References

C cert.pl/en/posts/2021/04/keeping-an-eye-on-guloader-reverse-engineering-the-loader/



CloudEye (originally GuLoader) is a small malware downloader written in Visual Basic that's used in delivering all sorts of malicious payloads to victim machines. Its primary function is to download, decrypt and run an executable binary off a server (commonly a legitimate one like Google Drive or Microsoft OneDrive).

At the time of writing this article, the malicious code can be split into two parts:

- The core of the program that performs VM checks, downloads the code, decrypts and runs it
- A small wrapper that hides the core by encrypting it with a simple xor algorithm

While the outer layer is pretty tiny and straightforward, mimicking it and manually unpacking the core can be a bit of a headache. In this article, we'll explain how one can leverage IDA Pro functionalities to simplify this process.

Sample analysed:

The first thing you want to do while reverse-engineering Visual Basic binaries in IDA is grab a copy of <u>vb.idc</u>. It's a super useful IDA script that parses the embedded VB metadata and provides you with much more information about the binary than the original analysis.

Compare the number of detected event entry points before running the script:

Name		Address	Ordinal
🚺 start		004017E0	[main entry]
Line 1 of 1			
	OK Cancel Se	earch Help	

And after:

Name	Address	Ordinal	
1 start	004017E0	[main entry]	
[f] _O_Pub_Obj_Inf1_Method0x1	00413650		
[f] _O_Pub_Obj_Inf1_Event0x2	00413941		
[f] _O_Pub_Obj_Inf1_Event0x3	004139F7		
[f] _O_Pub_Obj_Inf1_Event0x4	00413AD7		
[f] _O_Pub_Obj_Inf1_Event0x5	00413B74		
[f] _O_Pub_Obj_Inf1_Event0x6	00413D9F		
[f] _O_Pub_Obj_Inf1_Event0x7	00413FFB		
[f] _O_Pub_Obj_Inf1_Event0x8	0041407E		
[f] _O_Pub_Obj_Inf1_Event0x9	004147BB		
[f] _O_Pub_Obj_Inf1_Event0xA	00414900		
[f] _O_Pub_Obj_Inf1_Event0xB	00415584		
_O_Pub_Obj_Inf1_Event0xC	0041571A		
_O_Pub_Obj_Inf1_Event0xD	00415D31		
_O_Pub_Obj_Inf1_Event0xE	00413034		
_O_Pub_Obj_Inf1_Event0xF	004130A0		
_O_Pub_Obj_Inf1_Event0x10	004131E6		
_O_Pub_Obj_Inf1_Event0x11	004133FD		
_O_Pub_Obj_Inf1_Event0x12	004134F0		
_O_Pub_Obj_Inf1_Event0x13	00413557		
_O_Pub_Obj_Inf1_Event0x14	0041371B		
_O_Pub_Obj_Inf1_Event0x15	004138D2		
_O_Pub_Obj_Inf1_Event0x16	00414244		
_O_Pub_Obj_Inf1_Event0x17	00414425		
_O_Pub_Obj_Inf1_Event0x18	00414B66		
_O_Pub_Obj_Inf1_Event0x19	00414BE8		
_O_Pub_Obj_Inf1_Event0x1A	00414E11		
_O_Pub_Obj_Inf1_Event0x1B	00414EB7		-
f O Bub Obi Infl Eventoric	00415100		•
OK Cancel	Search Help		

Locating the malware entry point is still not trivial, though. You can iterate over all discovered entry points and judge if there's anything suspicious or not, but that can become quite tedious, and you can still miss some better-hidden code.

Sometimes, a good method is to search for all add instructions and find the "odd one" with a large immediate value sticking out. You can do that in IDA either by selecting Search -> Text or if you're aspiring to be a power-user: by quickly tapping Alt + t.

Text sear	ch (slow!)
<u>S</u> tring	add
	Match <u>c</u> ase <u>R</u> egular expression <u>I</u> dentifier Search <u>U</u> p <u>F</u> ind all occurrences
	O <u>K</u> Cancel Help

Make sure you check the Find all occurrences box, this will take IDA a bit longer, but it will allow you to inspect all matches at once.

