A Deep Dive into Brute Ratel C4 payloads

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

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. The malware implements the API hashing technique and comes up with a configuration that contains the C2 server, the user-agent used during the network communications, a password used for authentication with the C2 server, and a key used for encrypting data transmitted to the C2 server. The badger takes control of the infected machine by executing 63 different commands issued by the C2 server. The first 20 commands will be described in this blog post, while the rest of them will be detailed in an upcoming blog post.

Technical analysis

SHA256: d71dc7ba8523947e08c6eec43a726fe75aed248dfd3a7c4f6537224e9ed05f6f

This is a 64-bit executable. The malware pushes the code to be executed on the stack in order to evade Antivirus and EDR software:

```
.text:0000000000401000                mov     eax, 7C7C70h
.text:0000000000401005                push    rax
.text:0000000000401006                mov     rax, 68702E746E65746Eh
.text:0000000000401010                push    rax
.text:0000000000401011                mov     rax, 6F632F7C33323140h
.text:000000000040101B                push    rax
.text:000000000040101C                mov     rax, 646362617C333231h
.text:0000000000401026                push    rax
.text:0000000000401027                mov     rax, 64726F7773736150h
.text:0000000000401031                push    rax
.text:0000000000401032                mov     rax, 7C6E632E6D6F632Eh
.text:000000000040103C                push    rax
.text:000000000040103D                mov     rax, 657474696F6C6564h
.text:0000000000401047                push    rax
.text:0000000000401048                mov     rax, 406C616972747C30h
.text:0000000000401052                push    rax
.text:0000000000401053                mov     rax, 387C38322E323731h
.text:000000000040105D                push    rax
.text:000000000040105E                mov     rax, 2E37372E35347C30h
.text:0000000000401068                push    rax
.text:0000000000401069                push    4Bh ; 'K'
```

Figure 1

It implements the API hashing technique, which uses the "ROR EDI,0xD" instruction to compute 4-byte hashes that are compared with pre-computed ones (Figure 2).

Figure 2

The VirtualAllocEx API is used to allocate a new memory area that will store a DLL file (0x3000 = **MEM_COMMIT** | **MEM_RESERVE**, 0x40 = **PAGE_EXECUTE_READWRITE**):

Figure 3

The Brute Ratel C4 configuration is stored in clear text however, in recent versions, the config is encrypted and Base64-encoded. It contains the C2 IP address and port number, the user-agent used during the network communications, a password used to authenticate with the C2 server, a key used to encrypt data transmitted to the C2 server, and the URI:

Figure 4

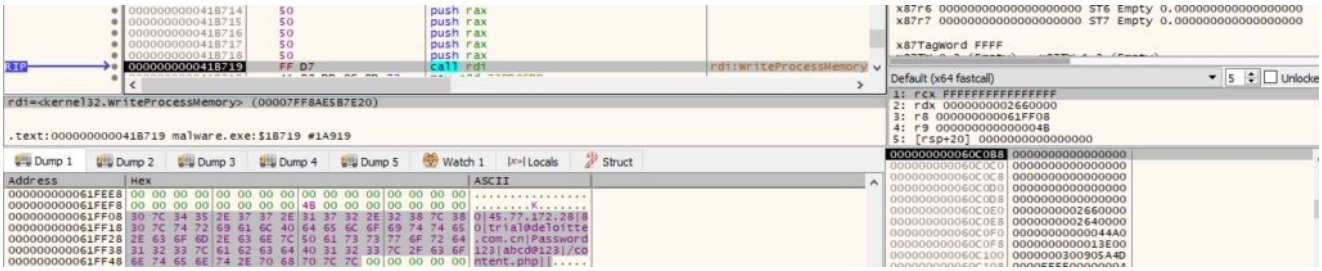

Figure 5

A thread that executes the entry point of the new DLL is created via a function call to CreateRemoteThread:

Figure 6

The process extracts a pointer to the PEB from gs:[0x60] and another one to the PEB_LDR_DATA structure (+0x18), which contains information about the loaded DLLs. The InMemoryOrderModuleList doubly-linked list contains the loaded DLLs for the current process:

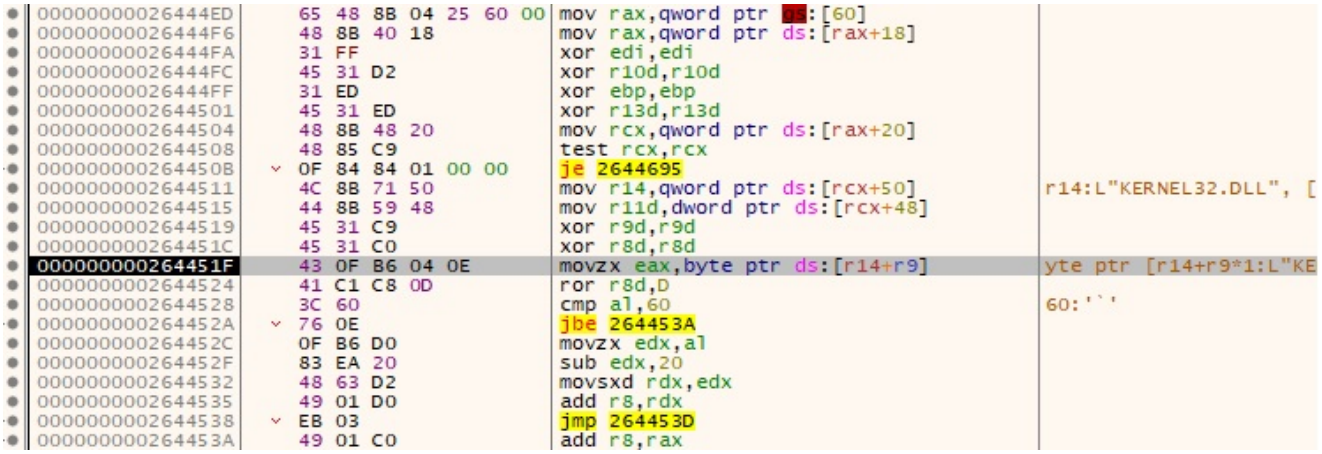

Figure 7

The malicious binary allocates new memory for another DLL that implements the main functionality using VirtualAlloc:

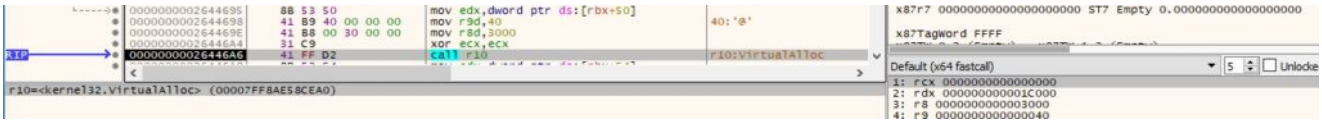

Figure 8

LoadLibraryA is utilized to load multiple DLLs into the address space of the current process:

Figure 9

The malware retrieves the address of relevant functions by calling the GetProcAddress method:

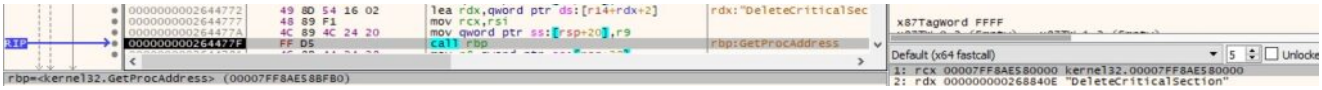

Figure 10

The binary flushes the instruction cache for the current process using the NtFlushInstructionCache function (see Figure 11).

Figure 11

Finally, the malware passes the execution flow to the newly constructed DLL:

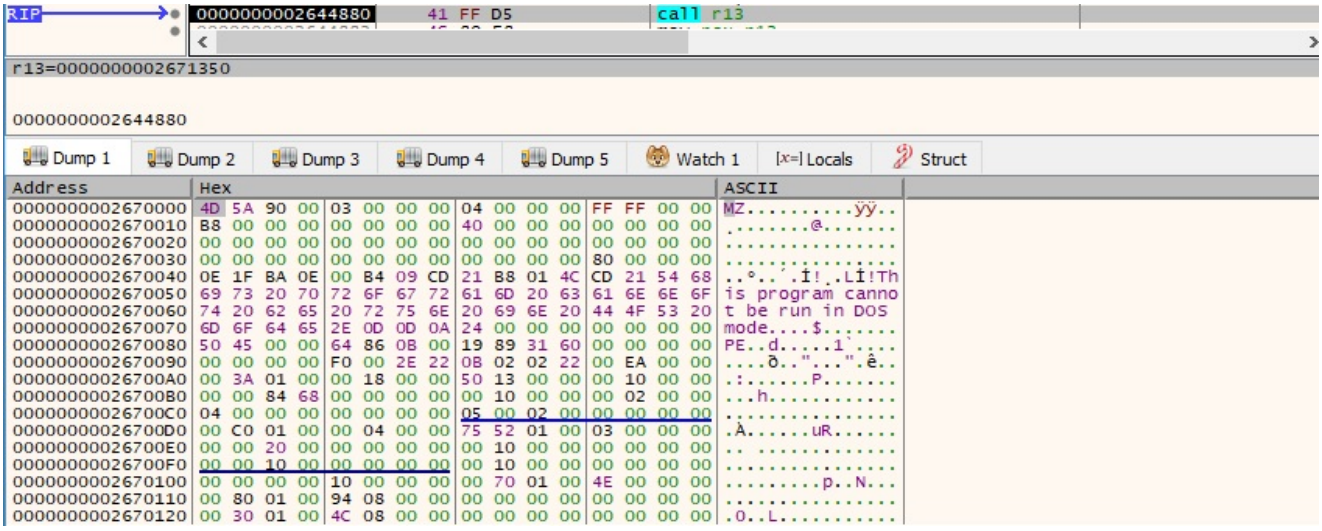

Figure 12

As we can see below, one of the export functions of the DLL is called “badger_http_1”, which reveals a Brute Ratel agent/badger.

Name	Address	Ordinal
	00000000026750A0	1
TlsCallback_0	00000000026716B0	
TlsCallback_1	0000000002671680	
DllEntryPoint	0000000002671350	[main entry]

Figure 13

```
.text:00000000026750A0 ; Exported entry   1.
.text:00000000026750A0
.text:00000000026750A0
.text:00000000026750A0
.text:00000000026750A0 public badger_http_1
.text:00000000026750A0 badger_http_1 proc near
.text:00000000026750A0
.text:00000000026750A0 var_58= qword ptr -58h
.text:00000000026750A0 var_50= qword ptr -50h
.text:00000000026750A0 arg_0= qword ptr  8
.text:00000000026750A0
.text:00000000026750A0 int     3               ; Trap to Debugger
.text:00000000026750A1 push    rdi
.text:00000000026750A2 push    r14
.text:00000000026750A4 push    r13
.text:00000000026750A6 push    r12
.text:00000000026750A8 push    rbp
.text:00000000026750A9 push    rdi
.text:00000000026750AA push    rsi
.text:00000000026750AB push    rbx
.text:00000000026750AC sub     rsp, 38h
.text:00000000026750B0 mov     [rsp+78h+arg_0], rcx
.text:00000000026750B8 call    sub_2675620
.text:00000000026750BD mov     r8, rax
.text:00000000026750C0 mov     rax, gs:40h
.text:00000000026750C9 lea     rsi, [r8+rax]
```

