raZZia's Tutorial on Key Generators

Tools!

For tools you need a minimum of debugger like SoftIce for Windows (hence WinIce), and a C compiler with Dos libraries.

Content!

In this tutorial I will show how to make a key-gen for Ize and Swiftsearch. The protection that these programs use is the well known Enter-Name-and-Registration-Number method. After selecting 'register', a window pops up where you can enter your name and your registration number. The strategy here is to find out where in memory the data you enter is stored and then to find out what is done with it. Before you go on make sure you configure the SoftIce dat file according to the PWD tutorial #1.

Part 1: Scanline Swiftsearch 2.0!

Swiftsearch is a useful little program that you can use to search on the web. I will explain step by step how to crack it.

step 1. Start the program :)

step 2: Choose register from the menus. You will now get a window where you can enter your name and your registration number.

step 3: Enter SoftIce (ctrl-d)

step 4: We will now set a breakpoint on functions like GetWindowText(a) and GetDlgItemText(a) to find out where in memory the data that we just entered is stored. The function that is used by this program is GetDlgItemTexta (trial and error, just try yourself :) so, in SoftIce type BPX GetDlgItemTexta and exit SoftIce with the g command.

step 5: Now type a name and a registration number (I used razzia and 12345) and press OK, this will put you back in SoftIce. Since you are now inside the GetDlgItemTexta function press F11 to get out of it. You should see the following code:

```
lea eax, [ebp-2C]          :<--- we are looking for this location
push eax
push 00000404
push [ebp+08]
call [USER32!GetDlgItemTextA]
mov edi, eax               :<--- eax has the length of the string
                            and is stored in edi for later usage.
```

We see that EAX is loaded with a memory address and then pushed to the stack as a parameter for the function GetDlgItemTextA. Since the function GetDlgItemTextA is already been run we can look at EBP-2c (with ED EDP-2c) and see that the name we entered is there. Now we know where the name is stored in memory, normally it would be wise to write that address down, but we will see that in this case it wont be necessary.

So, what next? Now we have to allow the program to read the registration number we entered. Just type g and return and when back in SoftIce press F11. You should see the following code:

```
push 0000000B
lea ecx, [ebp-18]          : <--So, ebp-18 is where the reg. number
push ecx                   :    is stored.
push 0000042A
push [ebp+08]
call [USER32!GetDlgItemTextA]
mov ebx, eax               : <--save the lenght of string in EBX
test edi, edi              : <--remember EDI had the lenght of the
jne 00402FBF               :    name we entered?
```

We see that the registration number is stored at location EBP-18 , check it with ED EBP-18. Again, normally it would be wise to note that address down. Also we see that it is checked if the length of the name we gave was not zero. If it is not zero the program will continue.

Step 6: Ok, now we know where the data we entered is stored in memory. What next?
Now we have to find out what is DONE with it. Usually it would we wise to put breakpoints on those memory locations and find out where in the program they are read. But in this case the answer is just a few F10's away. Press F10 until you see the following code :

```
cmp ebx, 0000000A       :<--remember EPX had the length of the
je 00402FDE             :   registration code we entered?
```

These two lines are important. They check if the length of the registration code we entered is equal to 10. If not the registration number will be considered wrong already. The program wont even bother to check it. Modify EBX or the FLAG register in the register window to allow the jump. Continue Pressing F10 until you get to the following code (note that the adresses you will see could be different) :

```
:00402FDE xor esi, esi          :<-- Clear ESI
:00402FE0 xor eax, eax          :<-- Clear EAX
:00402FE2 test edi, edi
:00402FE4 jle 00402FF2
:00402FE6 movsx byte ptr ecx, [ebp + eax - 2C] :<-- ECX is loaded with a letter
                                of the  name we entered.
:00402FEB add esi, ecx          :<-- Add the letter to ESI
:00402FED inc eax               :<-- Increment EAX to get next letter
:00402FEE cmp eax, edi          :<-- Did we reach the end of the string?
:00402FF0 jl 00402FE6           :<-- If not, go get the next letter.
```

Well, we see that the program adds together all the letters of the name we entered. Knowing that ESI contains the sum of the letters, lets continue and find out what the program does with that value :

```
:00402FF2 push 0000000A
:00402FF4 lea eax, [ebp-18]     :<-- Load EAX with the address of the reg. number we
                                entered
:00402FF7 push 00000000
:00402FF9 push eax              :<-- Push EAX (as a parameter for the following
                                function)
:00402FFA call 00403870         :<-- Well, what do you think this function does? :)
:00402FFF add esp, 0000000C
:00403002 cmp eax, esi          :<-- Hey!
:00403004 je 00403020
```

We see that a function is called and when RETurned ESI is compared with EAX. Hmm, lets look at what's in EAX. A '? EAX' reveals :

00003039 0000012345 "09"

Bingo. That's what we entered as the registration number. It should have been what's inside ESI. And we know what's inside ESI, the sum of the letters of the name we entered!

