Reverse Engineering Redosdru String Decryption

By Andrew Petrus

Published: 2024-06-17 · Archived: 2026-04-05 13:40:07 UTC

Redosdru is malware primarily functioning as a downloader, commonly associated with APT27/Iron Tiger.

Reported by Microsoft as early as July 2009

Surprisingly, for its age, I couldn’t find much information about it, which made dissecting it even more appealing. I wanted to share the decryption process of the C2 URL, which I found quite interesting.

Press enter or click to view image in full size

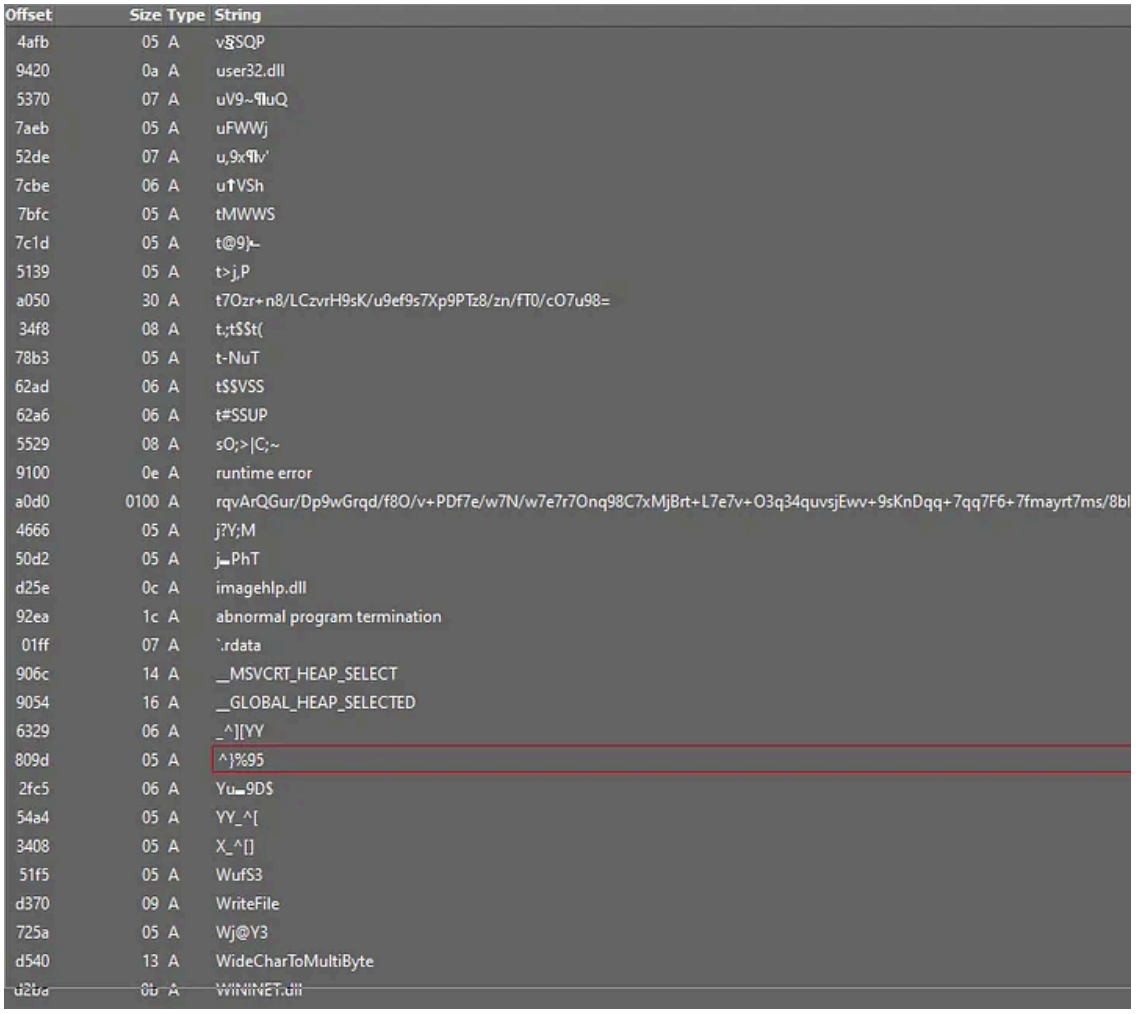
Date (UTC)	SHA256 hash	Type	Signature	Tags	Reporter	DL
2024-06-12 19:20	506c946ecc0877b13de...	exe	Redosdru	exe Redosdru	sentotayam	
2024-06-12 19:15	064497427357409fcd63...	exe	Redosdru	exe Redosdru	sentotayam	
2024-06-12 18:08	04e5c9467245df7b1beb...	exe	Redosdru	exe Redosdru	sentotayam	
2023-12-21 05:09	79c061e457eae6fe5e1e...	exe	Redosdru	exe Redosdru	adm1n_usa32	
2023-06-06 04:52	523461b6e1b7beb0ea5...	exe	Redosdru	32 exe Redosdru	zbetcheckin	
2023-06-06 04:09	41fe567d55eb7815d15f...	exe	Redosdru	32 exe Redosdru	zbetcheckin	
2023-01-19 08:30	5a74312dcaeb621a2c4...	exe	Redosdru	32 exe Redosdru	zbetcheckin	
2022-11-06 00:34	1575c84f1a5579e58584...	zip	Redosdru	Redosdru zip	DesdinovaOsint	
2022-07-10 07:58	161d412d6102172d9a9...	exe	Redosdru	exe Redosdru	zbetcheckin	
2022-04-03 01:45	4def25bfde1457ef315eb...	exe	Redosdru	exe Redosdru	abuse_ch	
2022-01-31 07:32	7dfd68fb1e4b9739c080...	exe	Redosdru	exe Redosdru	abuse_ch	
