A step-by-step analysis of the Russian APT Turla backdoor called TinyTurla – CYBER GEEKS

Published: 2022-03-28 · Archived: 2026-04-05 22:02:17 UTC

Summary

Turla is a Russian-based group that has impacted government, embassies, military, education, and research companies since 2004. Our analysis focuses on a backdoor called TinyTurla that was installed on an endpoint via a Windows Service. The list of C2 servers and a password used for authentication with the servers are stored in the Windows registry. The malware implements 12 different commands that include spawning and killing processes, creating and exfiltrating files, creating pipes for process communication, and modifying registry values used during the execution.

Analyst: @GeeksCyber

Technical analysis

SHA256: 030cbd1a51f8583ccfc3fa38a28a5550dc1c84c05d6c0f5eb887d13dedf1da01

The file is a 64-bit DLL that was installed as a service called “Microsoft Windows Time” (https://blog.talosintelligence.com/2021/09/tinyturla.html). We’ve manually created a service called “W64Time” and the corresponding registry keys/values by simulating the execution of the batch script mentioned in the Talos article:

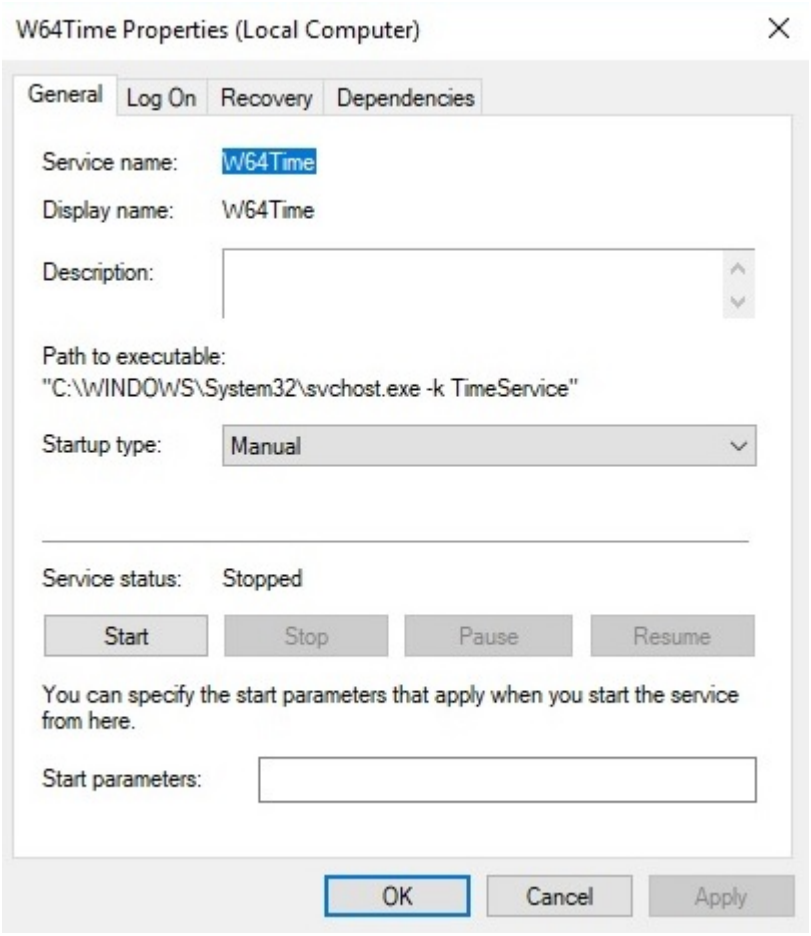


Figure 1

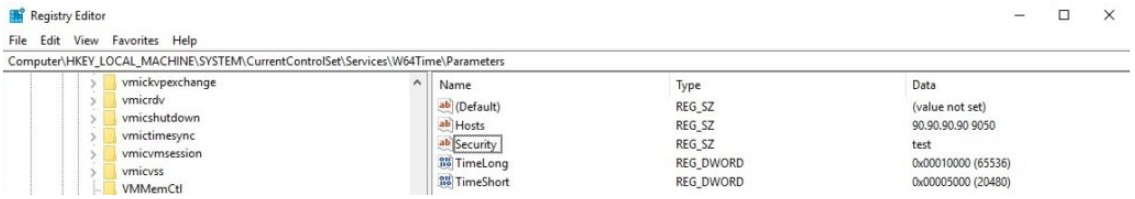


Figure 2

Because we're analyzing a 64-bit file, the calling convention is different, and the function arguments are passed to the RCX, RDX, R8, and R9 registers. Additional arguments are pushed onto the stack (right to left).

RegisterServiceCtrlHandlerW is utilized to register a function to handle service control requests:

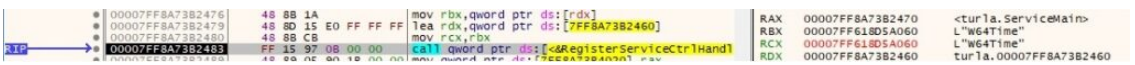


Figure 3

The service status for the above service is set to 0x4 (**SERVICE_RUNNING**) via a function call to SetServiceStatus:

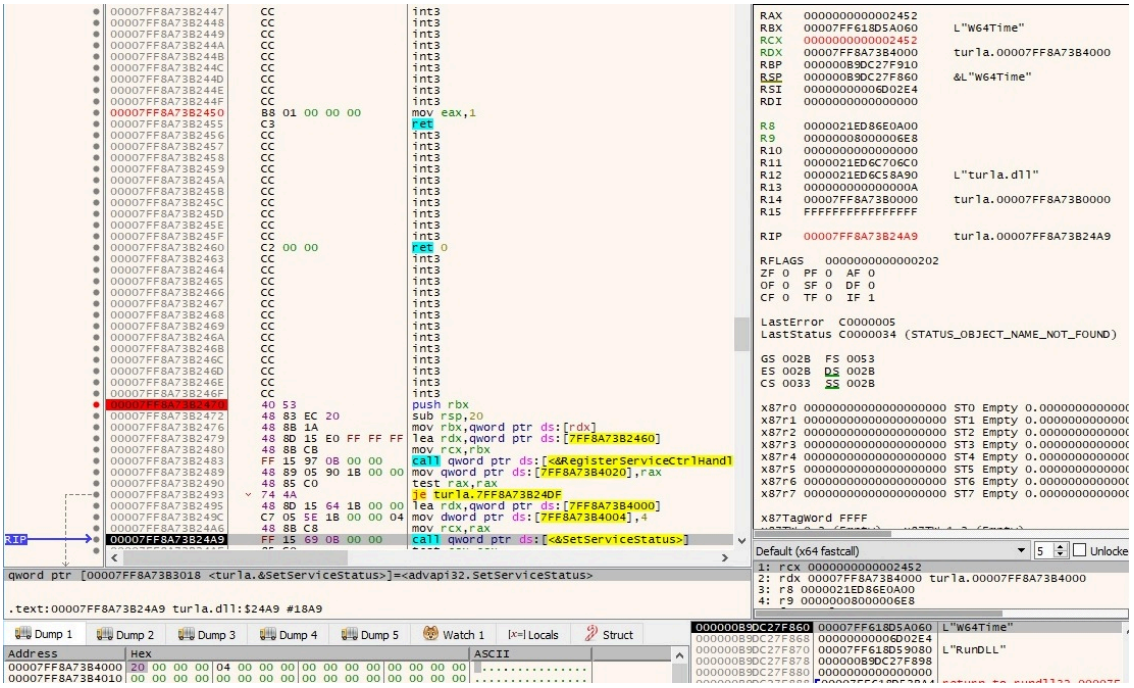


Figure 4

After the main function finishes, the service status is set to 0x1 (**SERVICE_STOPPED**).

The RegOpenKeyExW API is used to open the “SYSTEM\CurrentControlSet\Services\W64Time\Parameters” registry key (0x80000002 = **HKEY_LOCAL_MACHINE**, 0x20119 = **KEY_READ** | **KEY_WOW64_64KEY**):

Figure 5

The process extracts the following registry values using RegQueryValueExW:

- TimeLong – the number of milliseconds that the malware waits when the C2 servers are not responding

- TimeShort – the number of milliseconds between requesting different commands from the C2 server
- Security – password used to perform some sort of authentication
- Hosts – list of C2 domains and port numbers

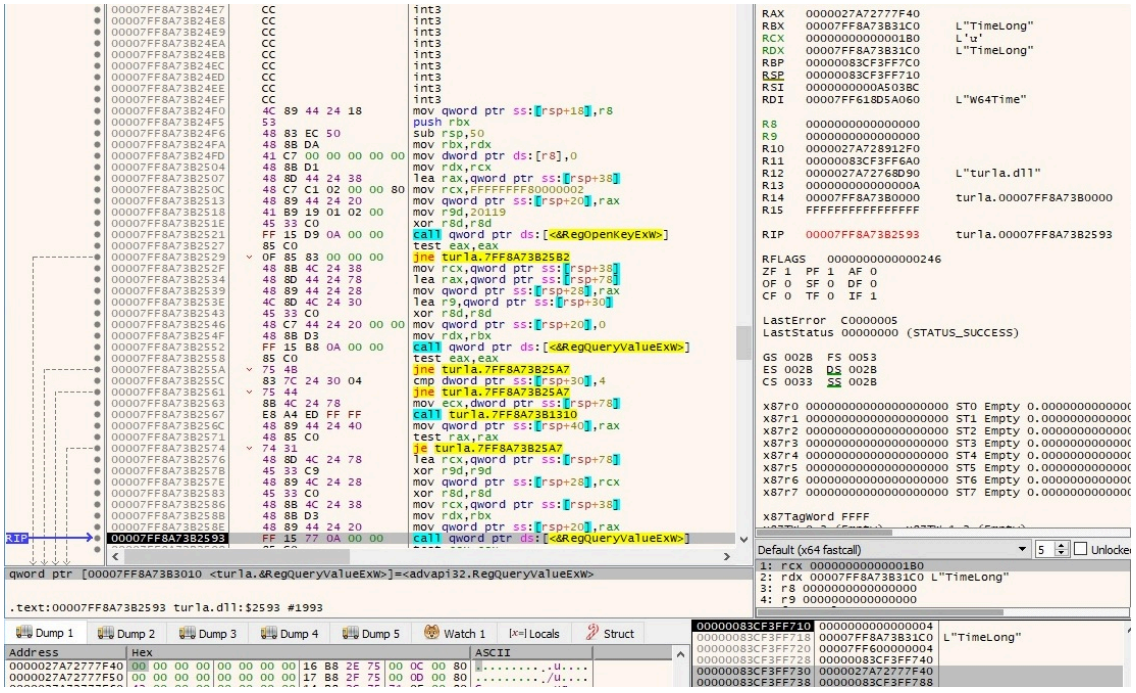


Figure 6

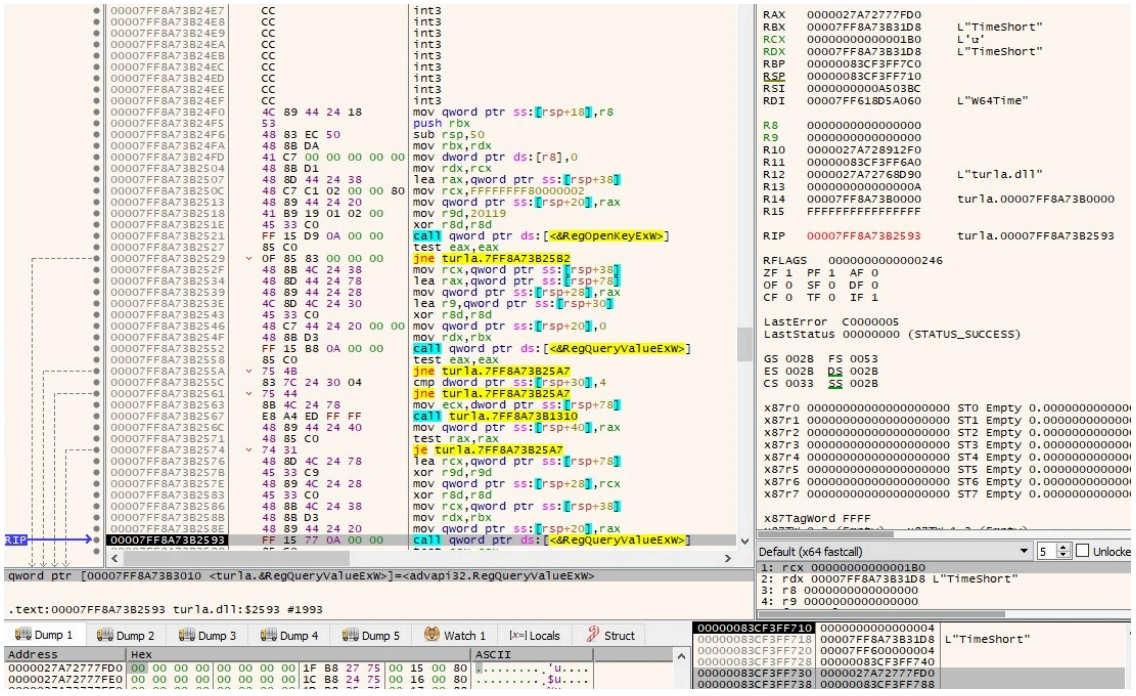


Figure 7

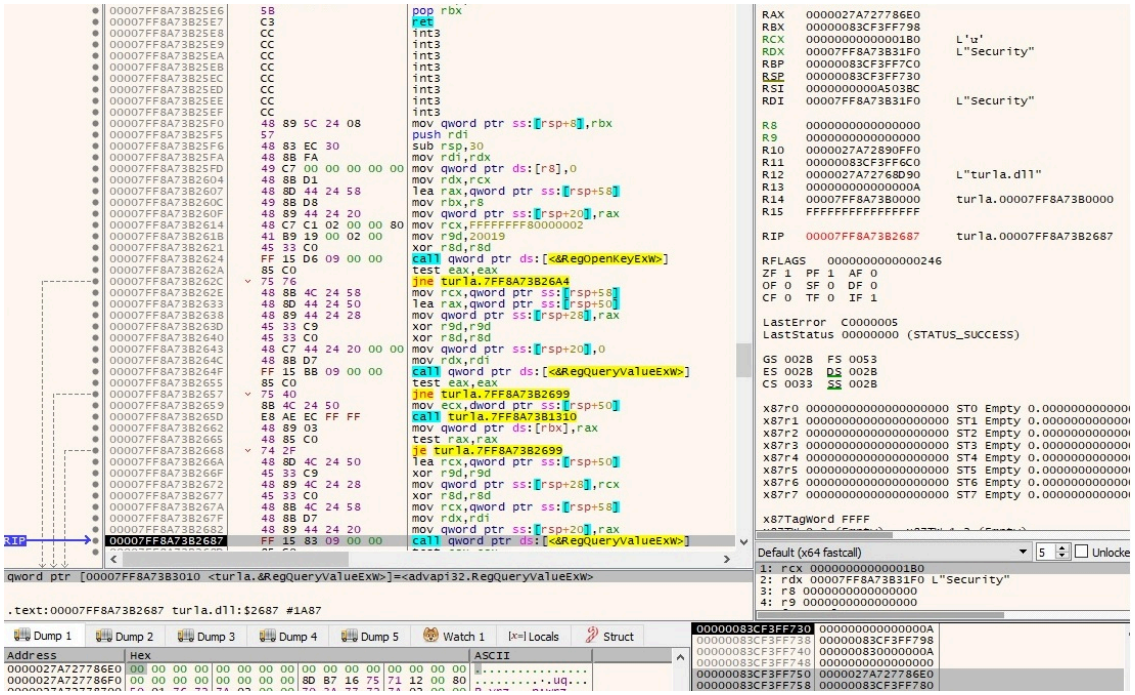


Figure 8

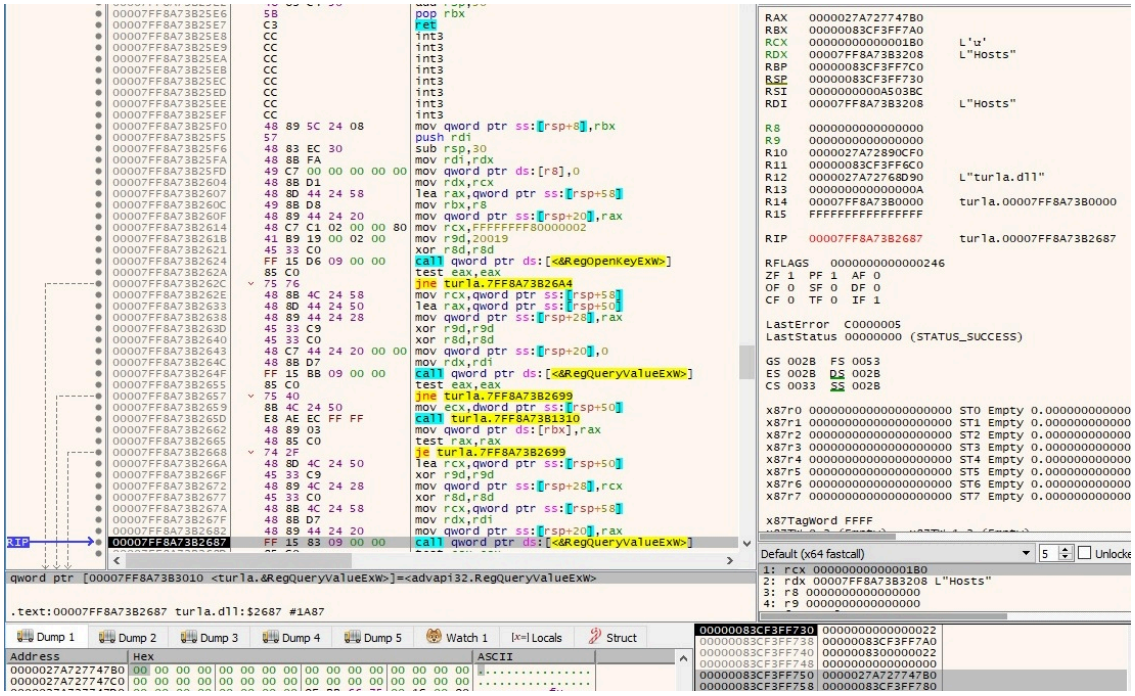


Figure 9

The malware passes the C2 IPs and port numbers to the CommandLineToArgvW routine and extracts an array of pointers to them (the C2 server is randomly chosen for testing purposes):

Figure 10

We’ve emulated network connections using FakeNet.

The malicious process opens the “SOFTWARE\Microsoft\Cryptography” registry key using RegOpenKeyExW (0x80000002 = **HKEY_LOCAL_MACHINE**, 0x20019 = **KEY_READ**):

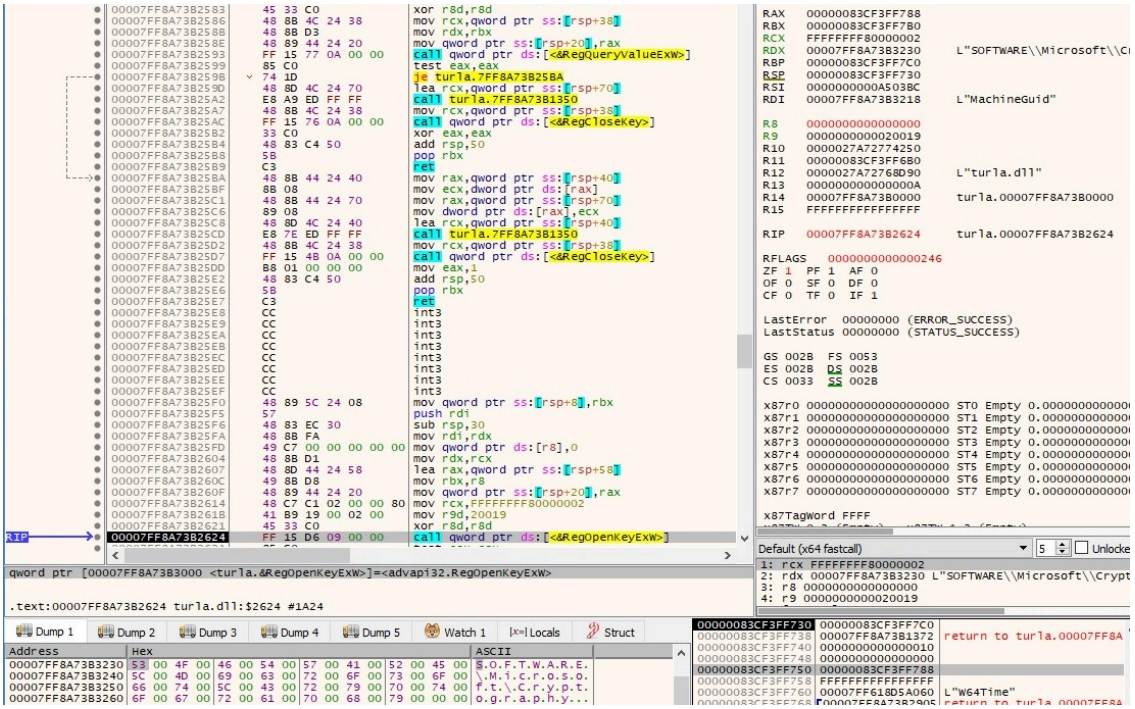


Figure 11

The “MachineGuid” value is extracted via a function call to RegQueryValueExW:

