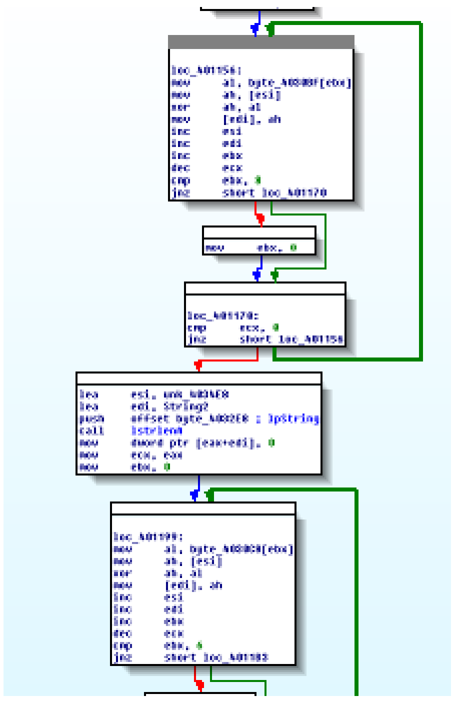
We could simply patch the target to return true at this point – but we want to reverse it properly. Let’s have a quick peek through IDA and see where this might be happening.



```
loc_4010FE:                    ; cchMax
push    200h
push    offset byte_4032E8 ; lpString
push    3EAh                   ; nIDDlgItem
push    [ebp+hDlg]             ; hDlg
call    GetDlgItemTextA
call    sub_40112E
```

GetDlgItemTextA followed by a call to a subroutine ... got to be worth a peek.

The subroutine looks like this :



```
loc_401156:
mov     al, byte_4030BF[ebx]
mov     ah, [esi]
xor     ah, al
mov     [edi], ah
inc     esi
inc     edi
inc     ebx
dec     ecx
cmp     ebx, 8
jnz     short loc_401170
```

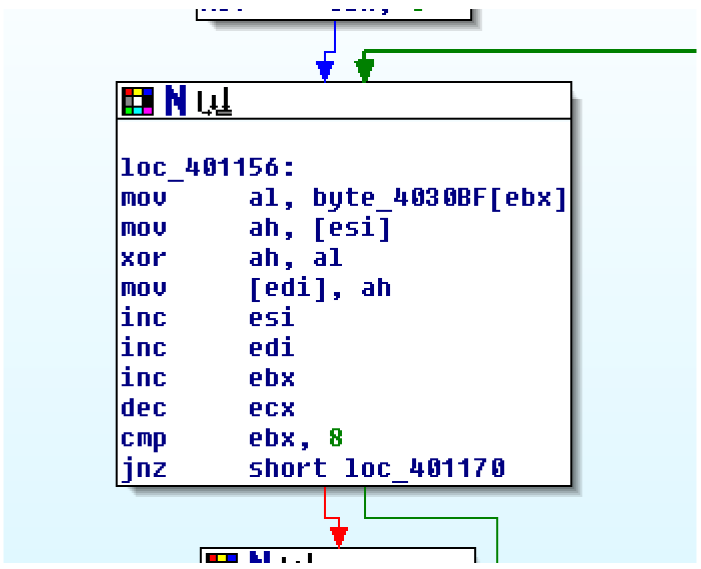
```
mov     ebx, 0
```

```
loc_401170:
cmp     ecx, 0
jnz     short loc_401156
```

```
lea     esi, unk_4034E8
lea     edi, String2
push    offset byte_4032E8 ; lpString
call    lstrlenA
mov     dword ptr [eax+edi], 0
mov     ecx, eax
mov     ebx, 0
```

```
loc_401199:
mov     al, byte_4030C8[ebx]
mov     ah, [esi]
xor     ah, al
mov     [edi], ah
inc     esi
inc     edi
inc     ebx
dec     ecx
cmp     ebx, 6
jnz     short loc_4011B3
```