	IDA View-A	X 🕅 Occurrences of	: add 🗶 [🧿	Hex View-1	XA	Structures	×	E	Enums	×		Imports	×	P	Exports	×	
Address	5	Function	Instru	uction													*
.text:00	40D30C							; Jmp to M	ethod Addr 0	x413650)						
.text:00	41306D	_O_Pub_Obj_Inf1_Even	t0×E		add	edi, 27A3410h											
.text:00	04133CB	_O_Pub_Obj_Inf1_Even	t0x10		add	esp, 0Ch											
.text:00	4133E5	_O_Pub_Obj_Inf1_Even	t0x10		add	esp, 0Ch											
.text:00	4136CD	_O_Pub_Obj_Inf1_Meth	od0x1		add	esp, 0Ch											
.text:00	4136E7	_O_Pub_Obj_Inf1_Meth	od0x1		add	esp, 0Ch											
.text:00	41379B	_O_Pub_Obj_Inf1_Even	t0x14		add	esp, 0Ch											
.text:00	41389C	_O_Pub_Obj_Inf1_Even	t0x14		add	esp, 0Ch											
.text:00	413F8F	_O_Pub_Obj_Inf1_Even	t0x6		add	esp, 0Ch											
.text:00	0413FB9	_O_Pub_Obj_Inf1_Even	t0x6		add	esp, 0Ch											
.text:00	414114	_O_Pub_Obj_Inf1_Even	t0x8		add	esp, 0Ch											
.text:00	04143F3	_O_Pub_Obj_Inf1_Even	t0x16		add	esp, 0Ch											
.text:00	41440D	_O_Pub_Obj_Inf1_Even	t0x16		add	esp, 0Ch											
.text:00	41474D	_O_Pub_Obj_Inf1_Even	t0x17		add	esp, 0Ch											
.text:00	41475F	_O_Pub_Obj_Inf1_Even	t0x17		add	esp, 0Ch											
.text:00	414793	_O_Pub_Obj_Inf1_Even	t0x17		add	esp, 0Ch											
.text:00	0414DB0	_O_Pub_Obj_Inf1_Even	t0x19		add	esp, 0Ch											
.text:00	414DD9	_O_Pub_Obj_Inf1_Even	t0x19		add	esp, 0Ch											
.text:00	415161	_O_Pub_Obj_Inf1_Even	t0x1B		add	esp, 0Ch											
.text:00	415528	_O_Pub_Obj_Inf1_Even	t0x1E		add	esp, 0Ch											
.text:00	415554	_O_Pub_Obj_Inf1_Even	t0x1E		add	esp, 0Ch											
.text:00	415905	_O_Pub_Obj_Inf1_Even	t0xC		add	esp, 0Ch											
.text:00	41591F	_O_Pub_Obj_Inf1_Even	t0xC		add	esp, 0Ch											
.text:00	4159CC	_O_Pub_Obj_Inf1_Even	t0x1F		add	esp, 0Ch											
.text:00	416B0C	_O_Pub_Obj_Inf1_Even	t0x25		add	esp, 0Ch											
.text:00	4174DD	_O_Pub_Obj_Inf1_Even	t0x25		add	esp, 0Ch											
.text:00	41794A	_O_Pub_Obj_Inf1_Even	t0x25		add	esp, 0Ch											
.text:00	417CC2	_O_Pub_Obj_Inf1_Even	t0x25		add	esp, 0Ch											
.text:00	417EBD	_O_Pub_Obj_Inf1_Even	t0x25		add	esp, 0Ch											
.text:00	41896F	_O_Pub_Obj_Inf1_Even	t0x25		add	esp, 0Ch											
.text:00	418E17	_O_Pub_Obj_Inf1_Even	t0x25		add	esp, 0Ch											
.text:00	4191EF	_O_Pub_Obj_Inf1_Even	t0x25		add	esp, 0Ch											
.text:00	4196DC	_O_Pub_Obj_Inf1_Even	t0x25		add	esp, 0Ch											
.text:00	419ED1	_O_Pub_Obj_Inf1_Even	t0x25		add	esp, 0Ch											
Line 13	7 of 229	0 8-6 05 1-6 5	60			och											

Now, navigate to the function in question:

.text:00413034		public	_O_Pub_Obj_Inf1_Event0xE	
.text:00413034	_O_Pub_Obj_Inf1_Event0xE	proc ne	ar	<pre>; CODE XREF: .text:0040D65C+j</pre>
.text:00413034				
.text:00413034	var_14	= dword	l ptr -14h	
.text:00413034	var_8	= dword	ptr -8	
.text:00413034	var_4	= dword	ptr -4	
.text:00413034			-	
.text:00413034 55		push	ebp	<pre>; _O_Pri_Obj_Inf1_Event0xE</pre>
.text:00413035 8B EC		mov	ebp, esp	
.text:00413037 51		push	ecx	
.text:00413038 51		push	ecx	
.text:00413039 68 B6 15 40 00		push	offsetvbaExceptHandler	
.text:0041303E 64 A1 00 00 00 00		mov	eax, large fs:0	
.text:00413044 50		push	eax	
.text:00413045 64 89 25 00 00 00 00		mov	large fs:0, esp	
.text:0041304C 6A 10		push	10h	
.text:0041304E 58		pop	eax	
.text:0041304F E8 5C E5 FE FF		call	vbaChkstk	
.text:00413054 53		push	ebx	
.text:00413055 56		push	esi	
.text:00413056 57		push	edi	
.text:00413057 89 65 F8		mov	[ebp+var_8], esp	
.text:0041305A C7 45 FC 78 11 40 00		mov	<pre>[ebp+var_4], offset dword_401178</pre>	
.text:00413061 C7 45 EC 33 33 33 33		mov	[ebp+var_14], 33333333h	
.text:00413068 BF BE EF C5 FD		mov	edi, OFDC5EFBEh	
.text:0041306D 81 C7 10 34 7A 02		add	edi, 27A3410h	
.text:00413073 57		push	edi	
.text:00413074 C3		retn		
.text:00413074	_O_Pub_Obj_Inf1_Event0xE	endp ;	sp-analysis failed	
.text:00413074				

And press **tab** to see the matching decompilation (as usual, the decompiler does most of the job for us):



With the malware entry address 0x4023CE we can now begin analyzing the real loader. Let's jump to the entry address by selecting Jump -> Jump to address (shortcut g)

Surprisingly, there's no code there, just a bunch of data.



That's because IDA didn't know to follow the code reference; to fix that, we'll have to mark the data as code ourselves. Start by undefining the fragment Edit -> Undefine (shortcut u)

This will split the large chunk of data into single-byte lines:

jie Edit jump Search View Debugger Lumija Options Windows Help																			
····································																			
5																			\$ · · · ·
Library function 📃 Regular funct	on 📕 Instruction 📗 Data 📕 Unexp	lored Exte	mal symbol 📕 Lumina	function															
📝 Functions window		8 🛛 🖪	IDA View-A	× 🗈	Pseudocode-A	× 🖸	Hex View-1	×	Structures	X 🖽	Enums	×	3 1	Imports	×	1	Exports	×	
Function name	Segment	St.*	.text:00401FAC				dd 7C40h, 4 dup	(0)											
_vbaChkstk	.test	OC	.text:00401FC0				db 90h												
vbaExceptHandler	.text	OC	.text:00401FC4				dd 32001Fh. 100	(h dup (0)											
_vbaVarForNext	text	oc	.text:004023C8				db 0	an and (a)											
_vbaOnError	.text	oc	.text:004023C3				db 0												
_vbaVarForinit	.text	oc	.text:004023CR				db 072h ; 0												
_vbaFreeStrList	.text	OC	.text:004023CC				db 3Ch ; <												