Figure 14

The FreeConsole method is used to detach the process from its console:

```
RIP  00000000026748EE  FF 15 40 39 01 00  call qword ptr ds:[<&FreeConsole>]
qword ptr [0000000002688234 <&FreeConsole>]=<kernel32.FreeConsole>
```

Figure 15

The DLL repeats the process of finding functions address, as highlighted in Figure 16.

```
.text:0000000002677377 mov     ecx, 0EC0E4E8Eh
.text:000000000267737C lea     rdi, [rsp+168h+var_26]
.text:0000000002677384 mov     rdx, rax
.text:0000000002677387 call    sub_2675D40
.text:000000000267738C mov     ecx, 16B3FE72h
.text:0000000002677391 mov     cs:qword_2686040, rax
.text:0000000002677398 call    sub_2675D40
.text:000000000267739D mov     ecx, 88A9223Ch
.text:00000000026773A2 mov     cs:qword_2685B60, rax
.text:00000000026773A9 call    sub_2675D40
.text:00000000026773AE mov     ecx, 0BF608091h
.text:00000000026773B3 mov     cs:qword_2685948, rax
.text:00000000026773BA call    sub_2675D40
.text:00000000026773BF mov     ecx, 0FFD97FBh
.text:00000000026773C4 mov     cs:qword_2686060, rax
.text:00000000026773CB call    sub_2675D40
.text:00000000026773D0 mov     ecx, 99EC895Eh
.text:00000000026773D5 mov     cs:qword_2685B90, rax
.text:00000000026773DC call    sub_2675D40
.text:00000000026773E1 mov     ecx, 9FCF5965h
.text:00000000026773E6 mov     cs:qword_2685B48, rax
.text:00000000026773ED call    sub_2675D40
.text:00000000026773F2 mov     ecx, 7C0017A5h
.text:00000000026773F7 mov     cs:qword_2685B68, rax
.text:00000000026773FE call    sub_2675D40
.text:0000000002677403 mov     ecx, 56C61229h
.text:0000000002677408 mov     cs:qword_2686168, rax
.text:000000000267740F call    sub_2675D40
.text:0000000002677414 mov     ecx, 7C0017BBh
.text:0000000002677419 mov     cs:qword_2685D48, rax
.text:0000000002677420 call    sub_2675D40
.text:0000000002677425 mov     ecx, 4EE4A045h
.text:000000000267742A mov     cs:qword_2685CB8, rax
.text:0000000002677431 call    sub_2675D40
.text:0000000002677436 mov     ecx, 170C8F80h
.text:000000000267743B mov     cs:qword_2685FE0, rax
.text:0000000002677442 call    sub_2675D40
.text:0000000002677447 mov     ecx, 72BD9CDDh
.text:000000000267744C mov     cs:qword_2685C98, rax
.text:0000000002677453 call    sub_2675D40
```

Figure 16

The process extracts the system time and passes the result to the srand function:

Figure 17

The atoi method is utilized to convert the port number to integer:

Figure 18

The malicious process creates an unnamed mutex object by calling the CreateMutexA API, as displayed in Figure 19.

Figure 19

GetUserNameW is used to obtain the username associated with the current thread:

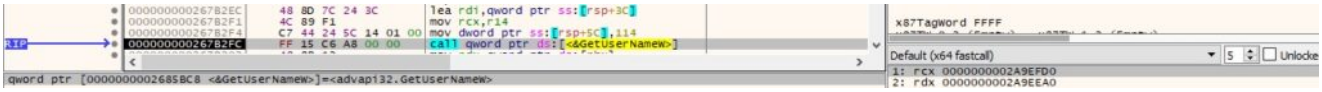

Figure 20

GetComputerNameExW is used to obtain the NetBIOS name associated with the local machine:

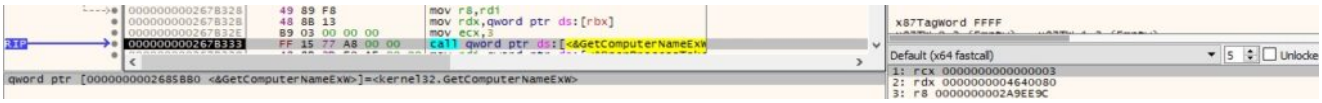

Figure 21

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

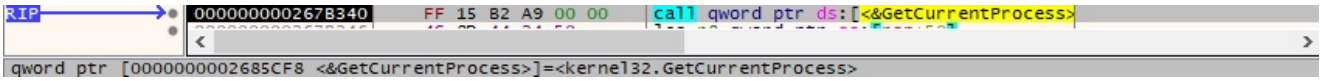

Figure 22

The OpenProcessToken API is utilized to open the access token associated with the process (0x8 = **TOKEN_QUERY**):

Figure 23

The malware verifies if the token is elevated using the GetTokenInformation method (0x14 = **TokenElevation**):

Figure 24

It obtains the current process ID via a function call to GetCurrentProcessId:

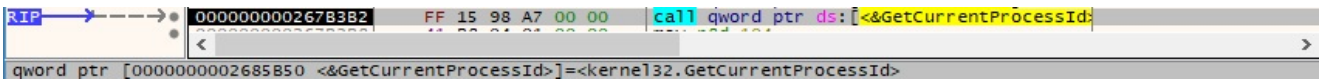

Figure 25

GetModuleFileNameW is utilized to extract the path of the executable file of the process:

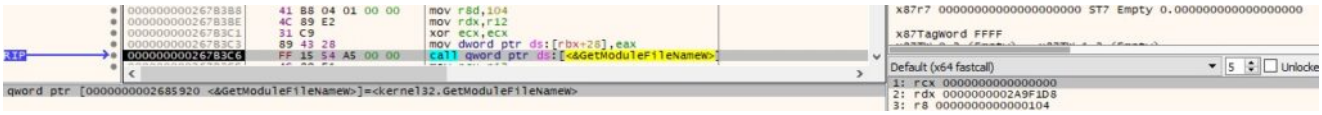

Figure 26

The above path is Base64-encoded using the CryptBinaryToStringW API (0x40000001 = **CRYPT_STRING_NOCRLF** | **CRYPT_STRING_BASE64**):

Figure 27

The process retrieves version information about the current operating system using RtlGetVersion:

Figure 28

The WSAStartup function initiates the use of the Winsock DLL by the current process:

Figure 29

The badger constructs a JSON that stores the password extracted from the configuration, the computer name, the OS version, the Base64-encoded executable path, the username, and the process ID:

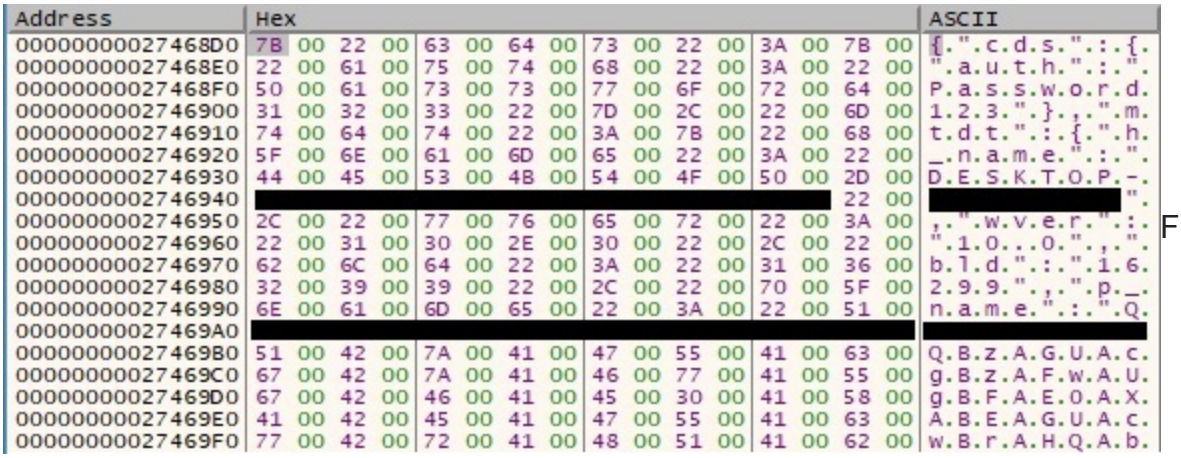

Figure 30

The JSON is encrypted using the XOR operator (key = “abcd@123” from configuration) and transformed by other operations:

Figure 31

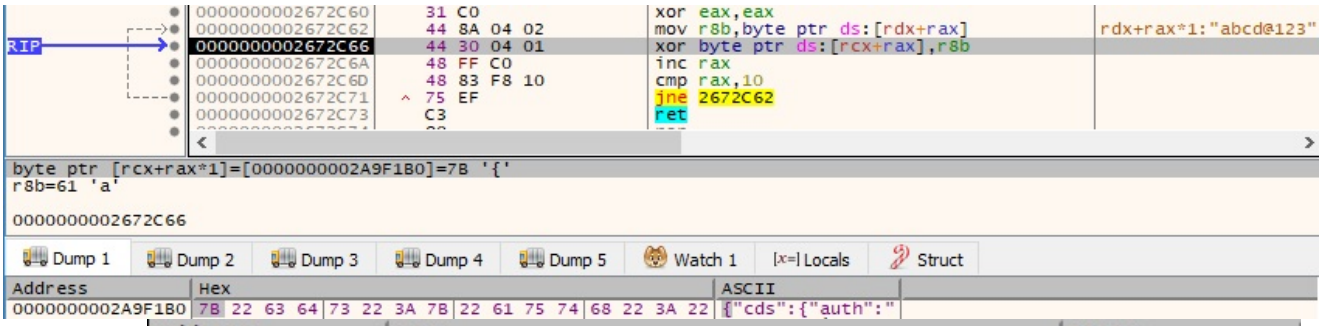

Figure 32

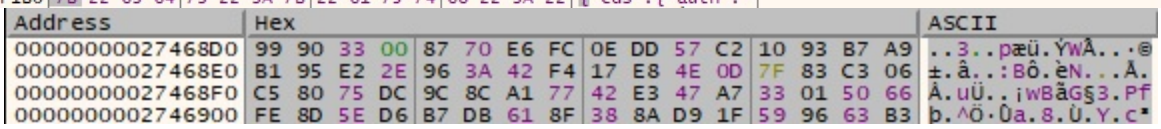

Figure 33

The user-agent passed to the InternetOpenW function seems to indicate that the product was used by Deloitte China (Figure 34).

