

# Malware Crypters – the Deceptive First Layer

DECEMBER 2, 2015 | BY [HASHEREZADE](#)



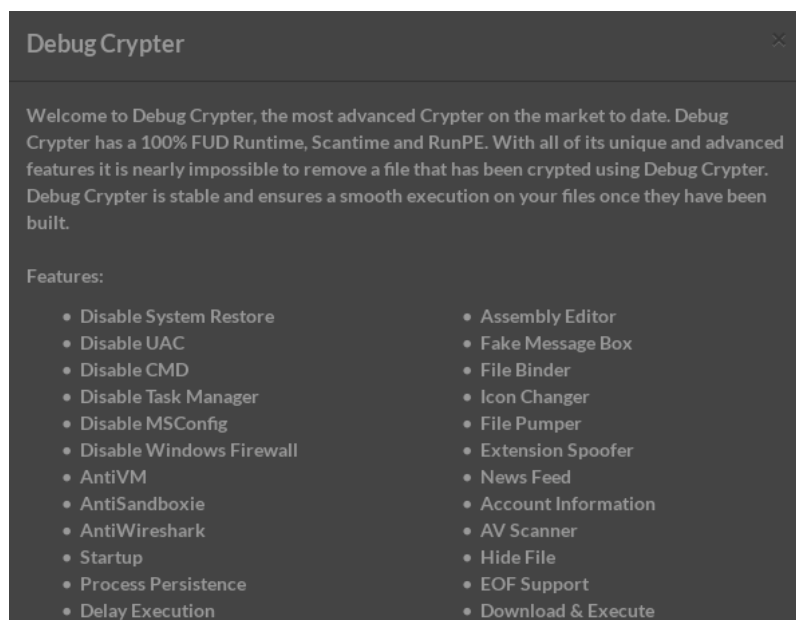
Recently, [two suspects were arrested](#) for selling **Cryptex Reborn** and other FUD tools (helping to install malware in a Fully UnDetectable way). Today, we will study some examples to make sure that everyone knows what this type of tools are and why they are dangerous. We will also present some example of identifying and unpacking a malware crypter.

## Crypters – what are they?

Most modern malware samples, in addition to built-in defensive techniques, are protected by some packer or crypter. A crypter's role is basically to be the first – and most complex – layer of defense for the malicious core. They try to deceive pattern-based or even behavior-based detection engines – often slowing down the analysis process by masquerading as a harmless program then unpacking/decrypting their malicious payload.

They may also add some icons and metadata that make the sample look like a legitimate product.

Underground crypters, created to defend malware against antivirus/anti-malware products, are sold in typical cybercriminal hangouts. Below, you can see examples of crypters being advertised on the black market and the tricks they use:



### Related Posts

[“INTUIT Security Warning” Emails Lead to Fake Browser Update Malware](#)

[Vonteera Adware Uses Certificates to Disable Anti-Malware](#)

[No money, but Pony! From a mail to a trojan horse](#)

[Three Reasons Why Anti-Virus Alone No Longer Enough](#)

[A Technical Look At Dyreza](#)

### Tweets

[Follow @Malwarebytes](#)

**Re/code** @Recode  
Mossberg: An encryption "backdoor" is a bad idea [on.recode.net/1PZW3Hy](#) by [@waltmossberg](#) [pic.twitter.com/7BY2iFL5WD](#)  
Retweeted by Malwarebytes  
Show Photo

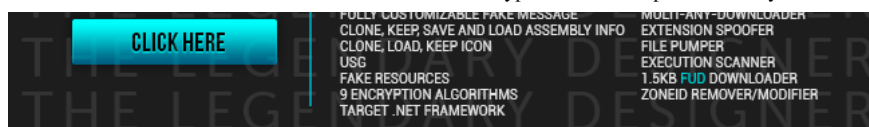
**SCMagazine** @SCMagazine  
Intuit again in hackers crosshairs, this time with phishing scam | [ow.ly/VoUst](#)  
Retweeted by Malwarebytes  
Show Summary

**The Verge** @verge  
After privacy ruling, Facebook now requires Belgium users to log in to view pages [theverge.com/2015/12/2/9838...](#) [pic.twitter.com/wl0HFqn9gO](#)  
Retweeted by Malwarebytes  
Show Photo

**Mercury News** @mercnews  
Google accused of invading students' privacy [bayareane.ws/1XFbm7w](#)  
Retweeted by Malwarebytes  
Show Summary