_vbal4Var	.text	OC	.text:004023CD				db 4Fh ; 0												
vbaLateldCallLd	.text	oc	.text:004023CF				db OERh ; a												
7 rtcUpperCaseBstr	.text	oc	.text:004023D0				db 215h ; .												
7 rtcEnvironBstr	.test	oc	.text:004023D1				db 0A4h ; =												
📝 rtcLeftBstr	.text	OC	.text:004023D3				db 0AEh ; @												
7 rtcEndOfFile	.text	oc	.text:004023D4				db 54h ; T												
_vbaVarAdd	.text	oc	.text:004023D5				db 7Fh ; ⊖ db 7Fh ; ⊖												
📝 rtcGetDateVar	.text	oc	.text:004023D7				db 7Fh ; ⊖												
📝 rtcDateAdd	.text	OC	.text:004023D8				db 7Fh ; ⊖												
📝 rtcRandomize	.text	OC	text:004023D3				db 717h : 0												
7 rtcGetHourOfDay	.text	oc	.text:004023DB				db 7Fh ; ⊙												
📝 rtcIsNumeric	.text	oc	.text:004023DC				db 7Fh ; ⊖												
7 rtcPPMT	.test	oc	.text:004023DE				db 711h;0												
📝 rtcKilFiles	.text	OC	.text:004023DF				db 7Fh ; ⊙												
📝 rtcSpaceBstr	.text	oc	.text:004023E0				db 7Fh ; ⊖												
7 rtcGetPresentDate	.text	oc	.text:004023E2				db 71°h;⊖												
_vbaVarNove	.text	oc	.text:004023E3				db 717h ; ⊖												
📝 rtcFreeFile	.text	OC	text:004023E4				db 7Fh ; 0												
📝 rtcinputBox	.text	oc	.text:004023E6				db 7Fh ; ⊖												
7 rtcInStrRev	.text	oc	.text:004023E7				db 71°h;⊝												
7 _vbal214	.text	oc	text:004023E3				db 77h : 0												
_vbaLateidSt	.text	oc	.text:004023EA				db 7Fh ; ⊖												
📝 rtcCos	.text	OC	.text:00402328				db 7Fh ; 0												
📝 rtcShell	.text	OC.	.text:004023ED				db 717h ; ⊖												
1 rtcVarFromError	.text	oc	.text:004023EE				db 7Fh ; ⊙												
vbaVarTstNe	.text	oc	text:004023E0				db 7Fh : 0												
1 rtcFileSeek	.text	oc	.text:004023F1				db 717h ; ⊖												
✓ _vbaLenBstrB	.test	oc	.text:004023F2				db 717h;⊖ db 717h;⊖												
vbaVar2Vec	text	OC .	.text:004023F4				db 7Fh ; ⊖												
_vbaAryMove	.text	oc	.text:00402385				db 71°h;⊝												
_vbaAryDestruct	text	oc	.text:004023F6				do nh;⊖ do 71h;⊖												
1 rtcjoin	.text	oc	.text:004023F8				db 7Fh ; 🗇												
	.text	OC.	.text:004023F9				db 7Fh ; ⊖												
VoaAryConstruct2	.text	OC.	.text:004023FB				db 717h : ⊖												
1 recenangeour	.1688	OC.	.text:004023FC				db 717h;⊙												
Volume Parts	test.	00	text:004023FD				000 7£h;⊖ 40h 7£h;⊖												
2 what has me	.text	00	.text:004023FF				db 47h ; G												
_vusStrump	.text	-	.text:00402400				db OXRh ; #												
2 InconcerciaryName	text	00	text:00402401 text:00402402				400 330 ; 3 600 4Ch ; L												
Z dationation in a second	Jane.	~	.text:00402403				db 5Eh												
Z do Environmente	.text	00	.text:00402404				db 0ADh ; -												
Z deble Telle	.text	00	.text:00402406				db 008h ; #												
Na Anakai Iored	.text		.text:00402407				db 0D8h ; ∅												

