

Reverse Engineering Redosdru String Decryption

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

June 17, 2024



Andrew Petrus

--

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.

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	admin_usa32	
2023-06-06 04:52	523461b6e1b7beb0ea5...	exe	Redosdru	32 exe Redosdru	zbtcheckin	
2023-06-06 04:09	41fe567d55eb7815d15f...	exe	Redosdru	32 exe Redosdru	zbtcheckin	
2023-01-19 08:30	5a74312dcaeb621a2c4...	exe	Redosdru	32 exe Redosdru	zbtcheckin	
2022-11-06 00:34	1575c84f1a5579e58584...	zip	Redosdru	Redosdru zip	DesdinovaOsint	
2022-07-10 07:58	161d412d6102172d9a9...	exe	Redosdru	exe Redosdru	zbtcheckin	
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	a1619735baec9312f45...	exe	Redosdru	32 exe Redosdru	zbtcheckin	
2021-08-13 02:28	323ea92408f9dfb0598c...	exe	Redosdru	32 exe Redosdru	zbtcheckin	
2021-08-07 19:42	799866f1237d484f5e55...	exe	Redosdru	32 exe Redosdru	zbtcheckin	
2021-08-01 16:37	7e6b769afc67e5e76904...	exe	Redosdru	32 exe Redosdru	zbtcheckin	
2021-07-29 15:06	06c56274fc1db5ddff596...	exe	Redosdru	exe Redosdru	abuse_ch	
2021-07-02 06:27	7e2d72b0a1edfcbff327...	exe	Redosdru	32 exe Redosdru	zbtcheckin	
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.

Offset	Size	Type	String
4afb	05	A	v\$SQP
9420	0a	A	user32.dll
5370	07	A	uV9~9luQ
7aeb	05	A	uFWWj
52de	07	A	u,9x9lv'
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\$St(
78b3	05	A	t-NuT
62ad	06	A	t\$SVSS
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
02ba	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.

```
.text:004028D7 _WinMain@16      proc near                ; CODE XREF: start+C94p
.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:004028D9                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)+84p
.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 loc_40289E:                ; CODE XREF: sub_40287F+191j
.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.

```

.text:00401CBC sub_401CBC      proc near                                ; CODE XREF: sub_40287F+234p
.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:00401CB0      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:00401CE8      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.

Here's a [link](#) to the two main functions that make up sub_401917.

Address	Hex	ASCII
0040A050	74 37 4F 7A 72 2B 6E 38 2F 4C 43 7A 76 72 48 39	t70Zr+n8/LCzvrH9
0040A060	73 4B 2F 75 39 65 66 39 73 37 58 70 39 50 54 7A	sk/u9ef9s7Xp9PTz
0040A070	38 2F 7A 6E 2F 66 54 30 2F 63 4F 37 75 39 38 3D	8/zn/ft0/c07u98=
0040A080	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	

Encrypted string before preamble decryption function (sub_401917)

Address	Hex	ASCII
006C1B00	B7 B3 B3 AF E9 FC FC B0 B3 BE B1 FD B0 AF EE F5	.**~ëüü~*%+y~10
006C1B10	E7 FD B3 B5 E9 F4 F4 F3 F3 FC E7 FD F4 F4 FD C3	çy*µëööööüçyööyA
006C1B20	BB BB DF 00 00 00 00 00 00 00 00 00 00 00 00	»»B.....
006C1B30	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	

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).