2022-01-01 04:20	ae5f6a5007c02c48f4bb...	exe	Redosdru	exe Redosdru	abuse_ch	
2021-11-26 18:00	5eb5ed756b14dd3e689...	exe	Redosdru	exe Redosdru	abuse_ch	
2021-10-03 03:20	69f2373b506710775bbd...	exe	Redosdru	exe Redosdru	r3dbU7z	
2021-09-08 21:11	a1619735fbaec9312f45...	exe	Redosdru	32 exe Redosdru	zbetcheckin	
2021-08-13 02:28	323ea92408f9dfb0598c...	exe	Redosdru	32 exe Redosdru	zbetcheckin	
2021-08-07 19:42	799866f1237d484f5e55...	exe	Redosdru	32 exe Redosdru	zbetcheckin	
2021-08-01 16:37	7e6b769afc67e5e76904...	exe	Redosdru	32 exe Redosdru	zbetcheckin	
2021-07-29 15:06	06c56274fc1db5ddff596...	exe	Redosdru	exe Redosdru	abuse_ch	
2021-07-02 06:27	7e2d72b0a1edfdcbff327...	exe	Redosdru	32 exe Redosdru	zbetcheckin	
2021-03-17 00:07	81d6978af7320d1e0631...	exe	Redosdru	exe Redosdru	r3dbU7z	
2021-03-07 22:26	cee26a7cf2c48b461b3e...	exe	Redosdru	exe Redosdru	r3dbU7z	
2021-02-22 19:19	b19f2c2eec099ce61163...	exe	Redosdru	exe Redosdru	abuse_ch	
2021-02-12 08:56	b65776ee765163d7668...	dll	Redosdru	dll Redosdru	r3dbU7z	
2021-02-11 08:39	420c4a968b184211b9e...	exe	Redosdru	Downloader exe Redosdru	r3dbU7z	
2020-12-19 08:34	1305139a7ad70ad2c9af...	exe	Redosdru	Redosdru	SecuriteInfoCom	
2020-12-18 09:18	ab6c0f4ae9d79e5dca90...	exe	Redosdru	exe Redosdru	abuse_ch	
2020-09-01 09:27	97896c9917071080e38...	exe	Redosdru	Redosdru	JAMESWT_MHT	

Low amount of submissions on Malware Bazaar considering age

Finding the C2 URL

The majority of strings in this binary are either encrypted or refer to APIs/libraries.