Now we can once again jump to 0×4023 (the original line contained many bytes, and IDA doesn't know which one to follow and decides to stay on the first byte) and mark the data as code Edit -> Code (shortcut c).

File Edit Jump Search View Debugge	r Lumina Options Windows	Help													
📅 🖬 🔍 ㅋㅋㅋ 🐁 🛍 🐁 🚳	🔰 🔬 i 🖾 🥥 i 📾 🖄 🤉	1 🖈 - 🖈 1	🖌 📉 🕨 💷 🖬 No debr	ugger 👻 🤹 🚼	3 D % 💷	🗟 🗟 i 🚺 🚼 💕									
5															5
Library function Regular function	Instruction 📃 Data 🗮 Unexplo	red Extern	al symbol 📕 Lumina function												
7 Functions window		8 1	IDA More A	Decoderade 4	× 🖂	Harr Minur 1 V	Chrysleyer	v 📼	Farmer	v 📼	Immunity	v /37	Ferraria	~	
Eurotion name	Serment	G1 A	Los risina A	ag recoucours	× 📟	dd 2040b 4 day (0)	scructures	A 160	Enoms	~ ~	imports	~ @	Exports	^	
7 vbaChkstk	test	oc	.text:00401FC0			db 90h									
VhaEvrentHandler	text	00	.text:00401FC1			db 7Fh, 0AEh, 0									
vbaVarForNext	test	oc.	text:00401FC4			dd 32001Fh, 100h dup(0)									
VbaOnError	.test	oc	.text:004023C9			db 0									
vbaVarForInit	.text	oc	text:004023CA			db 75h; u db 0F2b; ò									
VbaFreeStrList	.text	oc	.text:004023CC			db 3Ch / <									
J _vbal4Var	.text	oc	.text:004023CD			db 4Ph ; 0									
vbaLateidCallLd	.text	oc	.text:004023CE			dec edi									
7 rtcUpperCaseBstr	.text	oc	.text:004023CF			jmp short loc_4023FF									
7 rtcEnvironBstr	.text	oc	.text:004023CF ;			db 034b - 8									
📝 rtcLeftBstr	.text	oc	.text:004023D2			db 6Th ; o									
7 rtcEndOfFile	.text	oc	.text:004023D3			db 0AEh ; 0									
VbaVarAdd	.text	oc	.text:004023D5			db 7Fh;⊖									
1 rtcGetDateVar	.text	oc	.text:004023D6			db 717h;⊝									
📝 rtcDateAdd	.text	oc	.text:004023D7			db 717h ; ⊖ db 717h ; ⊖									
ftcRandomize	.text	oc	.text:004023D9			db 7Fh ; ⊖									
/ rtcGetHourOfDay	.text	oc	.text:004023DA			db 7Fh ; 0									
1 rtclsNumeric	text	oc	.text:004023DC			db 777h; 0									
1 rtcPPMT	text	oc	.text:004023DD			d0o 717h ; ⊖									
/ rtcKillPines	.text	oc	.text:004023DE			db 71°h;⊖ db 71°h;⊖									
/ rtcSpaceBstr	test	oc.	.text:004023E0			db 717h ; ⊖									
/ rtcGetPresentDate	test	oc	.text:004023E1			db 7Fh ; ⊖									
/ _vbavarNove	.text	00	.text:004023E3			db 71°h;⊖									
Z delegade	text	~	.text:004023E4			db 717h ; ⊖									
T dtripStrBou	test	~	text:004023E5			db 7Fh ; 0									
V ubal214	test	~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~	.text:004023E7			db 7Fh ; ⊖									
C ubalate data	test		.text:00402328			db 7Fh ; 0									
T deCos	test		.text:004023EA			db 777h; ☉									
7 dr.Shell	text	oc.	.text:004023EB			d0o 717h ; ⊖									
T rtcVarFromFron	text	oc.	text:004023EC			db 7Fh ; ⊖ db 7Fb : ⊖									
7 vbaVarTstNe	test	oc	.text:004023EE			db 717h; ⊖									
7 rtcFileSeek	.text	oc	text:004023EF			db 7Fh ; 0 dh 7Fh ; 0									
_vbaLenBstrB	.text	oc	.text:004023F1			db 71°h; ⊖									
_vbaVar2Vec	.text	oc	.text:004023F2			db 77h; 0									
C _vbaAryMove	.text	oc	text:004023F4			db 7Fh 1 0									
_vbaAryDestruct	.text	oc	.text:004023F5			db 7Fh ; ⊖									
📝 rtsjoin	.text	oc	text:004023F6			db 717h; 0 db 717h; 0									
_vbaGenerateBoundsError	.text	oc	.text:004023F8			db 7Fh ; ⊙									
_vbaAryConstruct2	.text	oc	.text:004023F9			db 7Fh ; ⊖									
📝 rtcChangeDir	.text	oc	.text:004023FB			db 717h; 0									
VbaEnd	.text	oc	.text:004023FC			db 717h ; ⊖									
1 rtcRightTrimBstr	.text	oc	text:004023FD			db 7Fh ; ⊕ db 7Fh ; ⊕									
_vbaStrCmp	text	oc	.text:004023FF ;												
ftcWeekdayName	.text	oc	.text:004023FF												
_vbaStrCopy	.test	oc	+= .text:004023FF 100_4	Neder I		inc edi		, coos axer: .t	exc. over23CF1]						
1 rtcMonthName	.text	oc	.text:00402400			jmp short loc_402435									
1 rtcEnvironVar	text	oc	.text:00402400 ;			db 40b : 1									
_vbaVarTstEq	.text	- D0	.text:00402403			db 5Eh ; ^									
•															

This will automatically disassemble all reachable code blocks and functions.

We can almost immediately notice that this isn't an ordinary function, but something rather weird is going on: there are many jumps with random data between them. We can clear it up a bit by grouping the data between code blocks together and adding a few arrows:

e <u>E</u> dit Jump Searc <u>h V</u> iew Deb <u>ugg</u> er	Lumina Options Wind	ows Help	>																				
5 🖬 이 아이 아	4 i ka i 🛛 🌢 i 🖄 i	ഷ് 🕈 🚽	•• # si	á×∶⊳⊡⊡[No debugge	er v '	10 🛃 i	3050	3	🗊 🕈 🕅													
																							5
Library function Regular function	nstruction 📃 Data 📕 Une	xplored	Externa	il symbol 📕 Lumina f	function																		
Functions minders			-			-											-			-			
Functions window		88	181	IDA View-A	×	Pseudoco	de-A	x	He	x View-1	X	Structures	XE		Enums	×	80	Imports	×	025	Exports	×	
cbon name	Segment	St.*		.text:004023CE					dec	adi			· loader a										
vbaCrikstk	M91.	00		.text:004023CF					1mp	short 1	oc_4023FF		,										
	.text			.text:004023CF																			
vbava For Next	text			.text:004023D1	aor				0.0 7	Th, 7Fh, 7Fh,	7Fh, 7Fh, 7Fh, 7Fh, 7	rh, 7Fh, 7Fh, 7Fh, 7Fh, 7	h, 7Fh, 7Fh, 7Fh, 7Fh, 7	Th, 775.	JTh .								
uhaVarEorinit	text	00		.text:004023D1					db 7	Ph, 7Ph, 7Ph,	7Fh, 7Fh, 7Fh, 7	rh, 7Fh, 7Fh, 7Fh											
vbaFreeStrList	.text	oc		.text:004023FF	,																		
vbal4Var	.text	00		.text:004023FF	loc_4023	FT :							; CODE XRS	r: .tes	st:004023CF+j								
vbaLateldCallLd	.text	00		.text:004023FF		_			inc	edi short 1	00 402435												
- cUpperCaseBstr	.text	00	1	.text:00402400																			
cEnvironBstr	.text	00		.text:00402402	a1000000	0000000			_	,19h,'00	0000000000000000												
cLeftBstr	.text	oc		.text:00402435	,		_																
cEndOfFile	.text	00	1 L	.text:00402435	100_4024	35:							; CODE XRS	F: .tes	kt:004024001j								
vbaVarAdd	.text	00		.text:00402435					nop	short 1	00 402481												
cGetDateVar	.text	00	[.text:00402436	;				244														
cDateAdd	.text	00		.text:00402438	aAh				db i	58,8,8	,8,8,8,8,8,8,8,	8,8,8,8,8,8,8,8,8											
cRandomize	.text	00		.text:00402438				-	db 8	8.8.8.8	0,0,0,0,0,0,0,0	, 0, 0, 0, 0, 0, 0, 0, 0, 0	0,0,0,0,0,0,0,0,0										
:GetHour0fDay	.text	oc		.text:00402481	;		_																
IsNumeric	.text	0C		text:00402481	100 4024	81							: CODE X88		1:0040243611								
PPMT	.text	0C		.text:00402481					nov	ecx, 6E	CEEFh												
KillFiles	.text	oc		.text:00402486					cld														
:SpaceBstr	.text	oc	1	.text:00402487					244	anore	oc_vozvaz												
GetPresentDate	.text	oc		.text:00402489	a2hVooco	0000000			_	ER V0000000	00000000000000	ō'											
VDavarMove	.text	oc		.text:004024A2	;		_																
chreefile	.text	00		.text:004024A2	loc_4024	A2:							; CODE XR	F: .tes	kt:004024871j								
cinputeox	.text			.text:004024A2					dec	ebx short 1	00 4024CF												
ubs/214	text	~		.text:004024A3	;				·····														
ubal stoldSt	text.	~		.text:004024A5					-	on, caaxxxx	****		200000										
rEos	text	00		.text:004024CF	,																		
Shell	test	oc		.text:004024CF	loc_4024	CT :							; CODE XRS	F: .tes	kt:004024A31j								
WarFromFrror	text	oc		.text:004024CF					100	short 1	oc 4024EE												
vbaVarTstNe	.text	oc		.text:004024D0																			
cFileSeek	.text	oc		.text:004024D2	alfCeeee	eeeeee				ry, ceeèèèèè	ecceccééééééé	eece'											
vbaLenBstrB	.text	oc		.text:004024EE			_																
vbaVar2Vec	.text	oc		.text:004024EE	100_4024	:33			-44	een 12	Deports		; CODE XRS	F: .tes	kt:004024D013								
vbaAryMove	.text	oc		text:004024F4					clc	wdR, 17													
vbaAryDestruct	.text	oc	- I I	text:004024F5					jap	short 1	oc_40252A												
cJoin	.text	0C		.text:004024F5 .text:004024F7	aAwdz				-	WUZERKERK	kkkkkkkkkkk	****	kkkkkkkkkkk										
vbaGenerateBoundsError	.text	0C		.text:0040252A	;																		
baAryConstruct2	.text	oc		.text:0040252A	100 4025	28.							- CODE 102		+								
ChangeDir	.text	oc	1 1	.text:0040252A	100_1045				dec	ebx _			,										
baEnd	.text	oc	l r	text:0040252B					1mp	short 1	oc_402554												
RightTrimBstr	.text	00		 text:0040252B text:0040252D 	aCN11111	1111111			_	-Cawl1111111	11111111111111												
bastrump	.text	00		.text:00402554			_																
weekoaywame	.text	00		.text:00402554	100 4025	14							· CODE X88		** - 00402528+1								
/basercopy	text	00	1 1	.text:00402554					inc	ebx _			,										
Fourier	txet.	00	1	.text:00402555					jap	short 1	oc_402589												
uballarTetCo	.test	00		.text:00402557					db 0	C7h ; Ç													
FINE R ME F JOINT	.0200	00.4		.text:00402558					db 0	P3b - 6													

Some of our avid readers will surely recognize this pattern from our <u>Dissecting Smoke</u> <u>Loader</u> article. The main takeaway was that while manually reconstructing the code flow and creating a new disassembly is possible; usually, the best method is to let IDA decompiler deal with such obfuscations.