.text:00401CED	loc_401CED:	mov	edx, [ebp+var_8]	; CODE XREF: sub_401CBC+70↑j
.text:00401CED		add	edx, 1	
.text:00401CF0		mov	[ebp+var_8], edx	
.text:00401CF3				
.text:00401CF6	decryption_loop:			; CODE XREF: sub_401CBC+2F↑j
.text:00401CF6		mov	eax, [ebp+var_8]	
.text:00401CF9		cmp	eax, [ebp+length_of_string]	; counter which checks the length of decrypted string
.text:00401CFC		jge	short loc_401D2E	; if buffer equals expected length of decrypted string, end loop
.text:00401CFE		mov	ecx, [ebp+buffer_for_decrypt]	
.text:00401D01		add	ecx, [ebp+var_8]	
.text:00401D04		movsx	edx, byte ptr [ecx]	
.text:00401D07		movsx	eax, [ebp+subtract_key]	
.text:00401D08		sub	edx, eax	; subtract the encrypted byte with 0x86
.text:00401D0D		mov	ecx, [ebp+buffer_for_decrypt]	
.text:00401D10		add	ecx, [ebp+var_8]	
.text:00401D13		mov	[ecx], dl	
.text:00401D15		mov	edx, [ebp+buffer_for_decrypt]	
.text:00401D18		add	edx, [ebp+var_8]	
.text:00401D1B		movsx	eax, byte ptr [edx]	
.text:00401D1E		movsx	ecx, [ebp+XOR_key]	
.text:00401D22		xor	eax, ecx	; XOR the encrypted byte with 0x59
.text:00401D24		mov	edx, [ebp+buffer_for_decrypt]	
.text:00401D27		add	edx, [ebp+var_8]	
.text:00401D2A		mov	[edx], al	; move decrypted byte to buffer
.text:00401D2C		jmp	short loc_401CED	

sub_401CBC (main decryption function) part 2

We can observe the subtraction instruction in our debugger.

EIP → 00401D08

Address	Hex	Assembly
00401D04	0FBE11	movsx edx,byte ptr ds:[ecx]
00401D07	0FBE45 F0	movsx eax,byte ptr ss:[ebp-10]
00401D08	28D0	sub edx,edx
00401D0D	884D FC	mov ecx,dword ptr ss:[ebp-4]
00401D10	034D F8	add ecx,dword ptr ss:[ebp-8]
00401D13	8811	mov byte ptr ds:[ecx],dl
00401D15	8855 FC	mov edx,dword ptr ss:[ebp-4]
00401D18	0355 F8	add edx,dword ptr ss:[ebp-8]
00401D1B	0FBE02	movsx eax,byte ptr ds:[edx]
00401D1E	0FBE4D EC	movsx ecx,byte ptr ss:[ebp-14]
00401D22	33C1	xor eax,ecx
00401D24	8855 FC	mov edx,dword ptr ss:[ebp-4]
00401D27	0355 F8	add edx,dword ptr ss:[ebp-8]
00401D2A	8802	mov byte ptr ds:[edx],al
00401D2C	EB BF	jmp sus.401CED
00401D2E	8845 FC	mov eax,dword ptr ss:[ebp-4]
00401D31	8BE5	mov esp,ebp
00401D33	5D	pop ebp
00401D34	C3	ret
00401D35	55	push ebp
00401D36	8BEC	mov ebp,esp
00401D38	6A FF	push FFFFFFFF
00401D3A	68 30814000	push sus.408130
00401D3F	64:A1 00000000	mov eax,dword ptr [0]
00401D45	50	push eax
00401D46	64:8925 00000000	mov dword ptr [0],esp
00401D4D	51	push ecx
00401D4E	81EC 40040000	sub esp,440
00401D54	53	push ebx
00401D55	56	push esi
00401D56	57	push edi
00401D57	8965 F0	mov dword ptr ss:[ebp-10],esp
00401D5A	C745 EC 00000000	mov dword ptr ss:[ebp-14],0
00401D61	C785 BCFBFFFF 00000000	mov dword ptr ss:[ebp-444],0

edx=FFFFFFB7
eax=FFFFFF86
.text:00401D08 sus.exe:\$1D08 #1D08

Address	Hex	ASCII
00931800	B7 B3 B3 AF E9 FC FC B0 B3 BE B1 FD B0 AF EE F5	...éüü°*%±ý°-îð
00931810	E7 FD B3 B5 E9 F4 F4 F3 F3 FC E7 FD F4 F4 FD C3	çý°µéððóóüçýððýÀ
00931820	BB BB DF 00 00 00 00 00 00 00 00 00 00 00 00	»»B.....