Press enter or click to view image in full size

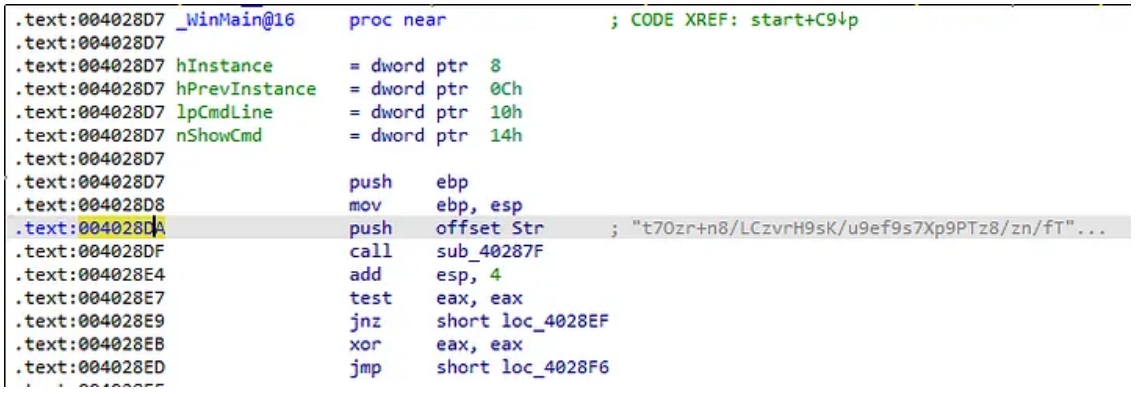


Offset	Size	Type	String
4afb	05	A	v§SQP
9420	0a	A	user32.dll
5370	07	A	uV9~¶luQ
7aeb	05	A	uFWWj
52de	07	A	u,9x¶v'
7cbe	06	A	u↑VSh
7bfc	05	A	tMWWS
7c1d	05	A	t@9}–
5139	05	A	t>j.P
a050	30	A	t7Ozr+n8/LCzvrH9sK/u9ef9s7Xp9PTz8/zn/fT0/cO7u98=
34f8	08	A	t;t$$t(
78b3	05	A	t-NuT
62ad	06	A	t$$VSS
62a6	06	A	t#SSUP
5529	08	A	sO;>|C;~
9100	0e	A	runtime error
a0d0	0100	A	rqvArQGur/Dp9wGrqd/f8O/v+PDf7e/w7N/w7e7r7Onq98C7xMjBrt+L7e7v+O3q34quvsjEwv+9sKnDqq+7qq7F6+7fmayrt7ms/8bl
4666	05	A	j?Y;M
50d2	05	A	j▬PhT
d25e	0c	A	imagehlp.dll
92ea	1c	A	abnormal program termination
01ff	07	A	`.rdata
906c	14	A	__MSVCRT_HEAP_SELECT
9054	16	A	__GLOBAL_HEAP_SELECTED
6329	06	A	_^][YY
809d	05	A	^}%95
2fc5	06	A	Yu▬9D$
54a4	05	A	YY_^[
3408	05	A	X_^[]
51f5	05	A	WufS3
d370	09	A	WriteFile
725a	05	A	Wj@Y3
d540	13	A	WideCharToMultiByte
d2b8	0b	A	WININET.dll

Encrypted strings

After loading the binary into IDA, I came across an intriguing string in Main that was passed to another function.

Press enter or click to view image in full size



```
.text:004028D7 _WinMain@16     proc near               ; CODE XREF: start+C9↓p
.text:004028D7
.text:004028D7 hInstance       = dword ptr  8
.text:004028D7 hPrevInstance   = dword ptr  0Ch
.text:004028D7 lpCmdLine       = dword ptr  10h
.text:004028D7 nShowCmd        = dword ptr  14h
.text:004028D7
.text:004028D7                 push    ebp
.text:004028D8                 mov     ebp, esp
.text:004028DA                 push    offset Str      ; "t7Ozr+n8/LCzvrH9sK/u9ef9s7Xp9PTz8/zn/fT"...
.text:004028DF                 call    sub_40287F
.text:004028E4                 add     esp, 4
.text:004028E7                 test    eax, eax
.text:004028E9                 jnz     short loc_4028EF
.text:004028EB                 xor     eax, eax
.text:004028ED                 jmp     short loc_4028F6
```