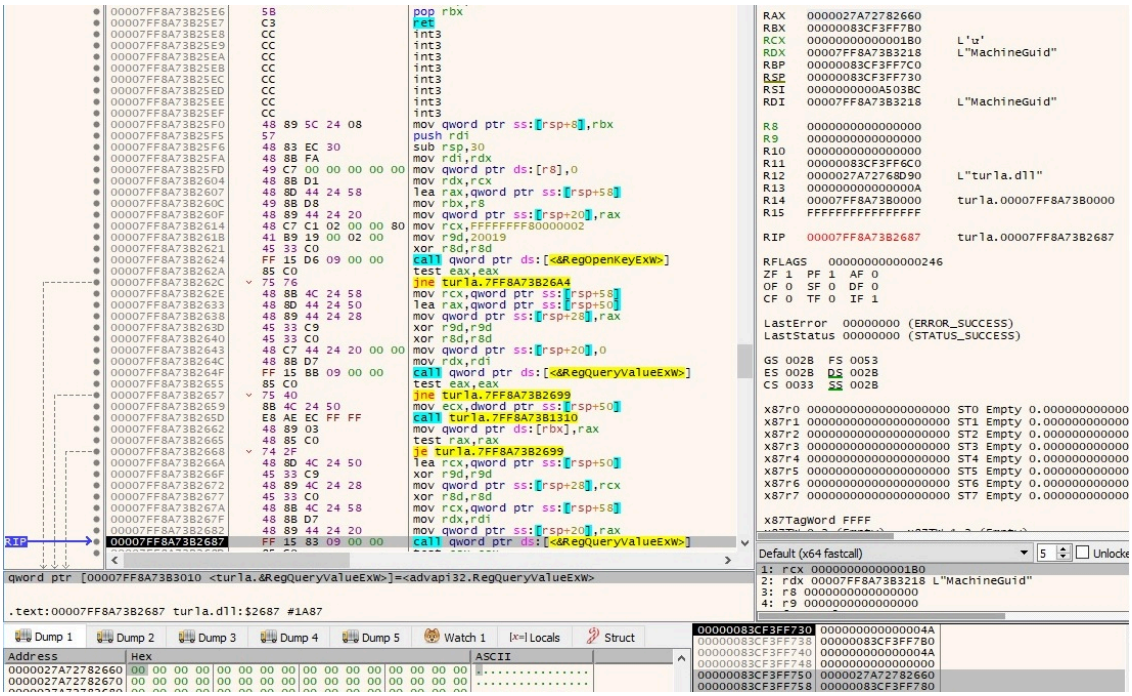


Figure 12

WinHttpOpen is utilized to initialize the use of WinHTTP functions:

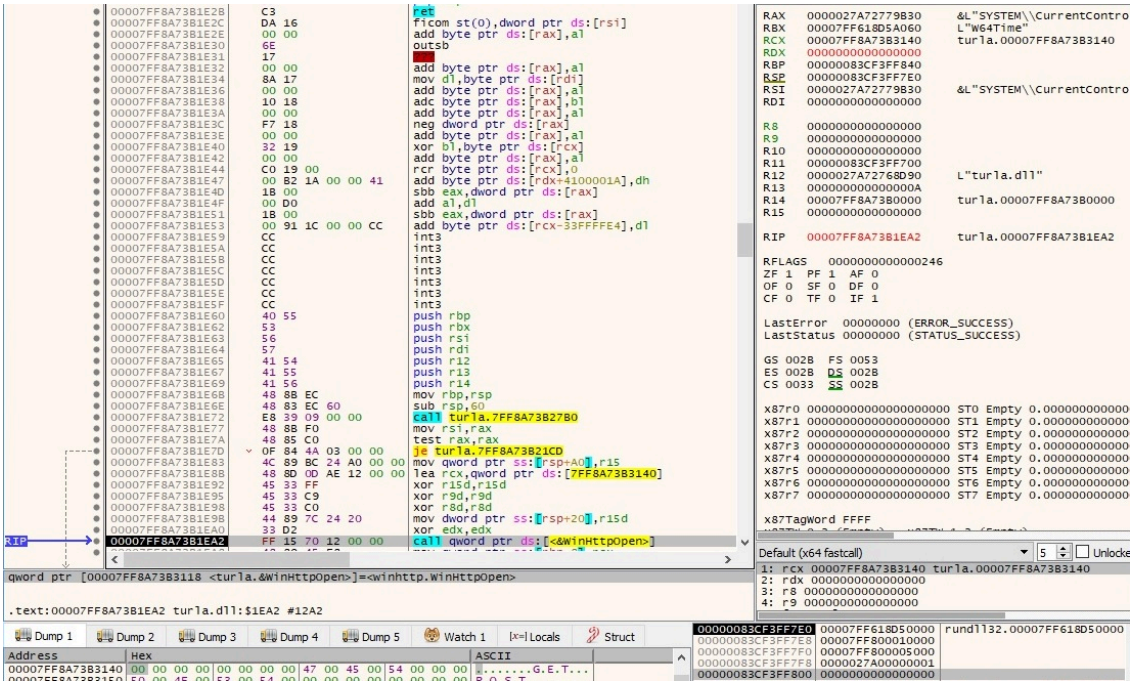


Figure 13

The file initializes a connection to the C2 server by calling the WinHttpConnect API:

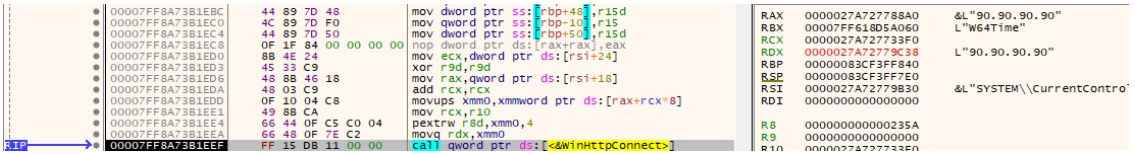


Figure 14

The WinHttpOpenRequest function is used to create a GET request handle (0x800000 = **WINHTTP_FLAG_SECURE**):

