

A Deep Dive into Brute Ratel C4 payloads – Part 2

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

Summary

Brute Ratel C4 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 [here](#).

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:

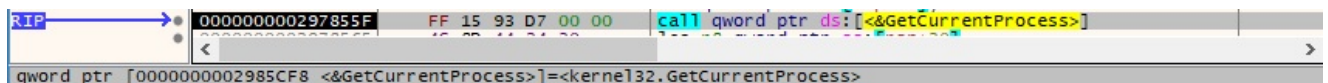


Figure 2

OpenProcessToken is utilized to open the access token associated with the process (0x28 = **TOKEN_ADJUST_PRIVILEGES | TOKEN_QUERY**):

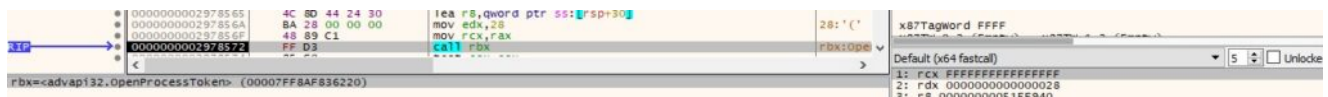


Figure 3

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

```

000000002978578 31 C9 xor ecx,ecx
00000000297857A 48 8D 5C 24 3C lea rdx,qword ptr ss:[rsp+3C]
00000000297857F 48 8D 54 24 4F lea rdx,qword ptr ss:[rsp+4F]
000000002978584 4C 8D 44 24 40 lea r8,qword ptr ss:[rsp+40]
000000002978589 FF 15 79 DC 00 00 call qword ptr ds:[<&LookupPrivilegeValue>]
qword ptr [000000002986208 <&LookupPrivilegeValue>]=advapi32.LookupPrivilegeValue>

```

Figure 4

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

```

000000002978593 31 D2 xor edx,edx
000000002978595 48 8B 4C 24 30 mov rcx,qword ptr ss:[rsp+30]
00000000297859A 41 89 10 00 00 00 mov r9d,10
0000000029785A0 49 89 D8 mov r8,rbx
0000000029785A3 C7 44 24 3C 01 00 00 mov dword ptr ss:[rsp+3C],1
0000000029785A8 C7 44 24 48 02 00 00 mov dword ptr ss:[rsp+48],2
0000000029785B3 48 C7 44 24 28 00 00 mov qword ptr ss:[rsp+28],0
0000000029785B8 48 C7 44 24 20 00 00 mov qword ptr ss:[rsp+20],0
0000000029785C5 FF 15 60 DA 00 00 call qword ptr ds:[<&AdjustTokenPrivileges>]
qword ptr [000000002986038 <&AdjustTokenPrivileges>]=advapi32.AdjustTokenPrivileges>
0000000029785C5

```

Figure 5

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

```

0000000029799A6 48 89 C1 mov rcx,rax
0000000029799A9 48 8D 7C 24 54 lea rdi,qword ptr ss:[rsp+54]
0000000029799AF 48 89 F2 mov rdx,rsl
0000000029799B3 FF 15 59 BF 00 00 call qword ptr ds:[<&Process32FirstW>]
qword ptr [000000002985910 <&Process32FirstW>]=kernel32.Process32FirstW>

```

Figure 6

```

0000000029799B8 48 89 F2 mov rdx,rsl
0000000029799BE 4C 89 E1 mov rcx,r12
0000000029799C1 FF 15 B9 C1 00 00 call qword ptr ds:[<&Process32NextW>]
qword ptr [000000002985B80 <&Process32NextW>]=kernel32.Process32NextW>

```

Figure 7

The agent is looking for the “LogonUI.exe”, “winlogon.exe”, and “lsass.exe” processes:

```

0000000029799C8 48 89 DA mov rdx,rbx
0000000029799CE 48 89 F9 mov rcx,r12
0000000029799D1 EB 0A BE FF FF call <wscmp>
rdx:L"Li
rcx:L"S
<wscmp>

```

Figure 8

It opens the first process found using the OpenProcess method (0x400 = **PROCESS_QUERY_INFORMATION**):

```

0000000029799E2 41 89 C0 mov r8d,eax
0000000029799E3 85 C0 test eax,eax
0000000029799E4 74 1A je 2979D4F
0000000029799E5 31 D2 xor edx,edx
0000000029799E7 89 00 04 00 00 mov ecx,400
0000000029799E9 FF 15 58 C2 00 00 call qword ptr ds:[<&OpenProcess>]
qword ptr [000000002985F98 <&OpenProcess>]=kernel32.OpenProcess>

```

Figure 9

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

```

0000000029799E3 49 8B 8C 24 A0 04 00 mov rcx,qword ptr ds:[r12+4A0]
0000000029799E8 FF 15 5F 80 00 00 call qword ptr ds:[<&ImpersonateLoggedOnUser>]
qword ptr [000000002985C30 <&ImpersonateLoggedOnUser>]=advapi32.ImpersonateLoggedOnUser>

```

Figure 10

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

```

0000000029799E5 48 8D 54 24 5C lea rdx,qword ptr ss:[rsp+5C]
0000000029799EA 48 89 F9 mov rcx,r12
0000000029799ED 8B 00 00 00 00 mov ebx,0
0000000029799E2 FF 15 E0 BC 00 00 call qword ptr ds:[<&GetUserNameW>]
qword ptr [000000002985BC8 <&GetUserNameW>]=advapi32.GetUserNameW>

```

Figure 11

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

rcx=82299023EFEF0000
r12=00000000045B08A0 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 “lsass.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**):

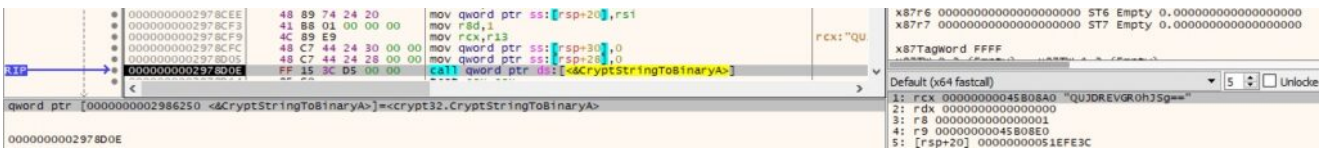


Figure 14

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**):

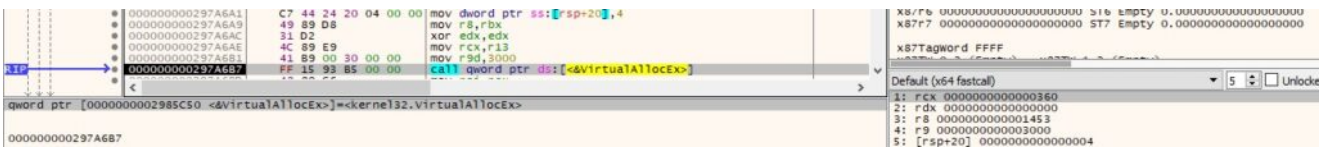


Figure 16

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**):



Figure 18

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:

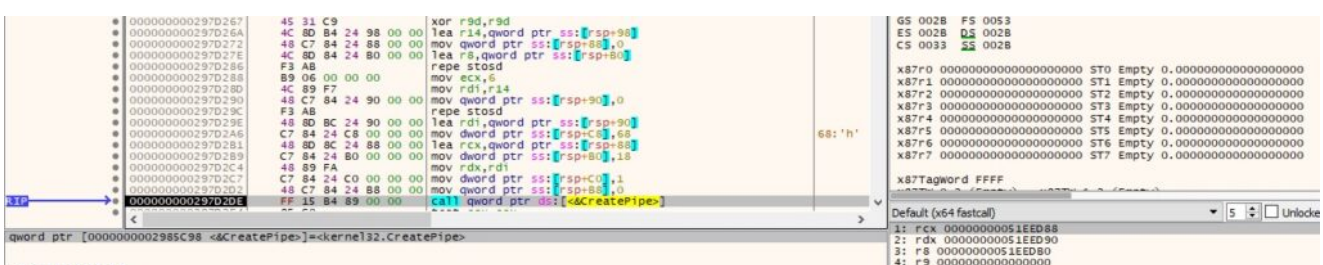


Figure 20

The pipe is set to be inherited via a call to SetHandleInformation (0x1 = HANDLE_FLAG_INHERIT):



Figure 21

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

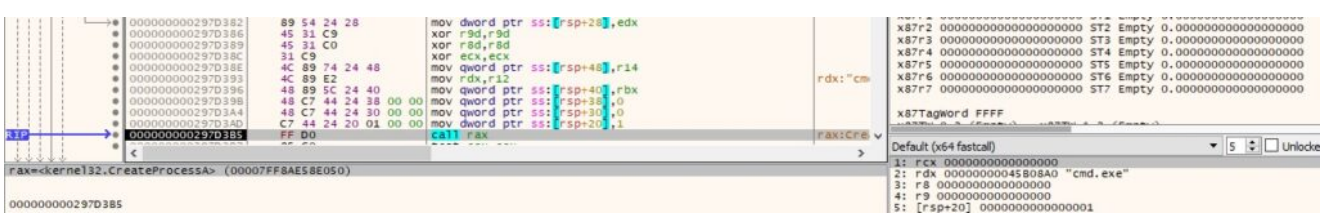


Figure 22

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



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):



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:

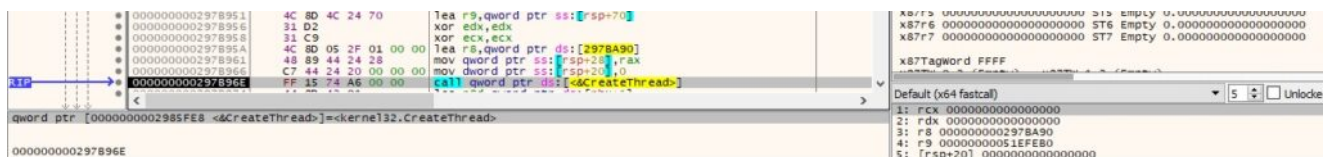


Figure 27

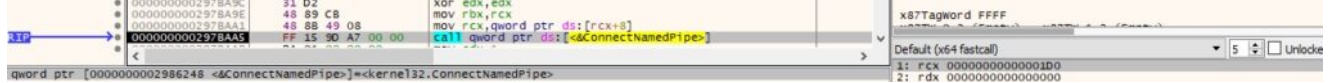


Figure 28

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

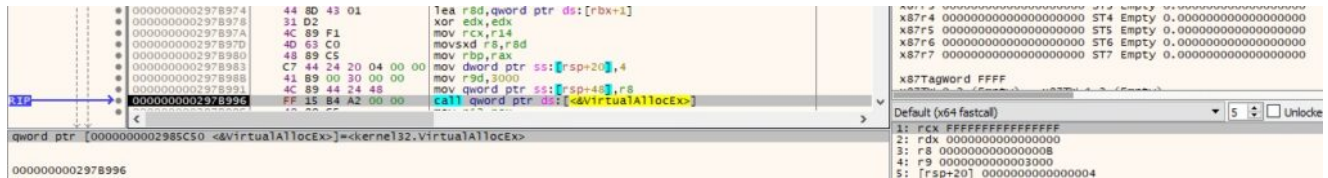


Figure 29

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

```

0000000029789B0 48 63 CB movsxd rcx,ebx
0000000029789B3 48 8D 44 24 60 lea rax,qword ptr ss:[rsp+60]
0000000029789B6 F3 A4 repe movsb
0000000029789BA 4C 8B 44 24 48 mov r8,qword ptr ss:[rsp+48]
0000000029789BF 48 89 44 24 20 mov qword ptr ss:[rsp+20],rax
0000000029789C4 4C 89 EA mov rdx,r13
0000000029789C7 41 89 20 00 00 00 mov r9d,20
0000000029789CD 4C 89 F1 mov rcx,r14
0000000029789D0 FF 15 42 9F 00 00 call qword ptr ds:[<&VirtualProtect>]
qword ptr [000000002985918 <&VirtualProtect>]=kernel32.VirtualProtect
0000000029789D0

```

Figure 30

A new thread runs the shellcode copied earlier:

```

0000000029789DF 45 31 C9 xor r9d,r9d
0000000029789E2 4D 89 E8 mov r8,r13
0000000029789E5 31 D2 xor edx,edx
0000000029789E7 31 C9 xor ecx,ecx
0000000029789E9 48 89 44 24 28 mov qword ptr ss:[rsp+28],rax
0000000029789EE C7 44 24 20 00 00 00 mov dword ptr ss:[rsp+20],0
0000000029789F6 FF 15 EC A5 00 00 call qword ptr ds:[<&CreateThread>]
qword ptr [000000002985FE8 <&CreateThread>]=kernel32.CreateThread
0000000029789F6

```

Figure 31

0x1719 ID – Enable SeDebugPrivilege

The malicious process calls the LookupPrivilegeValueA function with the “SeDebugPrivilege” parameter:

```

000000002978578 31 C9 xor ecx,ecx
00000000297857A 48 8D 5C 24 3C lea rbx,qword ptr ss:[rsp+3C]
00000000297857F 48 8D 54 24 4F lea rdx,qword ptr ss:[rsp+4F]
000000002978584 4C 8D 44 24 40 lea r8,qword ptr ss:[rsp+40]
000000002978589 FF 15 79 DC 00 00 call qword ptr ds:[<&LookupPriv11egeValueA>]
qword ptr [000000002985208 <&LookupPriv11egeValueA>]=advapi32.LookupPriv11egeValueA
000000002978589

```

Figure 32

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

```

00000000297895D 48 8D 54 24 4C lea rdx,qword ptr ss:[rsp+4C]
000000002978962 4C 8D 44 24 24 lea r8,qword ptr ss:[rsp+24]
000000002978967 C7 44 24 5C 02 00 00 mov dword ptr ss:[rsp+5C],2
00000000297896F 48 89 44 24 4C mov qword ptr ss:[rsp+4C],rax
000000002978974 48 8B 44 24 30 mov rax,qword ptr ss:[rsp+30]
00000000297897E FF 15 04 D5 00 00 call qword ptr ds:[<&Priv11egeCheck>]
qword ptr [000000002985F58 <&Priv11egeCheck>]=advapi32.Priv11egeCheck
00000000297897E

```

Figure 33

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

```

rdx=10
qword ptr [rsp+28]=[00000000051EFEF8 &L"[+] Enabled debug privilege"]

```

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).

```

00000000297C049 48 89 5C 24 20 mov qword ptr ss:[rsp+20],rbx
00000000297C04E 48 8B 4C 24 50 mov rcx,qword ptr ss:[rsp+50]
00000000297C053 49 89 C0 mov r8,rax
00000000297C056 BA 03 00 00 00 mov edx,3
00000000297C05B FF 15 DF 9C 00 00 call qword ptr ds:[<&GetTokenInformation>]
qword ptr [000000002985D40 <&GetTokenInformation>]=advapi32.GetTokenInformation
00000000297C05B

```

Figure 35

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



Figure 36

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	Hex	ASCII
0000000029F8210	5B 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 69 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	l.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 00D.i.
0000000029F8280	73 00 61 00 62 00 6C 00 65 00 64 00 0A 00 5B 00	s.a.b.l.e.d..[.
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
0000000029F82F0	20 00 20 00 20 00 20 00 20 00 44 00 69 00 73 00D.i.s.
0000000029F8300	61 00 62 00 6C 00 65 00 64 00 0A 00 5B 00 2B 00	a.b.l.e.d..[.+.

Figure 37

0x9DE0 ID – Extract Username, PPID, PID, and Executable path 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:



Figure 38

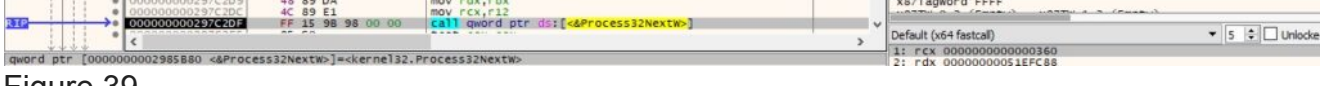


Figure 39

The agent tries to open the local process object using OpenProcess (0x410 = **PROCESS_QUERY_INFORMATION | PROCESS_VM_READ**):

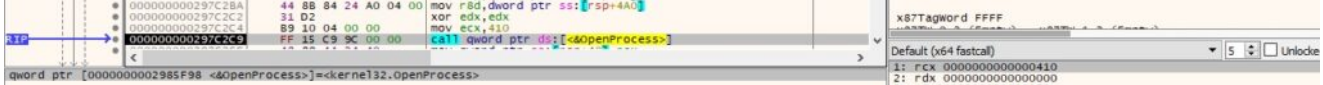


Figure 40

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).