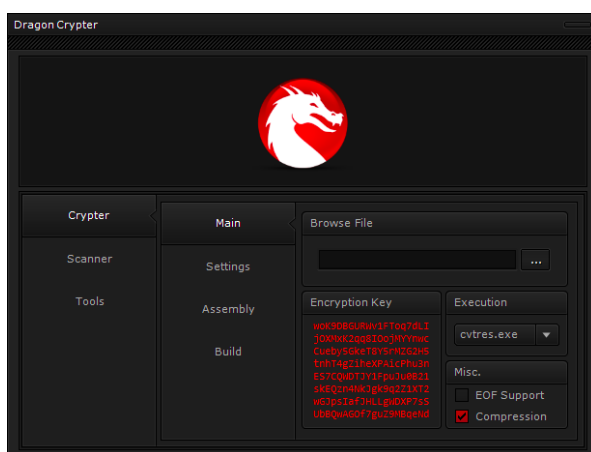
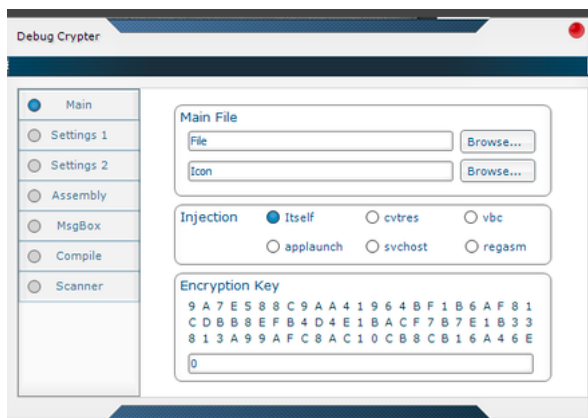
**Malwarebytes** @Malwarebytes  
Come to #RSA on us! Use this code to register for complimentary Exhibit Hall Only Pass: XEMWREB-rsaconference.com/events/us16/re...  
Expand

**Malwarebytes** @Malwarebytes  
Malware Crypters – the Deceptive First Layer | Malwarebytes Unpacked [blog.malwarebytes.org/development/20... via @hasherezade](#)



Expand

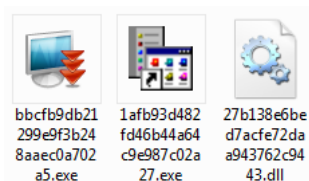
These products are designed to cater to simple criminals, those who do not need (or want) a deep technical knowledge. That's why authors provide a GUI to configure all the options in a very easy way. For example, it allows the configuration of the encryption method and key as well as where the payload should be injected.



As you can see, a crypter is a completely independent module. Cybercriminals can use it to protect *any* malware that they want to deliver. That's why knowing the crypter that is used does not help in identifying the malware sample. As an example, I would like to present you several different malware samples packed by the same/similar crypter.

## Analyzed samples

- [27b138e6bed7acf72daa943762c9443](#) – a DLL delivered by [Magnitude Exploit Kit](#) (will be referred as: **Magnitude.dll**)
  - carrying payload: [d890bd08180d69ee6ee5f7658be33030](#)
- [bbcfb9db21299e9f3b248aaec0a702a5](#) – an executable captured under the name: **makta.exe**
  - carrying payload: [3cf25fa56e8e8ecec90d8f2e8f123e8](#)
- [1afb93d482fd46b44a64c9e987c02a27](#) – an executable delivered by [Blackhole Exploit Kit](#) (will be referred as: **blackhole.exe**)
  - carrying payload: [5a58395fda49c8f3f4571a007cf02f4d](#)



## Identifying similarities

Before we start unpacking, let's have a look at similarities in the code that made me to believe that the above three samples (captured in different distribution campaigns) are all packed by the same tool.

Tracing the flow of execution, we notice similarities. At the beginning of execution, all of the samples make some meaningless API calls (i.e. trying to read some random keys from the registry). Then, they call a function to allocate memory (VirtualAlloc or VirtualAllocEx). They unpack something into this memory and redirect execution there. After

some time, execution comes back to the memory space of the original image. However, it now executes code that was not present before (the code images have been overwritten).

We can guess that all of the samples use the RunPE technique to overwrite the image of the original file with the payload. It all happens with the shellcode that is first unpacked into allocated memory. Let's set a breakpoint at VirtualAlloc/VirtualAllocEx and follow execution to see what is written into this newly allocated memory. Unpacking usually includes two stages: Some encrypted content is copied from the original image then stage 1 decryption is applied. After this, some of the shellcode is revealed. This same shellcode is responsible for decrypting the actual payload—this is now stage 2 decryption—and loading it into memory.

This is how the content unpacked to the allocated memory looks for each respective samples (after the stage 1 decryption):

Magnitude:

C "G.P.U" - main thread, module Magnitud		
10001A16	• PUSH ECX	
10001A17	• MOV EDX,DWORD PTR DS:[0x1003C0E4]	
10001A1D	• PUSH EDX	
10001A1E	• CALL Magnitud.10001120	
10001A23	• ADD ESP,0x8	
10001A25	• MOV BYTE PTR SS:[EBP-0x302],0x5C	
10001A2D	• MOV ECX,ECX	
10001A2F	• JMP SHORT Magnitud.10001A37	
10001A31	• LEA ESI,DWORD PTR DS:[EAX+0x93BB25A]	
ESP=0012FC5C		
Address	Hex dump ASCII	
00330000	47 65 74 50 72 6F 63 41 64 64 72 65 73 73 00 00	GetProcAddress...
00330010	00 56 69 72 74 75 61 6C 41 6C 6C 6F 63 00 00 00	VirtualAlloc...
00330020	00 00 56 69 72 74 75 61 6C 46 72 65 65 00 00 00	VirtualFree...
00330030	00 00 00 55 6D 61 70 56 69 65 77 4F 66 46 69 6C	UnmapViewOfFile
00330040	6C 65 00 00 56 69 72 74 75 61 6C 50 72 6F 74 65	le..VirtualProte
00330050	63 74 00 00 00 4C 6F 61 64 4C 69 62 72 61 72 79	ct...LoadLibrary
00330060	45 78 41 00 00 00 47 65 74 4D 6F 64 75 6C 65 48	ExA...GetModuleLeH
00330070	61 6E 64 6C 65 41 00 43 72 65 61 74 65 46 63 6C	andleA.CreateFile
00330080	65 41 00 00 00 00 00 00 53 65 74 46 69 6C 65 50	eA.....SetFileP
00330090	6F 69 6E 74 65 72 00 00 57 72 69 74 65 46 69	ointer...WriteFi
003300A0	6C 65 00 00 00 00 00 00 43 6C 6F 73 65 48	le.....CloseH
003300B0	61 6E 64 6C 65 00 00 00 00 00 47 65 74 54	andleA.....GetTe
003300C0	6D 70 50 61 74 68 41 00 00 00 00 6C 73 74 72	mpPathA.....lstr
003300D0	6C 65 6E 41 00 00 00 00 00 00 00 6C 73 74	lenA.....lstr
003300E0	72 63 61 74 41 00 00 00 00 00 00 00 00 00	rcatA.....
003300F0	E9 03 00 00 50 06 03 00 00 00 00 00 00 00	Se0..pVE..W...W...
00330100	FE FB 00 00 31 03 00 00 E9 03 00 00 29 04 00 00	u...i...0...0...0...
00330110	E9 03 00 00 E9 03 00 00 E9 03 00 00 E9 03 00 00	0...0...0...0...0...
00330120	E9 03 00 00 E9 03 00 00 E9 03 00 00 E9 03 00 00	0...0...0...0...0...
00330130	04 00 00 00 E7 10 0A 0E E9 AF 09 CD C8 08 01 4C	...s...j...0...0...0...
00330140	AC 25 54 68 00 77 20 70 FB 6A 67 72 C8 68 20 63	C\Th..w p...j...h c
00330150	C8 69 6E 6F DD 23 62 65 09 76 75 6E 09 6D 6E 20	inoI#be.vun.mn
00330160	85 4A 53 20 C4 6A 64 65 FF 08 00 0A CD 03 00 00	aJS -jde...W...W...
00330170	E9 03 00 00 50 06 03 00 00 00 09 04 0E 01 F2 04 09 00	0...0...0...0...0...
00330180	FC 66 D6 F9 F9 34 09 F9 F0 66 D6 F9 E7 1E 37 F9	Afi"-4..-fi"sA7"
00330190	C9 66 D6 F9 D7 1E 0A F9 C7 1E 08 F9 C7 1E 08 F9	ffi"A..=fi"sA0"
003301A0	00 66 D6 F9 1B 6D 63 68 AC 66 D6 F9 E9 03 00 00	2fi+mhCfiU0..
003301B0	E9 03 00 00 39 41 00 00 1D 05 04 00 30 03 03 00	U...90..+...<...i...
003301C0	E9 03 00 00 E9 03 00 00 89 03 02 21 F2 04 0C 00	0...0...e...0...0...
003301D0	E9 59 01 00 E9 B9 00 00 E9 03 00 00 39 6D 01 00	UV0..0j...0...9m0..
003301E0	E9 13 00 00 E9 73 01 00 E9 03 00 00 10 E9 13 00 00	U...0...0...0...0...
003301F0	E9 03 00 00 E9 03 00 00 E9 03 00 00 E9 03 00 00	U...0...0...0...0...
00330200	E9 03 00 00 E9 53 02 00 E9 FF FF FF E9 03 00 00	0...0...0...0...0...
00330210	EB 03 40 00 E9 03 10 00 E9 13 00 00 E9 03 10 00	0...0...0...U...0...0...
00330220	E9 13 00 00 E9 03 00 00 09 03 00 00 59 74 01 00	U...0...0...0...0...
00330230	0F 00 00 00 E9 03 00 00 E9 03 00 00 E9 03 00 00	W...0...0...0...0...
00330240	E9 03 00 00 E9 03 00 00 E9 03 00 00 E9 03 00 00	0...0...0...0...0...
00330250	E9 03 00 00 E9 43 02 00 85 00 00 00 E9 03 00 00	U...0...0...0...0...
00330260	E9 03 00 00 E9 03 00 00 E9 03 00 00 E9 03 00 00	0...0...0...0...0...