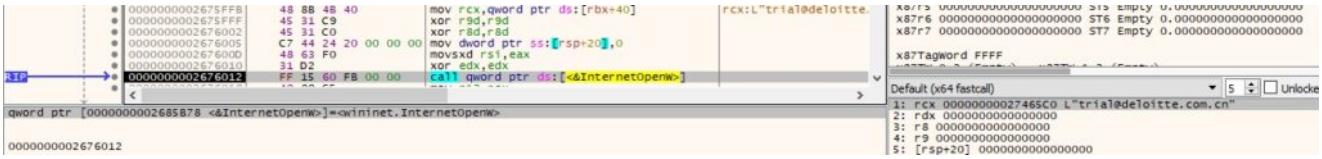

Figure 34

The process connects to the C2 server on port 80 by calling the InternetConnectW function:

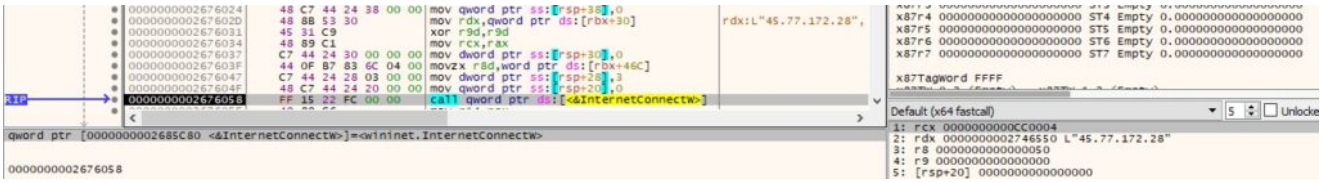

Figure 35

It creates a POST request to the “/content.php” resource using HttpOpenRequestW, as displayed below.

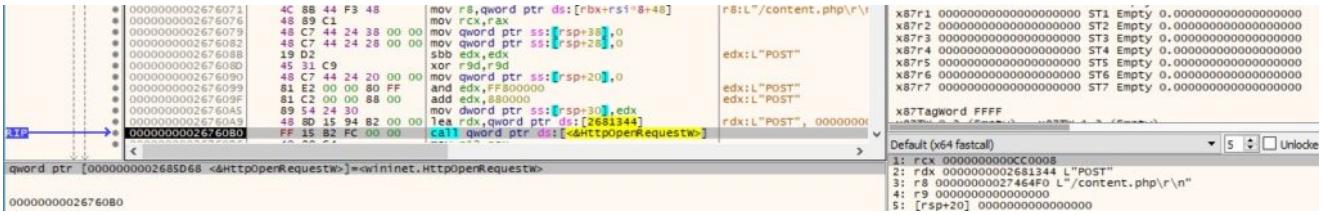

Figure 36

The security flags for the handle are changed using the InternetSetOptionW API (0x1100 = **SECURITY_FLAG_IGNORE_CERT_CN_INVALID** | **SECURITY_FLAG_IGNORE_UNKNOWN_CA**):

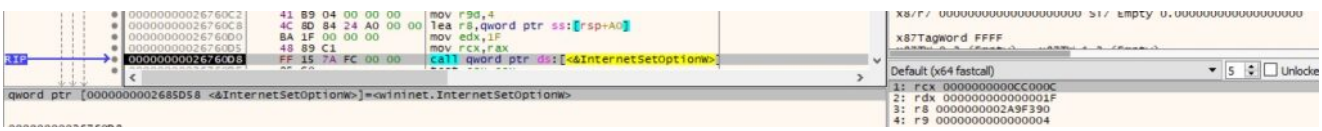

Figure 37

HttpAddRequestHeadersW can be used to add one or more HTTP request headers to the handle however, the second parameter is NULL during malware's execution (0x20000000 = **HTTP_ADDREQ_FLAG_ADD**):

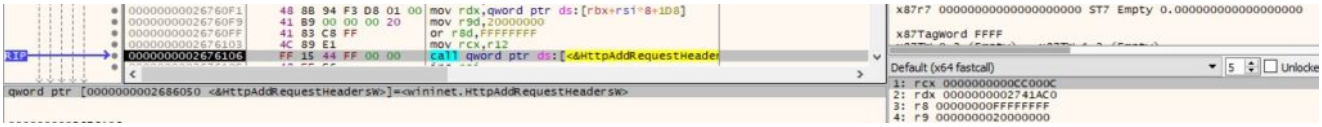

Figure 38

The process encodes the encrypted JSON using Base64 and exfiltrates the resulting data using HttpSendRequestW:

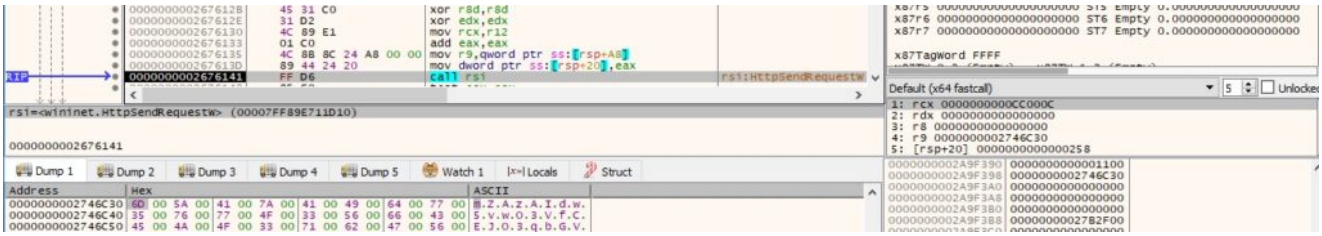

Figure 39

It verifies whether the C2 server sends any data back via a function call to InternetQueryDataAvailable:

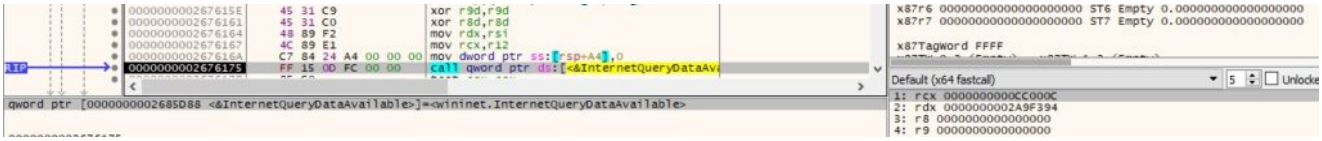

Figure 40

The C2 server's response is read using InternetReadFile:

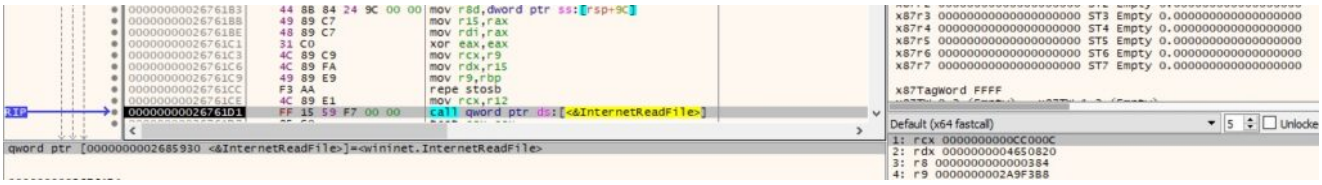

Figure 41

The response is Base64-decoded and decrypted using the same key that was previously mentioned. The "auth" field is set to the decrypted information, and another request is made to the C2 server, asking for commands:

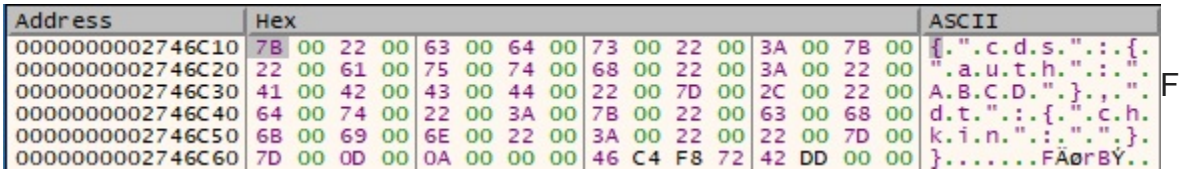

Figure 42

FakeNet-NG was used to simulate the network communications with the C2 server. After decoding and decrypting the response, the first 2 bytes represent the command to be executed followed by additional parameters if necessary. A new thread handles the commands execution:

Figure 43

We'll now describe the commands that can be issued by the C2 server.

0x2C74 ID – Exfiltrate file content to the C2 server

The PathFileExistsA API is utilized to confirm if the target file exists on the system:

Figure 44

The file is opened via a function call to CreateFileA (0x80000000 = **GENERIC_READ**, 0x1 = **FILE_SHARE_READ**, 0x3 = **OPEN_EXISTING**):

Figure 45

The content is read by calling the ReadFile method, as shown in Figure 46.

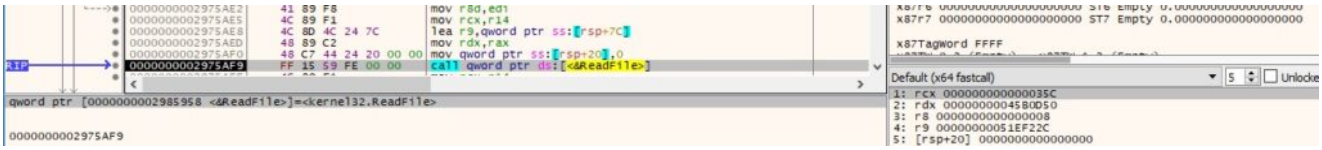

Figure 46

The data is sent to the C2 server along with the "[+] Download complete" message or the message shown in the figure below.

```
.text:0000000002975A5F
.text:0000000002975A5F loc_2975A5F:
.text:0000000002975A5F xor     eax, eax
.text:0000000002975A61 mov     ecx, ebp
.text:0000000002975A63 mov     rdi, r13
.text:0000000002975A66 mov     r9, r15
.text:0000000002975A69 rep stosb
.text:0000000002975A6B xor     eax, eax
.text:0000000002975A6D mov     ecx, 82h ; ','
.text:0000000002975A72 mov     edx, 104h       ; SizeInWords
.text:0000000002975A77 lea     rdi, [rsp+0D58h+var_AC8]
.text:0000000002975A7F lea     r10, [rsp+0D58h+var_AC8]
.text:0000000002975A87 rep stosd
.text:0000000002975A89 mov     dword ptr [rsp+0D58h+var_D38], r8d
.text:0000000002975A8E mov     rcx, r10        ; Dst
.text:0000000002975A91 lea     r8, Format      ; "[+] Download Status %S (%lu %%)"
.text:0000000002975A98 mov     [rsp+0D58h+var_CF0], r10
.text:0000000002975A9D call    swprintf_s
.text:0000000002975AA2 mov     r10, [rsp+0D58h+var_CF0]
.text:0000000002975AA7 mov     rcx, r12
.text:0000000002975AAA mov     rdx, r10
.text:0000000002975AAD call    sub_29783D0
.text:0000000002975AB2 jmp     loc_29759AA
```

Figure 47

0xA905 ID – Copy files

The malware copies an existing file to a new file using CopyFileA:

Figure 48

0x9B84 ID – Move files

The process moves an existing file to another using the MoveFileA function (Figure 49).