Step 7: Now we know how the program computes the registration code we can make a key-gen. But we should not forget that the program checks also that the registration number has 10 digits.
A simple C code that will compute the registration number for this program could look like this:

```
#include   <stdio.h>
#include   <string.h>
main()
{
     char Name[100];
     int NameLength,Offset;
     long int Reg = 0, Dummy2 = 10;
     int Dummy = 0;
     int LengtDummy = 1;
     int Lengt , Teller;
     printf("Scanline SwiftSearch 2.0 crack by raZZia.\n");
     printf("Enter your name: ");
      gets(Name);
      NameLength=strlen(Name);

/* the for lus calculates the sum of the letters in Name */
/* and places that value in Reg                          */
      for (Offset=0;Offset<NameLength;Offset=Offset+1)
        {
           Reg=Reg+Name[Offset];
        }
/* the while lus calculates the lenght of the figure in */
/* Reg and places it in Lengt                           */
      while (Dummy != 1)
        {
             if ( Reg < Dummy2 )
               { Lengt = LengtDummy ; Dummy =1;
               }
            else
               { LengtDummy=LengtDummy + 1; Dummy2=Dummy2*10;
               }
        };
     printf("\nYour registration number is : " );
/* First print 10-Lengt times a 0                        */
     Lengt=10-Lengt;
     for (Teller=1;Teller<=Lengt;Teller=Teller+1) printf("0");
/* Then print the registration number                    */
     printf("%lu\n",Reg);
}
```

Case 2 Ize 2.04 from Gadgetware

Ize from Gadgetware is a cute little program that will put a pair of eyes on your screen which will follow your mousepointer. It has a register function where you can enter your name and a registration number. The strategy in this case is still the same : Find out where in memory the entered information is stored and then find out what is done with that information.

Step 1: Start Ize. Chose register and enter a name and a number. I used 'razzia' and '12345'.

Sterp 2: Enter (CTRL-D) Softice and set a breakpoint on GetDlgItemTextA.

Step 3: Leave SoftIce and press OK. This will put you back in Softice. You will be inside the GetDlgItemTextA function. To get out of it press F11. You should see the following code :

```
mov esi, [esp + 0C]
push 00000064
push 0040C3A0          :<--On this memory location the NAME we entered will be stored.
mov edi, [USER32!GetDlgItemTextA]   :<--Load edi with adress of GetDlgItemTextA
push 00004EE9
push esi
call edi               :<-- Call GetDlgItemTextA
push 00000064          :<-- (you should be here now)
push 0040C210          :<--On this memory location the NUMBER we entered will be stored
push 00004EEA
push esi
call edi               :<-- Call GetDlgItemTextA
```

We see that the function GetDlgItemTextA is called twice in this code fragment. The first call has already happened. With ED 40C3A0 we can check that the name we entered is stored on that location.
To allow the program to read in the number we entered we type G and enter. Now we are inside the Get-DlgItemTextA function again and we press f11 to get out of it. We check memory location 40C210 and we see the number we entered is stored there.
Now we know the locations were the name and the number are stored,we note those down!

Step 4: Ok, what next? We now know where in memory the name and the number are stored. We need to find out what the program does with those values. In order to do that we could set breakpoints on those memory locations to see where they are read. But in this case it wont be necessary. The answer is right after the above code :

```
push 0040C210  :<--save the location of the number we entered (as a parameter for the next call)
call 00404490   :<-- call this unknown function
add esp, 00000004
mov edi, eax  :<-- save EAX  (hmmmm)
```

We see a function being called with the number-location as a parameter. We could trace into the function and see what it does, but that is not needed. With your experience of the Swiftsearch example you should be able to guess what this function does. It calculates the numerical value of the registration number and puts it in EAX. To be sure we step further using F10 untill we are past the call and check the contents of EAX (with ? EAX). In my case it showed : 00003039 0000012345 "09".

Knowing that EDI contains our registration number we proceed:

```
        push 0040C3A0  :<-- save the location of the name we entered (as a parameter for the next call)
        push 00409080  :<-- save an unknown memory-location (as a parameter for the next call)
        call 004043B0 :<--call to an unknown function
        add esp, 00000008
        cmp edi, eax   :<--compare EDI (reg # we entered) with EAX (unknown, since the previous
call                                                              changed it)
        jne 004018A1   :<--jump if not equal
```

We see that a function is called with two parameters. One of the parameters is the location of the name
we entered. The other we dont know, but we can find out with ED 409080. We see the text 'Ize'. This function calculates the right registration number using those two parameters. If you just want to crack this program, you can place a breakpoint right after the call and check the contents of EAX. It will
contain the right registration number. But since we want to know HOW the reg. # is calculated we will trace inside the function (using T). We will then try to find out HOW the contents of EAX got in there.