Within sub_40287F, the encrypted string is passed to strlen and then to sub_401CBC, which serves as the main decryption function.

```
.text:0040287F sub_40287F      proc near               ; CODE XREF: WinMain(x,x,x,x)+8↓p
.text:0040287F
.text:0040287F Src             = dword ptr -8
.text:0040287F var_4           = dword ptr -4
.text:0040287F Str             = dword ptr  8
.text:0040287F
.text:0040287F                 push    ebp
.text:00402880                 mov     ebp, esp
.text:00402882                 sub     esp, 8
.text:00402885                 mov     eax, [ebp+Str]
.text:00402888                 push    eax             ; Str
.text:00402889                 call    _strlen
.text:0040288E                 add     esp, 4
.text:00402891                 mov     [ebp+var_4], eax
.text:00402894                 cmp     [ebp+var_4], 10h
.text:00402898                 jnb     short loc_40289E
.text:0040289A                 xor     eax, eax
.text:0040289C                 jmp     short loc_4028D3
.text:0040289E ; ---------------------------------------------------------------------------
.text:0040289E
.text:0040289E loc_40289E:                             ; CODE XREF: sub_40287F+19↑j
.text:0040289E                 mov     ecx, [ebp+Str]  ; encrypted_string
.text:004028A1                 push    ecx
.text:004028A2                 call    sub_401CBC      ; main decryption function
```

Delving into sub_401CBC, I've labeled some variables for easier understanding. We have an XOR key of 0x59 and another subtraction key set to 0x86.

A couple of other points worth mentioning:

- Firstly, there's function sub_401917, which I've identified as a kind of preamble decryption function. It takes the encrypted string through the initial stage of decryption, returning it in a ready-to-XOR state for subsequent processing in the decryption loop.

  Press enter or click to view image in full size

```
.text:00401CBC sub_401CBC      proc near               ; CODE XREF: sub_40287F+23↓p
.text:00401CBC
.text:00401CBC XOR_key         = byte ptr -14h
.text:00401CBC subtract_key    = byte ptr -10h
.text:00401CBC length_of_string= dword ptr -0Ch
.text:00401CBC var_8           = dword ptr -8
.text:00401CBC buffer_for_decrypt= dword ptr -4
.text:00401CBC encrypted_string= dword ptr  8
.text:00401CBC
.text:00401CBC                 push    ebp
.text:00401CBD                 mov     ebp, esp
.text:00401CBF                 sub     esp, 14h
.text:00401CC2                 mov     [ebp+buffer_for_decrypt], 0 ; zero out the buffer
.text:00401CC9                 mov     [ebp+subtract_key], 86h
.text:00401CCD                 mov     [ebp+XOR_key], 59h ; 'Y' ; set the xor key
.text:00401CD1                 lea     eax, [ebp+buffer_for_decrypt]
.text:00401CD4                 push    eax             ; buffer
.text:00401CD5                 mov     ecx, [ebp+encrypted_string]
.text:00401CD8                 push    ecx
.text:00401CD9                 call    sub_401917      ; take the encrypted string, process it through stage 1 decryption, then assign to buffer
.text:00401CDE                 add     esp, 8
.text:00401CE1                 mov     [ebp+length_of_string], eax ; length of the stage 1 decrypted string
.text:00401CE4                 mov     [ebp+var_8], 0
.text:00401CEB                 jmp     short decryption_loop
```

  sub_401CBC (main decryption function) part 1

Since sub_401917 is quite complex, explaining it fully would require a dedicated blog post. Therefore, I'll simplify it as much as possible.
In the main decryption loop of sub_401917, the function handles the encrypted_string in 4-byte parts. It uses another function, sub_401B13, to decode each byte, using bitwise operations to rebuild the original data.

