Cracking With +Vip-Vop

Written by : +Vip-Vip **Published by : Splatter Industries,Inc.**

Getting registration numbers with Softice 3.2

**In this tutorial you will learn how to retrieve registration numbers using Softice.
The "crackme.exe" example file should be included in this package. If not, you can download the full lesson from www.splatter.net**

Ok this crackme has 5 parts to the registration scheme. Here they are:

1. Getting the Name and Reg Number from the user:

Enter your name and number (vipvop/111222). Put a bpx on getdlgitemtexta.
Press OK and you'll pop into s-ice, press F12 and you will be here:

```
:004012B5 6A0B                  push 0000000B
:004012B7 688E214000            push 0040218E ; this will be your name
:004012BC 68E8030000            push 000003E8
:004012C1 FF7508                push [ebp+08]

* Reference To: USER32.GetDlgItemTextA, Ord:0000h

:004012C4 E807020000            Call 004014D0
:004012C9 83F801                cmp eax, 00000001
:004012CC C74510EB030000        mov [ebp+10], 000003EB
:004012D3 72CC                  jb 004012A1
:004012D5 6A0B                  push 0000000B
:004012D7 687E214000            push 0040217E ; this will be your reg num
:004012DC 68E9030000            push 000003E9
:004012E1 FF7508                push [ebp+08]

* Reference To: USER32.GetDlgItemTextA, Ord:0000h

:004012E4 E8E7010000            Call 004014D0
```

You will be on line 004012C9 now. Look before the call to GetDlgItemTextA, we are trying to figure out which one of those numbers being pushed is the area for our name. Well its not likely to be B or 3E8 (too low in memory), so "d 0040218E" and you will see your name. Lets put a breakpoint on it, "bpm 0040218E RW" and now disable your getdlgitemtexta breakpoint, because we can see 0040217E will hold our reg number, but we don't need to put a bpm on it. Press ctrl-D to pop out of soft-ice and now we are at the next part.

2. Making the name uppercase:

After pressing ctrl-D, you will arrive in this code:

```
:00401383 8A06          mov al, byte ptr [esi] ; put letter into al
:00401385 84C0          test al, al ; if al is 0, we are done
:00401387 7413          je 0040139C ; jump to done
:00401389 3C41          cmp al, 41 ; 41 = A
:0040138B 721F          jb 004013AC ;if below jump to bad code
:0040138D 3C5A          cmp al, 5A ; 5A = Z
:0040138F 7303          jnb 00401394 ; jump to make it uppercase
:00401391 46            inc esi ; next letter of our name
:00401392 EBEF          jmp 00401383 ; go to start of loop
:00401394 E839000000    call 004013D2 ; make letter uppercase
:00401399 46            inc esi ; next letter of our name
:0040139A EBE7          jmp 00401383 ; go to start of loop
```

and heres the call we see:

```
:004013D2 2C20          sub al, 20
:004013D4 8806          mov byte ptr [esi], al
:004013D6 C3            ret
```

Ok an explanation of the code. When we start, [esi] points to the first letter of our name (do a "d esi" to see your whole name). A quick explanation of ASCII codes - 41hex = A, 5A hex = Z, 61 hex = a, 7A hex = z
Now the program checks the current letter of our name. If the ascii code is below 41, for characters like +, we automatically fail the reg check. Next if the current letters ascii code is higher than 5A, we jump to the make letter uppercase call. Then esi points to the next letter and we continue the jump, until the current letters is NULL, which is the end of a string. When al finally holds 0 we jump to code that does a pop esi, and now esi holds our name, but all the letters are uppercase (do a "d esi").

3. Make name code:

All this code does it take a letter from our name, make sure its not the NULL character, if it is we jump to the ret, if its not we add the ASCII code of our current letter to edi, then do the next letter of our name. In the end, edi holds the value of all the letters added together. IE:
if your name was ab, edi would hold 83 (A = 41, B = 42, 41 + 42 = 43). Now after this call there is a xor edi, 00005678, which means edi now equals the value of all the letters added together, XOR'ed by 5678. Now this value is moved into eax to be saved (we will call this our name hash), and we jump to the next part of the protection.

