# A Deep Dive into Brute Ratel C4 payloads – Part 2

**cybergeeks.tech**/a-deep-dive-into-brute-ratel-c4-payloads-part-2/

#### Summary

<u>Brute Ratel C4</u> is a Red Team & Adversary Simulation software that can be considered an alternative to Cobalt Strike. In this blog post, we're presenting a technical analysis of a Brute Ratel badger/agent that doesn't implement all the recent features of the framework. There aren't a lot of Brute Ratel samples available in the wild. This second part of the analysis presents the remaining commands executed by the agent. The commands include: user impersonation, inject shellcode into processes, create and stop processes, extract information about the processes and services, create TCP listeners, block keyboard and mouse input events, extract Windows registry keys and values, and others. You can consult the first part of the analysis <u>here</u>.

Technical analysis

SHA256: d71dc7ba8523947e08c6eec43a726fe75aed248dfd3a7c4f6537224e9ed05f6f

We continue to describe the commands that can be used by the Brute Ratel agent.

**0x0703 ID** – Stop the current process

The malware stops the current process by calling the ExitProcess API:



#### Figure 1

0x6BAE/0x6F39 ID – User impersonation

The binary retrieves a pseudo handle for the current process using GetCurrentProcess:

	000000000297855F	FF 15 93	D7 00 00	call qword ptr ds: [<&GetCurrentProcess>]	>
qword ptr [00000	00002985CF8 <&GetCur	rentProcess	s>]= <kernel3< td=""><td>2.GetCurrentProcess&gt;</td><td></td></kernel3<>	2.GetCurrentProcess>	

Figure 2

OpenProcessToken is utilized to open the access token associated with the process (0x28 = **TOKEN\_ADJUST\_PRIVILEGES** | **TOKEN\_QUERY**):

000000002978555     00000000297856A     00000000297856F	4C 8D 44 24 30 BA 28 00 00 00 48 89 C1	iea r8,qword ptr ss:[rsp+30] mov edx,28 mov rcx,rax	28: ' ('	x87Tagword FFFF	
000000002978572	PF DS	CALL FOX	>	Default (x64 fastcall)	▼ 5 € Unlocke
rbx= <advapi32.0penprocesstoken> (0</advapi32.0penprocesstoken>	0007FF8AF836220)			2: rdx 000000000000028	

#### Figure 3

The process extracts the locally unique identifier (LUID) for the "SeDebugPrivilege" privilege (Figure 4).

000000002978578     000000002978578     0000000002978577     0000000002978578     0000000002978584	31 C9 48 8D 5C 24 3C 48 8D 54 24 4F 4C 8D 44 24 40	<pre>xor ecx,ecx lea rbx,qword ptr ss: rsp+BC lea rdx,qword ptr ss: rsp+4F lea r8,qword ptr ss: rsp+46</pre>		x87r7 00000000000000000000 ST7 Empty 0.00000000000000000000000000000000000
000000002978589	FF 15 79 DC 00 00	[call qword ptr ds:[<&LookupPrivilegeValueA>]	, v	Default (x64 fastcall) 👻 5 💠 🗌 Unlocke
qword ptr [0000000002986208 <&Looku	pPrivilegeValueA>]= <ad< td=""><td>vapi32.LookupPrivilegevalueA&gt;</td><td>,</td><td>1: rcx 000000000000000 2: rdx 0000000051EF95F "SeDebugPrivilege"</td></ad<>	vapi32.LookupPrivilegevalueA>	,	1: rcx 000000000000000 2: rdx 0000000051EF95F "SeDebugPrivilege"

The executable enables the above privilege via a function call to AdjustTokenPrivileges:



# Figure 5

The running processes are enumerated using the Process32FirstW and Process32NextW functions:



# Figure 7

The agent is looking for the "LogonUI.exe", "winlogon.exe", and "Isass.exe" processes:

<ul> <li>0000000029799CB</li> <li>00000000029799CE</li> </ul>	48 89 DA 48 89 F9	mov rdx,rbx mov rcx,rd1	rdx:L"L rcx:L"S	x87TagWord FFFF	
00000000000000000000000000000000000000	ES OA SE FF FF	call «wcscmp»	~	Default (x64 fastcall)	▼ 5 ÷ Unlocke
<			>	1: rcx 0000000051EF744 L"System"	

# Figure 8

It opens the first process found using the OpenProcess method (0x400 = **PROCESS\_QUERY\_INFORMATION**):

00000000297902E 41 89 C0     000000002979031 85 C0     000000002979033 74 IA     000000002979035 31 02     000000002979037 89 00 04 00 00	mov rSd,eax TeSt eax,eax <b>je 23704F</b> xor edx,edx mov ecx,400	x8/F6 000000000000000000000000000000000000	Empty 0.00000000000000000000000000000000000
81P → 00000000297903C FF 15 56 C2 00 00	[call qword ptr ds:[<&OpenProcess>]	 Default (x64 fastcall)	▼ 5 ¢ 🗌 Unlocke
qword ptr [000000002985F98 <&OpenProcess>]= <kernel32.0< td=""><td>penProcess&gt;</td><td> 1: rcx 0000000000000400 2: rdx 00000000000000000</td><td></td></kernel32.0<>	penProcess>	 1: rcx 0000000000000400 2: rdx 00000000000000000	

# Figure 9

ImpersonateLoggedOnUser is used to impersonate the security content of the user extracted from the process identified above:



# Figure 10

In order to confirm that the operation was successful, the malware calls the GetUserNameW API (see Figure 11).

	<ul> <li>000000002979ED5</li> <li>000000002979EDA</li> <li>000000002979EDD</li> </ul>	48 8D 54 24 5C 48 89 F9 BB 0D 00 00 00	lea rdx,qword ptr ss:[rsp+5C] mov rcx,rdi mov ebx,D	0D;'\r'	x87Tagword FFFF	195
qword ptr [0	000000002985BC8 <&GetUse	rNamew>]= <advap132.get< td=""><td>UserNamew&gt;</td><td>&gt;</td><td>Default (x64 fastcal) 1: rcx 00000000051EFA96 2: rdx 00000000051EF90C</td><td>▼ 5 C Unlocke</td></advap132.get<>	UserNamew>	>	Default (x64 fastcal) 1: rcx 00000000051EFA96 2: rdx 00000000051EF90C	▼ 5 C Unlocke

# Figure 11

The message displayed in Figure 12 will be sent to the C2 server:

rcx=82299023EFEF0000 r12=0000000045B08A0 L"[+] Impersonated 'SYSTEM' via technique 2" Figure 12 On another branch, the binary calls the DuplicateTokenEx method in order to duplicate the access token extracted from "winlogon.exe" or "Isass.exe". Finally, a new process is created using CreateProcessWithTokenW.

**0xA86A ID** – Inject code into a remote process

The malicious executable converts the process ID passed as a parameter using atoi:



### Figure 13

The shellcode to be executed is Base64-decoded by calling the CryptStringToBinaryA API (0x1 = CRYPT\_STRING\_BASE64):



#### Figure 14

The badger opens the target process using OpenProcess (0x1F0FFF = **PROCESS\_ALL\_ACCESS**):



# Figure 15

VirtualAllocEx is utilized to allocate a new memory area in the remote process (0x3000 = **MEM\_COMMIT** | **MEM\_RESERVE**, 0x4 = **PAGE\_READWRITE**):

<ul> <li>00000000297A5A1</li> <li>00000000297A6A9</li> <li>00000000297A6AC</li> <li>00000000297A6AC</li> <li>00000000297A6A1</li> <li>00000000297A6B1</li> </ul>	C7         44         24         20         04         00         mov         dword ptr         ss:[rsp+20],4           49         89         D8         mov r8,rbx         mov r8,rbx         st.         st.	X8/F6 000000000000000000000000000000000000
B12 ● 00000000297A687 <	FF 15 93 BS 00 00 [call gword ptr ds:[<&virtual/	> Default (x64 fastcall)
qword ptr [000000002985C50 <&Virtu	(]AllocEx>]= <kernel32.virtualallocex></kernel32.virtualallocex>	11 rcx 000000000000350 21 rdx 00000000000000 31 r8 000000000001453 41 r9 000000000003000

### Figure 16

The malware writes the shellcode to the above area via a function call to WriteProcessMemory, as shown in Figure 17.



### Figure 17

The page's protection is changed using the VirtualProtectEx API (0x20 = **PAGE\_EXECUTE\_READ**):

	<ul> <li>00000000297A70B</li> <li>00000000297A70E</li> <li>00000000297A714</li> <li>00000000297A716</li> <li>00000000297A718</li> <li>00000000297A71E</li> </ul>	48 89 F2 41 89 20 00 00 00 31 F6 48 89 44 24 20 49 89 D8 4C 89 E9	mov rdx,rs1 mov rdx,20 xor es1,es1 mov qword ptr sst[rsp+20],rax mov r8,rbx mov rcx,r13	20: ' '	x87r6 0000000000000000000000000000000000 x87r7 00000000000000000000000000000000000	Empty 0.000000000000000000 Empty 0.00000000000000000000000000000000000
2114	e (	FF 15 F1 B1 00 00	call dword per dst [kay incualindectexs]		Default (x64 fastcall)	▼ 5 ≑ 🗋 Unlock
qword ptr	0000000002985918 <&Virtu	alProtectEx>]= <kernel3< td=""><td>2.VirtualProtectEx&gt;</td><td></td><td>→ 1: rcx 00000000000000000 2: rdx 0000000000000000 3: r8 0000000000001453 4: r9 000000000000000 5: [rsp+20] 00000000051EFEOC</td><td></td></kernel3<>	2.VirtualProtectEx>		→ 1: rcx 00000000000000000 2: rdx 0000000000000000 3: r8 0000000000001453 4: r9 000000000000000 5: [rsp+20] 00000000051EFEOC	

Finally, the binary creates a thread in the remote process that executes the shellcode:



### Figure 19

0xE9B0 ID - Create a process and read its output via a pipe

The agent creates an anonymous pipe using the CreatePipe method:

	0000003970526 4 8 80 BC 24 90 00 0000023970264 C 7 84 24 C 80 00 00000023970281 48 80 8C 24 88 00 00000023970281 48 89 7A 00000023970264 48 89 7A 00000023970264 48 89 7A 00000023970264 48 89 7A 00000023970264 48 89 7A 18 88 80 00 00000023970264 48 89 00 00 00000023970264 48 89 00 00 00000023970264 88 88 00 00 00000023970264 88 00 00 00000023970264 88 00 00 00000023970264 88 00 00000023970264 88 00 00000023970264 88 00 00000023970264 88 00 00000023970264 88 00 00000000000000000000000000000000	0 00 1 28 rdl.qword ptr ss:[rsp+30] 00 mov dword ptr ss:[rsp+30] 01 call qword ptr ds:[cscreatePipes]	68: 'h' >	x87r4         000000000000000000000000000000000000	000000000000 00000000000 00000000000 0000
DOOD     DOOD     ODO     ODO	000002370267         45         31         C9           000002370264         45         80         84         24         98         00           000002370272         48         C7         84         24         88         00           0000002370274         42         80         80         84         24         80         00           0000002370286         B9         06         00	<pre>xor r9d,r9d 00 lea r14,qword ptr ss: [rsp+80] 00 mov qword ptr ss: [rsp+86],0 00 lea r6,qword ptr ss: [rsp+86] repe stosd mov ect,r14 00 mov qword,ptr ss: [rsp+90],0</pre>		GS 0028         F5 0053           ES 0028         DES 0028           CS 0033 <u>SS</u> 0028           K87r0         000000000000000000000000000000000000	10000000000000 10000000000000 100000000