Makta :

C *G.P.U* - main thread, module bbcb9db		
0041DB45	. PUSH ECX	
0041DB4A	. MOV EDX,DMWORD PTR DS:[0x426024]	
0041DB4B	. PUSH EDX	
0041DB4B	. CALL bbcb9db.0041DFF0	
0041DB50	. ADD ESP,0x8	
0041DB53	. MOV BYTE PTR SS:[EBP-0x34F],0x78	
0041DB5A	. SHL EAX,0x0	
0041DB5D	> PUSH bbcb9db.0041DB64	
0041DB62	. RETN	
0041DB63	. XCHG EAX,ESI	
ESP=0012FBB0		
Address Hex dump ASCII		
00250020	00 00 56 69 72 74 75 61 6C 46 72 65 65 00 00 00	VirtualFree...
00250030	00 00 00 55 6D 61 70 56 69 65 77 4F 66 46 69	UnmapViewOfFile
00250040	6C 65 00 00 56 69 72 74 75 61 6C 50 72 6F 74 65	le..VirtualProte
00250050	63 74 00 00 00 4C 6F 61 64 4C 69 62 72 61 72 79	ct...LoadLibrary
00250060	45 78 41 00 00 00 47 65 74 4D 6F 64 75 6C 65 48	ExA...GetModuleLeH
00250070	61 6E 64 6C 65 41 00 43 72 65 61 74 65 46 63 6C	andleA.CreateFile
00250080	65 41 00 00 00 00 00 00 53 65 74 46 69 6C 65 50	eA.....SetFileP
00250090	6F 69 6E 74 65 72 00 00 57 72 69 74 65 46 69	ointer...WriteFi
002500A0	6C 65 00 00 00 00 00 00 43 6C 6F 73 65 48	le.....CloseH
002500B0	61 6E 64 6C 65 00 00 00 00 00 47 65 74 54	andleA.....GetTe
002500C0	6D 70 50 61 74 68 41 00 00 00 00 6C 73 74 72	mpPathA.....lstr
002500D0	6C 65 6E 41 00 00 00 00 00 00 00 6C 73 74	lenA.....lstr
002500E0	72 63 61 74 41 00 00 00 00 00 00 00 00 00	rcatA.....
002500F0	E9 03 00 00 50 06 03 00 00 00 09 04 0E 01 F2 04 09 00	Se0..pVE..W...W...
00250100	FE FB 00 00 31 03 00 00 E9 03 00 00 29 04 00 00	u...i...0...0...0...
00250110	E9 03 00 00 E9 03 00 00 E9 03 00 00 E9 03 00 00	0...0...0...0...0...
00250120	E9 03 00 00 E9 03 00 00 E9 03 00 00 E9 03 00 00	0...0...0...0...0...
00250130	04 00 00 00 E7 10 0A 0E E9 AF 09 CD C8 08 01 4C	...s...j...0...0...0...
00250140	AC 25 54 68 00 77 20 70 FB 6A 67 72 C8 68 20 63	C\Th..w p...j...h c
00250150	C8 69 6E 6F DD 23 62 65 09 76 75 6E 09 6D 6E 20	inoI#be.vun.mn
00250160	85 4A 53 20 C4 6A 64 65 FF 08 00 0A CD 03 00 00	aJS -jde...W...W...
00250170	E9 03 00 00 50 06 03 00 00 00 09 04 0E 01 F2 04 09 00	0...0...0...0...0...
00250180	E9 03 00 00 E9 03 00 00 09 04 0E 01 F2 04 09 00	0...0...0...0...0...
00250190	E9 D3 00 00 E9 73 00 00 E9 03 00 00 36 D5 00 00	0E...s...0...eA...
002501A0	E9 03 00 00 E9 03 00 00 09 03 00 00 36 D5 00 00	0...0...0...0...0...
002501B0	E9 03 00 00 E9 03 00 00 09 03 00 00 36 D5 00 00	U...0...0...0...0...
002501C0	E9 03 00 00 E9 53 01 00 E9 13 00 00 E9 03 00 00	U...0...0...U...0...0...
002501D0	EB 03 00 00 E9 03 10 00 E9 13 00 00 E9 03 10 00	U...0...0...U...0...0...
002501E0	E9 13 00 00 E9 03 00 00 09 03 00 00 E9 03 00 00	U...0...0...0...0...
002501F0	E9 03 00 00 E9 03 00 00 09 03 00 00 E9 03 00 00	U...0...0...0...0...
00250200	E9 03 00 00 E9 03 00 00 E9 03 00 00 E9 03 00 00	U...0...0...0...0...
00250210	E9 03 00 00 E9 33 01 00 5D 08 00 00 E9 03 00 00	U...0...0...0...0...
00250220	E9 03 00 00 E9 03 00 00 09 03 00 00 E9 03 00 00	U...0...0...0...0...
00250230	E9 03 00 00 E9 03 00 00 E9 03 00 00 E9 03 00 00	U...0...0...0...0...
00250240	E9 03 00 00 E9 03 00 00 E9 03 00 00 E9 03 00 00	U...0...0...0...0...
00250250	19 06 00 00 E9 03 00 00 E9 03 00 00 E9 03 00 00	+...0...0...0...0...
00250260	E9 03 00 00 E9 03 00 00 E9 03 00 00 07 6F 65 78	U...0...0...0...ioeK
00250270	25 03 00 00 E9 C4 00 00 E9 13 00 00 E9 03 00 00	U...0...0...U...0...

