

## Analyzing XOR encrypted payload with xdbg

In this section, we will be analyzing, the .exe file, in which the payload is encrypted with XOR.

We must follow the same procedure we followed in the base64 case, so first run the program the cmd, then attach the program in xdbg, then from then analyze the program.

So, in the dump1, we can see the encrypted xor payload:

Dump 1	Dump 2	Dump 3	Dump 4	Dump 5	Watch 1	[x=] Locals	Struct
Address	Hex	Hex	Hex	Hex	Hex	Hex	ASCII
0000006EE30FFB30	CD 7A 80 D0	C5 DE F7 38	39 41 03 12	70 62 61 65	Iz°DAb=89A..pbae		
0000006EE30FFB40	63 7E 06 EA	5C 09 C9 11	51 7A B8 66	2D 7E BC 6A	c~.è\..É.Qz.f~%4j		
0000006EE30FFB50	19 09 C9 31	61 7A 3C 83	7F 7C 7A 09	F0 09 73 83	..É1az<.. z.ð.s.		
0000006EE30FFB60	9D 0E 52 48	37 1A 17 79	F8 88 4F 02	30 F3 D1 D9	..RH7..yø.O.0óNÜ		
0000006EE30FFB70	67 77 66 70	B2 13 62 C8	73 0E 78 35	E5 8D 87 80	gwfP°.bEs.{5à%°		
0000006EE30FFB80	39 41 42 08	B4 F2 47 53	7D 37 E7 68	B2 09 5A 07	9AB.òGS}7çh°.Z.		
0000006EE30FFB90	BA 72 13 7D	34 E6 D4 6E	71 BE 8B 02	BA 06 BB 7C	°r.}4æOnq%..°.»		
0000006EE30FFBA0	34 E0 7A 09	F0 09 73 83	9D 73 F2 FD	38 77 36 F9	4az.ð.s...søy8w6u		
0000006EE30FFBB0	01 A1 37 B2	7D 31 7F 10	3D 73 0E E9	4C 99 1A 07	.i7°}1...s.èL...		
0000006EE30FFBC0	BA 72 17 7D	34 E6 51 79	B2 4D 0A 07	BA 72 2F 7D	°r.}4æQY°M...°r/}		
0000006EE30FFBD0	34 E6 76 B3	3D C9 0A 42	E1 73 6B 75	6D 68 6E 62	4æv°=È.Baskumhnb		
0000006EE30FFBE0	78 19 03 1A	70 68 78 B7	D9 16 76 6A	C6 A1 1A 02	x...ph{.Ü.vjÆi...		
0000006EE30FFBF0	68 68 78 BF	27 DF 60 C7	C6 BE 1F 08	88 33 33 34	hh{ç'B.CA%...334		
0000006EE30FFC00	35 36 37 38	39 09 CF CE	30 33 33 34	74 8C 06 B3	56789..iI0334t...*		
0000006EE30FFC10	56 C6 8D 96	8A D2 2E 1E	3F 77 8D 9E	AC FC DF BC	VÆ%..ð...?w...-Üß%		
0000006EE30FFC20	E4 7A 80 F0	1D 0A 31 44	33 C1 B9 A3	44 37 88 73	äz°ð...1D3A'£D7.s		
0000006EE30FFC30	26 44 58 52	39 18 03 CA	EB CD E6 5A	5A 42 52 48	&DXR9...ÈætaZZBRH		
0000006EE30FFC40	58 25 6C 26	49 57 33 00	00 00 00 00	00 00 00 00	X%7&IW3.....		
0000006EE30FFC50	31 32 33 34	35 36 37 38	39 41 42 43	00 7F 00 00	123456789ABC....		
0000006EE30FFC60	00 00 00 00	00 00 00 00	24 16 4A B3	F7 7F 00 00	.....\$.J°*...<		

In dump2, we can see VirtualAlloc has allocated the space:

Dump 1	Dump 2	Dump 3	Dump 4	Dump 5	Watch 1	[x=] Locals	Struct
Address	Hex	Hex	Hex	Hex	Hex	Hex	ASCII
000001FA9F410000	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
000001FA9F410010	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
000001FA9F410020	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
000001FA9F410030	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
000001FA9F410040	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
000001FA9F410050	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
000001FA9F410060	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
000001FA9F410070	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
000001FA9F410080	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
000001FA9F410090	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
000001FA9F4100A0	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
000001FA9F4100B0	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
000001FA9F4100C0	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
000001FA9F4100D0	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
000001FA9F4100E0	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
000001FA9F4100F0	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
000001FA9F410100	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
000001FA9F410110	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
000001FA9F410120	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
000001FA9F410130	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....

Command: Commands are comma separated (like assembly instructions): mov eax, ebx

Here when we follow the address, which was there in dump1, if we follow that address in the memory map, we can see a thread value(stack)

Address	Size	Party	Info	Content	Type	Protection	Initial
000000007FFE0000	0000000000001000	User	KUSER_SHARED_DATA		PRV	-R---	-R---
000000007FFE8000	0000000000001000	User			PRV	-R---	-R---
00000000EE2E0000	0000000000008000	User	Reserved		PRV	-RW--	-RW--
00000000EE2E8000	0000000000003000	User	PEB, TEB (2464)		PRV	-RW--	-RW--
00000000EE2ED000	000000000000125000	User	Reserved (0000006EE2E00000)		PRV	-RW--	-RW--
00000000EE300000	000000000000FA000	User	Reserved		PRV	-RW--	-RW--
00000000EE30FA000	0000000000006000	User	Stack (2464)		PRV	-RW-G	-RW--

Even when following it in the threads, we can see that the same number pops up: “2464”

Number	ID	Entry	TEB	RIP	Suspend Count	Priority	Wait Reason	Last Error	User Time	Kernel Time	Creation Time	CPU cycles	Name
1	1040	00007FFBD3C2B320	00000000EE2E0000	00007FFBD310A4	0	Normal	Executive	00000000	00:00:00.0000000	00:00:00.0000000	14:57:32.2891051	234828	Main Thread
Main	2464	0000000000000000	00000000EE2E0000	00007FFBD30DC4	0	Normal	Executive	00000000	00:00:00.0000000	00:00:00.0937500	14:52:32.2891674	9137800	Main Thread

So, now just press enter in the cmd, and we will see that in dump2, the ASCII column will be flooded with decrypted payload.

As we can see here, as soon as we hit enter, we can see the decrypted payload.

And we can see that even the protection got changed:

000001FA9F410000	0000000000001000	User		PRV	EX---	-RW--
------------------	------------------	------	--	-----	-------	-------

Now if we further run the program, it will exit, and Notepad will be opened.