(See Page 3 for code)

After the pop esi we see this code:

```
:0040139D E820000000        call 004013C2
:004013A2 81F778560000      xor edi, 00005678
:004013A8 8BC7              mov eax, edi
:004013AA EB15              jmp 004013C1
```

the call contains this code:

```
:004013C2 33FF              xor edi, edi
:004013C4 33DB              xor ebx, ebx
:004013C6 8A1E              mov bl, byte ptr [esi]
:004013C8 84DB              test bl, bl
:004013CA 7405              je 004013D1
:004013CC 03FB              add edi, ebx
:004013CE 46                inc esi
:004013CF EBF5              jmp 004013C6
:004013D1 C3                ret
```

4. Making the Reg Number hash:

Pressing F10 a bit we will execute a ret, and then we see this code:

```
:00401232 50                push eax
:00401233 687E214000        push 0040217E
:00401238 E89B010000        call 004013D8
:0040123D 83C404            add esp, 00000004
:00401240 58                pop eax
:00401241 3BC3              cmp eax, ebx
:00401243 7407              je 0040124C
```

0040217E is our reg number, and it is passed to a function. Remember that eax holds our name hash, and we see after the call our name has is compared to ebx, which we can guess is probably a hash of our reg number.
Well lets look into the call 004013d8:

```
* Referenced by a CALL at Address:
:00401238

:004013D8 33C0              xor eax, eax
:004013DA 33FF              xor edi, edi
:004013DC 33DB              xor ebx, ebx
:004013DE 8B742404          mov esi, dword ptr [esp+04]
```

(Continued on Page 4)

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
:004013E8(C)

:004013F5 81F734120000        xor edi, 00001234
:004013FB 8BDF                mov ebx, edi
:004013FD C3                  ret
```

Ok the first part of this call is just zero'ing out the registers and making esi point to our reg number. The last part XORs edi by 1234,moves edi into ebx, and then returns. The middle part is the main part of our code.

```
:004013E2 B00A                mov al, 0A
:004013E4 8A1E                mov bl, byte ptr [esi]
:004013E6 84DB                test bl, bl ; if bl is NULL we end the call
:004013E8 740B                 je 004013F5
:004013EA 80EB30               sub bl, 30
:004013ED 0FAFF8              imul edi, eax
:004013F0 03FB                 add edi, ebx
:004013F2 46                   inc esi
:004013F3 EBED                jmp 004013E2
```

Ok heres whats happening. Al holds 0A. The current number from our reg number is moved into bl. If bl = NULL we end the call. We subtract 30 from bl. What this does is turn the ASCII code of our number to our actual number. IE. if we had entered 1 as our reg number, bl would hold 31. After subtracting 30 bl holds 1. Edi is then multiplied by eax (0A), then added to ebx. ESI is increased to point to the next number, and we continue our loop. Now I hope you like math heres a chart showing the value of edi as we execute the loop:

```
EDI=x
x * A
EDI=Ax + y
(Ax + y) * A
EDI=64x + Ay + z
(64x + Ay + z) * A
EDI=3E8x + 64y + Az + w
(3E8x + 64y + Az) * A
EDI=2710x + 3E8y + 64z + Aw + t
```

where x is our first number, y is the 2nd, z is the 3rd, w is the 4th, and t is the 5th. Look carefully over the code/chart until you understand whats happening. Now when this loops done, the following happens:

```
:004013F5 81F734120000        xor edi, 00001234
:004013FB 8BDF                mov ebx, edi
:004013FD C3                  ret
```

So then edi is XOR'ed by 1234, then the call ends. When we are back from the call ebx holds our reg number hash, eax holds our name hash, and if they are equal we are regged, else we are bad. Now as complicated as this seems, there is an easy way to get a reg number for yourself. Enter your reg info again, use the same breakpoints, and then once you get your name hash write that down and start up calculator. If can't find your name hash do a "? eax" on line 00401232. for the name splatter the hash is 5417. Remember we are using hex numbers here too. Ok so start the calculator that comes with windows, select scientific, then choose Hex. Enter your name hash (5417), then click XOR then type 1234. What we are doing is doing the registration scheme in reverse, and we know we have to make it equal 5417. Since XORing by 1234 is the last operation done we start with that. So 5417 XOR 1234 = 4633. So now we know that one middle loop of the reg number hash must make edi equal to 4623. How do we figure out what values makes that? Just use the chart we made earlier. Look at the last line.EDI=2710x + 3E8y + 64z + Aw + t.
So we need to divide and use the remainders, but let me explain. First dividew 4623 by 2710. We get 1 (no decimals in hex), which means our first number will be 1. Now 1 times 2710 is 2710, so subtract 2710 from 4623, and get 1F13. Now divide 1F13 by 3E8, we get 7. 7 is our second number, and 7 times 3E8 is 1B58. 1F13 - 1B58 = 3BB. 3BB divided by 64 is 9, so 9s our third number. 9 times 64 is 384, so 3BB - 384 is 37. 37 divided by A is 5, so thats our next number. 5 times A is 32, and 37 - 32 is 5, so thats our last number. So our number is 17955 and reg name is splatter. If this essay seems a bit confusing re-read it a couple of times, then try to reg a different name.

Cracking With +Vip-Vop
(Getting registration numbers with Softice 3.2)
Was written by : +Vip-Vop
And published by : Splatter Industries,Inc.

The End