### Figure 20

The pipe is set to be inherited via a call to SetHandleInformation (0x1 = **HANDLE\_FLAG\_INHERIT**):

00000000297D2EC     00000000297D2EF     000000000297D2F7	45 31 C0 48 8B 8C 24 88 00 00 BA 01 00 00 00	<pre>xor r8d,r8d mov rcx,qword ptr ss:[rsp+88] mov edx,1</pre>	x87Tagword FFFF	
NIP → 00000000297D2FC	FF 15 2E 8C 00 00	<pre>call qword ptr ds:[&lt;&amp;SetHandleInformation&gt;]</pre>	 Default (x64 fastcall)	▼ 5 🗘 🗌 Unlocke
qword ptr [000000002985F30 <&SetHa	ndleInformation>]= <kerr< td=""><td>nel32.SetHandleInformation&gt;</td><td> 1: rcx 000000000000000000 2: rdx 00000000000000000000000000000000000</td><td></td></kerr<>	nel32.SetHandleInformation>	 1: rcx 000000000000000000 2: rdx 00000000000000000000000000000000000	

# Figure 21

The malicious executable creates a process specified by the C2 server using the CreateProcessA API, as shown in the figure below.



### Figure 22

The process' output that resides in the anonymous pipe is copied into a buffer by calling PeekNamedPipe (Figure 23).



The output is read using ReadFile and then transmitted to the C2 server:



# Figure 24

**0x91B3 ID** – Inject code into the current process

The CryptStringToBinaryA method is utilized to decode from Base64 the shellcode that will be executed:



### Figure 25

The agent creates a named pipe (0x80000003 = FILE\_FLAG\_WRITE\_THROUGH | PIPE\_ACCESS\_DUPLEX):

<ul> <li>0000000029788F8</li> <li>0000000029788F8</li> <li>000000002978878</li> <li>000000000297893</li> <li>0000000002978914</li> <li>0000000002978914</li> <li>0000000002978914</li> <li>0000000002978914</li> <li>0000000002978914</li> <li>0000000002978914</li> <li>0000000002978914</li> <li>0000000002978914</li> </ul>	45       31       CO       xor r8d,r8d         C7       44       24       30       00       00       mov rdword ptr ss:       [rsp+30],0         9       88       C2       CO       04       00       mov rcs,qword ptr ds:       [r12+4CO]         41       89       01       00       00       mov rds,1       ss:       [rsp+78],FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF	rcx:"//'	x8773 0000000000000000000 573 Empty 0.0000000000000000000 x8774 0000000000000000000 574 Empty 0.0000000000000000 x8775 000000000000000000 575 Empty 0.0000000000000000 x8777 00000000000000000 575 Empty 0.00000000000000000 x8777 0000000000000000 575 Empty 0.000000000000000000 x8777 00000000000000000 575 Empty 0.00000000000000000000000000000000000
<	re 15 DA AB 00 00 Call quord pr 351 car eatenaneur pewsj	>	Default (x64 fastcal) 1: rcx 000000004580690 "\\\\.\pipe\pipe"
qword ptr [000000002986018 <&Creat	eNamedP1pew>]= <kernel32.createnamedp1pew></kernel32.createnamedp1pew>		2: rdx 00000008000003 3: r8 00000000000000 4: r9 000000000000001 5: [rspt20] 000000000FFFF

### Figure 26

A new thread is created using the CreateThread function. In this thread, the malware connects to the pipe and reads data using the ConnectNamedPipe and ReadFile methods:

000000002978951 000000002978956 00000000297856 00000000237856 000000002378966 000000002378966 000000002378966 000000002378966	4C 8D 4C 24 70 31 D2 31 C9 4C 8D 05 2F 01 00 0 48 89 44 24 28 C7 44 24 20 00 00 0 FF 15 74 A6 00 00	lea r9,qword ptr ss:[rsp+70] xor edx,edx 0 hea 65,ecnd ptr ds:[3978A90] mov qword ptr ss:[rsp+20],rea 0 mov dword ptr ss:[rsp+20],0 call qword ptr ds:[460reateThread-]	~	X8775         000000000000000000000000000000000000	
gword ptr [0000000002985FE8 <&Creat	teThread>]= <kernel32.cr< td=""><td>eateThread&gt;</td><td></td><td>1: rcx 00000000000000000000000000000000000</td><td></td></kernel32.cr<>	eateThread>		1: rcx 00000000000000000000000000000000000	
Figure 27					
	31 D2 48 89 CB 48 88 49 08 FF 15 90 A7 00 00	<pre>xor edx,edx mov rbx,rcx mov rcx,qword ptr ds:[rcx+8] call qword ptr ds:[c&amp;ConnectNamedPipe&gt;]</pre>	> ~	x87Tagword FFFF	▼ 5 🜩 🗆 Unlocke
qword ptr [000000002986248 <&Conne	ectNamedPipe>]= <kernel3< td=""><td>2.ConnectNamedPipe&gt;</td><td></td><td>2: rdx 000000000000000</td><td></td></kernel3<>	2.ConnectNamedPipe>		2: rdx 000000000000000	

# Figure 28

VirtualAllocEx is used to allocate a new memory area in the current process:

<ul> <li>000000002578574</li> <li>000000002578574</li> <li>00000000257857A</li> <li>00000000257857A</li> <li>00000000257857A</li> <li>00000000257853</li> <li>00000000257853</li> <li>000000002578538</li> <li>000000002578588</li> <li>000000002578588</li> <li>000000002578581</li> </ul>	44         8D         43         01         1e           31         D2         X0           4C         89         F1         mo           4D         63         C0         mo           4B         89         C5         04         00         00         mo           41         89         05         04         00         00         mo           42         89         42         24         48         mo	a r8d,qword ptr ds:[rbx+1] r edx,edx vrCx,r14 vsAd r6,r8d v dword ptr ss:[rsp+20],4 v 904,3000 v qword ptr ss:[rsp+48],r8	x87r4 00000000000000000000 514 Emp x87r5 000000000000000000000 515 Emp x87r6 000000000000000000000 515 Emp x87r7 000000000000000000000 517 Emp x87r7 00000000000000000000 517 Emp	Ty 0.0000000000000000 Ty 0.0000000000000000 Ty 0.0000000000000000000 ty 0.00000000000000000000000000000000000
R1P 000000002978996	FF 15 B4 A2 00 00 Ca	<pre>qword ptr ds:[&lt;&amp;VirtualAllocEx&gt;]</pre>	 Default (x64 fastcall)	▼ 5 ¢ 🗌 Unlocke
qword ptr [000000002985C50 <&virtu 0000000002978996	alAllocEx>]= <kernel32.virt< td=""><td>JallocEx&gt;</td><td>1: rcx FFFFFFFFFFFFF 2: rdx 000000000000000 3: r8 00000000000000 4: r9 000000000000000 5: [rsp+20] 00000000000000000</td><td></td></kernel32.virt<>	JallocEx>	1: rcx FFFFFFFFFFFFF 2: rdx 000000000000000 3: r8 00000000000000 4: r9 000000000000000 5: [rsp+20] 00000000000000000	

### Figure 29

The shellcode is copied into the new area and its page is made executable, as highlighted below:

	000000002978980     000000002978983     000000002978988     000000002978988     000000002978986     00000000297896     00000000297896     000000002978967     0000000002978967	48 63 CB 48 6D 44 24 60 F3 A4 4C 8B 44 24 48 48 89 44 24 48 48 89 44 24 20 4C 89 EA 41 89 20 00 00 00 4C 89 F1	movsxd rCx,ebx lea rax,gword ptr ss:[rsp+60] mov r6,gword ptr ss:[rsp+20],rax mov rdx,r13 ss:[rsp+20],rax mov rdx,r14	rdx: "AB 20: "	X8/F3 00000000000000000000000000000 X8774 000000000000000000000000 X8775 00000000000000000000000000 X8777 0000000000000000000000000000000000	Empty 0.00000000000000000000000000000000000
RIP	00000002978900	FF 15 42 9F 00 00	call gword ptr ds:[ <dvirtualprotectex>]</dvirtualprotectex>		Default (x64 fastcall)	▼ 5 0 Unlock
qword ptr	0000000002985918 <&virtu	alProtectEx>]= <kernel3< td=""><td>2.VirtualProtectEx&gt;</td><td>,</td><td>1: rcx FFFFFFFFFFFFFF 2: rdx 00000000053F0000 "ABCDEF 3: r8 00000000000008 4: r9 000000000000020 5: [rsp+20] 00000000051EFEA0</td><td>GHIJ"</td></kernel3<>	2.VirtualProtectEx>	,	1: rcx FFFFFFFFFFFFFF 2: rdx 00000000053F0000 "ABCDEF 3: r8 00000000000008 4: r9 000000000000020 5: [rsp+20] 00000000051EFEA0	GHIJ"

A new thread runs the shellcode copied earlier:

<ul> <li>0000000029789DF</li> <li>000000002978952</li> <li>0000000002978955</li> <li>000000002978957</li> <li>000000002978959</li> <li>000000002978956</li> </ul>	45 31 C9 xor r9d,r9d 40 80 E8 mov r8,r13 31 C9 xor eck,eck 48 89 44 24 28 mov qword ptr ss:[rsp+28],rax 67 44 24 20 00 00 00 mov dword ptr ss:[rsp+28],0	r 8: "ABCI	x8/r5 0000000000000000000 515 1 x87r6 000000000000000000 515 1 x87r7 00000000000000000000 517 7 x87Tagword FFFF	Empty 0.00000000000000000000000000000000000
	FF 15 EC AS 00 00 [call gword ptr ds:[<&CreateThread>		/ Default (x64 fastcall)	▼ 5 😫 🗌 Unlacke
qword ptr [0000000002985FE8 <&Create	Thread>]= <kernel32.createthread></kernel32.createthread>		<pre>1: rcx 00000000000000000000000000000000000</pre>	13"

# Figure 31

### 0x1719 ID - Enable SeDebugPrivilege

The malicious process calls the LookupPrivilegeValueA function with the "SeDebugPrivilege" parameter:

	00000000029785     00000000029785     00000000029785     00000000029785	78 31 C9 7A 48 8D 5C 24 3C 7F 48 8D 54 24 4F 84 4C 8D 44 24 40	<pre>xor ecx,ecx lea rbx,qword ptr ss:[rsp+3C] lea rdx,qword ptr ss:[rsp+4F] lea r8,qword ptr ss:[rsp+40]</pre>		x87r7 000000000000000000 ST7 Empty 0.00000000000000000000000000000000000
RIP	0000000029785	59 FF 15 79 DC 00 00	call gword ptr ds:[<&LookupPrivilegevalueA>]		▼ Default (x64 fastcall)
qword ptr	0000000002986208 <&u	ookupPrivilegeValueA>]=<	dvap132.LookupPr1v1legeValueA>		1: rcx 00000000000000 2: rdx 0000000051EFEAF 3: r8 00000000051EFEA0 4: r9 000000002985DA0 5: [rsp+20] 0000000051EFF20
Ump 1	Ump 2 Ump 2	3 1 Dump 4 1 Dump 5	😸 Watch 1 🛛 🖉 Struct		00000000051EFE60 F000000002985DA0 00000000051EFE68 00007FF8AEF5984C return to msvcrt.00007FF8
Addr ess 0000000005	Hex LEFEAF 53 65 44 65 63	75 67 50 72 69 76 69 60	ASCII 65 67 65 SeDebugPrivilege	^	0000000051EFE70 000000002985DA8 00000000051EFE78 000000000000000000000000000000000000

#### Figure 32

The PrivilegeCheck API is utilized to determine if the above privilege is enabled in the access token:

		<ul> <li>00000000297895D</li> <li>000000002978962</li> <li>000000002978967</li> <li>000000002978957</li> <li>000000002978974</li> <li>000000002978974</li> </ul>	48         8D         54         24         4C         1eardx.qw           4C         8D         44         24         1eardx.qw         1eardx.qw           67         44         24         5C         00         00         mov dword           68         89         44         24         0         mov qword         mov qword           48         89         44         24         30         mov rax.qw         qword	ord ptr ss:[rsp+4C] rd ptr ss:[rsp+24] ptr ss:[rsp+5C],2 ptr ss:[rsp+4C],rax ord ptr ss:[rsp+30] ptr ss:[rsp+54],rax		x8/r5 000000000000000000000000000000000000	mpty 0.00000000000000000000000000000000000
RIP		< 00000000297897E	FF 15 D4 D5 00 00   Call gword	ptr ds:[<&PrivilegeCheck>]	>	Default (x64 fastcall)	▼ 5 🗘 🗆 Unlocke
qwo	d ptr	0000000002985F58 <&Privi	legeCheck>]= <advapi32.privilegeche< td=""><td>do</td><td></td><td>2: rdx 00000000051EFEAC</td><td></td></advapi32.privilegeche<>	do		2: rdx 00000000051EFEAC	

#### Figure 33

The message displayed in Figure 34 will be sent to the C2 server as a confirmation.

rdx=10
qword ptr [rsp+28]=[0000000051EFEF8 &L"[+] Enabled debug privilege"]
Figure 34
0x4FFE ID - Extract the status of the token's privileges

The badger obtains the TOKEN\_PRIVILEGES structure that contains the privileges of the token using GetTokenInformation (see Figure 35).



#### Figure 35

It retrieves the name of the privileges represented by a locally unique identifier (LUID) via a function call to LookupPrivilegeNameW:

00000000297C0     00000000297C0     00000000297C0     00000000297C0     00000000297C0     00000000297C0	49         89         FO         mov r8,r51           084         48         89         DA         mov rdx,rbx           87         C7         44         24         48         00         100         mov rdx,rbx           887         C7         44         24         48         00         100         mov rdx,rbx           887         C7         44         24         48         00         100         mov rdx,rbx           887         S1         C9         xor         ecx,ecx         mov r9,r14	x8/r6 0000000000000000 516 EmpLy 0.00000000000000000000000000000000000
31P → 00000000297C0	FF 15 46 A1 00 00 call gword ptr ds: [cdLookupPrivilegeNam	> Default (x64 fastcall) • 5 🗘 Unlock
qword ptr [0000000029861E0 <&	LookupPrivilegeNamew>]= <advapi32.lookupprivilegenamew></advapi32.lookupprivilegenamew>	1: rcx 00000000000000000000000000000000000

The list of privileges and their status is written in the memory. The following statuses can be specified: "[+] %-50ls Enabled (Default)", "[+] %-50ls Enabled (Adjusted)", "[+] %-50lsDisabled\n", "[+] Elevated", or "[+] Restricted".

Address	He	ĸ											2				ASCII
0000000029F8210	5 B	00	2B	00	5D	00	20	00	53	00	65	00	49	00	6E	00	[.+.]S.e.I.n.
0000000029F8220	63	00	72	00	65	00	61	00	73	00	65	00	51	00	75	00	c.r.e.a.s.e.Q.u.
0000000029F8230	6F	00	74	00	61	00	50	00	72	00	69	00	76	00	69	00	o.t.a.P.r.i.v.i.
0000000029F8240	6C	00	65	00	67	00	65	00	20	00	20	00	20	00	20	00	1.e.g.e
0000000029F8250	20	00	20	00	20	00	20	00	20	00	20	00	20	00	20	00	
0000000029F8260	20	00	20	00	20	00	20	00	20	00	20	00	20	00	20	00	
0000000029F8270	20	00	20	00	20	00	20	00	20	00	20	00	44	00	69	00	· · · · · D. i. Figure 07
0000000029F8280	73	00	61	00	62	00	6C	00	65	00	64	00	0A	00	5 B	00	s.a.b.1.e.d[. Figure 37
0000000029F8290	2B	00	5D	00	20	00	53	00	65	00	53	00	65	00	63	00	+.]S.e.S.e.c.
0000000029F82A0	75	00	72	00	69	00	74	00	79	00	50	00	72	00	69	00	u.r.i.t.y.P.r.i.
0000000029F82B0	76	00	69	00	6C	00	65	00	67	00	65	00	20	00	20	00	v.i.l.e.g.e
0000000029F82C0	20	00	20	00	20	00	20	00	20	00	20	00	20	00	20	00	
0000000029F82D0	20	00	20	00	20	00	20	00	20	00	20	00	20	00	20	00	
0000000029F82E0	20	00	20	00	20	00	20	00	20	00	20	00	20	00	20	00	
00000000029F82F0	20	00	20	00	20	00	20	00	20	00	44	00	69	00	73	00	D.i.s.
0000000029F8300	61	00	62	00	6C	00	65	00	64	00	0A	00	5 B	00	2B	00	a.b.l.e.d[.+.
0x9DE0 ID - Ext	ract	t Us	ser	nar	ne.	PF	PID	. P	ID.	an	d E	xe	cut	abl	еp	ath	for every running process

The binary obtains a snapshot of all processes in the system using CreateToolhelp32Snapshot. It enumerates them using the Process32FirstW and Process32NextW methods:

>■ 00000000297C296 ● 00000000297C299	48 89 DA 48 89 C1	mov rdx,rbx mov rcx,rax		x87Tagword FFFF	n a a cenan o	
	FF 15 6E 96 00 00	call gword ptr ds:[<&Process32Firstw>]	>	Default (x64 fastcall)	👻 5 🔅 🗌 Unlocke	
qword ptr [000000002985910 <&Proces	s32FirstW>]= <kernel32< th=""><th>.Process32FirstW&gt;</th><th></th><th>1: rcx 00000000000000000000000000000000000</th><th></th></kernel32<>	.Process32FirstW>		1: rcx 00000000000000000000000000000000000		
Figure 38						
000000000297C2D9     00000000297C2DC	48 89 DA 4C 89 E1	mov rdx,rbx mov rcx,r12		x87Tagword FFFF		
00000000297C2DF	FF 15 98 98 00 00	call qword ptr ds:[<&Process32NextW>]		Default (x64 fastcall)	▼ 5 😫 🗌 Unlocke	
gword ptr [000000002985880 <&Proces	s32NextW>]= <kernel32.< td=""><td>Process32NextW&gt;</td><td></td><td>1: rcx 00000000000000000000000000000000000</td><td></td></kernel32.<>	Process32NextW>		1: rcx 00000000000000000000000000000000000		

# Figure 39

The agent tries to open the local process object using OpenProcess (0x410 = **PROCESS\_QUERY\_INFORMATION | PROCESS\_VM\_READ**):



# Figure 40

For each of the access token extracted from the processes, the executable calls the GetTokenInformation function and retrieves the user account of the token (Figure 41).

00000000297C361 00000000297C366 00000000297C368 00000000297C373	48 89 7C 24 20 44 88 4C 24 50 49 89 C0 8A 01 00 00 00 48 88 4C 24 78	mov qword ptr ss: rsp+20, rdi mov r8d,dword ptr ss: rsp+50 mov r8,rax mov edx,1 mov rck,qword ptr ss: rsp+78		x87r6 0000000000000000000 ST6 Empty 0.000000000000000 x87r7 00000000000000000 ST7 Empty 0.00000000000000000000000000000000000	000
	FF 15 C2 99 00 00	call gword ptr ds:[ <adettokeninformation>]</adettokeninformation>	Ň	Default (x64 fastcall) 🔹 5 🗘 🗌 t	Jnlocke
gword ptr [000000002985D40 <&GetTok	cenInformation>]= <adva< td=""><td>p132.GetTokenInformation&gt;</td><td></td><td>1: rcx 000000000000384 2: rdx 00000000000000</td><td></td></adva<>	p132.GetTokenInformation>		1: rcx 000000000000384 2: rdx 00000000000000	
00000000297C378				3: r8 000000004580050 4: r9 00000000000001C 5: [rsp+20] 0000000001EF854	

# Figure 41

The malware extracts the name of the account for the security identifier (SID) and the first domain on which the SID is found:

00000000297C3A6     00000000297C3A6     00000000297C3AE     00000000297C3AE     00000000297C3B6     00000000297C3B6     00000000297C3B6     00000000297C3B6     000000000297C3B6	48 89 44 24 30 49 89 E9 4C 89 54 24 20 4D 89 D8 4C 89 54 24 58 4C 89 52 24 50 FF 15 82 9C 00 00	<pre>mov qword ptr ss:[rsp+30],rax mov qword ptr ss:[rsp+30],r10 mov qword ptr ss:[rsp+50],r10 mov qword ptr ss:[rsp+50],r11 mov qword ptr ss:[rsp+50],r11 call qword ptr ds:[dkuokupAccountSidw-]</pre>		X8/F5 0000000000000000000000 STS ET x8/F7 0000000000000000000000 STE ET x8/F7 00000000000000000000 ST ET x8/F7 00000000000000000000 ST ET x8/F1 agword FFFF	npty 0.00000000000000000000000000000000000
• <	** **		>	Default (x64 fastcall)	🔻 5 🔹 🗌 Unlocke
qword ptr [000000002986048 <&Lookup 000000000297C3C0	AccountSidW>]= <advapi< td=""><td>32.LookupAccountSidw&gt;</td><td></td><td><pre>- 1: rcx 000000000000000 2: rdx 000000004580060 3: r8 0000000051EF878 4: r9 0000000051EF858 5: [rsp+20] 0000000051EFA80</pre></td><td></td></advapi<>	32.LookupAccountSidw>		<pre>- 1: rcx 000000000000000 2: rdx 000000004580060 3: r8 0000000051EF878 4: r9 0000000051EF858 5: [rsp+20] 0000000051EFA80</pre>	

Figure 42 0xEBC0 ID – Kill processes

The target process is opened via a function call to OpenProcess (0x1 = **PROCESS\_TERMINATE**):

000000000297A2DC     41 89 C0     000000000297A2DF     5 C0     000000000297A2E1     7 4 47     000000000297A2E3     9 01 00 00 00     000000000297A2E3     102	mov r8d,eax test eax,eax je 297-A32A mov eCx,1 xor edx,edx		x87r6 000000000000000000 ST6 x87r7 00000000000000000 ST7 x87TagWord FFFF	Empty 0.00000000000000000000000000000000000
10000000000000000000000000000000000000	0 [call gword ptr ds:[<&OpenProcess>]	×	Defends for the forebase ID	
		>	Default (X64 fastcall)	• D • Uniocke
**	-		1: rcx 000000000000000	
qword ptr [000000002985F98 <&OpenProcess>]= <kerne132.< td=""><td>openProcess&gt;</td><td></td><td>2: rdx 0000000000000000</td><td></td></kerne132.<>	openProcess>		2: rdx 0000000000000000	

# Figure 43

The process is killed using the TerminateProcess API:



#### Figure 44

**0xF584 ID** – Create a new process using the Domain, Username, and Password received from the C2 server

The binary spawns a new process using the CreateProcessWithLogonW method. The parameters are modified according to the command's arguments:

Address	Hey			ASCIT			00000000000000000000000000000000000000	. cmd. exe
Ump 1	Dump 2	Dump 3 🛛 👹 Dum	p 4 📲 Dump 5	🛞 Watch 1 🛛 🕅 🖉 🕅 🕅 🕅	2 Struct		00000000051EFDA0 00000000000000000000000000000000000	"Dom"
qword ptr	[000000000298617 97DA78	8 <&CreateProce	ssWithLogonW>]=	<advap132.createproces< th=""><th>swithLogonw&gt;</th><th></th><th>1: rcx 000000004580960 L"User" 2: rdx 000000004580940 L"Dom" 3: r8 000000004580980 L"Pass" 4: r9 00000000000001 5: [rsp+20] 0000000004580050</th><th></th></advap132.createproces<>	swithLogonw>		1: rcx 000000004580960 L"User" 2: rdx 000000004580940 L"Dom" 3: r8 000000004580980 L"Pass" 4: r9 00000000000001 5: [rsp+20] 0000000004580050	
30-	COUNTRY     COUNTRY	197DA1F         48           197DA2A         48           197DA2D         48           197DA3D         48           197DA3A         48           197DA3A         48           197DA3A         48           197DA3A         48           197DA3A         48           197DA5A         48           197DA5A         48           197DA5A         46           197DA5A         48           197DA5A         46           197DA6A         41           197DA6A         48           197DA6B         48           197DA78         FF	89         6C         24         50           C7         44         24         40         00           89         44         24         48         88         84         24         98         00         14         24         88         84         24         98         00         14         24         30         00         10         14         24         30         00         10         14         24         30         00         14         24         30         10         12         12         12         14         20 <td>New quord ptr s:: may quord ptr s:: may quord ptr s:: may quord ptr s:: 00 moy rax, quord ptr s: 00 moy quord ptr s: may quord ptr s: 00 moy quord ptr s: 00 moy rax, quord ptr s: 00 moy rax, quord ptr s: 00 moy rax, quord ptr s: 00 moy rcx, quord ptr s: 00 moy rcx,</td> <td>spFs0, rb5 spH=0, 0 spH=0, 0 spH=0, 0 spH=0, 0 spH=0, 0 spH=0, 0 spH=0, 0 spH=0, 0 spH=0, 0 sH=spH=0, 0 sH=spH=spH=0, 0 sH=spH=spH=spH=spH=spH=spH=spH=spH=spH=s</td> <td>[rsp+88; [rsp+76; [rsp+80]</td> <td>x87r0 00000000000000000000000000000000000</td> <td>mpty         0.00000000000000000000000000000000000</td>	New quord ptr s:: may quord ptr s:: may quord ptr s:: may quord ptr s:: 00 moy rax, quord ptr s: 00 moy quord ptr s: may quord ptr s: 00 moy quord ptr s: 00 moy rax, quord ptr s: 00 moy rax, quord ptr s: 00 moy rax, quord ptr s: 00 moy rcx,	spFs0, rb5 spH=0, 0 spH=0, 0 spH=0, 0 spH=0, 0 spH=0, 0 spH=0, 0 spH=0, 0 spH=0, 0 spH=0, 0 sH=spH=0, 0 sH=spH=spH=0, 0 sH=spH=spH=spH=spH=spH=spH=spH=spH=spH=s	[rsp+88; [rsp+76; [rsp+80]	x87r0 00000000000000000000000000000000000	mpty         0.00000000000000000000000000000000000

#### Figure 45

**0xBED0 ID** – Execute the "open", "runas", or "print" command

The first parameter is compared with the above commands, as shown in Figure 46.



We could use the runas command to spawn a cmd.exe process:

	00000000297E9DA     00000000297E9DB     00000000297E9EB     00000000297E9EB     000000000297E9F9     000000000297E9F9     000000000297EADA     00000000297EADA	48         8D         4C         24         58         1e           48         C7         84         24         80         00         00         mo           C7         84         24         80         00         00         mo           48         C7         84         24         80         00         00         mo           48         85         C0         00         00         mo         46         67         84         24         90         00         00         mo           48         85         C0         00         00         mo         46         69         44         24         90         00         00         mo           48         89         44         24         76         mo         56         mo         56	a rCx.eword ptr ss:[rcps5] v dword ptr ss: rcp+80,0 strax,rCx v eax,0 v eax,0 v qword ptr ss:[rcp+90],0 owne rax,rI3 v qword ptr ss:[rcp+70],rax		x87r4         000000000000000000000000000000000000	Empty 0.00000000000000000000000000000000000
RIP	000000002978418	FF 15 5F 73 00 00 Ca	word ptr ds:[<&shellExecuteExa>]	, i i i i i i i i i i i i i i i i i i i	Default (x64 fastcall)	▼ 5 🗘 🗌 Unlocka
qword ptr	[000000002985D78 <&She] 97EA13	1ExecuteExA>]= <she1132.she1< td=""><td>1ExecuteExA&gt;</td><td></td><td>1: rCx 00746E6972700000 3: r8 00000000000064 4: r9 FFFFFFFFFFFF78 5: [r5p+20] 00000000000000000000000000000000000</td><td></td></she1132.she1<>	1ExecuteExA>		1: rCx 00746E6972700000 3: r8 00000000000064 4: r9 FFFFFFFFFFFF78 5: [r5p+20] 00000000000000000000000000000000000	
Utu Dump 1	Dump 2 Dump 3	🗱 Dump 4 📲 Dump 5 👹	Watch 1 🛛 🗐 Locals 🎾 Struct		00000000000000000000000000000000000000	600 F80
Addr ess 0000000005 0000000005 0000000005 000000	Hex           1EFC68         70         00         00         00         00           1EFC78         CPD         1E         05         00         00           1EFC78         O         00         00         00         00         00           1EFC38         O         00         00         00         00         00         10           1EFC38         O         00         00         00         00         00         10           1EFC38         00         00         00         00         00         00         10           1EFC48         00         00         00         00         00         00         10           1EFC58         00         00         00         00         00         00         10           1EFC68         56         64         00         00         00         00         00           1EFC08         56         64         00         00         00         00         00         00         00         00         00         00         00         00         00         00         00         00         00         00         00	$ \begin{array}{cccccccccccccccccccccccccccccccccccc$	ASCII 00 B	^	00000000051EFC20         0000000000000           00000000051EFC20         000000000000           00000000051EFC30         000000000000           0000000051EFC30         000000000000           0000000051EFC30         000000000000           0000000051EFC30         000000000000           0000000051EFC36         6167572006E6           0000000051EFC36         00746E8972700           0000000051EFC56         00746E8972700           0000000051EFC56         000000000000	000 000 002 000 000 000 000 000 000 000

GetProcessId is utilized to obtain the PID of the newly created process:

>e         000000000297EA2D         48         88         82         24         CO         000         mov rcx,qword ptr sst[rsp+C0]           e         000000000297EA35         48         85         C9         test rcx,rcx           e         000000000297EA35         48         85         C9         test rcx,rcx           f         297EA68         74         25         jf         297EA68	x87Tagword FFFF	18-14 A
CODDDDDDD2572584 FF 15 58 71 00 00 call qword ptr.ds:[≪46etProcessId>]	Default (x64 fastcall)	▼ 5 🗘 🗌 Unlocke
<pre>qword ptr [000000002985898 &lt;&amp;GetProcessId&gt;]=<kernel32.getprocessid></kernel32.getprocessid></pre>	2: rdx 0000000000000000	

#### Figure 48

**0xE2EA ID** – Copy bytes into memory

The second parameter is Base64-decoded by calling the CryptStringToBinaryA API:

	D00000002978CEC     000000002978CEE     000000002978CF3     000000002978CF3     000000002978CF9     000000002978CFC     000000002978D05	31 D2 48 89 74 24 20 41 88 01 00 00 00 4C 89 E9 48 C7 44 24 30 00 00 48 C7 44 24 28 00 00	xor edx,edx mov qword ptr ss:[rsp+20],rs1 mov r8d,1 mov rcx,r13 mov qword ptr ss: rsp+30,0 mov qword ptr ss: rsp+30,0	rcx: "QU	x8/r5 000000000000000000000000000000000000	. 000000000000000000000000000000000000
RIP	00000000297600E	FF 15 3C D5 00 00	call gword ptr ds:[<&CryptstringToBinaryA>]	, ×	Default (x64 fastcall)	▼ 5 C Unlocke
qword ptr	[0000000002986250 <&Crypts 2978D0E	itringToBinaryA≻]= <crypt< td=""><td>32.CryptStringToBinaryA&gt;</td><td></td><td>1: rCX 00000000045808A0 "QUIDREVGRUG" 2: rdX 00000000000000 3: r8 00000000000001 4: r9 000000004580860 5: [r5p+20] 0000000051EFE9C</td><td></td></crypt<>	32.CryptStringToBinaryA>		1: rCX 00000000045808A0 "QUIDREVGRUG" 2: rdX 00000000000000 3: r8 00000000000001 4: r9 000000004580860 5: [r5p+20] 0000000051EFE9C	

#### Figure 49

The address containing the resulting bytes is stored in a table that contains functions pointers (see Figure 50).