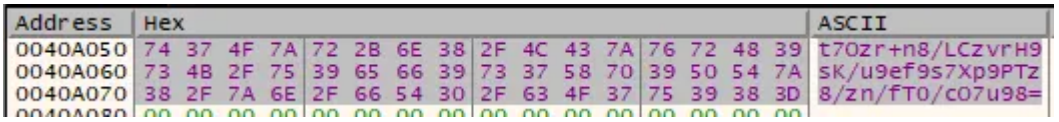
Get Andrew Petrus's stories in your inbox

Join Medium for free to get updates from this writer.

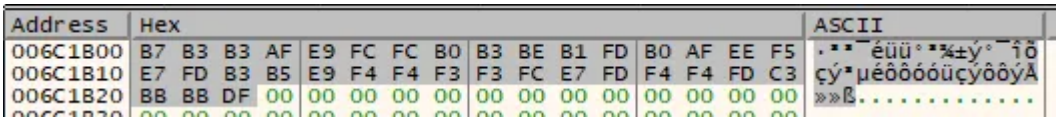
Remember me for faster sign in

Here's a link to the two main functions that make up sub_401917.

Press enter or click to view image in full size



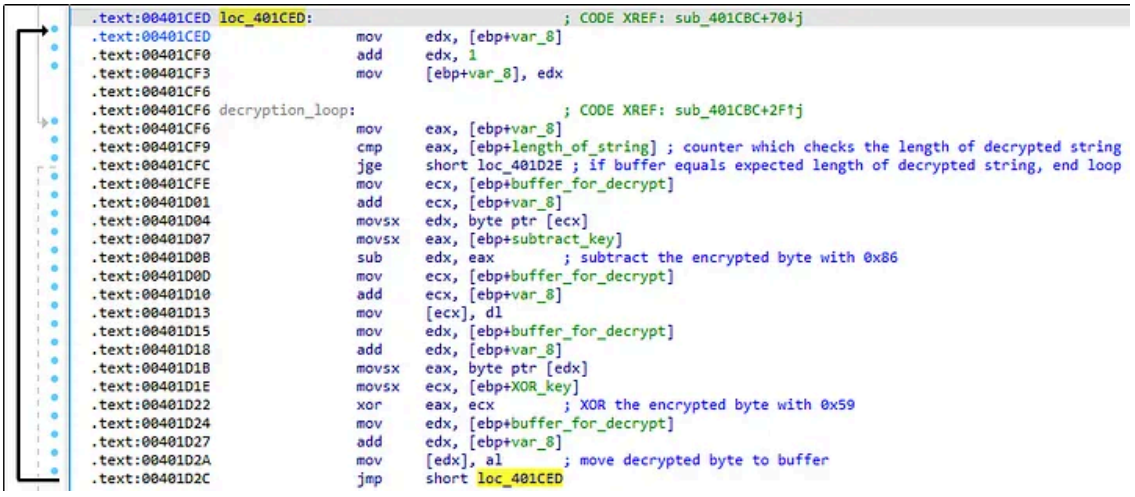
Encrypted string before preamble decryption function (sub_401917)



Encrypted string after preamble decryption function (sub_401917)