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).

Assembly code window showing instructions and their assembly code:

Address	Hex	Assembly
00401CFE	8B4D FC	mov ecx,dword ptr ss:[ebp-4]
00401D01	034D F8	add ecx,dword ptr ss:[ebp-8]
00401D04	0FBE11	movsx edx,byte ptr ds:[ecx]
00401D07	0FBE45 F0	movsx eax,byte ptr ss:[ebp-10]
00401D08	28D0	sub edx,eax
00401D0D	8B4D FC	mov ecx,dword ptr ss:[ebp-4]
00401D10	034D F8	add ecx,dword ptr ss:[ebp-8]
00401D13	8811	mov byte ptr ds:[ecx],dl
00401D15	8B55 FC	mov edx,dword ptr ss:[ebp-4]
00401D18	0355 F8	add edx,dword ptr ss:[ebp-8]
00401D1B	0FBE02	movsx eax,byte ptr ds:[edx]
00401D1E	0FBE4D EC	movsx ecx,byte ptr ss:[ebp-14]
00401D22	33C1	xor eax,ecx
00401D24	8B55 FC	mov edx,dword ptr ss:[ebp-4]
00401D27	0355 F8	add edx,dword ptr ss:[ebp-8]
00401D2A	8802	mov byte ptr ds:[edx],al
00401D2C	EB BF	jmp sus.401CED
00401D2E	8B45 FC	mov eax,dword ptr ss:[ebp-4]
00401D31	8BE5	mov esp,ebp
00401D33	5D	pop ebp
00401D34	C3	ret
00401D35	55	push ebp
00401D36	8BEC	mov ebp,esp
00401D38	6A FF	push FFFFFFFF
00401D3A	68 30814000	push sus.408130
00401D3F	64:A1 00000000	mov eax,dword ptr [0]
00401D45	50	push eax
00401D46	64:8925 00000000	mov dword ptr [0],esp
00401D4D	51	push ecx

Registers: eax=68 'h', ecx=59 'Y'

.text:00401D22 sus.exe:\$1D22 #1D22

Dump 1 | Dump 2 | Dump 3 | Dump 4 | Dump 5 | Watch 1 | [x=] Locals | Struct

Address	Hex	ASCII
00991800	68 B3 B3 AF E9 FC FC B0 B3 BE B1 FD B0 AF EE F5	h***éüü**%+ý'-ïð
00991810	E7 FD B3 B5 E9 F4 F4 F3 F3 FC E7 FD F4 F4 FD C3	çý*µéðððóüçýðýÀ
00991820	BB BB DF 00 00 00 00 00 00 00 00 00 00 00 00	»»B... ..

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.

