Ripping VB code and making keygen out of it

deroko/ARTeam

December 2005

Keywords

VB, keygen, ripping, asm

1. Intro

Maybe idea is not new, but I haven't seen any reference about this technique, so I'm gona explain this on one simple crackme from http://www.crackmes.de because it has simple key check routine and also has everything that we need to consider when writing such keygens. I was thinking to focus on one commercial app (something about sending anonymous mails) but whole keycheck routine didn't have any import from vb, so this crackme seems like a good target for this article.

Tools that we are gona use here are :

- IDA
- p32dasm v2.1
- tasm32
- OllyDbg
- crackme is supplied with this pdf.

*** I prefer tasm32 but any asm compiler will work.

2. Target overview

Well there is not much to talk about this target, it is simple VB crackme that uses computer name xored with string "Serializer" to get "hardware" key, and after that it will use hardware key xored with "Serializer" to get final key.

Load your target into p32dasm:

File: crackme.exe
P32Dasm v2.1

VB6 Application detected ... NCode

frm1
004028F4 1.1 cmdExit.Click()
004029C6 1.2 cmdRegister.Click()
00402BD3 1.3 Command1.Click()
00402D3D 1.4 Command2.Click()
0040315C 1.5 Form.Load()
00403332 1.6 txtReg.KeyPress(KeyAscii As Integer)

File processed OK.

Locate cmdRegister in Olly using address from p32dasm, click on register and trace a little bit till you get here:

```
00402AB2   . 68 24404000      PUSH crackme.00404024
00402AB7   . 50               PUSH EAX
00402AB8   . E8 00090000      CALL crackme.004033BD
00402ABD   . 8BD0             MOV EDX,EAX
00402ABF   . 8D4D DC          LEA ECX,DWORD PTR [EBP-24]
00402AC2   . E8 99E7FFFF      CALL <JMP.&msvbvm60.__vbaStrMove>
00402AC7   . 50               PUSH EAX
00402AC8   . E8 99E7FFFF      CALL <JMP.&msvbvm60.__vbaStrCmp>
```

Procedure marked with red is procedure responsible for keygening, so that's the procedure that we are going to rip from IDA and inline it into our asm keygen.

If you don't wanna read this pdf anymore you may fish your key at **00402AC8** and you have solved level 1 crackme :D

3. Ripping Procedure

Fire up IDA, and produce ASM file out of it (File -> Produce File -> Create Asm File…) and locate procedure that starts from :

```
seg000:004033BD          push    ebp
seg000:004033BE          mov     ebp, esp
seg000:004033C0          sub     esp, 0Ch
```

and ends here:

```
seg000:00403611          pop     ebx
seg000:00403612          leave
seg000:00403613          retn    8
```

This whole procedure should be copy/pasted to your keygen, but also we have to take care about 2 static variables:

```
seg000:004033DF          mov     dword ptr [ebp-8], offset dword_401160

seg000:00401158 dword_401158    dd 80001h
```

and __vbaStrCat :

```
seg000:00403572          push    offset dword_4027BC
seg000:00403577          push    dword ptr [ebp-28h]
seg000:0040357A          call    __vbaStrCat

seg000:0040264C                  dd 2
seg000:004027BC dword_4027BC     dd 30
```

well this is a little bit wrong disassembly because dword_4027BC is actually string “0” and 2 is nothing more then len of this string (remember VB is using unicode) so basically this should be:

```
                 dd          2       <----- len of string
string_0         dd          30h     <----- string
```

Ok, when we have inlined all of this in our asm file (check keygen\keygen.asm) we may ask ourselves, how to call this simple proc?

Let’s go again to our debugger and examine arguments passed to this proc at Form.Load (to get “hardware” key):

```
00403286   . 57              PUSH EDI         <----- static string
00403287   . 50              PUSH EAX         <----- computer name
00403288   . E8 30010000    CALL crackme.004033BD
```

Examine content of edi:

```
00404024  64 EF 15 00 00 00  00  00 70  D0  15  00   00  00  00   00    dï  .....pÐ  .....
          ^^^^^^^^^^^
0015EF64  53 00 65 00 72 00  69  00  61  00  6C  00   69  00  7A   00   S.e.r.i.a.l.i.z.
0015EF74  65 00 72 00 00 00 AD BA 0D F0 AD BA AB AB AB AB  e.r... º.ð º«««««
```

As you may see edi is nothing more then pointer to string pointer (C terminology) but there is also a little catch with VB strings, because they have their length at [string_pointer-4], and we have to take care of that when writing keygen for this crackme. Also I've shown that __vbaStrCmp is using dword_4027BC which is actually string "0".

Let's check our theory and go to 15EF60 :

```
0015EF54  00 00 00 00 07 00 58 00 7F 07 18 00 14 00 00 00  ....  .X.        .   ...
                                                ^^^^^^^^^^^
                                                0015EF60h
```

Yup that's size of unicode string "Serializer" without 2 terminating zeroes.

Content of EAX is similar, pointer to string pointer of my computer name, "SCORPION". including len of string at [string_pointer-4].