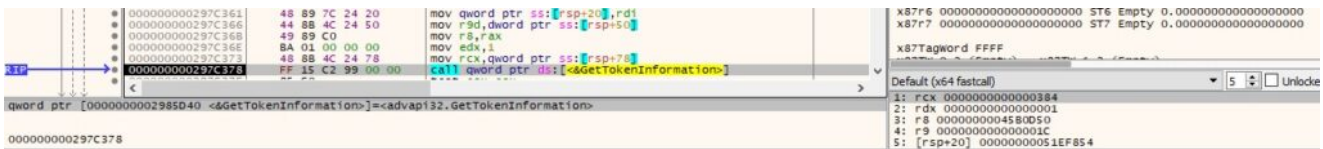


Figure 41

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

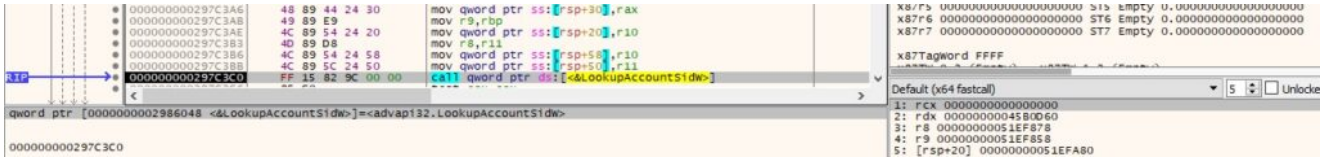


Figure 42
0xEBC0 ID – Kill processes

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



Figure 43
 The process is killed using the TerminateProcess API:

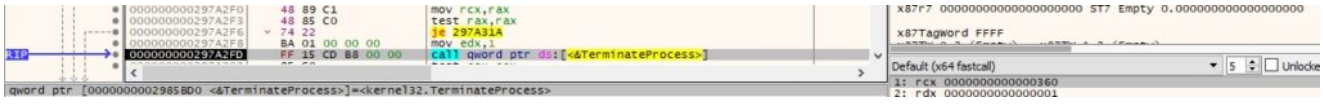


Figure 44
0xF584 ID – Create a new process using the Domain, Username, and Password received from the C2 server

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

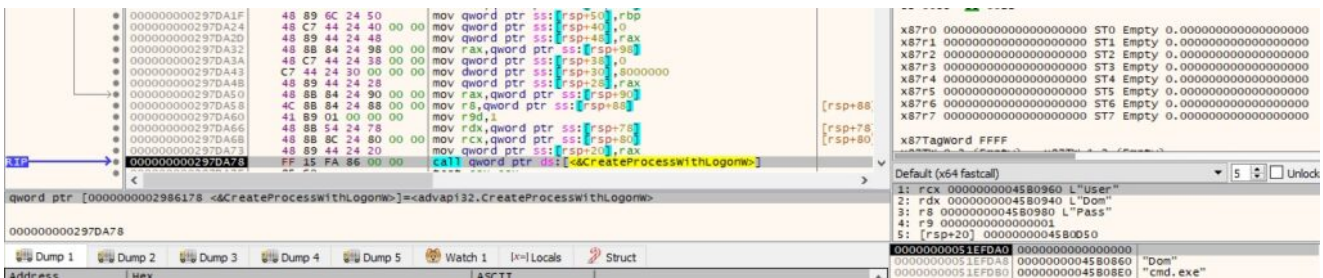


Figure 45
0xBED0 ID – Execute the “open”, “runas”, or “print” command

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

```
.text:00000000297E872 mov     dword ptr [rsp+2F8h+Str2], 'nepo'
.text:00000000297E87A movsxd rcx, ecx ; Count
.text:00000000297E87D mov     dword ptr [rsp+2F8h+var_2B4], 'anur'
.text:00000000297E885 mov     byte ptr [rsp+2F8h+var_2B0], 's'
.text:00000000297E88A mov     dword ptr [rsp+2F8h+var_2AE], 'nirp'
.text:00000000297E892 mov     byte ptr [rsp+2F8h+var_2AA], 't'
```