Blackhole:

*G.P.U* - main thread, module 1afb93d4	
00401848	PUSH ECX
00401852	MOV EDX,DWORD PTR DS:[0x41E6AC]
00401852	PUSH EDX
00401853	CALL 1afb93d4.004019D0
00401858	ADD ESP,0x8
0040185B	MOV DWORD PTR DS:[0x41E67C],1afb93d4.004014F0
00401865	MOV EAX,DWORD PTR DS:[0x41E6AC]
00401868	ADD ESP,0x4



0040185B	•	HUO	EAX, 0x191B9
0040186F	•	MOV	DWORD PTR DS:[0x41E6B4], EAX
00401874	•	MOV	EDX, EDX
←			
ESP=0012FF80			
Address	Hex dump	ASCII	
00260000	47 65 74 50 72 6F 63 41 64 64 72 65 73 73 00 00	GetProcAddress..	
00260010	00 55 69 74 75 61 6C 61 6C 6F 6F 00 00 00 00	VirtualAlloc...	
00260020	00 00 56 69 72 74 75 61 6C 46 72 65 65 00 00 00	..VirtualFree...	
00260030	00 00 55 6E 60 61 70 56 69 65 77 4F 66 46 69	..UnmapViewOfFi	
00260040	6C 65 00 00 56 69 72 74 75 61 6C 50 72 6F 74 65	le..VirtualProte	
00260050	63 74 00 00 00 4C 6F 61 64 4C 69 62 72 61 72 79	ct...LoadLibrary	
00260060	45 78 41 00 00 00 47 65 74 40 6F 64 75 6C 65 48	ExA...GetModuleH	
00260070	61 6E 64 6C 65 41 00 43 72 65 61 74 65 46 69 6C	andleA.CreateFil	
00260080	65 41 00 00 00 00 00 00 53 65 74 46 69 6C 65 50	eh.....SetFileP	
00260090	6F 79 6E 74 65 72 00 00 57 72 79 74 65 46 69	ointer...WriteFi	
002600A0	6C 65 00 00 00 00 00 00 00 00 43 6C 6F 73 65 48	le.....CloseH	
002600B0	61 6E 64 6C 65 00 00 00 00 00 00 47 65 74 54 65	andle.....GetTe	
002600C0	60 70 50 61 74 68 41 00 00 00 00 00 6C 73 74 72	npPathA.....lstr	
002600D0	6C 65 6F 61 00 00 00 00 00 00 00 6C 73 74	lenA...s...lstr	
002600E0	72 63 61 74 41 00 00 00 00 00 00 00 00 00 00	rcatA.....	
002600F0	00 86 01 00 04 59 90 00 0A 03 00 00 00 00 00	ac0..AVE..i...y0..	
00260100	FE FB 00 00 31 03 00 00 00 00 00 29 04 00 00	u...i...0...i...0...	
00260110	E9 03 00 00 00 00 00 00 00 00 00 00 00 00 00	U...U...U...U...	
00260120	E9 03 00 00 00 00 00 00 00 00 00 00 00 00 00	U...U...U...U...	
00260130	B9 04 00 00 00 00 00 00 00 00 00 00 00 00 00	U...U...U...U...	
00260140	HC 25 54 68 00 77 20 70 FB 6A 67 72 C8 68 20 63	CxTh..w p4jgr...h o	
00260150	03 69 6E 6F 00 23 62 65 09 76 75 6E 09 6D 6E 28	"incTabc..un..nn	
00260160	HE 4A 53 20 C4 6A 64 65 FF 08 00 0A CD 03 00 00	aJS...ide...=...	
00260170	E9 03 00 00 00 00 00 00 00 00 00 00 00 00 00	U...+...T00'T0	
00260180	42 7E 54 E2 92 7E 55 E2 76 7E 54 E2 3B 07 C7 E2	o'T0('U0U'T0;:ao	
00260190	29 7F 54 E2 29 7F 54 E2 29 7F 54 E2 29 7F 54 E2	2eT0w...0-0T0'0'0	
002601A0	10 7F 54 E2 29 7F 54 E2 29 7F 54 E2 29 7F 54 E2	+aT0'0f0'0T0;:mch	
002601B0	42 7F 54 E2 E9 03 00 00 00 00 00 00 00 00 00 00	BoT0U...U...U...	
002601C0	E9 03 00 00 00 00 00 00 00 00 00 00 00 00 00	U...+...+...SfAS	
002601D0	E9 03 00 00 00 00 00 00 00 00 00 00 00 00 00	U...+...+...SfAS	
002601E0	E9 11 00 00 00 00 00 00 00 00 00 00 00 00 00	U...U...U...U...	
002601F0	E9 13 00 00 00 00 00 00 00 00 00 00 00 00 00	U...U...U...U...	
00260200	E9 05 00 00 00 00 00 00 00 00 00 00 00 00 00	U...U...U...U...	
00260210	E9 05 00 00 00 00 00 00 00 00 00 00 00 00 00	U...U...U...U...	
00260220	EB 03 40 81 E9 03 00 00 00 00 00 00 00 00 00	U...U...U...U...	
00260230	E9 13 00 00 00 00 00 00 00 00 00 00 00 00 00	U...U...U...U...	
00260240	E9 03 00 00 00 00 00 00 00 00 00 00 00 00 00	U...U...U...U...	
00260250	E9 03 00 00 00 00 00 00 00 00 00 00 00 00 00	U...U...U...U...	
00260260	E9 03 00 00 00 00 00 00 00 00 00 00 00 00 00	U...U...U...U...	
00260270	E9 03 00 00 00 00 00 00 00 00 00 00 00 00 00	U...U...U...U...	
00260280	E9 03 00 00 00 00 00 00 00 00 00 00 00 00 00	U...U...U...U...	
00260290	E9 03 00 00 00 00 00 00 00 00 00 00 00 00 00	U...U...U...U...	
002602A0	E9 03 00 00 00 00 00 00 00 00 00 00 00 00 00	U...U...U...U...	
002602B0	E9 03 00 00 00 00 00 00 00 00 00 00 00 00 00	U...U...U...U...	
002602C0	E9 03 00 00 00 00 00 00 00 00 00 00 00 00 00	U...U...U...U...	
002602D0	E9 FF FF FF E9 03 00 00 00 00 00 00 00 00 00	U...U...U...U...	