A closer look at one of the subroutines shows :

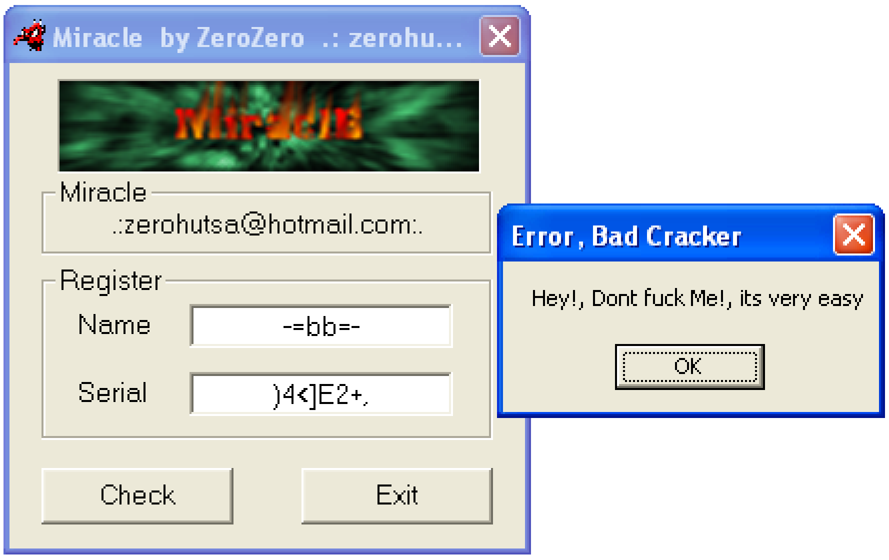


```
loc_401156:
mov     al, byte_4030BF[ebx]
mov     ah, [esi]
xor     ah, al
mov     [edi], ah
inc     esi
inc     edi
inc     ebx
dec     ecx
cmp     ebx, 8
jnz     short loc_401170
```

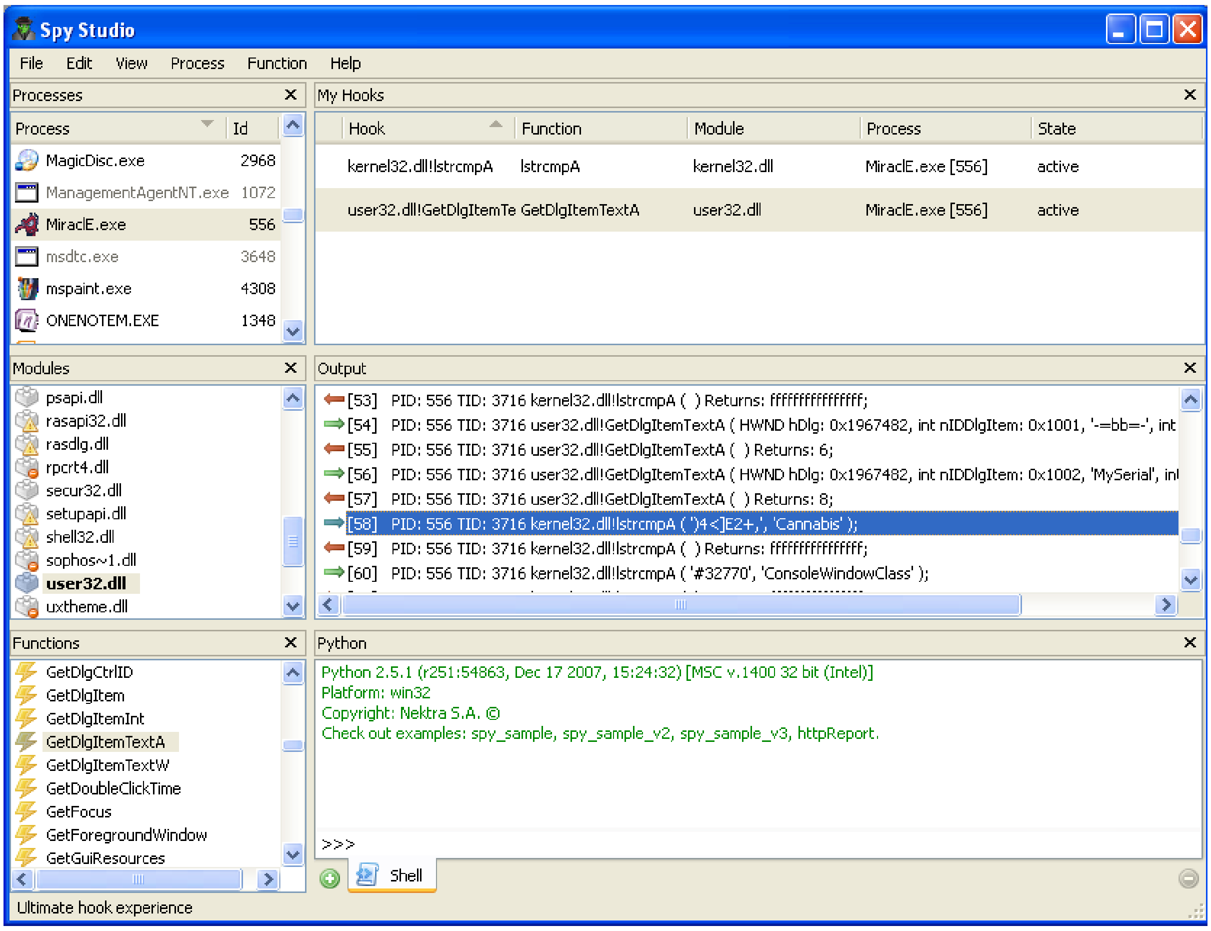
Looks like some XORing going on in a loop through each of the characters or our input and XORing it against the data at two different addresses.