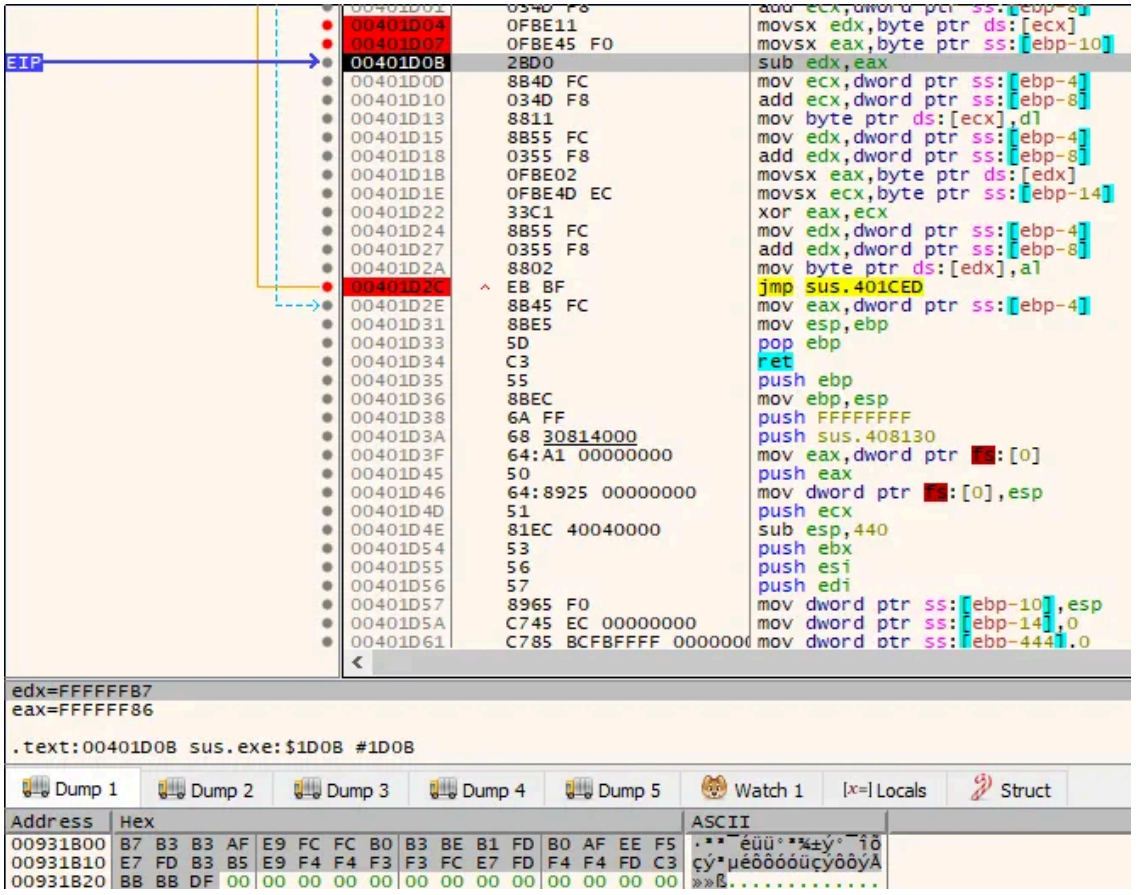
- Secondly, I've identified a loop as the main decryption loop, which decrypts the string byte by byte. To summarize this decryption process, it takes the first byte from the string returned by sub_401917, subtracts it by the subtraction key (0x86), and then performs an XOR operation using the XOR key (0x59).