But if we try to create a new function at the start address (Edit -> Functions -> Create Function shortcut p), all we get is this annoying error message:

.text:00402FEE: The function has undefined instruction/data at the specified address. Your request has been put in the autoanalysis queue.

That's because IDA wasn't able to disassemble the code at the given address; let's see what the fuzz is about.



Well, yes, it doesn't look too correct. Instructions in the form of jmp short near ptr <addr>+<number> should almost always raise a red flag for you. It very often means that the jmp (or any other code-flow-altering instruction) tries to jump into the middle of already defined code/data. In this case, though, it looks like IDA just made an error, and we have to mark the data as code manually, similarly as we had done previously.



Good as new! We may have to repeat this several times before we get all parts correct.

At some point, though, we'll come across a fragment that no longer looks like correct x86 code:

	text:00403F30	db 0EEb	: î	
	text:00403F31	db 0EEh	: 1	
	.text:00403F32	db 0EEh	; î	
	.text:00403F33 :		/ _	
	.text:00403F33			
	text:00403F33 loc 403F33:			: CODE XREF: .text:00403F07ti
- b e	.text:00403F33	inc	edi	,,
	.text:00403F34	amt	short loc 403F3F	
	.text:00403F34 ;			
•	.text:00403F36	db 92h	1	
•	.text:00403F37	db 85h		
•	.text:00403F38	db 0B0h	0	
•	.text:00403F39	db 29h		
•	.text:00403F3A	db 0BFh	12	
•	.text:00403F3B	db 0BFh	12	
•	.text:00403F3C	db 0BFh	12	
•	.text:00403F3D	db 0BFh	i è	
•	.text:00403F3E	db 0BFh	i è	
	.text:00403F3F ;			
	.text:00403F3F			
	.text:00403F3F loc_403F3F:			<pre>; CODE XREF: .text:00403F34+j</pre>
	.text:00403F3F			; .text:00403F66↓j
	.text:00403F3F			; DATA XREF:
9 0	.text:00403F3F	mov	gs, esp	
•	.text:00403F41	ficomp	dword ptr [ecx+67h]	
	.text:00403F44	db	65h	
	.text:00403F44	wait		
	.text:00403F46	fstp9	st (6)	
	.text:00403F48	outsb		
	.text:00403F49	mov	eax, 0E6C877Ah	
	.text:00403F4E	dec	ebx	
	.text:00403F4F	aas		
	.text:00403F50	test	dword ptr [ecx-4254BAACh], 5D9C5B03h	
	.text:00403F5A	mov	eax, ds:0BF23D604h	
- 1	.text:00403F5F	inc	edi	
- 1	.text:00403F60	and	eax, 1D8EBBA2h	
	.text:00403F65	xchg	eax, edx	
	.text:00403F66	ja	short near ptr loc_403F3F+1	
	.text:00403F68	push	es	
	.text:00403F69	mov	b1, 95n ; ***	
	.text:00403F6B	aec	epp	
	.text:00403F6C	add	a1, 0	
	.text:00403F6E	das	about the deliver and appression of the	
	text:00403F0F	TG2	ebp, [edi+eax*2-46Bff411n]	
	text:00403F77	pop	CDX 0C9F95435b	
	text:00403F77	and	Car, COPP J34551	
	text:00403F7C	db 8Fb		
	text:00403F7D	db 5Fb		
	text:00403F7E	db 12h	· _	
	text:00403F7F	db 41b	· A	
			/	

That's the code that gets decrypted in previous code blocks; naturally, IDA won't decompile invalid instructions. We can get around that using (at least) 2 methods:

- by selecting the code segments, we want to include in our newly-created function
- by patching the last jmp instruction to ret , which will cut off the last invalid block from our function

We'll go with the first method as it's a bit more elegant and simple; for any adventurous readers the Edit -> Patch program menu and a good x86 opcode reference (like <u>http://ref.x86asm.net/coder32.html</u>) should be more than enough to try out the other method.

Selecting the whole memory range by dragging the mouse is a bit boring and can sometimes deselect the selected code on its own. We'll use the Edit -> Begin selection (shortcut Alt + 1) command. Position the cursor just before the final jmp instruction, begin the selection, go to the loaders entry point (0x4023CE) and create a new function.