What do we know about XOR – it is reversible!

Let's enter that pre-computed value into the crackme and see what happens.


Miracle by ZeroZero .: zerohu...
Miracle
.:zerohutsa@hotmail.com:.
Register
Name
-=bb=-
Serial
)4<]E2+,
Check
Exit
Error , Bad Cracker
Hey!, Dont fuck Me!, its very easy
OK

Ho Hum … but wait. What can we see using Nektra?



Process	Id
MagicDisc.exe	2968
ManagementAgentNT.exe	1072
MiraclE.exe	556
msdtc.exe	3648
mspaint.exe	4308
ONENOTEM.EXE	1348

Hook	Function	Module	Process	State
kernel32.dll!lstrcmpA	lstrcmpA	kernel32.dll	MiraclE.exe [556]	active
user32.dll!GetDlgItemTe	GetDlgItemTextA	user32.dll	MiraclE.exe [556]	active

```
psapi.dll
rasapi32.dll
rasdlg.dll
rpcrt4.dll
secur32.dll
setupapi.dll
shell32.dll
sophos~1.dll
user32.dll
uxtheme.dll
```

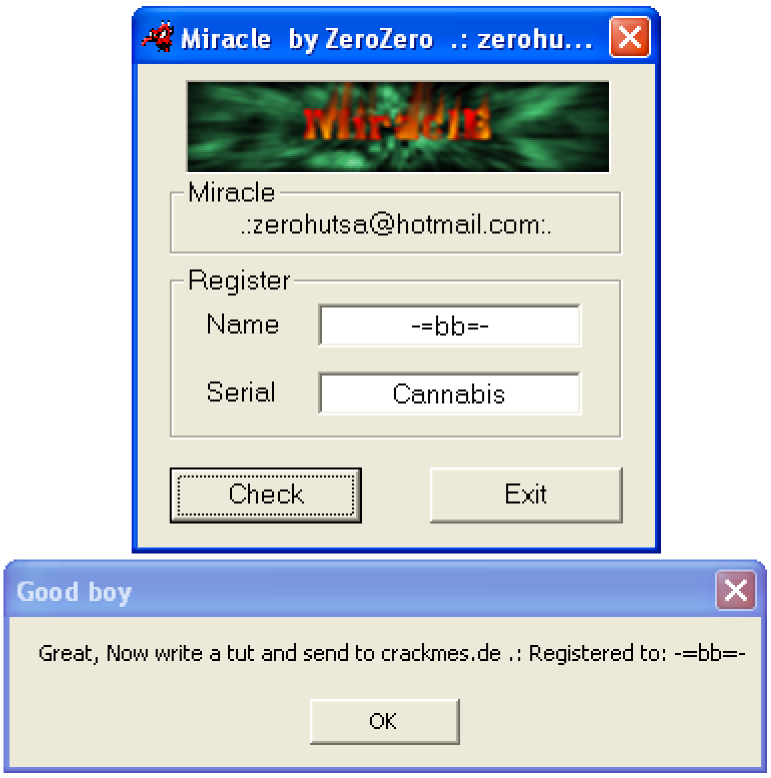
```
[53] PID: 556 TID: 3716 kernel32.dll!lstrcmpA (  ) Returns: ffffffffffffffff;
[54] PID: 556 TID: 3716 user32.dll!GetDlgItemTextA ( HWND hDlg: 0x1967482, int nIDDlgItem: 0x1001, '-=bb=-', int
[55] PID: 556 TID: 3716 user32.dll!GetDlgItemTextA (  ) Returns: 6;
[56] PID: 556 TID: 3716 user32.dll!GetDlgItemTextA ( HWND hDlg: 0x1967482, int nIDDlgItem: 0x1002, 'MySerial', in
[57] PID: 556 TID: 3716 user32.dll!GetDlgItemTextA (  ) Returns: 8;
[58] PID: 556 TID: 3716 kernel32.dll!lstrcmpA ( ')4<]E2+,', 'Cannabis' );
[59] PID: 556 TID: 3716 kernel32.dll!lstrcmpA (  ) Returns: ffffffffffffffff;
[60] PID: 556 TID: 3716 kernel32.dll!lstrcmpA ( '#32770', 'ConsoleWindowClass' );
```

```
GetDlgCtrlID
GetDlgItem
GetDlgItemInt
GetDlgItemTextA
GetDlgItemTextW
GetDoubleClickTime