The above content consists of the same elements in the same order. At the beginning, we can see a list of functions to be loaded. Next, we see an encrypted payload (independent PE file). Finally, we see the shellcode to be executed (loading the payload by the RunPE technique).

Below is the encrypted payload on the left and its decrypted version on the right:

Address	Hex dump	ASCII	
003300F4	A4 E9 90 00 EA 03 00 00	AVE..i...	
003300FC	ED 03 00 00 FE FB 00 00	Y...u...	
00330104	31 03 00 00 00 00 00 00	I...U...	
0033010C	29 04 00 00 00 00 00 00	I...U...	
00330114	E9 03 00 00 E9 03 00 00	U...U...	
0033011C	E9 03 00 00 E9 03 00 00	U...U...	
00330124	E9 03 00 00 E9 03 00 00	U...U...	
0033012C	E9 03 00 00 E9 03 00 00	U...U...	
00330134	E7 1A BA 0E E9 AF 09 CD	U...U...=...	
0033013C	C8 BB 01 4C AC 25 54 68	U...U...Th	
00330144	00 77 20 70 FB 6A 67 72	U...U...gr	
00330154	C9 69 6E 6F 00 23 62 65	U...U...no	
00330164	09 6D 6E 20 A5 4A 53 20	U...U...nn aJS	
0033016C	C4 6A 65 6F 00 00 00 00	U...U...=...	
00330174	50 06 B8 AA 0C 67 D6 F9	U...U...P+S..gi"	
0033017C	04 67 D6 F9 C6 66 D6 F9	U...U...+gi"Rfi"	
00330184	F9 24 05 F9 C6 66 D6 F9	U...U...A...fi"	
0033018C	E7 1E 37 F9 C6 66 D6 F9	U...U...A...fi"	
00330194	D7 1E 0A F9 C6 66 D6 F9	U...U...iA...fi"	
0033019C	C7 1E 08 F9 B0 66 D6 F9	U...U...A...2fi"	
003301A4	1B 6D 68 AA 66 D6 F9	U...U...mchCfi"	
003301AC	E9 03 00 00 E9 03 00 00	U...U...U...U...	
003301B4	39 41 00 00 10 05 04 00	U...U...9A...+i...	
003301BC	3C 33 B3 55 E9 03 00 00	U...U...<alU0...	
003301C4	E9 03 00 00 E9 03 00 00	U...U...U...U...	
003301CC	F2 04 00 00 E9 03 01 00	U...U...U...U...	
003301D4	E9 03 00 00 E9 03 00 00	U...U...U...U...	
003301DC	39 6D 01 00 E9 13 00 00	U...U...9n0.U...	
003301E4	E9 73 01 00 E9 03 00 00	U...U...10s0.U...	
003301EC	E9 03 00 00 E9 03 00 00	U...U...U...U...	
003301F4	EC 03 01 00 E9 03 00 00	U...U...y00.U...	
003301FC	EC 03 01 00 E9 03 00 00	U...U...y00.U...	
00330204	E9 03 00 00 E9 03 00 00	U...U...U...U...	
0033020C	E9 03 00 00 E9 03 00 00	U...U...U...U...	
00330214	E9 03 10 00 E9 13 00 00	U...U...U...U...	
0033021C	E9 03 10 00 E9 13 00 00	U...U...U...U...	
00330224	E9 03 00 00 D9 03 00 00	U...U...U...U...	
0033022C	74 01 00 CF 03 00 00	U...U...Vr0.P...	
00330234	E9 03 00 00 E9 03 00 00	U...U...U...U...	
0033023C	E9 03 00 00 E9 03 00 00	U...U...U...U...	
00330244	E9 03 00 00 E9 03 00 00	U...U...U...U...	
0033024C	E9 03 00 00 E9 03 00 00	U...U...U...U...	
00330254	E9 43 02 00 85 00 00 00	U...U...U...U...	
0033025C	E9 03 00 00 E9 03 00 00	U...U...U...U...	
00330264	E9 03 00 00 E9 03 00 00	U...U...U...U...	
0033026C	E9 03 00 00 E9 03 00 00	U...U...U...U...	

