http://www.CodeBreakers-Journal.com

Explaining Visual Basic

B. Kathras

Abstract

I am writing this essay in order to help my peers better understand how different code structures in Visual Basic are almost completely uniform when it is compiled to its native code. I will demonstrate how to reverse these structures. There is no specific target for this essay, all examples are provided from binaries compiled by yours truly.

Keywords: *Reverse Code Engineering; Visual Basic; Native Code*

I. Part 1 - The Dreaded Nag Screen

There are several ways a nag screen may be initialized in Visual Basic. One of the most common you will see is the Form.Show event. If you have read my previous essay about reversing VB nag screens, this is the event I was referring to. An example of the compiled translation of this event looks similar to this:

```
:0040203D FF92B0020000            call dword ptr [edx+000002B0]
```

A good string to do a text search for in this case would be ”+000002B0]”

The register in the above call can vary, but is almost always edx, ecx, or eax..

One can stop this event simply by removing this call. Sometimes it is necessary to fix the stack when removing this call because the value in esp is always 24h higher after the call is stepped over. An example of this modification may look like this:

```
:0040203D 83C424                  add esp, 00000024
:00402040 EB01                    jmp 00402043
:00402042 90                      nop
```

If a nag screen was created by initializing this even it is rather easily reversed by using the above piece of code.

Special note: In my previous essay I stated that one could easily find the above location by breaking on the Visual Basic API `__vbanew2`. I didn’t say why it sometimes worked or didn’t work. At that time I did not understand the function of that API. It is only called if the form is declared as a new object . In several references it states that this api is similar to the user32 API DialogBoxParamA. This is not true. It is in fact, as I just stated, a declaration. An example would look like this:

```
Dim NagScreen as New NagForm
NagScreen.Show
```

Another way that nag screens are initialized is by loading the nag screen’s form when the application loads. At load the nag has its boolean visible property set to false. During execution the property is set to true someplace in the code. In source form the property change looks something like this:

```
NagForm.Visible = True
```

An example of what this change in property looks like in compiled form:

```
:00401FF9 6AFF          push FFFFFFFF
                        <-This variable is how vb refers to
                          the boolean value of True
:00401FFB 50            push eax
:00401FFC 898568FFFFFF  mov dword ptr [ebp+FFFFFF68], eax
:00402002 8B10          mov edx, dword ptr [eax]
:00402004 FF92BC010000  call dword ptr [edx+000001BC]
                        <-This is the call that initializes
                          the property change
```

good string to do a text search for in this case would be ”+000001BC]”

There are a couple of ways you can approach reversing this. The easiest and some would say proper way would be to simply change the value of the boolean variable to False. Visual Basic represents a value of false by 00. An example of this approach would look like this:

```
:00401FF9 6A00                  push 00
:00401FFB 50                    push eax
:00401FFC 898568FFFFFF          mov dword ptr [ebp+FFFFFF68], eax
:00402002 8B10                  mov edx, dword ptr [eax]
:00402004 FF92BC010000          call dword ptr [edx+000001BC]
```

Another approach would be to simply remove the initialization call in the same way that the form.show event was reversed (see above)

II. Part 2 - Changing Object Properties

I will first start off by stating that there are way too many property changes to be explained in this essay. I have listed below some of the more common ones that are relevant for us reversers.

The vast majority of property changes can be located by setting a breakpoint in `__vbaSetObj`. Trying to break on this may be impractical in some circumstances if the program (as many do nowadays) have massive GUI's. Below I will list some common examples of various structures you can search for in order to reach your target destination.

The first object we will look at is the command button. Nothing is more annoying than to be testing an application and to discover that a particular button is missing or disabled. We will begin by reviewing how a command button is enabled or disabled. In source code an Enabled property change would look something like this:

```
Command1.Enabled = False
```

In compiled form it will look similar to this:

```
:00402018 57            push edi <--- value of edi = 00
                                      which in VB = False
:00402019 56            push esi
:0040201A 8B06          mov eax, dword ptr [esi]
:0040201C FF908C000000  call dword ptr [eax+0000008C] <--- Initialization call
```

A good string to do a text search for in this case would be "+0000008C]"

As before there are a couple of ways to reverse this. The "proper" way of dealing with this would be to change the flag being pushed to the stack at 402018 to -1 or FF which in Visual Basic means true. This is not always possible to do because of limited space for code insertion. A simpler method of just removing the initialization call would look something like this:

```
:0040201C E901000000            jmp 00402022
:00402021 90                    nop
```

Please note that the initialization call above is not exclusively used for command buttons. It is also used for enabling/disabling text box's as well.

Next we will look at making a command button visible/invisible. An example of making a button invisible at runtime would look like this:

```
Command1.Visible = False
```

In compiled form it will look something like this:

```
:00402018 57                    push edi <--- value of edi = 00 which in VB = False
:00402019 56                    push esi
:0040201A 8B06                  mov eax, dword ptr [esi]
:0040201C FF9094000000          call dword ptr [eax+00000094] <--- Initialization call
```

A good string to do a text search for in this case would be ””+00000094]”

Like before and with most boolean properties, the ”proper” way to reverse this would be to change the flag being pushed at 402018 to -1 or FF. And as with before you can also remove the initialization call with the same technique as above. Also note that this initialization call is not exclusive to command buttons as well. It is also used when enabling/disabling a label.

Next we will look at making labels visible/invisible. This would be useful in removing certain ”unregistered” labels and whatnot. An example of what this would look like in source code would be similar to this:

```
Label1.Visible = False
```

When compiled you will get something similar to this:

```
:00402018 57            push edi <--- value of edi = 00 which in VB = False
:00402019 56            push esi
:0040201A 8B06          mov eax, dword ptr [esi]
:0040201C FF909C000000  call dword ptr [eax+0000009C] <--- Initialization call
```

A good string to do a text search for in this case would be ”+0000009C]”

I understand we are beginning to sound like a broken record here. But as above one ”should” change the value being pushed to the stack at 402018 to -1 or FF which equals True in Visual Basic. Or once again you could remove the initialization call with a similar technique as above.

Our next trip down visual basic lane will involve enabling/disabling a menu item. The usefulness of this is rather obvious if you have any experience with crippleware or if you can’t for some reason get that ”Register Me” menu item to become enabled. An example in source code looks something like this:

```
mnuRegister.Enabled = False
```

In compiled form it will look something similar to:

```
:004020A8 57        push edi  <--- value of edi = 00 which in VB = False
:004020A9 56        push esi
:004020AA 8B06      mov eax, dword ptr [esi]
:004020AC FF5074    call [eax+74] <--- Initialization call
```

A good string to do a text search for in this case would be ”+74]”

Hmmm, this is getting kinda fishy... Everything looks the same as above... I will leave this one to you to decide how to reverse it.

Our next structure that we will review will be changing the text in a text box. An example for what this property change may look like in source:

```
Text1.Text = "Insert blah blah here"
```

In compiled for it will look something like this:

* Possible StringData Ref from Code Obj -¿"Insert blah blah here"

```
:00401C35 6890174000    push 00401790 <--- Push the address of the
                                           above string to the stack
:00401C3A 56            push esi
:00401C3B 8B06          mov eax, dword ptr [esi]
:00401C3D FF90A4000000  call dword ptr [eax+000000A4] <--- Initialization call
```

Depending upon what you are trying to do, you could do a couple of things here, one change the address of the string being pushed to the stack to point to another string. Another thing you could do would be to simply hex edit the string to what you want it to say. Lastly if you don't want this property change to take place, you can remove the initialization call as with the examples shown previously.

A good string to do a text search for in this case would be "+000000A4]"

Finally we will look at changing the Caption property for any object which has one. An example of how this knowledge may be useful is if a program has an annoying "Not Registered" in the title bar of a form. Or if a label in an about box has another annoying "Not Registered" message. In source code we will see something similar to this:

```
mainForm.Caption = "Blah Blah v1.0 - Unregistered"
```

When you compile this source you will get something similar to this:

```
* Possible StringData Ref from Code Obj ->"Blah Blah v1.0 - Unregistered"
                                        |
:004020CC 68FC194000                    push 004019FC <--- Location of the above string
:004020D1 50                            push eax
:004020D2 898564FFFFFF                  mov dword ptr [ebp+FFFFFF64], eax
:004020D8 8B10                          mov edx, dword ptr [eax]
:004020DA FF5254                        call [edx+54] <--- Initialization Call
```

A good string to do a text search for in this case would be "+54]"

Please note that you can't simply always rely on doing a text search for "Blah Blah v1.0 - Unregistered" because that string could rather easily be built dynamically during runtime. A more reliable method for searching for a Caption property change would be to search for a call [someregister+54]. The simplest technique for removing this is removing the initialization call or changing the string being pushed to the stack at 4020CC.