EIP

00401CF3	8955 F8	mov dword ptr ss:[ebp-8],edx
00401CF6	8B45 F8	mov eax,dword ptr ss:[ebp-8]
00401CF9	3B45 F4	cmp eax,dword ptr ss:[ebp-C]
00401CFC	7D 30	jge sus.401D2E
00401CFE	8B4D FC	mov ecx,dword ptr ss:[ebp-4]
00401D01	034D F8	add ecx,dword ptr ss:[ebp-8]
00401D04	0FBF11	movsx edx,byte ptr ds:[ecx]
00401D07	0FBF45 F0	movsx eax,byte ptr ss:[ebp-10]
00401D08	2BD0	sub edx,eax
00401D0D	8B4D FC	mov ecx,dword ptr ss:[ebp-4]
00401D10	034D F8	add ecx,dword ptr ss:[ebp-8]
00401D13	8B11	mov byte ptr ds:[ecx],dl
00401D15	8B55 FC	mov edx,dword ptr ss:[ebp-4]
00401D18	0355 F8	add edx,dword ptr ss:[ebp-8]
00401D1B	0FBF02	movsx eax,byte ptr ds:[edx]
00401D1E	0FBF4D EC	movsx ecx,byte ptr ss:[ebp-14]
00401D22	33C1	xor eax,ecx
00401D24	8B55 FC	mov edx,dword ptr ss:[ebp-4]
00401D27	0355 F8	add edx,dword ptr ss:[ebp-8]
00401D2A	8802	mov byte ptr ds:[edx],al
00401D2C	EB BF	jmp sus.401CED
00401D2E	8B45 FC	mov eax,dword ptr ss:[ebp-4]
00401D31	8BE5	mov esp,ebp
00401D33	5D	pop ebp
00401D34	C3	ret
00401D35	55	push ebp
00401D36	8BEC	mov ebp,esp
00401D38	6A FF	push FFFFFFFF

eax=5
dword ptr ss:[ebp-C]=[0019FEB8]=23 '#'
.text:00401CF9 sus.exe:\$1CF9 #1CF9

Dump 1		Dump 2		Dump 3		Dump 4		Dump 5		Watch 1		[x=] Locals		Struct				
Address	Hex											ASCII						
00C11AB0	77 69 6E 64	69 72 3D 43	3A 5C 57 69	6E 64 6F 77	windir=C:\window													
00C11AC0	73 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	s.....													
00C11AD0	2C 99 F3 40	EE A6 00 08	5F 5F 43 4F	4D 50 41 54	,.ô@!;...__COMPAT													
00C11AE0	5F 4C 41 59	45 52 3D 49	6E 73 74 61	6C 6C 65 72	_LAYER=Installer													
00C11AF0	00 00 00 00	00 00 00 00	2E 99 F3 42	EE A6 00 08	ô8i!..													
00C11B00	68 74 74 70	3A FC B0 B3	BE B1 FD B0	AF EE F5	http:üü*%zy°iö													
00C11B10	E7 FD B3 B5	E9 F4 F4 F3	F3 FC E7 FD	F4 F4 FD C3	cy*µeôôôüücyôôyA													

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

EIP

00401D22	33C1	xor eax,ecx
00401D24	8B55 FC	mov edx,dword ptr ss:[ebp-4]
00401D27	0355 F8	add edx,dword ptr ss:[ebp-8]
00401D2A	8802	mov byte ptr ds:[edx],al
00401D2C	EB BF	jmp sus.401CED
00401D2E	8B45 FC	mov eax,dword ptr ss:[ebp-4]
00401D31	8BE5	mov esp,ebp
00401D33	5D	pop ebp
00401D34	C3	ret
00401D35	55	push ebp
00401D36	8BEC	mov ebp,esp
00401D38	6A FF	push FFFFFFFF
00401D3A	68 30814000	push sus.408130
00401D3F	64:A1 00000000	mov eax,dword ptr [0]
00401D45	50	push eax
00401D46	64:8925 00000000	mov dword ptr [0]:f01.eso

edx=00BE1B22
dword ptr ss:[ebp-4]=[0019FEC0 &"http://star.sp168.tv:7744/8.77.dll"]=00BE1B00 "http://star.sp
.text:00401D24 sus.exe:\$1D24 #1D24

Dump 1		Dump 2		Dump 3		Dump 4		Dump 5		Watch 1		[x=] Locals		Struct				
Address	Hex											ASCII						
00BE1B00	68 74 74 70	3A 2F 2F 73	74 61 72 2E	73 70 31 36	http://star.sp16													
00BE1B10	38 2E 74 76	3A 37 37 34	34 2F 38 2E	37 37 2E 64	8.tv:7744/8.77.d													
00BE1B20	6C 6C 00 00	00 00 00 00	00 00 00 00	00 00 00 00	ll.....													

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!