If everything goes correctly, the relevant fragment in the sidebar should change its color to blue:



And you should be able to tap Tab and view the simple decompilation pseudocode:

```
📃 Pseudocode-A 🗶
                                             🕅 Occurrences of: add 🗶
                                                                     0
                                                                                           A
     IDA View-A
                 X
                                                                          Hex View-1
                                                                                      X
                                                                                                Structures
                                                                                                            X
  1 void sub_4023CE()
  2 {
  3
     int v0; // ecx
     char *v1; // eax
   4
     void (*v2)(void); // eax
  5
  6
     int i; // ecx
  7
•
  8
     v0 = 21564845;
  9
     do
 10
     {
11
          _asm { finit }
• 12
       --v0:
 13
     3
     while ( v0 );
• 14
• 15
     v1 = (char *)&rtcCos;
 16
     do
• 17
        --v1;
• 18
     while ( *(_DWORD *)v1 != 9460301 );
     v2 = (void (*) (void)) (*((int (__stdcall **) (_DWORD, int, int, int))v1 + 1075)) (0, 40960, 4096, 64);
• 19
0 20 for ( i = 0; i != 22396; i = i - 40 + 44 )
0 21
        *(_DWORD *)((char *)v2 + i) = _mm_cvtsi64_si32(
 22
                                        _m_pxor(
 23
                                           _mm_cvtsi32_si64(*(_DWORD *)((char *)&loc_403F3F + i)),
                                           _mm_cvtsi32_si64(0x59DA0A67u)));
 24
25
      v2();
     JUMPOUT (0x403F34);
26
• 27 }
```

The logic is actually quite simple, but the code can get much more bloated and confusing in other samples.

```
void sub_4023CE()
{
 int v0; // ecx
  char *v1; // eax
 void (*v2)(void); // eax
  int i; // ecx
 v0 = 21564845;
  do
  {
     _asm { finit }
    --v0;
  }
 while ( v0 );
 v1 = &rtcCos;
  do
    --v1;
 while ( *v1 != "\x90ZM" );
 v2 = (*(v1 + 1075))(0, 40960, 4096, 64);
  for ( i = 0; i != 22396; i = i - 40 + 44 )
    *(v2 + i) = _mm_cvtsi64_si32(_m_pxor(_mm_cvtsi32_si64(*(&loc_403F3F + i)),
_mm_cvtsi32_si64(0x59DA0A67u)));
  v2();
  JUMPOUT(0x403F34);
}
```

Going step by step:

```
v0 = 21564845;
do
{
  __asm { finit }
  --v0;
}
```

This is a simple sleep snippet, nothing super interesting there.

```
v1 = &rtcCos;
do
    --v1;
while ( *v1 != "\x90ZM" );
v2 = (*(v1 + 1075))(4300, 0, 0, 40960, 4096, 64);
```

This is a bit more interesting, it fetches the pointer to rtcCos from MSVBVM60.DLL and then iterates downrange to find the images base address. It then uses that address to calculate a function address by adding 1075 to the pointer.

If we load the dll in IDA and navigate to the fetched address (0x732A0000 + 1075 * 4) we can learn that it's VirtualAlloc.

```
.idata:732A10CC ; LPVOID __stdcall VirtualAlloc(LPVOID lpAddress, SIZE_T dwSize, DWORD flAllocationType, DWORD flProtect)
.idata:732A10CC extrn VirtualAlloc:dword
.idata:732A10CC ; CODE XREF: sub_732A6702+2A+p
.idata:732A10CC ; sub_732A6702+43+p ...
```

So this is all just a sneaky a way of calling it without clearly indicating it in imports. Let's move on.

```
for ( i = 0; i != 22396; i = i - 40 + 44 )
    *(v2 + i) = _mm_cvtsi64_si32(_m_pxor(_mm_cvtsi32_si64(*(&loc_403F3F + i)),
    _mm_cvtsi32_si64(0x59DA0A67u)));
```

This part copies 22396 bytes from $0 \times 403F3F$ into the newly allocated buffer dexoring it with the constant $0 \times 59DA0A67$ in the process.

We can get the decrypted core without debugging the binary using a short Python script:

```
import struct
from malduck import xor
data = xor(key=struct.pack("<I", 0x59DA0A67), data=get_bytes(0x403F3F, 22396))
with open("decrypted.bin", "wb") as f:
    f.write(data)</pre>
```

And finally, the program jumps into the newly copied buffer.

v2();

Tune in next time to the second part, where we'll describe some of the CloudEyE's functions and discuss how we can extract the download URLs and the encryption key from unpacked samples automatically using <u>Malduck</u>.