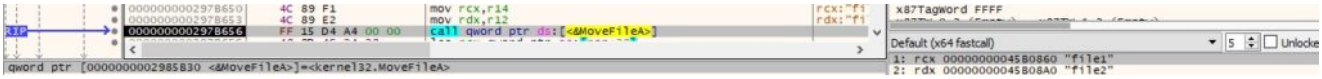

Figure 49

0x13A1 ID – Create files and populate them with content received from the C2 server

Firstly, the file is created via a function call to CreateFileA:

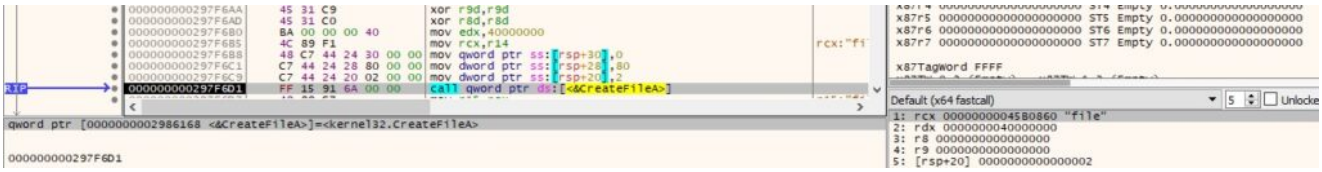

Figure 50

The received data is Base64-decoded using CryptStringToBinaryA and written to the file:

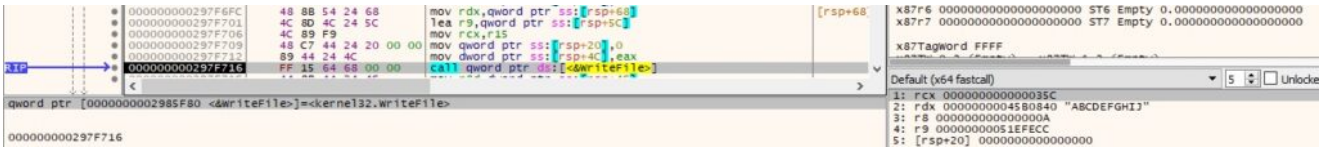

Figure 51

0xE993 ID – Delete files

DeleteFileA is used to delete the target files, as highlighted below:

Figure 52

0x0605 ID – Close handles

The badger closes an object handle (i.e. file, process) using the CloseHandle API:

Figure 53

0x3F61 ID – Create directories

The malicious binary has the ability to create directories using the CreateDirectoryA method:

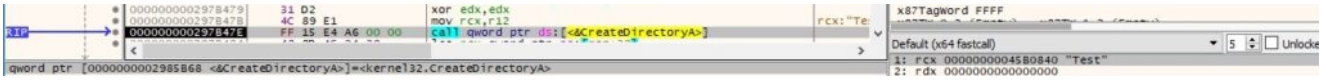

Figure 54

0x1139 ID – Change the current directory for the process

SetCurrentDirectoryA is utilized to perform the desired operation (see Figure 55).

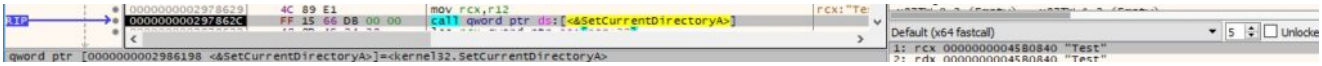

Figure 55

0x3C9F ID – Obtain the current directory for the process

The malware extracts the current directory for the process by calling the GetCurrentDirectoryW API:

Figure 56

0x8F40 ID – Delete directories

The process deletes a target directory only if it's empty using RemoveDirectoryA:

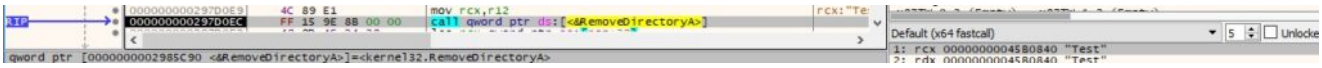

Figure 57

0x0A32 ID – Retrieve the Last-Write time for files/directories

The files are enumerated in the current directory using the FindFirstFileW and FindNextFileW functions:

Figure 58

Figure 59

For each of the file or directory that matches the pattern, the binary calls the CreateFileW API:

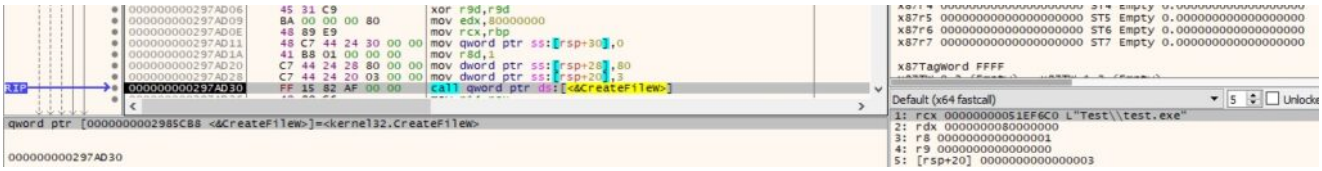

Figure 60

The process retrieves the Last-Write time via a function call to GetFileTime:

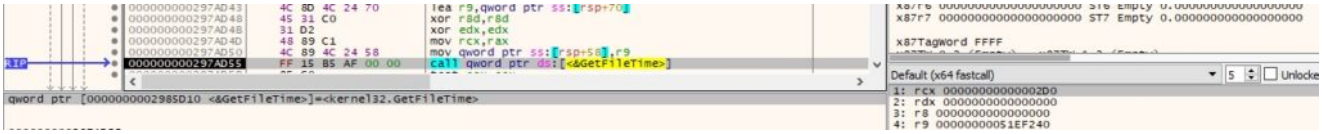

Figure 61

The file time is converted to system time format using FileTimeToSystemTime:

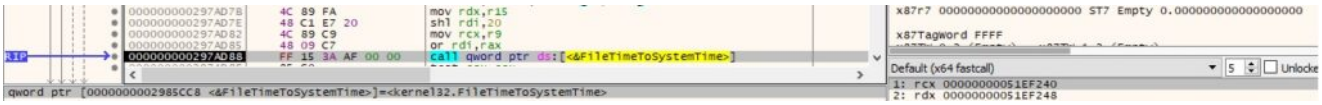

Figure 62

Finally, the above time is converted to the currently active time zone:

Figure 63

0x3D1D ID – Change the Desktop wallpaper

The malicious process opens the “TranscodedWallpaper” file that contains the Desktop wallpaper:

Figure 64

The above file is filled in with content received from the C2 server (Figure 65).

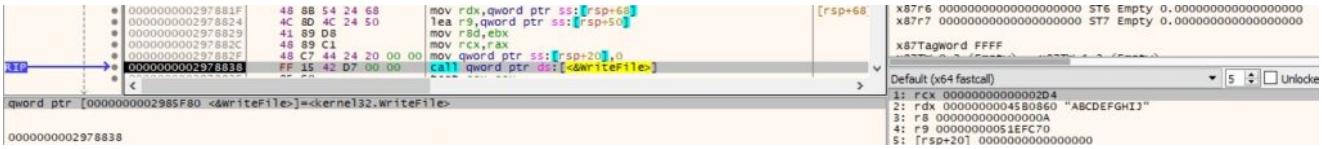

Figure 65

The SystemParametersInfoA method is utilized to change the Desktop wallpaper (0x14 = **SPI_SETDESKWALLPAPER**, 0x1 = **SPIF_UPDATEINIFILE**):

Figure 66

0xD53F ID – Retrieve the username

This command is used to obtain the username associated with the current thread:

Figure 67

0x0609 ID – Retrieve the available disk drives

The malware extracts a bitmask that contains the available disk drives by calling the GetLogicalDrives API, as shown in Figure 68.

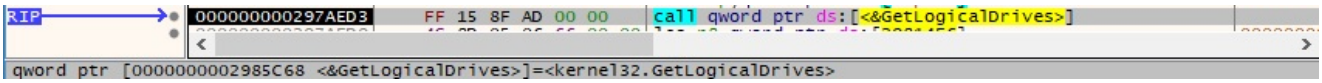

Figure 68

0xC144 ID – Extract all device drivers

EnumDeviceDrivers is utilized to obtain the load address for all device drivers:

Figure 69

Using the above address, the process retrieves the name of the device driver by calling the GetDeviceDriverBaseNameA method:

Figure 70

0x0A01 ID – Compute the number of minutes that have elapsed since the system was started

The GetTickCount function is used to extract the number of milliseconds and a simple calculation is performed (see Figure 71).

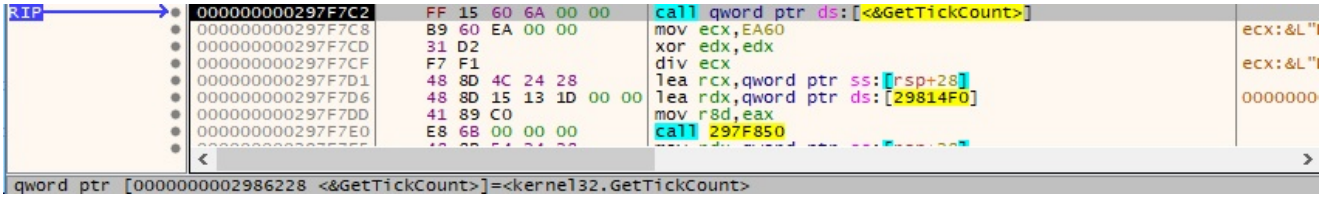

Figure 71

0x73E6 ID – Argument Spoofing

The badger has the ability to hide the arguments by modifying the process environment block (PEB):

Figure 72

0x8AFA ID – Parent PID Spoofing

This command can be used to spoof the parent process ID in order to evade EDR software or other solutions:

Figure 73

0xC929 ID – Extract child process name

The binary could spawn multiple processes that can be displayed using this command (Figure 74).

Figure 74

0x9E72 ID – Display pipes name

The malware displays the name of a previously created pipe:

Figure 75

The other 30 relevant commands will be detailed in a second blog post.

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/