Addres	55	He	x		1.2	2				zn.			36	2			ASCII	[				
000000	00002985D28	60	08	5 B	04	00	00	00	00	80	24	5A	AE	F8	7F	00 00	· [ · ·		\$Z@Ø			
000000	00002985D38	EO	25	5A	AE	F8	7F	00	00	10	5E	83	AF	F8	7F	00 00	a%Z@@	3	^. ø			
000000	00002985D48	40	DD	58	AE	F8	7F	00	00	00	BA	58	AE	F8	7F	00 00	@ÝX80	3	°X8Ø	•		
000000	00002985D58	00	BA	76	9E	F8	7F	00	00	90	15	3B	AF	F8	7F	00 00	.ºv.0		.; Ø	. I	rigure	e 50
000000	00002985D68	80	15	70	9E	F8	7F	00	00	40	62	58	AE	F8	7F	00 00	p.e	J@	bX⊖ø	-	-	
000000	00002985D78	60	53	A8	AF	F8	7F	00	00	FO	33	07	A9	F8	7F	00 00	SQ	ōō	3.00	•		
000000	00002985D88	A0	3E	71	9E	F8	7F	00	00	60	7C	5 B	AE	F8	7F	00 00	>q.@		[°Ø	•		
000000	00002985D98	DO	58	AB	93	F8	7F	00	00	00	00	00	00	00	00	00 00	DX«.0	3		-		

Depending on the number of bytes, the malware will send the "[+] Imported %d bytes" message to the C2 server:

00000000297C582 00000000297C582 000000000297C582 00000000297C582 00000000297C582 00000000297C582 000000000297C585	48 8D 4C 24 20 85 C0 41 89 C0 74 07 48 8D 15 7F 5C 00 00 E8 86 32 00 00	lea rcx,qword ptr ss:[rsp+20] test eax,eax mov r8d,eax je 297C595 lea rdx,qword ptr ds:[2982214] call 297F850	rdx:L"[]	x87r6         000000000000000000000000000000000000
00000000297F850			3	1: rcx 00000000051EFE0 2: rdx 00000000002982214 L"[+] Imported %d bytes" 3: r8 000000000000000

### Figure 51

**0x6154 ID** – Free the pointer storing the address of the imported bytes

The agent calls the free function with the pointer displayed in the above command. The message shown below is transmitted to the C2 server.

```
.text:00000000297C492 mov rcx, cs:qword_2985D28 ; Memory
.text:00000000297C499 call free
.text:00000000297C49E mov rcx, cs:qword_2986000
.text:00000000297C485 mov cs:qword_2985D28, 0
.text:00000000297C480 call cs:qword_2985D30
.text:00000000297C486 lea rcx, [rsp+38h+Memory]
.text:00000000297C48B lea rdx, aImportCleared ; "[+] Import cleared"
.text:00000000297C42 call sub_297F850
Ox699A ID - Create a TCP listener
```

The process creates a new thread that is responsible for the listener creation:

36	00000000237F034     0000000237F026     00000000237F028     00000000237F028     00000000237F038     00000000237F038     00000000237F038     00000000237F038     00000000237F048     00000000237F048     00000000237F048     00000000237F058     00000000037F058     0000000037F058     000000037F058     0000000037F058     00000000000000000000000000000000	31         D2         xor edx,edx           F3         A8         Fepe stood           F3         A8         Fepe stood           F3         A8         Fepe stood           F3         A8         Fepe stood           F4         Fepe stood         Ferge stood           F4         F8         A8         Fepe stood           F5         A8         F4         F4         Ferge stood           F6         F7         FF         FF         FE         Ferge stood           F6         F4         F5         Ferge stood         Fa         Ferge stood         Fa           F6         F4         F5         Ferge stood         Fa         Ferge stood         Fa           F6         F4         F5         Ferge stood         Fa         Ferge stood         Fa           F6         F5         F5         Ferge stood         Fa         Fa	x87r1         000000000000000000000000000000000000	00000000000000000000000000000000000000
gword ptr	[0000000002985FE8 <&Creat	eThread>]= <kernel32.createthread></kernel32.createthread>	1: rcx 000000000000000 2: rdx 0000000000000 3: r8 0000000002973820 4: r9 00000000000000000000000000000000000	

### Figure 53

It calls the getaddrinfo method with the port number and the first parameter being NULL, which returns all registered addresses on the local machine:

00000000029738A5     0000000029738A5     0000000029738A5     00000000029738A5     00000000029738A5     0000000029738A5     00000000029738A5     00000000029738A5     00000000029738A5     00000000029738A5	4C       8D       4C       8D       1 aa r9.qmord ptr ds:[rbsw6]         4S       8S       50.8       mov rdx.qmord ptr ds:[rbsw6]         4C       8D       42       78       1 aa r8.qmord ptr ds:[rbsw7]         F3       A6       repe stosd       repe stosd         48       85       01       00       00       02       mov rax.20000001         48       85       01       00       00       00       mov qword ptr ds:[rsp+78], rax         48       85       01       00       00       00       mov qword ptr ss:[rsp+78], rax         48       84       24       80       00       00       00       mov qword ptr ss:[rsp+78], rax	rdx: "88: x875 000000000000000000000000000000000000
	earl gword ptr ds: [«aget addr into»]	> Default (x64 fastcall)
qword ptr [000000002985820 <&get	addrinfo>]= <ws2_32.getaddrinfo></ws2_32.getaddrinfo>	2: rdx 0000000045808A0 "8888" 3: r8 0000000538EA8 4: r9 0000000532E90

# Figure 54

The badger creates a TCP socket (0x2 = **AF\_INET**, 0x1 = **SOCK\_STREAM**, 0x6 = **IPPROTO\_TCP**):

00000000297390F 000000002973912 000000002973915	88 48 04 88 50 08 44 88 40 0C	<pre>mov ecx,dword ptr ds:[rax+4] mov edx,dword ptr ds:[rax+8] mov r8d,dword ptr ds:[rax+C]</pre>		x87Tagword FFFF	a
	FF 15 19 26 01 00	Carl gword per ds:[casockets]	>	Default (x64 fastcall)	🔻 💈 🖨 Unlocke
qword ptr [000000002985F38 <&socke	t>]= <ws2_32.socket></ws2_32.socket>			1: rcx 000000000000002 2: rdx 000000000000000 3: r8 000000000000000	

# Figure 55

The bind function is used to associate the local address with the socket, as highlighted below:

	• 00000 • 00000 • 00000	00002973964 00002973969 0000297396D	48 88 4 48 88 5 44 88 4	4 24 60 0 20 0 10	mov rdx,0 mov rdx,0	word ptr s word ptr d word ptr d	rsp+60 [rax+20] [rax+10]		x87Tagword FFFF
RIP	00000	00002973971	FF 15 1	1 23 01 00	call qwor	d ptr ds:[	(&b1nd>]	2	Default (x64 fastcall) 🔹 5 🔹 🗌 Uniodi
qword ptr	00000000029	85C88 <&bind	>]= <ws2_32.< th=""><th>bind&gt;</th><th></th><th></th><th></th><th></th><th>1: rcx 000000000000000204 2: rdx 00000000012E40 3: r8 00000000000000 4: r9 000000000000000 5: [rspr20] 00000000000000000000000000000000000</th></ws2_32.<>	bind>					1: rcx 000000000000000204 2: rdx 00000000012E40 3: r8 00000000000000 4: r9 000000000000000 5: [rspr20] 00000000000000000000000000000000000
Dump 1	Dump 2	Ump 3	Ump 4	Ump 5	💮 Watch 1	Ix=I Locals	2 Struct		00000000053E8E30 0000000051EFE88 & 80.80.80.80.8888" 00000000053E8E38 0000000053E8F10
Addr ess	Hex	22 B8 00 00		0 00 00 00 0	ASC 00 00	11		^	00000000053E8E40 <b>E0000000053E8ED8</b> 00000000053E8E48 000000000109490 & .\r"

### Figure 56

The malware starts listening on the port specified in the command's arguments (in our case, 8888):

000000	0002973988 48 8 000297398F BA F	B 4B 10 F FF FF 7F	ov rcx, gword ptr ds:[rbx+10] ov edx,7FFFFFFF	x87Tagword FFFF	
34(2)	0002973964 FF 1	5 F6 27 01 00	all gword ptr ds:[<&listen>]	 Default (x64 fastcall)	▼ 5 ¢ Unlocke
qword ptr [00000000298	61C0 <&listen>]= <ws< td=""><td>2_32.1isten&gt;</td><td></td><td>1: rcx 000000000000204 2: rdx 000000007FFFFFF</td><td></td></ws<>	2_32.1isten>		1: rcx 000000000000204 2: rdx 000000007FFFFFF	

# Figure 57

Finally, the accept method is utilized to allow incoming connection attempts (Figure 58).

000000002973A57     000000002973A58     000000002973A58     000000002973A68     000000002973A68	48 88 48 10 C7 44 24 50 10 00 00 4C 88 44 24 38 48 88 54 24 40	<pre>mov rcx,qword ptr ds:[rbx+10] mov dword ptr ss:[rsp+50],10 mov rs,qword ptr ss:[rsp+80] mov rdx,qword ptr ss:[rsp+40]</pre>		x87r7 0000000000000000000000 ST7 E x87TagWord FFFF	mpty 0.000000000000000000
000000002973A6D	FF 15 35 25 01 00	call gword ptr ds:[<&accept>]	>	Default (x64 fastcall)	▼ 5 💠 🗌 Unlocke
qword ptr [000000002985FA8 <&acc	ept>]= <ws2_32.accept></ws2_32.accept>			1: rCx 00000000000000204 2: rdx 0000000053EBE98 3: r5 0000000053EBE98	

The IP address from the connection is converted into an ASCII string in dotted-decimal format:



#### Figure 59

A new thread that handles the receive operation is created:

	<ul> <li>000000002973AF2</li> <li>000000002973AF4</li> <li>000000002973AF9</li> <li>000000002973801</li> <li>000000002973803</li> <li>000000002973808</li> <li>000000002973812</li> <li>000000002973815</li> <li>000000002973815</li> <li>000000002973815</li> </ul>	31 D2 48 80 44 24 54 48 89 AC 24 C0 00 0 31 C9 4C 80 84 24 C8 00 0 48 C7 44 24 58 00 0 48 C7 44 24 28 C7 44 24 20 00 00 0	xor edx,edx learax,word ptr ss:[rsp+54] 0 mov qword ptr ss:[rsp+00],rbp xor edx,ecx 0 mov qword ptr ss:[rsp+20],r5 0 lear68,qword ptr ss:[2073700] mov qword ptr ss:[rsp+20],0 1 mov dword ptr ss:[rsp+20],0 1	[rsp+C8	x8773 0000000000000000000000 514 x8774 000000000000000000000 515 x8775 000000000000000000000 515 x8776 00000000000000000000 516 x8777 0000000000000000000 517 x877agword FFFF	Empty 0.000000000000000000 Empty 0.00000000000000000 Empty 0.00000000000000000 Empty 0.00000000000000000 Empty 0.00000000000000000000000000000000000
RIP	000000002973828	FF 15 BA 24 01 00	<pre>call gword ptr ds:[&lt;&amp;CreateThread&gt;]</pre>	v	Default (x64 fastcall)	▼ 5 ‡ Unlocke
word provided and	tr [0000000002985FE8 <&Create 002973828 IFE 60	Thread>]= <kernel32.cr< td=""><td>eateThread&gt;</td><td></td><td>1: rCX 000000000000000 2: rCX 0000000000000 3: r8 0000000002973PD0 4: r9 0000000058ED8 5: [rsp+20] 0000000000000000000</td><td></td></kernel32.cr<>	eateThread>		1: rCX 000000000000000 2: rCX 0000000000000 3: r8 0000000002973PD0 4: r9 0000000058ED8 5: [rsp+20] 0000000000000000000	
	<ul> <li>000000002974076</li> <li>000000002974079</li> <li>000000000297407F</li> <li>0000000002974082</li> </ul>	45 31 C9 41 B8 00 10 00 00 48 89 EA 4C 89 E1	xor r9d,r9d mov r8d,1000 mov rdx,rbp mov rcx,r12		x87r7 000000000000000000 ST7 x87Tagword FFFF	Empty 0.00000000000000000
		10 53 03 14 01 00	and dealer by an only an ectry		Default (x64 fastcall)	▼ 5 \$ Unlocke
qword p	tr [000000002985B40 <&recv>]	= <ws2_32.recv></ws2_32.recv>			1: rcx 0000000000000378 2: rdx 0000000055EEF1F 3: r8 000000000001000	

Figure 61

**0xB458 ID** – Extract information about Windows services

The binary opens the service control manager on the local machine using OpenSCManagerA (0x4 = **SERVICE\_QUERY\_STATUS**):

<ul> <li>00000000297E311</li> <li>00000000297E316</li> <li>00000000297E316</li> <li>00000000297E31E</li> </ul>	48         88         4C         24         48         mov rcx, gword ptr ss: [rsp+48]           41         88         04         00         00         mov r8d,4           31         D2         cor edx,edx         cr ave edx,edx           C7         80         6C         80         00         00         mov r8d,4		x87r7 000000000000000000 ST7 8	Empty 0.0000000000000000000
00000000297E328 <	FF 15 72 7E 00 00 [call gword ptr ds: [<&OpenSCManagerA>	·] //··································	Default (x64 fastcall)	▼ 5 🗘 🗌 Unlocke
qword ptr [0000000029861A0 <&OpenS	CManagerA>]= <advap132.0penscmanagera></advap132.0penscmanagera>		2: rdx 000000000000000 3: r8 000000000000000	

### Figure 62

EnumServicesStatusW is used to enumerate all services in the database (0x30 = **SERVICE\_WIN32**, 0x3 = **SERVICE\_STATE\_ALL**):



#### Figure 63

For every service, the malware calls the OpenServiceW API (0x1 = **SERVICE\_QUERY\_CONFIG**):



Figure 64

The agent extracts the configuration parameters of the service using QueryServiceConfigW. The following fields are relevant: display name, service name, service state, service path, service user, and service type.

00000000297E472     00000000297E477     000000000297E477     00000000297E47F	4C 8B 4C 24 58 44 8B 44 24 54 48 89 C2 4C 89 F1	<pre>mov r9,qword ptr ss:[rsp+58] mov r8d,dword ptr ss:[rsp+54] mov rdx,rax mov rcx,r14</pre>		x87r7 0000000000000000000 ST7 E	mpty 0.000000000000000000
000000000297E482	FF 15 88 74 00 00	call gword ptr ds:[<&QueryServiceConfigw>]		Default (x64 fastcall)	👻 5 ≑ 🗌 Unlocke
word ptr [000000002985940 <&QueryServiceConfigm>]= <advapi32.queryserviceconfigm></advapi32.queryserviceconfigm>				1: rcx 0000000000112040 2: rdx 000000000058240 3: r8 000000000000176	

# Figure 65

**0xE3CB ID** – Retrieve information about Domain Controllers and policies

The malicious executable obtains the name of a domain controller via a function call to DsGetDcNameW, as displayed in Figure 66.



### Figure 66

The DsGetDcOpenW API is utilized to open a new domain controller enumeration operation (0x2 = **DS\_NOTIFY\_AFTER\_SITE\_RECORDS**):

	<ul> <li>000000002978D85</li> <li>000000002978D88</li> <li>000000002978D88</li> <li>000000002978D90</li> <li>000000002978D94</li> <li>000000002978D94</li> <li>000000002978D94</li> <li>000000002978D91</li> <li>000000002978D31</li> </ul>	45         31 C9         xor r9d,r9d           45         31 C0         xor r8d,r8d           45         21 C0         xor r8d,r8d           46         80 00 00         000 veds,s           47         82 86         82           48         80 44 24 58         Tear xa, qoord ptr ss: [rsp=55]           7         42 42 80 00 00 mov dword ptr ss: [rsp=76],0           48 89 44 24 30 00 mov qword ptr ss: [rsp=30], rax           48 7 44 24 20 00 mov qword ptr ss: [rsp=30], rax	rcx: "Dor	Xarra 0000000000000000000000000000000000
RIP	000000002978DAF	FF 15 23 D4 00 00 call gword ptr ds: [<&DsGetDcOperw	»	Default (x64 fastcall) 👻 5 🔹 🗌 Unlocke
qword ptr	+ [2 [00000000029861D8 <&DsGett 978DAF	)cOpenw>]= <logoncli.dsgetdcopenw></logoncli.dsgetdcopenw>	,	1: rcx 0000000029F1378 "Domain" 2: rdx 00000000000000 3: r8 00000000000000 4: r9 000000000000000 5: rfsp-20 0000000000000

### Figure 67

The badger extracts the global password parameters and lockout information by calling the NetUserModalsGet function. The information is organized using the following structure:



0x0105 ID – Extract data from the clipboard

The process opens the clipboard by calling the OpenClipboard method:



#### Figure 69

The data is obtained from the clipboard in the Unicode format (0xD = CF\_UNICODETEXT):

	000000002979292	B9 0D 00 00 00	mov ecx,D	00: '\r'		
313	<	FF 15 D3 C8 00 00	call gword ptr ds:[<&GetC []pooardData>]	>	Default (x64 fastcall)	▼ 5 🗘 🗌 Unlocke
gword ptr [	000000002985B70 <&GetC1	ipboardData>]= <user32.< th=""><th>GetClipboardData&gt;</th><th></th><th>2: rdx 0000000000000000</th><th></th></user32.<>	GetClipboardData>		2: rdx 0000000000000000	
Liguro	70					

#### Figure 70

0x0B06 ID - Convert the time of the last input event in minutes

The binary obtains the number of milliseconds elapsed since the system was started using GetTickCount:



GetLastInputInfo is used to retrieve the time of the last input event:



Figure 72

**0xB63A ID** – Block keyboard and mouse input events

The BlockInput method is used to perform the operation, as displayed in the figure below.

000000000297A8E5 B9 01 00 00     00000000297A8E4 46 20 6C 24 28     00000000297A8EF 48 C7 44 24 28 00	mov ecx,1 lea r13,qword ptr ss:[rsp+28] on mov qword ptr ss:[rsp+28],0		x87Tagword FFFF	
00000000079745F8 FF 15 44 83 00 00	call gword ptr ds:[<&BlockInput>]	>	Default (x64 fastcall) 1: rcx 000000000000000	▼ 5 😫 🗌 Unlacke
<pre>qword ptr [000000002985C48 &lt;&amp;8lockInput&gt;]=<user32.bloc< pre=""></user32.bloc<></pre>	(Input>		2: rdx 000000002986400	

Figure 73

**0x0391 ID** – Lock the workstation's display

LockWorkStation is utilized to lock the display (see Figure 74).

RIP 00000000297A9E2	FF 15 30 B3 00 00	<pre>call qword ptr ds:[&lt;&amp;LockWorkStation&gt;]</pre>			
× <			>		
<pre>qword ptr [000000002985D18 &lt;&amp;LockWorkStation&gt;]=<user32.lockworkstation></user32.lockworkstation></pre>					
Figure 74					

**0xF999 ID** – Impersonate the context of a logged-on user

The badger attempts to log a user on to the local machine via a call to LogonUserA (0x2 = LOGON32\_LOGON\_INTERACTIVE):

	C7 44 24 20 00 00 00 mov rbd,5 41 89 00 00 00 00 mov rbd,5 89 44 24 28 mov rbd,5 C7 89 54 24 28 mov rbd,5 40 89 50 mov rd,712 40 89 50 mov rd,715	09: '\t' rcx: "Us: r8: "Pas: rdx: "Do:	X8/75 0000000000000000000000 SIS E X8776 0000000000000000000 SIS E X8777 0000000000000000000 SIT E X87Tagword FFFF	mpty 0.00000000000000000000000000000000000
	FF 15 61 AA 00 00 [call qword ptr ds:[<&LogonUserA>]		Default (x64 fastcall)	▼ 5 🗣 🗌 Unlocks
qword ptr [00000000029858C0 <&Logor 0000000002978159	UserA>]= <advap132.logonusera></advap132.logonusera>		1: rcx 0000000045808C0 "User" 2: rdx 0000000045808A0 "Dom" 3: r8 0000000045808E0 "Pass" 4: r9 0000000000000002 5: [rsp+20] 0000000000000000	

### Figure 75

The binary impersonates the context of the above user using the ImpersonateLoggedOnUser function:



Figure 76

**0xA959 ID** – Retrieve information about users

The first parameter is compared with "user" and "users". In the first case, the malware calls the NetUserGetInfo API to obtain information about the user account:





The information is organized in the following manner:

```
.text:000000002976A9A mov
                                     [rsp+328h+var_290], eax
                                     rax, aNumberOfLogons ; "Number of logons"
.text:0000000002976AA1 lea
.text:000000002976AA8 mov
                                   [rsp+328h+var 298], rax
.text:000000002976AB0 mov
                                 eax, [rcx+80h]
                                 [rsp+328h+var_2B0], rsi
[rsp+328h+var_2A0], eax
.text:0000000002976AB6 mov
.text:0000000002976ABB mov
                                  rax, aB
                                                      ; "B"
.text:0000000002976AC2 lea
.text:000000002976AC9 mov
                                 [rsp+328h+var_2A8], rax
                                 rax, aLastLogon ; "Last logon"
.text:0000000002976AD1 lea
                                  [rsp+328h+var 2B8], rax
.text:0000000002976AD8 mov
                                eax, [rcx+10h]
.text:0000000002976ADD mov
.text:000000002976AE0 mov [rsp+328h+var_2D0], rbx
                                    r8d
.text:0000000002976AE5 div
.text:000000002976AE8 lea r8, aUserName ; "User name"
.text:000000002976AEF lea rdx, asc_29824EE ; "%"
.text:000000002976AEF mov [rsp+328h+var_2C0], eax
                                                                                          Figure 78
.text:000000002976AFA lea rax, aPasswordLastSe ; "Password last set"
.text:000000002976B01 mov [rsp+328h+var_2C8], rax
.text:000000002976B06 lea rax, aAccountExpires ; "Account expires"
.text:0000000002976B0D mov [rsp+328h+var_2D8], rax
.text:0000000002976B12 mov eax, [rcx+90h]
.text:0000000002976B18 mov [rsp+328h+var_2E0], eax
.text:0000000002976B1C lea rax, aCountryRegionC ; "Country/region code"
                                 [rsp+328h+var_2E8], rax
.text:0000000002976B23 mov
                                rax, [rcx+20h]
.text:0000000002976B28 mov
.text:000000002976B2C mov [rsp+328h+var_2F0], rax
.text:000000002976B31 lea rax, aComment ; "Comment"
                                   [rsp+328h+var_2F8], rax
.text:0000000002976B38 mov
.text:0000000002976B3D mov rax, [rcx+40h]
.text:0000000002976B41 mov [rsp+328h+var_300], rax
.text:0000000002976B46 lea rax, aFullName ; "Full name"
.text:000000002976B4D mov
                                     [rsp+328h+var_308], rax
```