GetFocus
GetForegroundWindow
GetGuiResources
```

```
Python 2.5.1 (r251:54863, Dec 17 2007, 15:24:32) [MSC v.1400 32 bit (Intel)]
Platform: win32
Copyright: Nektra S.A. ©
Check out examples: spy_sample, spy_sample_v2, spy_sample_v3, httpReport.
>>>
```

Well well well – what do we see there. Our hash has been reversed by the program (reXOR'd really) into plain text.

What would happen if we were to enter that directly into our crackme …



-=bb=-

Cannabis

Great, Now write a tut and send to crackmes.de .: Registered to: -=bb=-

Success!

Whilst this is an immensely simple crackme, it is perfect for my needs at this point. I'm experimenting with different approaches. Of course, all of this could be done via IDA and a dead listing, such is the simplicity of the target. But what would you learn then?

Whilst I realise that this is of limited usefulness, imagine how we can build on it?

We have all come across worse protection schemes than this in the wild – but if this target was protected with a commercial protector to prevent debugging (assuming no code encryption, just anti-debug), this approach would still work as at no point do we attach a debugger to the program. No need to manually recode the routines to reverse the protection. No need to do it by hand on paper with the known data values that are used in the XORing.

For me this was a beginning step with Nektra. I think it shows potential and would be interested to see how anybody else has gone about reversing other targets using this software.

Greetz to : #GC, Inferno, W4rl0ck, ARTeam, relee, TheHiveMind, Phas and many more