If we wanna call this proc we have to get length of strings and make pointers to string_pointers:

NOTE: some macros that I use here (unis to make Unicode string) are included in includez\shitheap.inc

```
<++> keygen\keygen.asm <++>
.data

                dd      14h
static_string:  unis    <Serializer>
                dd      10h
computer_name:  unis    <SCORPION>

ptr1            dd      ?
ptr2            dd      ?

.code
start:
                mov     eax, offset static_string
                mov     ptr1, eax
                mov     eax, offset computer_name
                mov     ptr2, eax

                push    offset ptr1
                push    offset ptr2
                call    VB_KEYGEN_PROC

            ...
VB_KEYGEN_PROC:
            proc ripped from IDA

            end     start
<++> keygen.asm <++>
```

Compile this code, and run it trough olly so you can catch all exceptions instead causing infinite SEH loops.

We should get Exception (Access Violation) because our code is trying to read from wrong memory address. We have to locate this exception, also we have to fix code a little bit.

4. Going deeper underground

After a little bit of tracing, we have found problem:

```
004010C6   .  50                PUSH EAX
004010C7   .  E8 DD010000   CALL <JMP.&MSVBVM60.__vbaStrCat>
```

Step into __vbaStrCat:

```
660E5F3A >  55                  PUSH EBP
660E5F3B    8BEC                MOV EBP,ESP
660E5F3D    8D45 08             LEA EAX,DWORD PTR [EBP+8]
660E5F40    50                  PUSH EAX
660E5F41    FF75 08             PUSH DWORD PTR [EBP+8]
660E5F44    FF75 0C             PUSH DWORD PTR [EBP+C]
660E5F47    FF15 18EE1066   CALL DWORD PTR [6610EE18]
                                                ^^^^^^^^
```

6610EE18 is zero, and that's problem for us, there is our exception. But if we take a look at crackme and check that value we might see:

```
660E5F3A >  55                  PUSH EBP
660E5F3B    8BEC                MOV EBP,ESP
660E5F3D    8D45 08             LEA EAX,DWORD PTR [EBP+8]
660E5F40    50                  PUSH EAX
660E5F41    FF75 08             PUSH DWORD PTR [EBP+8]
660E5F44    FF75 0C             PUSH DWORD PTR [EBP+C]
660E5F47    FF15 18EE1066   CALL DWORD PTR [6610EE18] ;OLEAUT32.VarBstrCat
```

So somehow, ptr 6610EE18 is initialized in crackme, but not in keygen.exe, so we come to a conclusion that variables used by our key are not initialized with loading msvbvm60.dll.

We have to make workaround and initialize that local variable, only solution for us is IDA and disassembly of msvbvm60.dll:

```
.text:660E5F3A __vbaStrCat:
.text:660E5F3A              push    ebp
.text:660E5F3B              mov     ebp, esp
.text:660E5F3D              lea     eax, [ebp+8]
.text:660E5F40              push    eax
```

```
.text:660E5F41          push    dword ptr [ebp+8]
.text:660E5F44          push    dword ptr [ebp+0Ch]
.text:660E5F47          call    dword_6610EE18
```

Follow references to dword_6610EE18 :

```
.data:6610EE18 dword_6610EE18  dd 0    ; DATA XREF: .text:660053C4
.data:6610EE18                         ; .text:660E5F47
```

Follow .text:660053C4 and we end up here:

```
.text:660053BA          push    offset aVarbstrcat ; "VarBstrCat"
.text:660053BF          push    edi
.text:660053C0          call    esi ; GetProcAddress
.text:660053C2          test    eax, eax
.text:660053C4          mov     dword_6610EE18, eax
```

Nice, we have found good place, there is GetProcAddress, also if you check this part of code in IDA you will see a lots of GetProcAddress and many more variables initialized with exported procs from oleaut32.dll. So we have to trace references to this place till we end up at some exported proc which will for sure call this part of code:

Keep following references:

```
.text:66004F58 sub_66004F58    proc near ; CODE XREF: sub_66004DAD+3A   p
.text:66004F58                 push    esi
.text:66004F59                 push    edi
.text:66004F5A                 push    offset aOleaut32_dll_0 ; "oleaut32.dll"
```

This is entry of proc that calls all those GetProcAddresses to initialize global variables in msvbvm60.dll.

```
.text:66004DE2          call    sub_66004F1D
.text:66004DE7          call    sub_66004F58
.text:66004DEC          push    offset aOle32_dll ; "ole32.dll
```

Keep going:

```
.text:66004D81 sub_66004D81    proc near  ; CODE XREF: sub_66004D59+20   p
.text:66004D81                            ; CreateIExprSrvObj+3E   p
.text:66004D81
…
.text:66004DA4                 call    sub_66004DAD
.text:66004DA9                 pop     ebp
.text:66004DAA                 retn    24h
.text:66004DAA sub_66004D81    endp
```

Huh, finally we see some exported “api” from msvbvm60.dll which will eventually take us to the good place: