VB6 Variable Initialization and Tear Down

Site: <u>http://sandsprite.com</u>

David Zimmer <<u>dzzie@yahoo.com</u>>

Introduction:

In this article we will be looking at how VB6 initializes and cleans up module level variables. This query into the inner operations of the Runtime opens the door to some interesting new capabilities for binary analysis.

This includes:

- enumerating all live VB6 classes
- discovery of private variables and types
- extracting data from live class instances
- gaining arbitrary script access to any live class

Background:

Every VB6 code module, regardless of type, can hold its own module level variables.

- If a class goes out of scope and is being destroyed, how does the VB Runtime assure that all of its resources are automatically cleaned up?
- If a module contains an array of predefined size, how does the Runtime allocate the memory so its ready for first time use?

For instanced code modules, such as Forms and Classes, module level variables are stored on a per instance basis. The value returned by ObjPtr() actually returns a data block that holds a multitude of settings such as reference count, VTable pointer, local variables etc.

For BAS modules, where there is no instancing, the variables are held at static offsets starting at Object.aModulePublic.

If we want to explore how resource cleanup occurs, we can hold an instance of one of our classes in a BAS level variable and place a MsgBox in the class terminate code. This gives us a convenient place to attach a debugger and examine the call stack.

Consider the following code:

```
'Module 1
Dim a As Class1, b As Object, c As Object
Sub Main()
   Set a = New Class1
   Set b = New Class1
   Set c = New Class1
   a.name = "dog"
   b.name = "cat"
   c.name = "bird"
   InputBox "", , Hex(ObjPtr(a)) & " " & Hex(ObjPtr(b)) & " " & Hex(ObjPtr(c))
End Sub
'Class 1 code
Public name As String
Private Sub Class_Terminate()
   MsgBox "terminating " & name
End Sub
```

Attaching a debugger at the class terminate MsgBox gives us the following call stack:

Stack	Procedure / arguments
66028116	MSVBVM60.RUN_INSTMGR::ExecuteInitTerm
66027F83	MSVBVM60.RcmResetModulesPrepass
66027F04	MSVBVM60.RcmShutDownProject
66027B3C	MSVBVM60.RcmResetProject
66051B59	MSVBVM60.66027AC0
66027A39	MSVBVM60.EbResetProjectNormal
66027D28	MSVBVM60.CThreadPool::ResetProject
66051AEC	MSVBVM60.DbgReset
6600B1B5	MSVBVM60.DbgResetIfDoneRunning
	66028116 66027F83 66027F04 66027B3C 66051B59 66027A39 66027D28 66051AEC

As we examine these routines we discover that VB6 keeps a linked list of all class instances. The run time data we received in form load showed the following ObjPtrs for the 3 class instances:

```
objptr 1 = 0x565280
objptr 2 = 0x565C88
objptr 3 = 0x565CE8
```

As we step through the code in RcmResetModulesPrepass, we find it referencing the following VB6 structure:

```
Class1.ObjInfo.lpProjectData = 402018
```

As we analyze the code, we find that this offset is a pointer to a RUN INSTMGR structure

00402018 005650C8 *RUN_INSTMGR 005650C8 00565CE8 <-- objptr 3 005650CC 0000000 005650D0 004011E8 Class1.objinfo 005650D4 0000003 <-- live instances count

Following the ObjPtr() to the next live instance, we see a linked list emerge at ObjPtr() +4 to the next instance

```
00565CE8 00402348 Project1.00402348 class vtable

00565CEC 00565C88 <-- objptr 2

00565C88 00402348 Project1.00402348 class vtable

00565C8C 00565280 <-- objptr 1

00565280 00402348 Project1.00402348 class vtable

00565284 00000000 <-- no next instance
```

This is an interesting bit of trivia, but is it really useful? Well its more interesting than you might think!

Implications:

At this point, we can now scan a running a process and enumerate all of its live class instances. What would make this really interesting is if we knew what local variables were held at the class level, their offset, and type. This would allow us to remotely view every classes configuration state dumping live data without even attaching a debugger. Is that possible?

Yuuuup.

If you have been following along with my VB internals series there is an article titled <u>Recovery of function prototypes in Visual Basic 6 executables</u>. In this paper we detail how to parse the IDispatch type information to get public variable names, types, and offsets. When you combine these two techniques, you now get a pretty interesting information dumper that can be implemented using only ReadProcessMemory.

ObjPtr Refs Type Name*					DataOff	ProtoType *	Size	Value
2F32C8	1	Class	CVBProject		34	elapsedTime As String	28	26,906 seconds
2F2A80	2	Class	COcxItem		38	LoadedOk As Boolean	2	-1
2F28E0	2	Class	COcxItem		3A	isInitilized As Boolean	2	-1
2F25A0	2	Class	COcxItem		3C	Version As Long	4	6
2F24D0	2	Class	COcxItem		40	structNode As INode	4	0
2F20C0	2	Class	COcxItem		44	filePath As String	50	D:_code\vbdec2\vbdec.exe
2F2330	2	Class	COcxItem		48	fileBaseName As String	18	vbdec.exe
	2	Class	COcxItem		4C	fileExt As String	8	.exe
2F2810	2	Class	COcxItem		50	OutPutDir As String	64	D:_code\vbdec2\vbdec\vbdec.exe\
2F1F20	2	Class	COcxItem		54	parentDir As String	30	D:_code\vbdec2
2F29B0	2	Class	COcxItem		58	md5Hash As String	64	3FFC83920A1397D7DE82572EE67DC45C
2F2190	2	Class	COcxItem		5C	isDII As Boolean	2	0
2F2740	2	Class	COcxItem		60	VB50ffset As Long	4	42A2D0
2F2670	2	Class	COcxItem		64	objTable As New CObjectTable	4	5321850
2F1D80	2	Class	COcxItem		68	VBHeader As New CVBHeader	4	755E30
2D3118	1	Class	CDeclare		6C	ComRegData As New CComR	4	52FEF18
2D2DA8	1	Class	CDeclare		70	ComRegInfo As New CComRe	4	5305E38
2D32D0	1	Class	CDeclare		74	ProjectInfo As New CProjectInfo	4	5306850
2D2AE8	1	Class	CDeclare		78	ProjectInfo2 As New CProjectI	4	777CB0
2D2F08		Class	CDeclare		7C	Resources As New CResources	4	786868
2D3278	1	Class	CDeclare	×	80	globaWars As New CollectionEx	4	BD1C1A0
				>	84	VBStartOffset As Long	4	0
					88	StartUpName As String	12	(NONE)
lter					8C	FormCount As Long	4	20
					90	CompileType As String	12	Native
					94	ProjectTitle As String	10	vbdec
dGlobals 0	Linstan	ices		^	98	ProjectName As String	10	vbdec
dOutput 0				<u> </u>	9C	ProjectExename As String	10	vbdec
dPe 0 insta		-			<	public current and current	10	
nOptions 0 i								/
odČOM 0 in	istance	es			Filter			

This new form has already been added to Vbdec and is available from the Explore --> Live Class Instances menu item.

Ok, neat trick. With a live class instance pointer what else can we do? Can we call methods on these classes from the ObjPtr ?

Why yes we can. Here I will refer you to a previous paper titled <u>Scripting Arbitrary VB6 Applications</u>. Building on our previous work we are now able to access any live class instance through its ObjPtr.

```
int HandleIPCMsg(char* m_msg) {
    vector<string> cmd = split(m_msg, ":");
    if(cmd[0] == "regptr") { //regptr:hex_objptr:ROTNAME
        if(cmd.size()!=3) return -1;
        objPtr = strtoul(cmd[1].c_str(), NULL, 16);
        if(objPtr < 0x100) return -2;
        wstring rotName(A2W(cmd[2].c_str()));
        IUnknown *unk = (IUnknown*)objPtr;
        if( CreateFileMoniker(rotName.c_str(), &mon) == S_OK) {
            hr = rot->Register(ROTFLAGS_REGISTRATIONKEEPSALIVE, unk, mon, &appRotToken);
            if(hr == S_OK) rotTokens.push_back(appRotToken);
            if(hr == S_OK) = 0;
            if(hr == S_OK) = 0;
```

Once openscript is active by launching a process through Explore --> Script Remote Application these new features can be accessed using the new "remote" object in the vbdec Script Automation form.

```
💥 JScript Automation Showing last script: lastScript.txt
File Help
    1
    2
         if (remote.isAlive == 0) throw new Error ("No remote instance running..")
    3
    4
        f = remote.forms.item(0)
         tb.t("Main Form caption: " + f.caption)
    5
    6
        remote.regPtr(0x6C9BB0, "privC")
    7
         privC = remote.getObj("privC")
    8
    9
       tb.t(privC.pubVar1)
   10
   11
       remote.newObj("Class1", "newC")
         newC = remote.getObj("newC")
   12
         tb.t(newC.pubVar1)
   13
```

Here again, our knowledge keeps compounding upon itself allowing us to do new and magical things.

Going back:

Ok, so the live class instances list is interesting (and a bit unexpected) but there are still questions about that whole object tear down process. We know how the VB Runtime does its internal class cleanup. How does it handle the cleanup of other resources such as external COM objects?

For this query, I added a VB Collection object to our test module, and set a breakpoint on its VBACollection::Release method in the Vb Runtime. After some probing of the call stack I came across the following function:

```
.text:66061E42 void RESDESCTBL::DestructItem(
    RESDESCTBL *a1,
    void *a2,
    struct RESDESC *a3,
    unsigned int *a4
)
```

This function is pretty interesting. It reveals that the VB Runtime knows about private variables offsets and types to enable clean tear down. Where is this information held and what does it represent?

As we start analyzing this function, we find that the RESDESCTBL pointer corresponds to the Object.aPublicBytes field. Let us consider the following code in a module:

Private a As String Private b As Long Private c As New Collection Sub Main() a = "test" b = 3 c.Add a End Sub 4017B8 008 aPublicBytes 401914 4017C0 010 aModulePublic 402024 - data section 00402024 (+0) 004FF90C UNICODE "test" 00402028 (+4) 00000003 0040202C (+8) 004FF930 -> vtable for collection object .text:00401914 dw 14h |--;total data size .text:00401916 dd 10h | ;size in mem of data block .text:0040191A dd 2 | ;entries .text:0040191E dw 0 | ;unknown .text:00401920 dw 0 ;data offset 1 .text:00401922 dw 1 ;type id = string .text:00401924 dw 8 ;data offset 2 dw 3 ;type id = object .text:00401926

Since this code is in a non-instanced BAS module, the compiler had to define a data area for variable storage. This is where the <code>Object.aModulePublic</code> field comes into play. The <code>Object.aPublicBytes</code> field then points to a <code>RESDESCTBL</code> structure that defines the variables for the code unit.

Looking at our sample data we have 3 private variables, but only two entries in our RESDESCTBL. This is because of how this data is intended to be used. The Private b as long is not visible in this table because it does not require any cleanup on termination.

This table can reveal basic type information and data offsets for private variables, but unfortunately it is not as rich as the IDispatch information we extracted before. As we start to probe the format of this type information we find that it aware of Variants, Strings, Arrays, Objects and UDT types (*if the UDT contains a subtype requiring cleanup)

Here is a closer look at my current working definition for the header structure:

```
Private Type tResDescTbl
    size As Integer
    memDataSz As Integer 'end marker for class data block = ABABABAB
    reqAlloc As Integer
    entries As Long
    unk1 As Integer
End Type
```

The type data that follows this header contains dynamic field types and can get rather complex.

We also find that this table gets used during code initialization through RESDESCTBL::ConstructItem. This is how the Runtime can ensure that pre-dimensioned arrays are ready for first use. The following data defines an array for pre-execution initialization

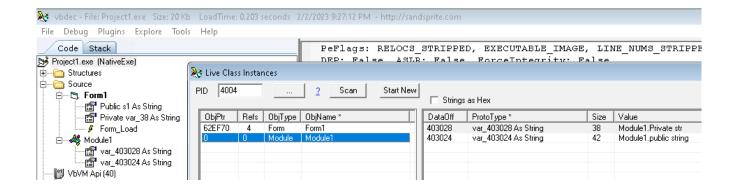
Dim x(&H11223344 To &H2233	4455) As Integer	'11111112 e	elements
----------------------------	------------------	-------------	----------

.text:0040165C	dw 38h
.text:0040165E	dw 20h
.text:00401660	dw 1 ; 660625AD cmp [esi+6], ax
.text:00401662	dw 1
.text:00401664	dw 0
.text:00401666	dw 0 ; RescDesctbl ends here
.text:00401668	dw 4 ; data offset - 660625B1 lea edi, [esi+0Ch]
.text:0040166A	dw 5 ; ary flag 6605606D MOV AX,WORD PTR DS:[ESI+2]
.text:0040166C	dw 0 ; array pre-def struct
.text:0040166E	dw 0
.text:00401670	dw 0 ; 660560F2 TEST BYTE PTR DS:[ECX+8],60
.text:00401672	dw 0
.text:00401674	dw 0
.text:00401676	dw 0 end of struct 2 (size based on 401670 flag)
.text:00401678	dw 1 raw safe array struct - dims
.text:0040167A	dw 92 features FADF_HAVEVARTYPE FIXEDSIZE STATIC
.text:0040167C	dd 2 element size (would be 4 for long)
.text:00401680	dd 0 locks
.text:00401684	dd 0 pvdata
.text:00401688	dd 11111112h ; SafeArrayBound.cElements
.text:0040168C	dd 11223344h ; SafeArrayBound.lbound
.text:00401690	dw 2 VT_I2 (if long value is 3 - VT_I4)

To gain quick visibility into these structures a new form has been added to vbdec under the Explore --> Resource Descriptors menu item.

00000498A0 00			
44004 96 CcodeBody PubBytes 24 24 24 65658 DE modEvents PubBytes 1 21 45000 76 modEvents PubBytes 14 9 44004 40 modEvents PubBytes 14 9 44004 40 modEvents PubBytes 14 14 44004 40 modEvents PubBytes 13 13 40264 30 CbcAtkPat PubBytes 3 3 65002 CcAtkPat PubBytes 3 3 40P+ + 40 Sting True 65002 CcAtkPat PubBytes 3 3 40P+ + 50 Sting True 40P+ + 62 65002 CcAtkPat PubBytes 3 3 40P+ + 50 Sting True 40P+ + 62 65002 CcCatbletot PubBytes 8 8 B B 40P+ + 50 Sting True 40P+ + 62 Diptet True 40P+ + 62 Diptet True 40P+ + 62 Diptet			
56568 DE mod6lobal PuBlytes 14 14 4890C 76 mod6lobal PuBlytes 14 9 4890C 76 mod6lobal PuBlytes 14 9 4890C 76 mod6lobal PuBlytes 14 14 4400 400 400 410 500 5000 7000 700 700 <t< td=""><td></td><td></td><td></td></t<>			
4590C 76 modSibabals Publicker 14 9 4590C 76 modSibabals Publicker 14 9 40040 40 finwhine Publicker 13 13 40040 40 ucStack Publicker 13 13 50204 42 ucStack Publicker 9 9 50204 42 ucStack Publicker 9 9 50204 42 ucStack Publicker 9 9 50202 C2 Christerstelm Publicker 8 8 50202 C2 Christerstelm Publicker 7 7 50202 C2 Christerstelm Publicker 7 7 50204 C2 Christerstelm Publicker 7 7 7 60049B20 4E 6F 70 49 6E 73 74 00 53 65 61 72 63 68 00 00 00 00 00 00 00 00 <td< td=""><td></td><td></td><td></td></td<>			
sigss 44 CCodeObject Publydes 14 14 14 dxD40 40 CCodeObject Publydes 13 13 14 14 dxD44 40 41 41 40			
Ab404 40 fm/Main PubByles 13 14 14 14 14 14 14 13 13 13 13 14 14 14 14 14 15 16<			
CAF4 30 CDoultem PubBytes 9			
D004 4E usStack PubBytes 9 9 1008 2C DAddWind PubBytes 9 9 9 1008 2C DAddWind PubBytes 8 8 0 </td <td></td> <td></td> <td></td>			
C364 30 CPublingTypeDesc PubBytes 9 9 9 SDCC 2C ChatHinerSplete PubBytes 8 8 0 00-0 True 00-0 SDCC 2C ChatHinerSplete PubBytes 8 8 0 00-0 True 00-0 SDCC 2C ChatHinerSplete PubBytes 8 8 0 </td <td></td> <td></td> <td></td>			
1008 2C CackHef PubByes 8 8 0000 25C CackHef PubByes 8 8 8 25C 26 CackHef PubByes 8 8 8 8 25C 26 CackHef PubByes 7			
SDCC 2C Christementer Publyes 8 8 8 ZEC 28 CDPDDbjrt 6C Dbject True D0049B90 4E 6F 70 49 6E 73 74 00 53 65 61 72 63 68 00 00 NopInst.Search True a Filter Filter VA Inf Off Name = V 00049B90 4E 6F 70 49 6E 73 74 00 53 65 61 72 63 68 00 00 NopInst.Search VA VA 00049B00 34 00 01 00 40 00 30 04 40 00 10 04 86 00 100 48 00 01 00 4eDH VA VA Inf Off Name = V 00049B00 64 00 03 00 68 00 30 06 C 00 33 00 70 00 30 08 00 03 00 A			
221C 28 COPUDUMO Public 7 7			
Image in the interval in the interval in the interval			
X X # Filter # Filter 000498B0 4E 6F 70 49 6E 73 74 00 53 65 61 72 63 68 00 00 NopInst.Search A Max Filter 000498D0 00 00 00 00 C4 00 1C 01 00 00 2E 00 00 00 00 00 NopInst.Search 000498D0 34 00 01 00 40 00 03 00 44 00 01 00 48 00 01 00 NopInst.Search 000498D0 4C 00 01 00 50 00 10 00 54 00 11 00 58 00 01 00 L.PT.X.X. 000498D0 4C 00 01 00 50 00 10 00 54 00 01 00 58 00 01 00 L.PT.X.X. 000498D0 9C 00 01 00 50 00 10 00 44 00 01 00 88 00 01 00 L.R.Y.L.Y.L. 000498D0 9C 00 01 00 A0 00 01 00 44 00 01 00 80 00 10 00 L.R.Y.L.Y.L. 000498D0 9C 00 01 00 A0 00 01 00 44 00 10 00 80 00 10 00 L.R.Y.L.Y.L. 000498D0 9C 00 01 00 A0 00 10 00 A4 00 10 00 80 00 10 00 L.R.Y.L.Y.L. 00049200 9C 00 10 00 A0 00 10 00 A4 00 10 00 80 00 10 00 L.R.Y.L.Y.L. 00049200 9C 00 01 00 A0 00 01 00 A4 00 10 00 80 00 10 00 L.R.Y.L.Y.L. 00049200 9C 00 01 00 A0 00 01 00 A4 00 10 00 80 00 10 00 L.R.Y.L.Y.L.Y.L. 00049200 9C 00 10 00 A0 00 01 00 A4 00 10 00 80		OUT THE	
D0049890 4E 6F 70 49 6E 73 74 00 53 65 61 72 63 68 00 00 NopInst.Search ▲ D0049800 00 00 00 00 00 (4 00 1C 01 00 00 2E 00 00 00 00 00 ▲ D0049800 44 00 10 00 10 00 40 00 30 04 40 00 100 48 00 01 00 ▲ ▲ D0049800 44 00 01 00 40 00 30 04 40 00 100 48 00 01 00 ▲ ▲ ▲ D0049800 64 00 30 00 68 00 30 00 6C 00 03 00 70 00 30 00 ↓ 4498A4 038 000 size = C4 00049800 64 00 03 00 78 00 30 06 C0 00 30 07 00 03 00 ↓			>
D0049890 4E 6F 70 49 6E 73 74 00 53 65 61 72 63 68 00 00 NopInst.Search ▲ D0049800 00 00 00 00 00 (4 00 1C 01 00 00 2E 00 00 00 00 00 ▲ D0049800 44 00 10 00 10 00 40 00 30 04 40 00 100 48 00 01 00 ▲ ▲ D0049800 44 00 01 00 40 00 30 04 40 00 100 48 00 01 00 ▲ ▲ ▲ D0049800 64 00 30 00 68 00 30 00 6C 00 03 00 70 00 30 00 ↓ 4498A4 038 000 size = C4 00049800 64 00 03 00 78 00 30 06 C0 00 30 07 00 03 00 ↓			
00049BA0 00			
00049BA0 00			
000049BA0 000000000000000000000000000000000000			
000 000 00 00 00 00 00 00 00 00 00 00 0	00 00 NopInst.Search • VA Thf Off N	opInst.Search • VA Inf Off Name	= Value
34 00 01 00 40 00 03 00 44 00 01 00 48 00 01 00 00049BD0 34 00 01 00 50 00 100 54 00 01 00 54 00 01 00 48 00 01 00 00049D0 64 00 03 00 68 00 30 00 60 00 30 00 60 00 30 00 00049D1 84 00 01 00 90 00 100 90 00 100 00 44 00 01 00 80 00 100 10049D10 92 00 11 00 40 00 300 68 00 30 00 7C 00 30 00 80 00 30 00 10049D10 94 00 30 00 88 00 30 00 7C 00 30 00 80 00 30 00 10049D10 94 00 30 00 80 00 30 00 7C 00 03 00 00 00 00 92 00 11 00 A0 00 01 00 A4 00 01 00 80 00 11 00 10049C00 92 00 01 00 A0 00 01 00 A4 00 01 00 80 00 11 00 10049C00 94 00 30 00 80 00 30 00 EC 00 03 00 D0 00 03 00 10049C10 94 00 30 00 80 00 30 00 EC 00 03 00 D0 00 03 00 10049C20 94 00 30 00 80 00 30 00 EC 00 03 00 E0 00 30 00 10049C30 94 00 30 00 F8 00 03 00 EC 00 03 00 E0 00 30 00 10049C4 94 00 30 00 F8 00 03 00 EC 00 03 00 E0 00 30 00 10049C4 94 00 30 00 F8 00 03 00 EC 00 03 00 E0 00 30 00 10049C4 94 00 30 00 F8 00 03 00 EC 00 03 00 E0 00 30 00 10049C4 94 00 30 00 F8 00 03 00 EC 00 03 00 E0			
4C 00 01 00 50 00 01 00 54 00 01 00 58 00 01 00 1ptx 44495A8 03A 032 membrals2 membrals2 000498D0 64 00 03 00 68 00 03 00 6C 00 03 00 70 00 03 00 1ptx 4495A8 03A 032 004 membrals2 000498D0 64 00 03 00 78 00 03 00 70 00 03 00 6C 00 03 00 70 00 03 00 tptx 4495A8 03C 004 membrals2 000498D0 64 00 03 00 78 00 03 00 70 00 03 00 6C 00 03 00 70 00 03 00 tptx 4495A8 03C 004 membrals2 000498D0 920 00 10 00 A0 00 01 00 94 00 01 00 98 00 01 00	1 00 1 0 D U	1155111 000 000 5110	
00049BD0 64 00 03 00 68 00 03 00 6C 00 03 00 70 00 03 00 dhl.p 00049BD0 74 00 03 00 78 00 03 00 7C 00 03 00 70 00 03 00 dh.llp 4499A8 03C 004 FeqAlloc = 00049BC0 88 00 01 00 90 00 01 00 94 00 01 00 98 00 01 00 7C 00 03 00 80 00 300 dh.llp 4499A8 03C 004 FeqAlloc = 00049BC0 9C 00 11 00 90 00 01 00 94 00 01 00 98 00 01 00 7C 00 03 00 80 00 300 dh.llp 4499A8 03C 004 FeqAlloc = 00049BC0 9C 00 11 00 A0 00 01 00 94 00 01 00 98 00 01 00 7C 00 30 00 80 00 300 C M FeqAlloc = 00049C0 9C 00 11 00 A0 00 11 00 94 00 01 00 98 00 01 00 7C 00 03 00 C0 00 300 FeqAlloc = 4499A8 03C 004 FeqAlloc = 00049C10 9C 00 10 00 A0 00 10 00 98 00 01 00 7C 00 03 00 00 00 00 00 FeqAlloc = FeqAlloc = 00049C20 C4 00 03 00 E8 00 03 00 EC 00 03 00 E0 00 30 0 FeqAlloc = FeqAlloc = FeqAlloc = 00049C30 D4 00 03 00 E8 00 03 00 EC 00 03 00 E0 00 30 0 Ae.i.i.sh FeqAlloc = Frivate var_44 As Stri 00049C50 F4 00 03 00 E8 00 03 00 EC 00 03 00 F0 00 03 00 Aeb. Aeb. Frivate var_46 As Stri 00049C50 F4 00 13 00 B8 01 03 00 12 00 00 7	91 99 T T Y 4495A6 03A 002 m	449BA6 USA UUZ memba	
000498E0 74 00 03 00 78 00 03 00 7C 00 03 00 80 00 03 00 1x 4498AA 03E 006 entries = 2 000498E0 88 00 10 0 90 00 01 00 94 00 01 00 98 00 01 00 98 00 01 00 98 00 01 00 00049C0 9C 00 01 00 A0 00 01 00 44 00 11 00 98 00 01 00 98 00 01 00			oc = 0
00049BE0 74 00 05 00 09 00 01 00 94 00 01 00 98 00 01 00 76 00 05 00 76 00 05 00 76 00 05 00 76 00 05 00 449BAE 042 00A unk1 = 0 00049E00 9C 00 01 00 A0 00 01 00 A4 00 01 00 B0 00 01 00 76 00 05 00 76 00 05 00 05 00 01 00 First event and a strip 00049E00 9C 00 01 00 A0 00 01 00 A4 00 01 00 B0 00 01 00 76 00 03 00 C0 00 30 0 First event and a strip 00049C10 9C 00 01 00 A0 00 01 00 A4 00 01 00 B0 00 01 00 76 00 03 00 C0 00 30 0 76 00 03 00 C0 00 30 0 77 00 00 03 00 00049C20 C4 00 03 00 C8 00 03 00 CC 00 03 00 D0 00 03 00 AEIA Firivate var_40 As 0bje 00049C30 D4 00 03 00 E8 00 03 00 CC 00 03 00 F0 00 03 00 AEIA Firivate var_44 As 0bje 00049C40 D4 00 03 00 E8 00 03 00 FC 00 03 00 F0 00 03 00 AEIA Firivate var_44 As 0bje 00049C40 D4 00 30 0F8 00 03 00 FC 00 03 00 F0 00 03 00 AEIA Firivate var_44 As 0bje 00049C40 D4 01 03 00 80 01 03 00 12 00 00 07 6 00 62 00 AEv.b. Firivate var_42 As 00 50 00049C40 D4 01 03 00 80 01 03 00 12 00 00 07 00 00 00 AE Firivate var_10 As 0bje 00049C40 D4 01 03 00 80 00 22 00 76 00 20 00 70 00 00 00 A Firivate var_10 As 05 <td></td> <td>449BAA 03E 006 entri</td> <td>s = 2E</td>		449BAA 03E 006 entri	s = 2E
00049C00 00	US UU CX		0
D0049C10 B4 00 03 00 B8 00 03 00 BC 00 03 00 BC 00 03 00 CC 00	01 00		
00049C20 C4 00 03 00 C8 00 03 00 CC 00 03 00 D0 00 03 00 ÅÈİD Private var_40 As Obje 00049C30 D4 00 03 00 D8 00 03 00 DC 00 03 00 DC 00 03 00 ÅÈİD Private var_44 As Stri 00049C40 E4 00 03 00 E8 00 03 00 EC 00 03 00 FC 00 03 00 F0 00 03 00 ÅÈID Private var_44 As Stri 00049C40 F4 00 03 00 F8 00 03 00 FC 00 03 00 F0 00 03 00 AEID Private var_44 As Stri 00049C40 O4 01 03 00 08 01 03 00 FC 00 03 00 00 10 300 OØØV.b. Private var_42 As Stri 00049C40 O4 01 03 00 08 01 03 00 12 00 00 00 76 00 62 00 AEv.b. Private var_40 As Stri 00049C40 64 00 65 00 63 00 2E 00 76 00 62 00 70 00 00 00 d.e.cv.b.p Private var_50 As Stri			
D0049C30 D4 00 03 00 D8 00 03 00 DC 00 03 00 E0 00 03 00 ÓØÜà Private var_44 As Stri D0049C40 E4 00 03 00 E8 00 03 00 EC 00 03 00 F0 00 03 00 ÓØÜà Private var_44 As Stri D0049C50 F4 00 03 00 F8 00 03 00 FC 00 03 00 01 03 00 OØÜb Private var_44 As Stri D0049C50 64 00 03 00 FC 00 03 00 76 00 62 00 OØÜv.b. Private var_4C As Stri D0049C70 64 00 65 00 63 00 2E 00 76 00 62 00 70 00 000 Private var_50 As Stri	03 00 (,¼Å Private var	,	String
D0049C30 D4 00 03 00 D8 00 03 00 DC 00 03 00 E0 00 03 00 00Uå Private var_44 As Stri D0049C40 E4 00 03 00 E8 00 03 00 EC 00 03 00 F0 00 03 00 á0Uå Private var_44 As Stri D0049C40 E4 00 03 00 E8 00 03 00 EC 00 03 00 F0 00 03 00 ó0Uå Private var_48 As Stri D0049C40 E4 00 03 00 F8 00 03 00 FC 00 03 00 00 01 03 00 ó0Uå Private var_48 As Stri D0049C40 04 01 03 00 08 01 03 00 12 00 00 76 00 62 00 ó Private var_4C As Stri D0049C40 04 00 65 00 63 00 2E 00 76 00 62 00 70 00 00 00 d.e.c Private var_50 As Stri	03 00 ÅÈÌĐ Private var	ÈÌĐ Private var 40 A	Object
D0049C40 E4 00 03 00 E8 00 03 00 EC 00 03 00 F0 00 03 00 aeis Private var_48 As Stri D0049C50 F4 00 03 00 F8 00 03 00 FC 00 03 00 00 103 00 beuu Private var_48 As Stri D0049C50 64 00 65 00 63 00 2E 00 76 00 62 00 d.e.ev.b. Private var_50 As Stri			
D0049C50 F4 00 03 00 F8 00 03 00 FC 00 03 00 00 01 03 00 6e.t.t.t Private Var_40 AS Stri D0049C60 04 01 03 00 08 01 03 00 12 00 00 07 60 06 20 00 6e.t.t.t Private Var_4C AS Stri D0049C70 04 00 65 00 63 00 2E 00 76 00 62 00 0e.t.t Private var_50 AS Stri	P2 00 K h k K	Privace var_44 A	
00049060 04 01 03 00 08 01 03 00 12 00 00 00 76 00 62 00 Private var_4C As Stri 00049070 64 00 65 00 63 00 2E 00 76 00 62 00 70 00 00 00 d.e.cv.b.p Private var_50 As Stri	Privace var_	Plivace val_40 A	
00049C70 64 00 65 00 63 00 2E 00 76 00 62 00 70 00 00 00 d.e.cv.b.p Private var_50 As Stri			String
			String
		.e.cv.n.p	
00049C90 73 8D 43 13 30 BE 96 47 8D 48 F7 36 12 C6 9A 2E s.C.O».G.H+6.Æ Private var_58 As Stri			
00049CA0 OC 00 54 00 00 00 00 00 54 00 04 04 09 00 00 00 TTT. Private var_64 As Obje			Object
00049CB0 6D 73 76 62 76 6D 36 30 00 00 00 00 0F 00 00 msvbvm60 Private var 68 As Obje	00 00 messbare60 performance		Object.
00049CC0 5F 5F 76 62 61 43 68 65 63 6B 54 79 70 65 00 00vbaCheckType Private var 6C As Obje	Private var	svbvm6U Private var 68 A	
00049CD0 B0 9C 44 00 C0 9C 44 00 00 00 04 00 C4 CC 60 00 .D.A.DAt .	IIIVacc var_		

While not every type of entry is fully supported yet, results so far are pretty good. With this extra information we are now able to start discovering private variables. This information can then be bubbled back into the main UI as well in the Live Class Instances form.



vbdec - File: Project1.exe Size: 16 Kb LoadTime: 0.172	seconds 2/2/	2023 9:32:07 PN	/ - http:/	/sandsprite.com		
File Debug Plugins Explore Tools Help						
Code Stack Project1.exe (NativeExe) Structures Form1 Private var_34 As String Private var_38 As String Private var_40 As Array (5) (5) As Bstr Live Class Instances	It New	VA 40174C 401750 401754 401758 40175C 401760 401764	Off 000 004 008 00C 010 014 018	Name aObjectInfo Const1 aPublicBytes aStaticBytes aModulePublic aModuleStatic aObjectName	Value 40136C FFFFFFF 401868 0 0 0 401790	
ObjPtr Refs ObjType ObjName* 698398 4 Form Form1 Source Private s As String Private s2 As String		Private v Private v Private v	ar_34 As 9 ar_38 As 9	itring 32 private		
Private a(5, 5) As String Private Sub Form_Load() s = "private string 1" s2 = "private string 2" For i = 0 To 5 a(i, i) = "string " & i Next End Sub	Address 6983 View As: Varie Date Curry Sing Doul Ary Assume Da Size 4 Bytes © Long © Bstr	ant y ency le ble ata As	dims featu elem locks Data Bound Bound Bound Bound	ures: Ox192 (S Size:4	6	

Not as comprehensive as the other data, but still a welcome addition. The next step is to start scanning the disasm to infer more types and data offsets. Knowing about the existence and basic type of private variables could also be coupled with more invasive run time data collection.

I am currently experimenting with a technique to call rtcTypeName through openscript to query object types dynamically. This many also be possible through ReadProcessMemory calls for internal objects.

p = tmp	<pre>tmp = remote.typName(0x81EF48) o = remote.readLong(0x81EF48 + 0x38); tmp += ", " + remote.typName(p); tb.alert(tmp) Live Class Instances</pre>							
	ND 1383			<u>?</u> Sca	n	Start New	□ String	s as Hex
	ObjPtr 81EF48 81EE58	Refs 2 1	ObjType Form Class vbdec Form1, I	ObjName * Form1 Class1 FileSystemObje	×		DataOff 34 38	ProtoType * Private var_34 As Object Private var_38 As Object
				ок				

Conclusion:

In this article we discovered how the VB6 Runtime tracks live class instances and certain types of private variables for initialization and cleanup.

This allowed us to enumerate all live class instances in a running process and dump known data offsets for external viewing.

Building on previous work, the revealed class ObjPtr() also allowed us to remotely script public functions on any arbitrary class instance.

The ability to call private class functions is now also <u>within our reach</u>. This same technique will also allow us to call arbitrary module functions remotely as well. The largest barrier at this point will be determining prototypes for the private functions.

While the type information held within the RESDESCTBL is not as rich as the IDispatch information, it is still a welcome addition and reveals new information to us. We may also be able to further refine the data it gives us through probing the live process.

At this point we still have gaps and more to explore but are making good progress none the less.

Our path has proven quite sinuous, but also very interesting. It appears that I am hooked on the intrigue and discovery.