Step 5: Once inside the interesting function you will see that we are dealing with a rather long function. It wont be necessary for me to include the complete listing of this function, because we wont need all of it to make our key-gen.
But in order find out which part of the code is essential for the computation of the right registration number, you have to trace STEP by STEP and figure out what EXACTLY is going on!

Afther doing this i found out that the first part of the function computes some kind of "key". Then this
"key" is stored in memory and in that way passed on to the second part of the function.
The second part of the function then computes the right registration number, based on this "key" AND
the name we entered.

The code that is essential and that we need for our key-gen is the following:

(Note that before the following code starts, the registers that are used will have the following values:
EBX will point to the first letter of the name we entered,
EDX will be zero,
EBP will be zero,
The "key" that we talked about earlier is stored in memory location 0040B828 and will have 0xA4CC as its initial value.)

```
:00404425 movsx byte ptr edi, [ebx + edx]     :<-- Put first letter of the name in EDI
:00404429 lea esi, [edx+01]      :<-- ESI gets the "letter-number"
:0040442C call 00404470          :<-- Call  function
:00404431 imul edi, eax          :<-- EDI=EDI*EAX (eax is the return value of the the previous
call)
:00404434 call 00404470          :<-- Call function
:00404439 mov edx, esi
:0040443B mov ecx, FFFFFFFF
```

```
:00404440 imul edi, eax       :<-- EDI=EDI*EAX (eax is the return value of the previous call)
:00404443 imul edi, esi       :<-- EDI=EDI*ESI ( esi is the number of the letter position)
:00404446 add ebp, edi        :<-- EBP=EBP+EDI  (beware that EBP will finally contain the right reg#)
:00404448 mov edi, ebx    :<--these lines compute the lenght of the name we entered
:0040444A sub eax, eax    :<--these lines compute the lenght of the name we entered
:0040444C repnz           :<--these lines compute the lenght of the name we entered
:0040444D scasb           :<--these lines compute the lenght of the name we entered
:0040444E not ecx         :<--these lines compute the lenght of the name we entered
:00404450 dec ecx         :<-- ECX now contains the lenght of the name
:00404451 cmp ecx, esi
:00404453 ja 00404425     :<-- If its not the end of the name , go do the same with the next letter
:00404455 mov eax, ebp    :<--  SAVE EBP TO EAX !!!!
:00404457 pop ebp
:00404458 pop edi
:00404459 pop esi
:0040445A pop ebx
:0040445B ret
```

```
:00404470 mov eax, [0040B828]         :<-- Put "key" in EAX
:00404475 mul eax, eax, 015A4E35      :<-- EAX=EAX * 15A4E35
:0040447B inc eax                     :<-- EAX=EAX + 1
:0040447C mov [0040B828], eax         :<-- Replace the "key" with the new value of EAX
:00404481 and eax, 7FFF0000           :<-- EAX=EAX && 7FFF0000
:00404486 shr eax, 10                 :<-- EAX=EAX >>10
:00404489 ret
```

The above code consists of a loop that goes trough all the letters of the name we entered. With each letter some value is calculated, all these values are added up together (in EBP). Then this value is stored in EAX and the function RETurns. And that was what we were looking for, we wanted to know how EAX got its value!

Step 6: Now to make a key-gen we have to translate the above method of calculating the right reg# into a c program. It could be done in the following way :
(Note : I am a bad c programmer :)

```
#include    <stdio.h>
#include    <string.h>
main()
{
      char Name[100];
      int NameLength,Offset;
      unsigned long Letter,DummyA;
      unsigned long Key = 0xa4cc;
      unsigned long Number = 0;
        printf("Ize 2.04 crack by razzia\n");
      printf("Enter your name: ");
        gets(Name);
        NameLength=strlen(Name);
        for (Offset=0;Offset<NameLength;Offset=Offset+1)
        {
               Letter=Name[Offset];
```

```
                 DummyA=Key;
                 DummyA=DummyA*0x15a4e35;
                   DummyA=DummyA+1;
                   Key=DummyA;
                   DummyA=DummyA & 0x7fff0000;
                   DummyA=DummyA >> 0x10;
                 Letter=Letter*DummyA;
                 DummyA=Key;
                 DummyA=DummyA*0x15a4e35;
                   DummyA=DummyA+1;
                   Key=DummyA;
                   DummyA=DummyA & 0x7fff0000;
                   DummyA=DummyA >> 0x10;
                 Letter=Letter*DummyA;
                 Letter=Letter*(Offset+1);
                 Number=Number+Letter;
        }
        printf("\nYour registration number is : %lu\n",Number);
}
```

Final Notes

For feedback and suggestions pls contact me :)

raZZia