Figure 46

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

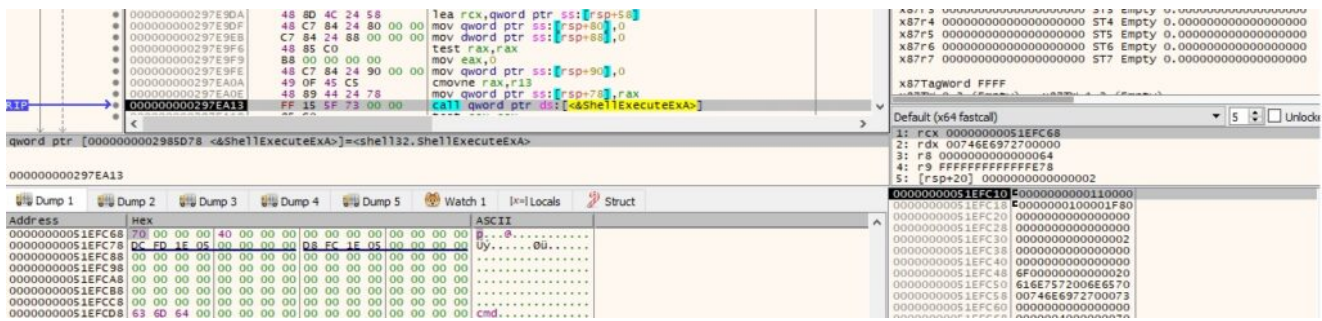


Figure 47

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



Figure 48

0xE2EA ID – Copy bytes into memory

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

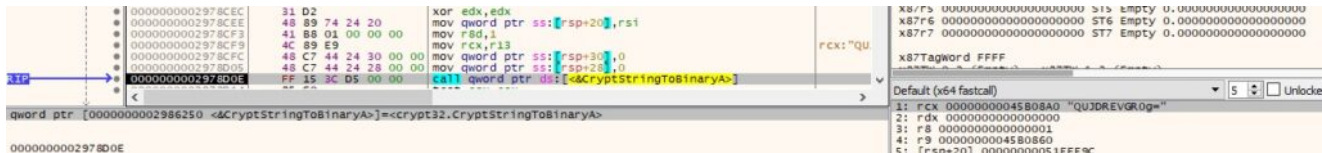


Figure 49

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

Address	Hex	ASCII
000000002985D28	60 08 5B 04 00 00 00 00	80 24 5A AE F8 7F 00 00
000000002985D38	E0 25 5A AE F8 7F 00 00	10 5E 83 AF F8 7F 00 00
000000002985D48	40 DD 58 AE F8 7F 00 00	00 BA 58 AE F8 7F 00 00
000000002985D58	00 BA 76 9E F8 7F 00 00	90 15 3B AF F8 7F 00 00
000000002985D68	80 15 70 9E F8 7F 00 00	40 62 58 AE F8 7F 00 00
000000002985D78	60 53 A8 AF F8 7F 00 00	E0 33 07 A9 F8 7F 00 00
000000002985D88	A0 3E 71 9E F8 7F 00 00	60 7C 5B AE F8 7F 00 00
000000002985D98	D0 58 AB 93 F8 7F 00 00	00 00 00 00 00 00 00 00

Figure 50

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

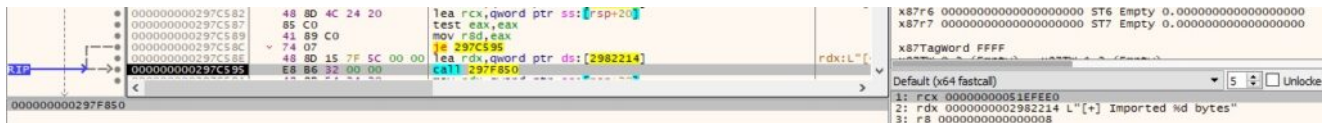


Figure 51

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:00000000297C4A5 mov cs:qword_2985D28, 0
.text:00000000297C4B0 call cs:qword_2985D30
.text:00000000297C4B6 lea rcx, [rsp+38h+Memory]
.text:00000000297C4BB lea rdx, aImportCleared ; "[+] Import cleared"
.text:00000000297C4C2 call sub_297F850

```

Figure 52

0x699A ID – Create a TCP listener

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

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:

Figure 54

The badger creates a TCP socket (0x2 = AF_INET, 0x1 = SOCK_STREAM, 0x6 = IPPROTO_TCP):

Figure 55

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

Figure 56

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

Figure 57

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



Figure 58

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

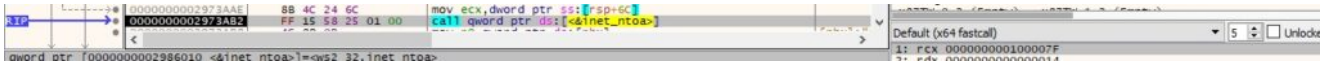


Figure 59

A new thread that handles the receive operation is created:

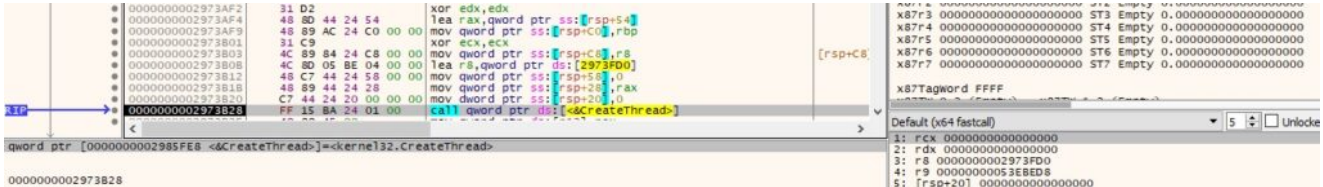


Figure 60

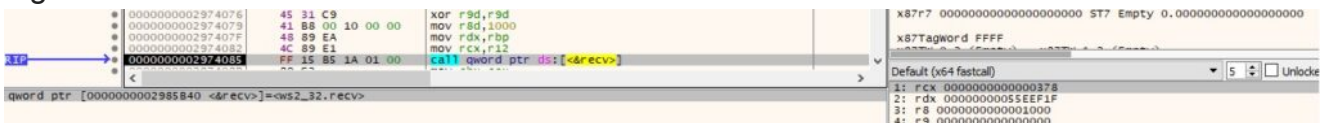


Figure 61

0xB458 ID – Extract information about Windows services

The binary opens the service control manager on the local machine using OpenSCManagerA (0x4 = **SERVICE_QUERY_STATUS**):

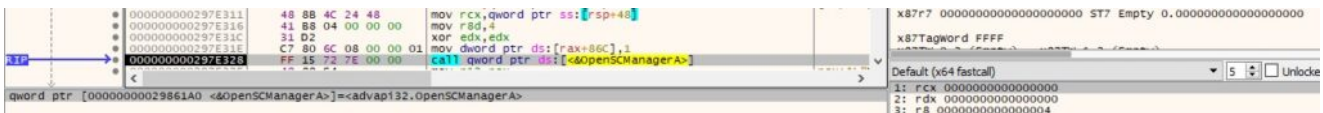


Figure 62

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.

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):

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:

```

.text:000000002978E22 mov     rax, [rsp+0A8h+var_58]
.text:000000002978E27 lea     rdx, aDomainControll ; "[+] Domain Controller:\n    %ls (%ls)\n"
.text:000000002978E2E mov     rcx, rbx
.text:000000002978E31 mov     r8, [rax]
.text:000000002978E34 mov     r9, [rax+8]
.text:000000002978E38 call    sub_297F850
.text:000000002978E3D mov     rax, [rsp+0A8h+var_58]
.text:000000002978E42 xor     edx, edx
.text:000000002978E44 mov     [rsp+0A8h+var_40], 0
.text:000000002978E4D mov     [rsp+0A8h+var_38], 0
.text:000000002978E56 lea     r8, [rsp+0A8h+var_40]
.text:000000002978E5B mov     rcx, [rax]
.text:000000002978E5E call    cs:qword_2985FF0
.text:000000002978E64 test    eax, eax
.text:000000002978E66 jnz     loc_2978EF3

```

```

.text:000000002978E6C cmp     [rsp+0A8h+var_40], 0
.text:000000002978E72 jz      short loc_2978EF3

```

```

.text:000000002978E74 lea     rdx, aDefaultDomainP ; "[+] Default Domain Password Policy:\n"
.text:000000002978E7B mov     rcx, rbx
.text:000000002978E7E mov     esi, 15180h
.text:000000002978E83 call    sub_297F850
.text:000000002978E88 mov     rax, [rsp+0A8h+var_40]
.text:000000002978E8D lea     rdx, asc_2981816 ; " "
.text:000000002978E94 mov     rcx, rbx
.text:000000002978E97 mov     r8d, [rax+10h]
.text:000000002978E9B call    sub_297F850
.text:000000002978EA0 mov     rax, [rsp+0A8h+var_40]
.text:000000002978EA5 xor     edx, edx
.text:000000002978EA7 mov     rcx, rbx
.text:000000002978EAA mov     eax, [rax+4]
.text:000000002978EAD div     esi
.text:000000002978EAF lea     rdx, aMaximumPasswor ; "    Maximum password age (d): %d\n"
.text:000000002978EB6 mov     r8d, eax
.text:000000002978EB9 call    sub_297F850

```

Figure 68

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**):

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:

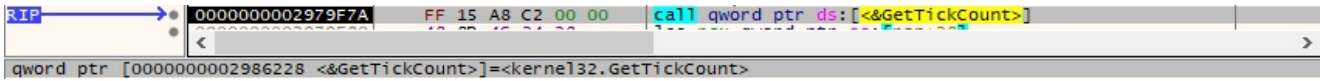


Figure 71

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.



Figure 73

0x0391 ID – Lock the workstation's display

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

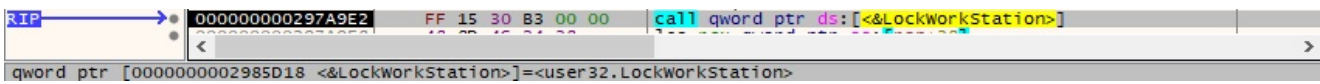


Figure 74

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):

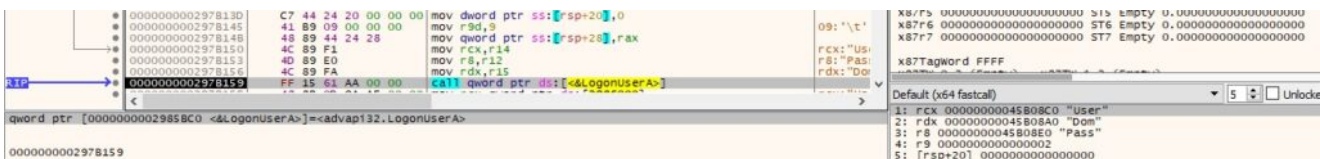


Figure 75

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

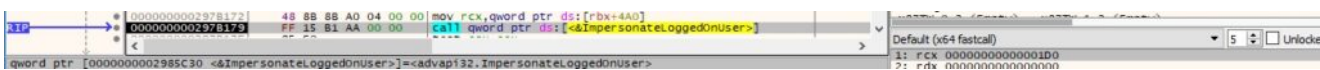


Figure 76

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:

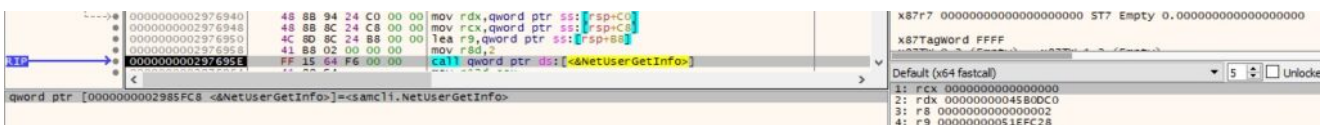


Figure 77

The information is organized in the following manner:

```
.text:000000002976A9A mov [rsp+328h+var_290], eax
.text:000000002976AA1 lea rax, aNumberOfLogons ; "Number of logons"
.text:000000002976AA8 mov [rsp+328h+var_298], rax
.text:000000002976AB0 mov eax, [rcx+80h]
.text:000000002976AB6 mov [rsp+328h+var_2B0], rsi
.text:000000002976ABB mov [rsp+328h+var_2A0], eax
.text:000000002976AC2 lea rax, aB ; "B"
.text:000000002976AC9 mov [rsp+328h+var_2A8], rax
.text:000000002976AD1 lea rax, aLastLogon ; "Last logon"
.text:000000002976AD8 mov [rsp+328h+var_2B8], rax
.text:000000002976ADD mov eax, [rcx+10h]
.text:000000002976AE0 mov [rsp+328h+var_2D0], rbx
.text:000000002976AE5 div r8d
.text:000000002976AE8 lea r8, aUserName ; "User name"
.text:000000002976AEF lea rdx, asc_29824EE ; "%"
.text:000000002976AF6 mov [rsp+328h+var_2C0], eax
.text:000000002976AFA lea rax, aPasswordLastSe ; "Password last set"
.text:000000002976B01 mov [rsp+328h+var_2C8], rax
.text:000000002976B06 lea rax, aAccountExpires ; "Account expires"
.text:000000002976B0D mov [rsp+328h+var_2D8], rax
.text:000000002976B12 mov eax, [rcx+90h]
.text:000000002976B18 mov [rsp+328h+var_2E0], eax
.text:000000002976B1C lea rax, aCountryRegionC ; "Country/region code"
.text:000000002976B23 mov [rsp+328h+var_2E8], rax
.text:000000002976B28 mov rax, [rcx+20h]
.text:000000002976B2C mov [rsp+328h+var_2F0], rax
.text:000000002976B31 lea rax, aComment ; "Comment"
.text:000000002976B38 mov [rsp+328h+var_2F8], rax
.text:000000002976B3D mov rax, [rcx+40h]
.text:000000002976B41 mov [rsp+328h+var_300], rax
.text:000000002976B46 lea rax, aFullName ; "Full name"
.text:000000002976B4D mov [rsp+328h+var_308], rax
```

Figure 78

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

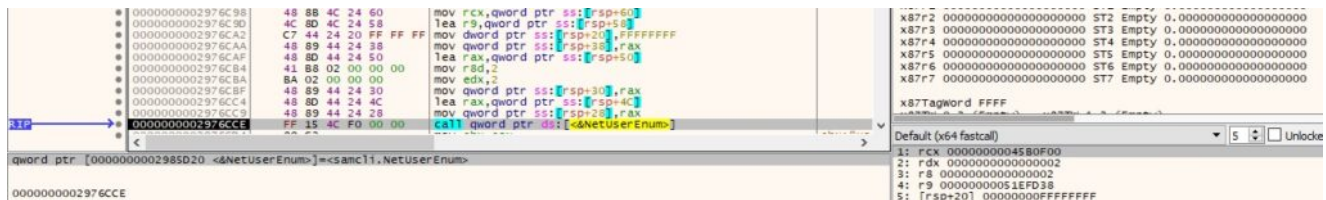


Figure 79

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**):



Figure 80

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



Figure 81

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

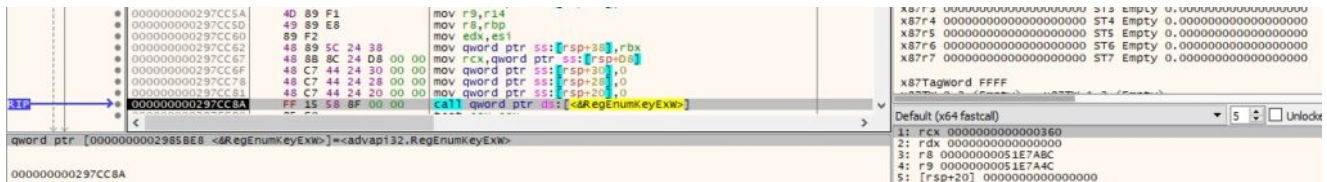


Figure 82

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

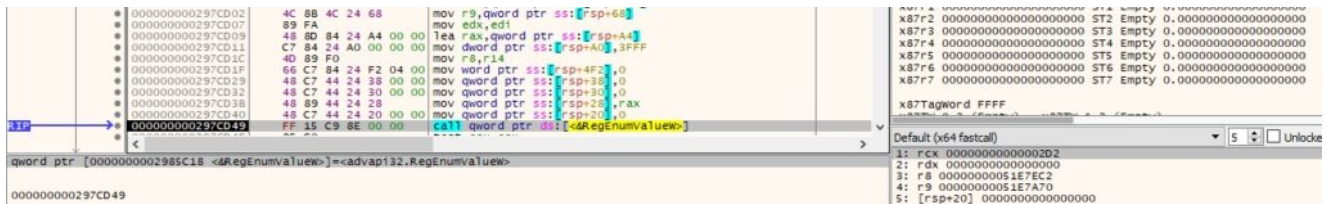


Figure 83

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

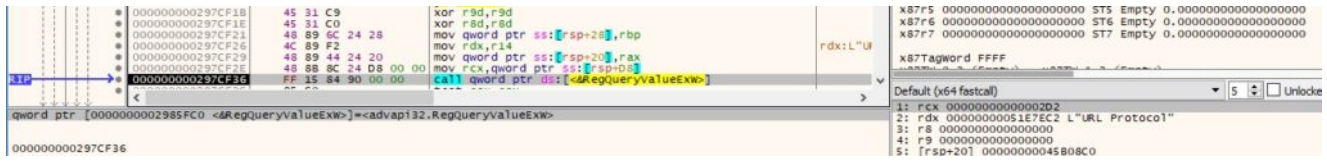


Figure 84

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

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