III. Part 3 - Resource Editing

As I could go on and on and on with this particular subject. I am going to limit this part of the essay to describing the properties of one object, in this case a command button. The properties for all other objects will be laid out in a similar fashion so there is no need to, as above, spend several pages describing almost the same thing over and over again.

Please note that the number of bytes describing the object's properties may vary a bit depending upon if some default properties are modified from the get go. Also note the bytes highlighted that specify the number of bytes used to describe the object's properties.

```
.004011D0:  00 0D 01 00-00 3E 00 00-00 00 05 00-46 6F 72 6D   --  >    - Form
.004011E0:  31 00 0D 01-05 00 46 6F-72 6D 31 00-19 01 00 42  1 --- Form1 -- B
.004011F0:  00 23 FF FF-FF FF 24 05-00 46 6F 72-6D 31 00 35   #    $- Form1 5
.00401200:  3C 00 00 00-59 01 00 00-48 12 00 00-7B 0C 00 00  <   Y-  H-  {-
.00401210:  46 03 FF 01-28 00 00 00-05 07 00 4F-70 74 69 6F  F- -(   -- Optio
.00401220:  6E 31 00 06-01 07 00 4F-70 74 69 6F-6E 31 00 05  n1 --- Option1 -
.00401230:  D0 02 08 07-57 03 C3 00-12 04 00 FF-03 26 00 00  ----W-- --  -&
.00401240:  00 04 06 00-43 68 65 63-6B 31 00 05-01 06 00 43   -- Check1 --- C
.00401250:  68 65 63 6B-31 00 05 D0-02 28 05 BF-04 FF 00 12  heck1 ---(---  -
.00401260:  03 00 FF 03-24 00 00 00-02 05 00 54-65 78 74 31  -  -$   -- Text1
.00401270:  00 02 04 38-04 D0 02 F7-08 77 01 0B-05 00 54 65   --8-----w--- Te
.00401280:  78 74 31 00-12 01 00 FF-03 2C 00 00-00 01 08 00  xt1 --  -,   --
.00401290:  43 6F 6D 6D-61 6E 64 31-00 04 01 08-00 43 6F 6D  Command1 --- Com
.004012A0:  6D 61 6E 64-31 00 04 E0-01 60 09 BF-04 77 01 08  mand1 ---'---w--
.004012B0:  00 11 00 00-FF 03 26 00-00 00 03 06-00 4C 61 62   -   -&   -- Lab
.004012C0:  65 6C 31 00-01 01 06 00-4C 61 62 65-6C 31 00 05  el1 --- Label1 -
.004012D0:  B0 04 F0 00-27 06 77 01-12 02 00 FF-02 04 00 00  --- '-w---  --
.004012E0:  50 00 00 00-AB 51 84 9E-E1 4E 9C 4A-90 DB 82 61  P   Q-NJ-a
.004012F0:  67 D3 DD F9-00 00 00 00-00 00 00 00-00 00 00 00  g---

Number of bytes used for object's properties
Name of object = Command1
Designates the type of object (see table below)
Object.Caption = Command1
Object.Left = 480
Object.Top = 2400
Object.Width = 1215
Object.Height = 375
Object.Enabled = False
```

Below is a table listing for the bytes that designate the type of the most common objects used in VB.

```
00 Picturebox
01 Label
02 TextBox
03 Frame
04 Command Button
05 CheckBox
06 Option Button
07 ComboBox
08 ListBox
0B Timer
0D Form
```

As an example upon how this might be useful for reversers, On the command Button, I have left the Enabled property as false to simulate a disabled Command button. The value highlighted in purple is 00 which is the VB equivalent of False. In order to enable this Command button one only needs to change the property to True, in VB, True equals FF or -1.

IV. Conclusion

Ok, if you have learned anything at all from this essay, I would certainly hope you at least learned that how uniform natively compiled VB code really is. As I said before its like a broken record. Also please note that the above examples aren't 100% universal among all VB 5 or 6 PE's variables of course are variables and the registers used are generally different as well. I very much hope that I have somewhat contributed to others' knowledge in reversing Visual Basic targets.

V. Greets (in no particular order)

Devine9, Crackz, disavowed, Maud 'Dib, Mankind, barcode_, Ordoc, dumbslut, death (where you been man), josephCo, tami, g0dMoNeY, MaTHieU, aerosmith, and to all my friends new and old

If anyone notices any discrepancies to the above listed structures or would have any comments they would like to make regarding this subject please email me at "k a t h r a s (at) m o o c o w (dot) c o m" you can figure out how to make that email work.