  Press enter or click to view image in full size



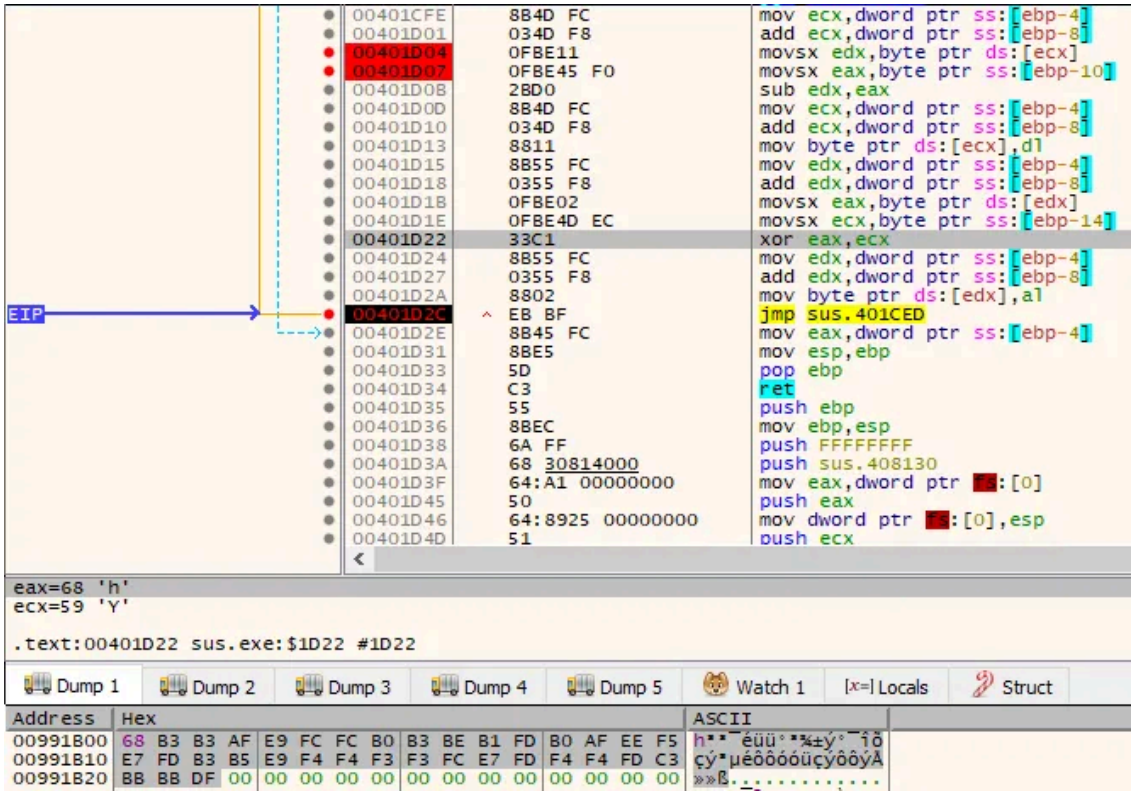
sub_401CBC (main decryption function) part 2

We can observe the subtraction instruction in our debugger.



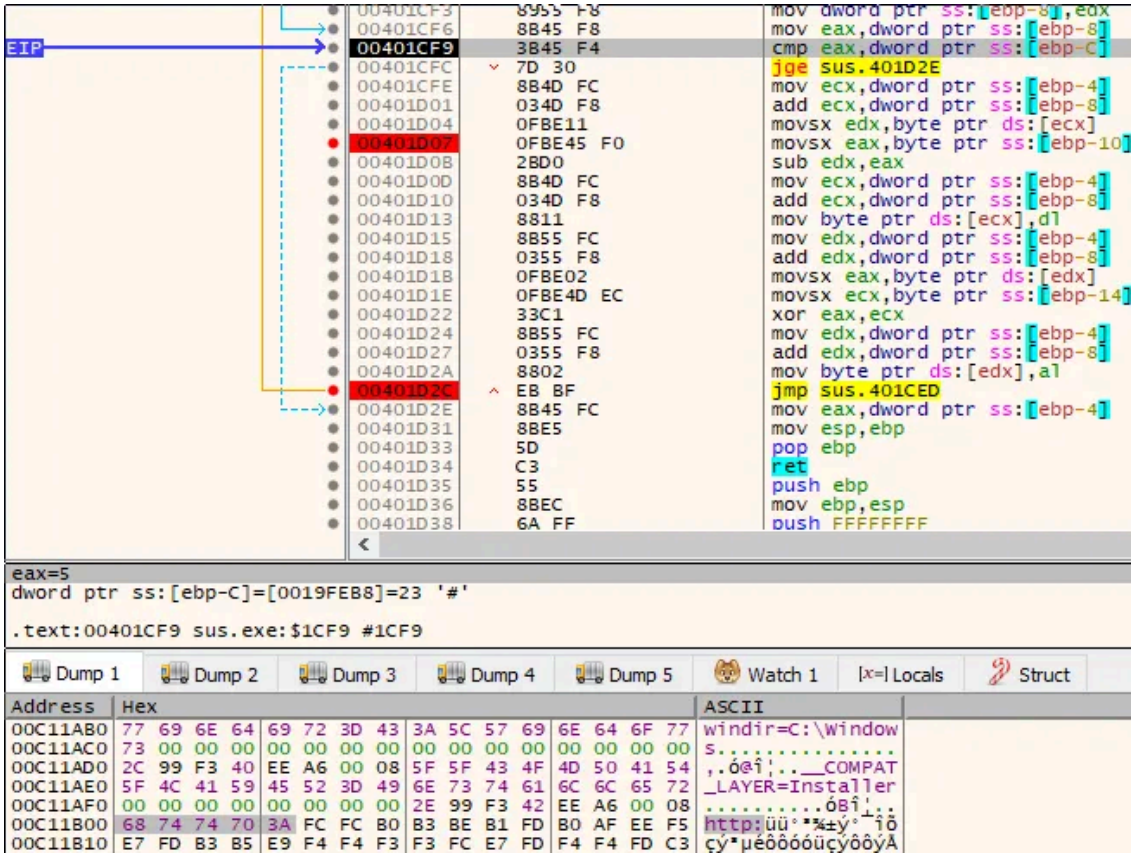
Subtracting 0x86 (subtracting_key) from 0xB7 (first byte) = 0x31