```

00000000297DD33 48 8D 94 24 E0 00 00 lea rdx,qword ptr ss:[rsp+E0]
00000000297DD38 48 8D 94 24 88 00 00 lea rcx,qword ptr ss:[rsp+88]
00000000297DD43 48 C7 84 24 A4 01 00 mov qword ptr ss:[rsp+1A4],0
00000000297DD4F 48 C7 84 24 AC 01 00 mov qword ptr ss:[rsp+1AC],0
00000000297DD55 48 C7 44 24 70 00 00 mov qword ptr ss:[rsp+70],0
00000000297DD59 48 C7 44 24 78 00 00 mov qword ptr ss:[rsp+78],0
00000000297DD6D 48 C7 84 24 80 00 00 mov qword ptr ss:[rsp+80],0
00000000297DD79 C7 44 24 64 00 00 00 mov dword ptr ss:[rsp+64],0
00000000297DD81 C7 84 24 E0 00 00 00 mov dword ptr ss:[rsp+E0],1
00000000297DD8C 48 C7 84 24 E8 00 00 mov qword ptr ss:[rsp+E8],0
00000000297DD98 48 C7 84 24 F0 00 00 mov qword ptr ss:[rsp+F0],0
RIP -> 00000000297DDA4 FF 15 3E 84 00 00 call qword ptr ds:[<&GdiplusStartup>]
qword ptr [0000000029861E8 <&gdiplusStartup>]=<gdiplus.GdiplusStartup>

```

Figure 85

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

```

RIP -> 00000000297DE4E FF 15 FC 7A 00 00 call qword ptr ds:[<&GetDesktopWindow>]
qword ptr [000000002985950 <&GetDesktopWindow>]=<user32.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):

```

00000000297DEB1 8A 0C 00 00 00 mov edx,c
00000000297DEB6 49 89 C5 mov r13,rax
00000000297DEB9 8B 84 24 98 00 00 00 mov esi,dword ptr ss:[rsp+98]
00000000297DE90 2B 84 24 90 00 00 00 sub esi,dword ptr ss:[rsp+90]
00000000297DE97 44 89 C8 mov eax,r9d
00000000297DE9A 44 89 4C 24 5C mov dword ptr ss:[rsp+5C],r9d
00000000297DE9F 4C 89 E9 mov rcx,r13
00000000297DEA2 44 29 C0 sub eax,r8d
00000000297DEA5 44 89 44 24 58 mov dword ptr ss:[rsp+58],r8d
00000000297DEA8 89 44 24 50 mov dword ptr ss:[rsp+50],eax
RIP -> 00000000297DEAE FF 15 64 83 00 00 call qword ptr ds:[<&GetDeviceCaps>]
qword ptr [000000002986218 <&GetDeviceCaps>]=<gdi32.GetDeviceCaps>

```

Figure 87

The BitBlt method is used to capture the image:

```

00000000297DF8A 41 89 F1 mov r9d,esi
00000000297DF8D 45 31 D2 xor r8d,r8d
00000000297DF90 31 D2 xor edx,edx
00000000297DF92 4C 89 6C 24 28 mov qword ptr ss:[rsp+28],r13
00000000297DF97 4C 89 F1 mov rcx,r14
00000000297DF9A 48 8D 84 24 A0 00 00 lea r8i,qword ptr ss:[rsp+A0]
00000000297DFA2 C7 44 24 40 20 00 CC mov dword ptr ss:[rsp+40],CC0020
00000000297DFAA C7 44 24 38 00 00 00 mov dword ptr ss:[rsp+38],0
00000000297DFB2 C7 44 24 30 00 00 00 mov dword ptr ss:[rsp+30],0
00000000297DFBA 89 44 24 20 mov dword ptr ss:[rsp+20],eax
RIP -> 00000000297DFBE FF 15 94 78 00 00 call qword ptr ds:[<&BitBlt>]
qword ptr [000000002986588 <&BitBlt>]=<gdi32.BitBlt>
00000000297DFBE

```

Figure 88

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

```

00000000297DFEA 31 D2 xor edx,edx
00000000297DFEC 4C 8D 44 24 70 lea r8,qword ptr ss:[rsp+70]
00000000297DFE1 48 89 F9 mov rcx,r9i
RIP -> 00000000297DFE4 FF 15 3E 7F 00 00 call qword ptr ds:[<&GdiPcreateBitmapFromHBITMAP>]
qword ptr [000000002985F88 <&GdiPcreateBitmapFromHBITMAP>]=<gdiplus.GdiPcreateBitmapFromHBITMAP>

```

Figure 89

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

```

00000000297E011 48 8D 94 24 80 00 00 lea rdx,qword ptr ss:[rsp+80]
00000000297E019 48 8D 80 16 30 00 00 lea rcx,qword ptr ds:[2981036]
00000000297E020 45 31 F6 xor r14d,r14d
RIP -> 00000000297E023 FF 15 7F 81 00 00 call qword ptr ds:[<&CLSIDFromString>]
qword ptr [0000000029861A8 <&CLSIDFromString>]=<combase.CLSIDFromString>

```

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.

```

00000000297E080 48 8B 54 24 78 mov rdx,qword ptr ss:[rsp+78]
00000000297E085 48 8B 4C 24 70 mov rcx,qword ptr ss:[rsp+70]
00000000297E08A 4C 8D 8C 24 F8 00 00 lea r8,qword ptr ss:[rsp+F8]
00000000297E092 49 89 F0 mov r9,r9i
RIP -> 00000000297E095 FF 15 95 7F 00 00 call qword ptr ds:[<&GdiPsaveImageToStream>]
qword ptr [000000002986030 <&GdiPsaveImageToStream>]=<gdiplus.GdiPsaveImageToStream>

```

Figure 91

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:



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:



Figure 95

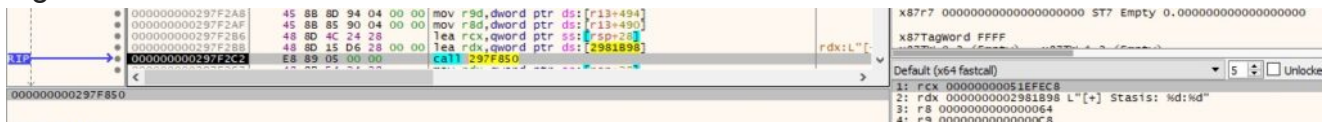


Figure 96

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

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

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