In the second case, the agent retrieves information about all user accounts on the local computer (0x2 = **FILTER\_NORMAL\_ACCOUNT**):

<ul> <li>000000002976C38</li> <li>000000002976C30</li> <li>000000002976C4</li> <li>000000002976C4</li> <li>000000002976C4</li> <li>000000002976C4</li> <li>000000002976C54</li> <li>000000002976C54</li> <li>000000002976C54</li> <li>000000002976C54</li> <li>000000002976C54</li> <li>000000002976C54</li> <li>0000000002976C54</li> <li>0000000002976C54</li> </ul>	48         8B         4C         24         60           4C         8D         4C         24         58           C7         44         24         20         FF         FF         FI           48         8D         44         24         38         0         40         00         00         00           41         88         02         00         00         00         00         48         89         44         24         30         48         40         48         44         40         48         89         44         24         20         48         89         44         24         28         48         89         44         24         28         36         46         48         89         44         28         28         36	<pre>mov rex.qword ptr sst[rsp+60] Hear9,qword ptr sst[rsp+60] Fmov qword ptr sst[rsp+60],FFFFFFF mov qword ptr sst[rsp+60],rax Hearrax,qword ptr sst[rsp+60] mov rdd,2 mov qword ptr sst[rsp+30],rax Hearrax,qword ptr sst[rsp+60] mov qword ptr sst[rsp+80],rax</pre>		x87r2         000000000000000000000000000000000000	y 0.00000000000000000000000000000000000
000000002976CCE	FF 15 4C F0 00 00	Call gword ptr os:[<@NetUserEnum>]	a last a flore	Default (x64 fastcall)	▼ 5 ¢ 🗌 Unlocke
qword ptr [0000000002985D20 <&NetUse	erEnum>]= <samcli.netus< td=""><td>er Enum&gt;</td><td></td><td>1: rcx 0000000004580F00 2: rdx 00000000000002 3: r8 000000000000002 4: r9 000000051EF038 5: [rse20] 00000005FFFFFFF</td><td></td></samcli.netus<>	er Enum>		1: rcx 0000000004580F00 2: rdx 00000000000002 3: r8 000000000000002 4: r9 000000051EF038 5: [rse20] 00000005FFFFFFF	

Figure 79

0x6C36 ID - Extract registry keys and values

The first argument can be "hklm", "hkcu", "root", "config", and "users". These are Windows registry hives.

The registry key passed as the second argument is opened using the RegOpenKeyExA method (0x20019 = **KEY\_READ**):

<ul> <li>00000000297C808</li> <li>00000000297C80E</li> <li>00000000297C811</li> <li>00000000297C814</li> <li>00000000297C819</li> </ul>	41         B9         19         00         02         mov r9d,20019           45         31         C0         xor r8d,r8d           42         89         EA         mov r0d,r13           48         89         44         24         20         mov qword ptr sst           48         C7         C1         03         00         080         mov rcx, FFFFFFF86	rsp+20 <b>]]</b> , rax 300003	x8/F6 0000000000000000000 S16 x87r7 00000000000000000000 S77 x87Tagword FFFF	Empty 0.00000000000000000000000000000000000
00000000297CB20 <	FF 15 EA 96 00 00 Call gword ptr ds:	[ <dregopenkeyexa>]</dregopenkeyexa>	Default (x64 fastcall)	▼ 5 ¢ Unlock
eax=0000000051E7AA8			1: rCx FFFFFFF80000001 3: r8 0000000045B0860 "AppEven 3: r8 00000000000000000 4: r9 000000000000000	nts"

The malicious process retrieves information about the registry key by calling the RegQueryInfoKeyW function:

1.1	000000000297C85A	45 31 C9	or r9d,r9d		CE O TE O TE I	
3.3	000000000297C85D	48 80 84 24 9C 00 00 1	ea rax, gword ptr ss: [rsp+9C]		CFO IFO IFI	
	00000000297C865	48 80 9C 24 E4 00 00 1	ea rbx, gword ptr ss: [rsp+E4]			
102.0	000000000297CB6D	48 89 44 24 50 m	ov gword ptr ss: rsp+50, rax		Lasterror 00000057 (ERROR_INVALI	D_PARAMETER)
3.3	000000000297CB72	48 80 84 24 98 00 00 1	ea rax, gword ptr ss: [rsp+98]		LastStatus 00000000 (STATUS_SUCCE	(\$5)
	000000000297CB7A	4C 8D 84 24 80 00 00 1	ea r8, gword ptr ss: [rsp+80]			
3.1	000000000297C882	48 88 8C 24 D8 00 00 m	ov rcx, gword ptr ss: [rsp+D8]		GS 0028 FS 0053	
11	000000000297CB8A	48 89 44 24 48 n	ov gword ptr ss: rsp+48, rax		ES 0028 DS 0028	
1.1	000000000297CB8F	48 80 84 24 94 00 00 1	ea rax, gword ptr ss: [rsp+94]		C5 0033 55 002B	
11	000000000297CB97	48 80 94 24 EA 02 00 1	ea rdx, qword ptr ss: rsp+2EA			
	000000000297CB9F	48 89 44 24 40 n	ov qword ptr ss: [rsp+40], rax		x87r0 0000000000000000000 STO Em	pty 0.000000000000000000000
	000000000297CBA4	48 80 84 24 90 00 00 1	ea rax, qword ptr_ss:[rsp+90]		x87r1 000000000000000000 ST1 Fm	nty 0.000000000000000000000
4.4	000000000297CBAC	48 89 44 24 38 n	ov gword ptr ss: rsp+38, rax		x87r2 000000000000000000 ST2 Fm	nty 0.00000000000000000000
	000000000297C881	48 80 84 24 8C 00 00 1	ea rax, qword ptr_ss: [rsp+60]		x87r3 000000000000000000 ST3 Fm	nty 0.00000000000000000000000000000000000
	000000000297CB89	48 89 44 24 30 m	ov gword ptr ss: rsp+30, rax		x87r4 0000000000000000000 ST4 E	nty 0 0000000000000000000000000000000000
	000000000297CB8E	48 80 84 24 88 00 00 1	ea rax, qword ptr_ss:[rsp+88]		x87r5 000000000000000000000000000000000000	pty 0.00000000000000000000000000000000000
	00000000297CBC6	48 89 44 24 28 n	ov gword ptr ss: rsp+28, rax		x8715 000000000000000000 STS E	pty 0.00000000000000000000000000000000000
	00000000297CBCB	48 80 84 24 84 00 00 1	ea rax, gword ptr ss: [rsp+84]		x87F6 000000000000000000000000000000000000	pty 0.00000000000000000000000000000000000
	000000000297C8D3	C7 84 24 84 00 00 00 m	ov dword ptr ss: rsp+84 ,0		x8/F/ 00000000000000000000000000000000000	bth 0.00000000000000000000000000000000000
	• 00000000297CBDE	C7 84 24 A0 00 00 00 m	ov aword ptr ss: rsp+Aur, 3FFF			
3.1	000000000237CBE3	48 89 5C 24 58 m	ov gword ptr ss: rsp+sa, rox		x87Tagword FFFF	
1111	000000000297CBEE	40 07 44 24 20 II	all award ate drift Page any ary to for a whole			
N.L.P.	00000000257CBFS	11 00 01 04 00 00	art duoin bei uss [sakegquei yin ukeyny]	Ŷ	Default (v64 factual)	T E Linkets
11.	* <			>	Default (xo4 fastcall)	• J • C Onloca
award ate f	000000000000000000000000000000000000000	arutofokauto1-zaduani22 B	annuaryInfoliates		1: rcx 00000000000000000	
duor a per l	oppoppoppe second revergin	ci yilli okeynyj=kauvapiszik	cyquel y 111 Diceyna		2: rdx 0000000051E7CBA	
					3: F8 0000000051E7A50	
0000000029	7CBE3				4: 19 0000000000000000	
0000000000					5; [rsp+20] 0000000051E/AS4	

# Figure 81

It enumerates the subkeys of the key using RegEnumKeyExW (Figure 82).

00000000237CC5A     00000000237CC5A     000000000237CC57     000000000237CC57     00000000237CC57     00000000237CC57     000000000237CC57     000000000237CC56     00000000037CC56     00000000037CC56	4D 89 F1 mov r9,r14 49 89 F2 mov r8,r05 48 89 C2 4 88 mov qword ptr sst[r5p+38],rbx 48 88 C2 4 D8 00 00 mov qword ptr sst[r5p+38],rbx 48 C7 44 24 30 00 00 mov qword ptr sst[r5p+36],0 48 C7 44 24 30 00 00 mov qword ptr sst[r5p+28],0 48 C7 44 24 20 00 00 mov qword ptr sst[r5p+28],0 48 C7 44 24 20 00 00 mov qword ptr sst[r5p+28],0		X8/r1 3         000000000000000000000000000000000000
000000000297CC8A	Call dword ptr ds:[ <akegenumkeyexw>]</akegenumkeyexw>	>	Default (x64 fastcall) 🔹 5 🔹 🗆 Unlock
qword ptr [000000002985BE8 <&RegEn 000000000297CC8A	umKeyExwb]= <advapi32.regenumkeyexwb< td=""><td></td><td>11 FCX 000000000000000 21 FCX 0000000000000 32 FS 00000000051E7ABC 41 F9 0000000051E7A4C 51 [Fsp=20] 00000000000000000</td></advapi32.regenumkeyexwb<>		11 FCX 000000000000000 21 FCX 0000000000000 32 FS 00000000051E7ABC 41 F9 0000000051E7A4C 51 [Fsp=20] 00000000000000000

# Figure 82

For each of the subkeys, the malware calls the RegEnumValueW API in order to enumerate the registry values:

<ul> <li>              0000000000000000000000000</li></ul>	<pre>mov r9.quord ptr ss:[r5p+68] mov edx,edi 0 lea rax,eword ptr ss:[r5p+A4] 0 mov dword ptr ss:[r5p+A4], 3FFF mov r8,r14 0 mov word ptr ss:[r5p+35,0 0 mov qword ptr ss:[r5p+36,0 0 mov qword ptr ss:[r5p+30,0 mov qword ptr ss:[r5p+20,0 mov qword ptr ss:[r5p+20,0]</pre>		X87r2 000000000000000000 572 Empty 0.0000000000000000000 X87r3 000000000000000000 573 Empty 0.00000000000000000 X87r4 0000000000000000 573 Empty 0.0000000000000000 X87r5 0000000000000000 575 Empty 0.00000000000000000 X87r7 00000000000000000 575 Empty 0.000000000000000000 X87r7 0000000000000000000 575 Empty 0.0000000000000000000 X87r7 00000000000000000000 575 Empty 0.00000000000000000000000000000000000
RIP 00000000297CD49 FF 15 C9 SE 00 00	call qword ptr ds:[<&RegEnumValuew>]	Ň	Default (x64 fastcall)
<pre>qword ptr [000000002985C18 &lt;&amp;RegEnumValuew&gt;]=<advap132.f 000000000297CD49</advap132.f </pre>	egEnumvaluew>		1: rcx 000000000000222 2: rck 0000000000000 3: r8 0000000051E7EC2 4: r9 00000000551E7A70 5: [rsp+20] 0000000000000000

# Figure 83

Finally, the type and data for all registry values identified are extracted:

00000000297CF     00000000297CF     00000000297CF     00000000297CF     00000000297CF     000000000297CF     000000000297CF	10         45         31         C9         xor r8d,r8d           45         31         C0         xor r8d,r8d           21         48         89         GC 24         mov qword ptr sst@rsp+200,rpp           24         68         89         42         28         mov qword ptr sst@rsp+200,rpx           25         46         88         9         42         20         mov qword ptr sst@rsp+200,rpx           26         68         86         52         80         00         mov rdx, qword ptr sst@rsp+200, rpx	ndx: L"U	x87r5 0000000000000000000000 ST x87r6 00000000000000000 ST x87r7 00000000000000000 ST x87Tagword FFFF	5 Empty 0.00000000000000000 6 Empty 0.0000000000000000 7 Empty 0.00000000000000000000000000000000000
	EG FF 15 84 90 00 00 Call gword ptr ds:[<4RegQueryValu	eexw>j	Default (x64 fastcall)	▼ 5 🗘 🗌 Unlocke
qword ptr [000000002985FC0 <&R	egQueryValueExW>]= <advapi32.regqueryvalueexw></advapi32.regqueryvalueexw>		<pre>1: rcx 0000000000000202 2: rdx 00000000051E7EC2 L"URL 3: r8 000000000000000 4: r9 00000000000000 5: [rsp+20] 00000000045808C0</pre>	Protocol"

### Figure 84

**0x9C41 ID** – Take a screenshot and send it to the C2 server

The GdiplusStartup function initializes Windows GDI+ (see Figure 85).

<u>819</u>	CONDOCODO27DD38     CONDOCOD027DD38     CONDOCOD027DD38     CONDOCOD027DD4     CONDOCOD027DD4     CONDOCOD027DD5     CONDOCOD027D5     COND027D5     CONDOCOD027D5     COND027D5     COND027D5     COND0275	48       89       94       24       60       00       1ea       rc4x, word ptr sst[rsp+80]         48       60       62       88       00       1ea       rc4x, word ptr sst[rsp+80]         48       67       84       24       40       00       mov qword ptr sst[rsp+1A4], 0         48       C7       84       24       A0       00       mov qword ptr sst[rsp+1A4], 0         48       C7       84       24       A0       00       mov qword ptr sst[rsp+1A4], 0         48       C7       84       24       A0       00       mov qword ptr sst[rsp+1A4], 0         48       C7       84       24       A0       00       mov qword ptr sst[rsp+10], 0         48       C7       84       24       80       00       mov qword ptr sst[rsp+10], 0         C7       84       24       80       00       mov qword ptr sst[rsp+10], 0       0         48       C7       84       24       80       00       mov qword ptr sst[rsp+10], 0       0         48       C7       84       24       80       00       mov qword ptr sst[rsp+10], 0       0         48       C7       84       24       80	 x8771 00000000000000000000000000000000000
qword ptr	[0000000029861E8 <&Gdip]	usStartup>]= <gdiplus.gdiplusstartup></gdiplus.gdiplusstartup>	2: rdx 00000000051EF070

The agent retrieves a handle to the desktop window via a call to GetDesktopWindow:



### Figure 86

It obtains the number of adjacent color bits for each pixel for the device context (DC) for the above window (0xC = BITSPIXEL):



# Figure 87

The BitBlt method is used to capture the image:

	0000000002370F8A     00000002370F80     000000002370F90     000000002370F92     000000002370F92     000000002370F92     0000000002370F93     0000000002370F93     0000000002370F84     0000000002370F84     0000000002370F84     0000000002370F84     0000000002370F83	41.89 F1         mov r8d,esi           45.31.00         xor r8d,r8d           31.02         xor r8d,r8d           46.89 6c 24.28         mov r0xd ptr ss:[r5p+28],r           46.89 F1         mov r0xd ptr ss:[r5p+28],r           47.43 80 84 24.40 000 0         lea rsi qword ptr ss:[r5p+38],c           67.44 24.40 000 0C mov dword ptr ss:[r5p+38],c         r           67.44 24.30 000 00 mov dword ptr ss:[r5p+38],c         r           68.42 40 000 00 mov dword ptr ss:[r5p+38],c         r           67.44 24.30 00 00 00 mov dword ptr ss:[r5p+38],c         r           68.42 40         mov dword ptr ss:[r5p+38],c	13 0] C0020 8×	x87r2         000000000000000000000000000000000000	Empty 0.00000000000000000000000000000000000
RIP	* 00000000029701E3E	FF 15 94 78 00 00   Call gword ptr ds:[<&BitBit	·	Default (x64 fastcall)	💌 5 🔹 🗌 Unlocke
qword ptr	[0000000002985858 <&81t81	<pre>&gt;]=<gd132.b1tb1t></gd132.b1tb1t></pre>		1: rcx 00000000000000000 2: rdx 000000000000000 3: r8 000000000000000 4: r9 000000000000000	

# Figure 88

The malware creates a Bitmap object based on a handle to a Windows GDI bitmap and a handle to a GDI palette:

212 → 000000002970FF4 000000002970FF1 000000002970FF1 <	31 D2 4C 8D 44 24 70 48 89 F9 FF 15 8E 7F 00 00	<pre>Xor edx.edx lea r8.quord ptr ss:[r5p+70] mov rCx.rdf call quord ptr ds:[&lt;&amp;GdipCreateBitmapFromHBITMAP&gt;]</pre>	>	x87Tagword FFFF Default (x64 fastcall)	▼ 5 🜩 🗌 Unlocke
qword ptr [000000002985F88 <&GdipCreateBitmapFromHBITMAP>]= <gdiplus.gdipcreatebitmapfromhbitmap></gdiplus.gdipcreatebitmapfromhbitmap>			1: rcx 0000000055050CE1 2: rdx 00000000000000000000000000000000000		

# Figure 89

The process calls the CLSIDFromString function with the "1d5be4b5-fa4a-452d-9cdd-5db35105e7eb" CLSID – Quality field:

<pre>     0000000002975011     48 80 94 24 80 00 00</pre>	rextl	x87Tagword FFFF
CODDOCODD25222028 FF 15 7F 81 00 00 Call qword ptr ds: ( <cusidfromstring)< td=""><td>&gt;</td><td>Default (x64 fastcall) - S C Unlocke</td></cusidfromstring)<>	>	Default (x64 fastcall) - S C Unlocke
gword ptr [0000000029861A8 <&CLSIDFromString>]= <combase.clsidfromstring></combase.clsidfromstring>		1: rcx 000000002981036 L 10500405-Ta4a-4520-9000-500351050/00 2: rdx 0000000051EF040

# Figure 90

GdipSaveImageToStream is utilized to save the screenshot to a stream (see Figure 91). The name of the image is derived from the current date and time.





**0x3C4D ID** – Read content from pipe and send it to the C2 server. Write server's response to the pipe

The agent opens an existing pipe using the CreateFileA API (0xC0000000 = **GENERIC\_READ** | **GENERIC\_WRITE**, 0x3 = **OPEN\_EXISTING**):



### Figure 92

The malware modifies the read and the blocking mode via a function call to SetNamedPipeHandleState (0x0 = **PIPE\_READMODE\_BYTE** | **PIPE\_WAIT**):



### Figure 93

The pipe's content is read using the ReadFile method:

	<ul> <li>00000000297704E</li> <li>000000002977057</li> <li>00000000297705C</li> <li>00000000297705F</li> <li>000000000297705F</li> </ul>	48 C7 44 24 20 00 00 mov qword ptr ss: [rsp+20],0 4C 88 4C 24 48 mov r9,qword ptr ss: [rsp+46] 4C 89 E1 mov r0x,r13 4C 89 E1 mov rCx,r12 41 88 FF FF 00 00 mov r64,FFFF	x: x: x:	8776 00000000000000000000000000000000000	Empty 0.000000000000000000 Empty 0.00000000000000000000000000000000000
1	R1P → 000000002977068	FF 15 EA E8 00 00 [call gword ptr ds:[<&ReadFile>]	De	fault (x64 fastcall)	▼ 5 🗘 🗌 Unlocke
	gword ptr [000000002985958 <&ReadF1	le>]= <kernel32.readfile></kernel32.readfile>	1	rcx 00000000000000000 rdx 00000004580050 r8 0000000000FFFF r9 00000000051EFE5C [rsp+20] 00000000000000000	

#### Figure 94

The content is exfiltrated to the C2 server, and the server's response is written back to the pipe.

**0x2129 ID** – Write two numbers into memory

The command takes two parameters and writes them in the following format:

<pre>00000000297F263 00000000297F266 </pre>	4C 89 F9 E8 E5 35 FF FF	mov rcx,r15 call <acoly< th=""><th>· cx: 10</th><th>Default (x64 fastcal)</th></acoly<>	· cx: 10	Default (x64 fastcal)
Figure 95				
	45 88 8D 94 04 00 45 88 85 90 04 00 48 8D 4C 24 28 48 8D 15 D6 28 00 18 8D 15 D6 28 00 18 85 15 05 00 00	00 mov r8d.dword ptr ds:[r13+494] 00 mov r8d.dword ptr ds:[r13+490] 1ea rcx.gword ptr ds:[r13+490] 1ea rcx.gword ptr ds:[r3p+28] call 297650	rdx:L"[- >	x87r7 00000000000000000 ST7 Empty 0.000000000000000000           x87r3 agword FFFF           Default (x64 fastcall)           1: rcx 0000000000000000           2: rdx 00000000000000000000000000000000000

Figure 96

INDICATORS OF COMPROMISE

SHA256: d71dc7ba8523947e08c6eec43a726fe75aed248dfd3a7c4f6537224e9ed05f6f

C2 server: 45.77.172.28

User-agent: trial@deloitte.com.cn

### References

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

FakeNet-NG: <u>https://github.com/mandiant/flare-fakenet-ng</u>

Unit42: https://unit42.paloaltonetworks.com/brute-ratel-c4-tool/

MDSec: <u>https://www.mdsec.co.uk/2022/08/part-3-how-i-met-your-beacon-brute-ratel/</u>