Here, for instance, 0x31 XOR'd with 0x59 decrypted the first byte to 'h' (as shown in the hex/ASCII dump below).

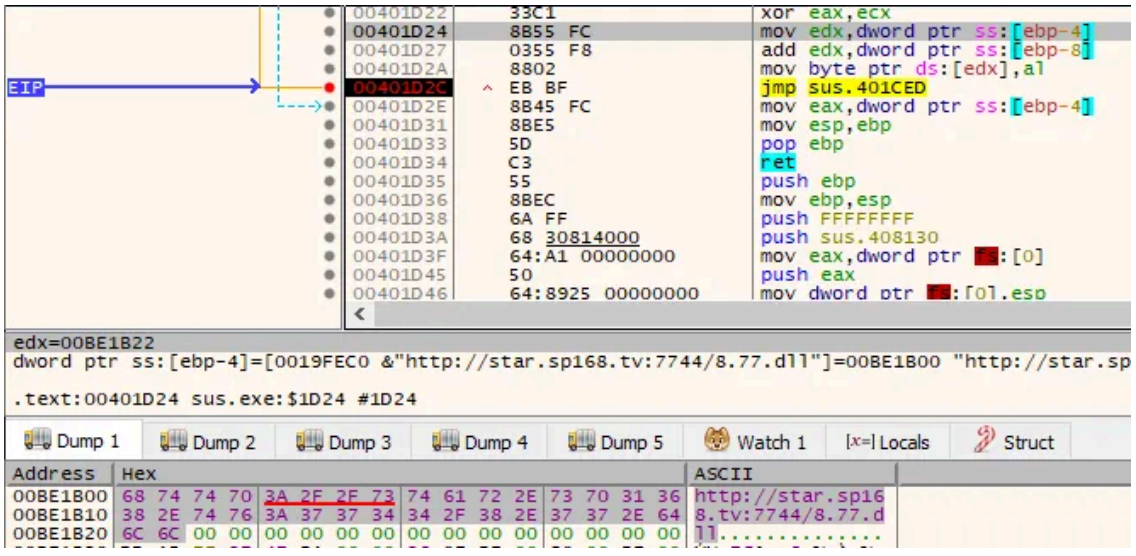


XOR 0x31 (from last operation) with 0x59 (XOR key) = 0x68 (‘h’)

The decryption function compares the current length of the decrypted string with 0x23 (0x401CF9), which is 35 in decimal.



When the comparison succeeds, the loop ends, revealing our final C2 URL.



Decrypted C2 URL which appears to download a malicious DLL file

Oh yeah!

Hash, dynamic analysis, and samples

SHA256:
506c946ecc0877b13de8fb977de24a7b9e14054d44ca547e518084c914334a6b
Dynamic Analysis:
https://www.vmray.com/analyses/_vt/506c946ecc08/report/network.html
Samples:
https://bazaar.abuse.ch/browse.php?search=tag%3Aredosdru

Thanks for tuning in

So, that wraps up our dive into this decryption method used by Redosdru. Writing this post has been really fun, and I hope you learned something new.

Until next time, stay safe out there!

Source: https://medium.com/@andrew.petrus/reverse-engineering-redosdru-string-decryption-595599087dbb