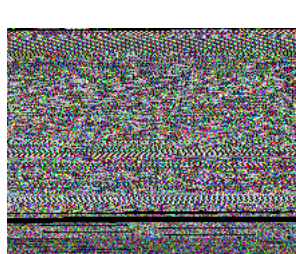
## Visual analysis

The decrypting procedure is heavily obfuscated, but by having memory dumps made before and after each stage of decryption, we can try to get some hints of what is going on by comparing the changes.

Visual analysis may help in discovering the algorithm by which the data is packed. I have decided to dump the allocated memory before each stage of decryption + the revealed payload (new PE file). You can see this stages on the first and second pictures in the row. At the third position, you can see the visualization of the dumped payload.

Similar patterns are present in all three files:

**Magnitude.dll** (encrypted, first stage decrypted, payload)

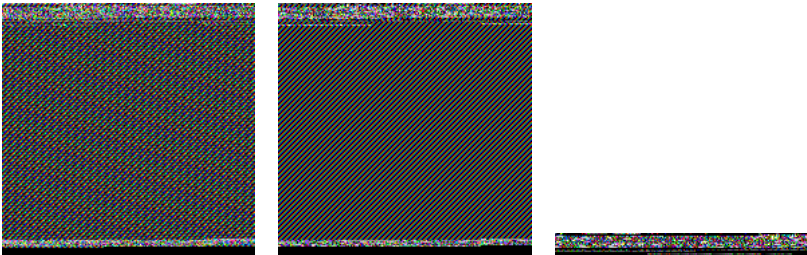




*makta.exe* (encrypted, first stage decrypted, payload)



*blackhole.exe* (encrypted, first stage decrypted, payload)



What information can we get from such a visual dump? Look at this last case:

The payload is tiny, that's why we can see a lot of padding between the encrypted payload (that is at the beginning of the allocated memory) and the shellcode (that is at the end). The padding allows us to discover the encryption pattern.

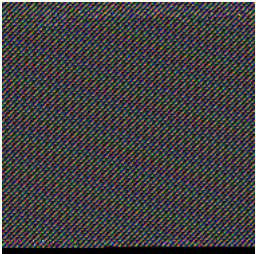
Looking at the regularities, we can guess that: the first stage, as well as the second stage, are both encrypted by XOR with some key (key length > 1). The key seems to be longer at the first stage and shorter at the second. Let's look inside the memory dump.

At first stage, the key is composed by some repetitive pattern:

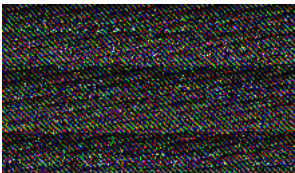
Address	Hex dump	ASCII
00348D20	0A 7F 00 00 0A 7F 00 00 FA 7E 00 00 FA 7E 00 00	.A...A...F...F...
00348D30	0A 7F 00 00 0A 7F 00 00 FA 7E 00 00 FA 7E 00 00	.A...A...F...F...
00348D40	CA 7E 00 00 CA 7E 00 00 BA 7E 00 00 BA 7E 00 00	..E...E...B...B...
00348D50	CA 7E 00 00 CA 7E 00 00 BA 7E 00 00 BA 7E 00 00	..E...E...B...B...
00348D60	8A 86 00 00 8A 86 00 00 7A 86 00 00 7A 86 00 00	..R...R...V...V...
00348D70	8A 86 00 00 8A 86 00 00 7A 86 00 00 7A 86 00 00	..R...R...V...V...
00348D80	4A 86 00 00 4A 86 00 00 3A 86 00 00 3A 86 00 00	..J...J...Z...Z...
00348D90	4A 86 00 00 4A 86 00 00 3A 86 00 00 3A 86 00 00	..J...J...Z...Z...
00348DA0	0A 86 00 00 0A 86 00 00 FA 85 00 00 FA 85 00 00	..A...A...a...a...
00348DB0	0A 86 00 00 0A 86 00 00 FA 85 00 00 FA 85 00 00	..A...A...a...a...
00348DC0	CA 85 00 00 CA 85 00 00 BA 85 00 00 BA 85 00 00	..E...E...B...B...
00348DD0	CA 85 00 00 CA 85 00 00 BA 85 00 00 BA 85 00 00	..E...E...B...B...
00348DE0	8A 85 00 00 8A 85 00 00 7A 85 00 00 7A 85 00 00	..R...R...V...V...
00348DF0	8A 85 00 00 8A 85 00 00 7A 85 00 00 7A 85 00 00	..R...R...V...V...
00348E00	4A 85 00 00 4A 85 00 00 3A 85 00 00 3A 85 00 00	..J...J...Z...Z...
00348E10	4A 85 00 00 4A 85 00 00 3A 85 00 00 3A 85 00 00	..J...J...Z...Z...
00348E20	0A 85 00 00 0A 85 00 00 FA 84 00 00 FA 84 00 00	..A...A...a...a...
00348E30	0A 85 00 00 0A 85 00 00 FA 84 00 00 FA 84 00 00	..A...A...a...a...

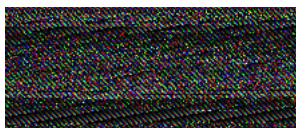
To verify if it is really XOR, we can do reverse XOR—input with output—and see if the result is a regular pattern. The experiment has given the following results:

Blackhole:

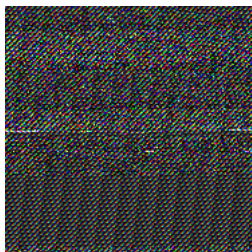


Magnitude:





Makta:



Looking at the visualization, we can guess that encryption is more than just plain XOR and that the key is probably modified during execution.

At the second stage, the visual pattern is denser, so it suggests that the key may be shorter.

## Unpacking

In each of the 3 files, the decoding functions are heavily obfuscated with lot of junk code and redundant API calls in between valuable instructions. Also, known tricks (i.e. PUSH-to-RET) are used in order to hide the real flow.

After deobfuscating it, we can see that in each case the algorithm is exactly the same—for each three files and for both stages (only parameters differ).

```

1  bool decode(DWORD *inbuf, //encrypted input
2      DWORD *outbuf, //buffer to store the output
3      size_t bufsize,
4      const DWORD key,
5      const size_t max_size = SIZE_MAX
6  )
7  {
8      if (inbuf == NULL || outbuf == NULL) return false;
9
10     for (size_t i = 0; i < bufsize; i++) {
11         DWORD val = inbuf[i];
12         DWORD step = i * sizeof(DWORD);
13         if (step >= max_size) {
14             outbuf[i] = val;
15             continue;
16         }
17         outbuf[i] = (val + step) ^ (key + step);
18     }
19     return true;
20 }

```

As we have guessed by visual analysis, it is based on an XOR operation, and the key is modified as the decoding progress.

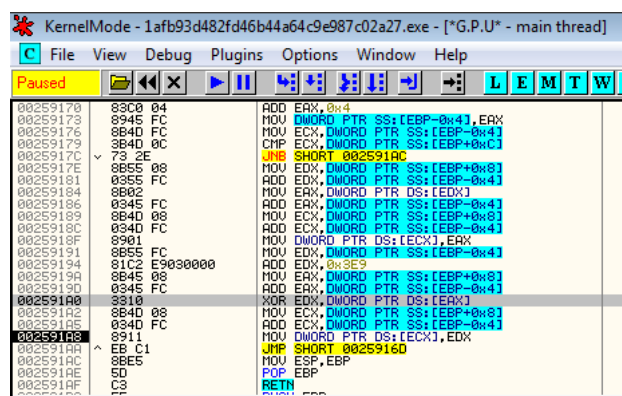
Used parameters:

stage#1

- **makta.exe**: key = 0x57FC
- **blackhole.exe**: key = 0x82A3, max\_size = 0x19400
- **Magnitude.dll**: key = 0x0A42

stage#2

- all 3 files: key = 0x03E9






## Writing Auto-unpacker

1. Find the encrypted chunks (by patterns) and glue them together
2. Find the XOR key (by XOR with expected output)
3. Use it to decrypt the memory fragment (stage#1)
4. Decrypt stage#2
5. Save the decrypted PE file (payload)

Unpacker in action:

## Conclusion

 OFFICIAL SECURITY BLOG

HomeAuthorsVideosScamsAbout UsArchives +Categories +

The described crypter seems to be popular nowadays. However, it's not any advanced tool. For example, there is no defense deployed against the debugger or virtual environment. The author puts a lot of effort in obfuscating code in order to hide the encryption method but looking at visualization, we can recognize that it is an XOR-based encryption and not even implemented well (encrypting DWORD size unit with WORD size key leads to visible artifacts). This is why we could easily write a static unpacker for the future use.

Be the first to comment.

DISQUS

FOLLOW MALWAREBYTES

[f](#)
[twitter](#)
[reddit](#)
[g+](#)
[YouTube](#)
[t](#)
[instagram](#)

Products	Support	Company	Partners
<a href="#">Malwarebytes Anti-Malware Free</a>	<a href="#">Consumer Support</a>	<a href="#">About</a>	<a href="#">Affiliates</a>
<a href="#">Malwarebytes Anti-Malware Premium</a>	<a href="#">Business Support</a>	<a href="#">Security Blog</a>	<a href="#">Corporate Partnerships</a>
<a href="#">Malwarebytes Anti-Malware Mobile</a>	<a href="#">Release History</a>	<a href="#">Forums</a>	<a href="#">Resellers</a>
<a href="#">Malwarebytes Anti-Exploit</a>		<a href="#">Management</a>	<a href="#">Reseller Login</a>
<a href="#">Malwarebytes Anti-Exploit Premium</a>		<a href="#">Careers</a>	
<a href="#">Malwarebytes Secure Backup</a>		<a href="#">Press Center</a>	
<a href="#">Malwarebytes Anti-Malware Remediation Tool</a>		<a href="#">Contact Us</a>	
<a href="#">Malwarebytes Anti-Malware for Business</a>		<a href="#">Awards/Testimonials</a>	
<a href="#">Malwarebytes Anti-Exploit for Business</a>			
<a href="#">Malwarebytes Endpoint Security</a>			
<a href="#">Malwarebytes Techbench</a>			

© 2015 MALWAREBYTES CORPORATION | [PRIVACY POLICY](#) | [TERMS OF SERVICE](#) | [EULA](#)