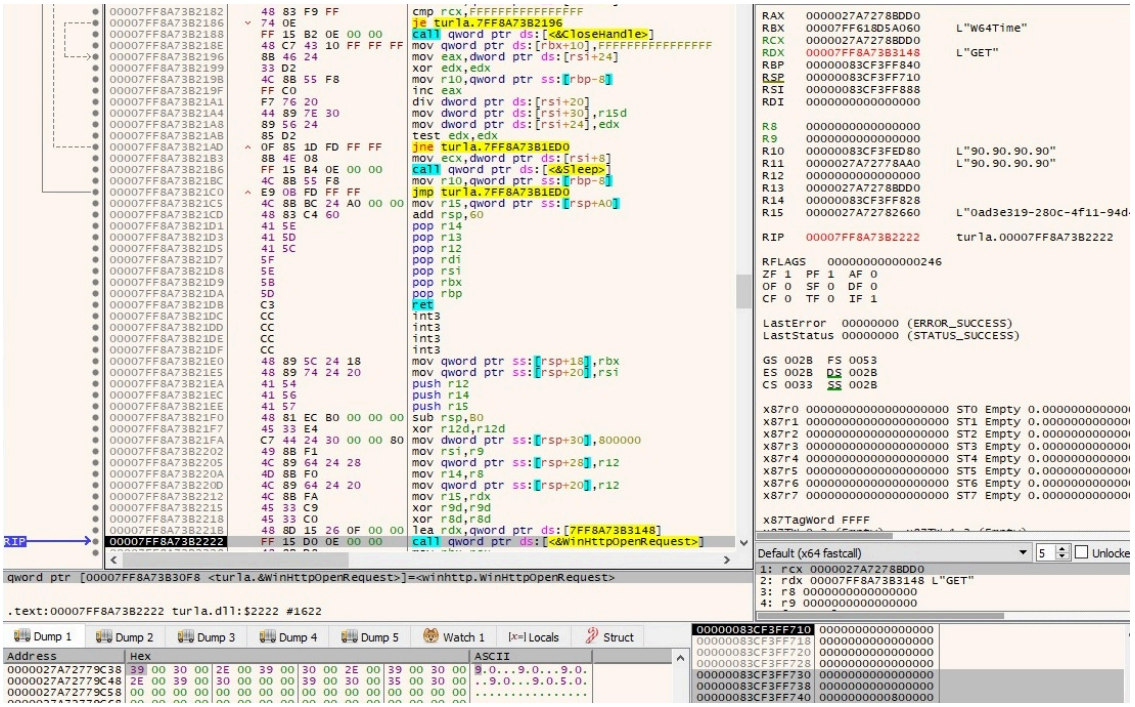


Figure 15

The process adds an HTTP request header called “Title” containing the Machine GUID to the HTTP request handle (0x20000000 = **HTTP_ADDREQ_FLAG_ADD**):

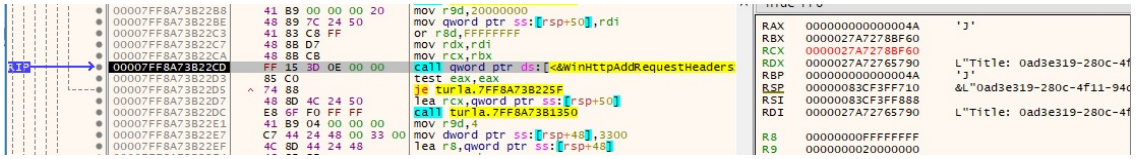


Figure 16

The security flags for the handle are set using WinHttpSetOption (0x1F = **WINHTTP_OPTION_SECURITY_FLAGS**, 0x3300 = **WinHttpRequestOption_SslErrorIgnoreFlags**):

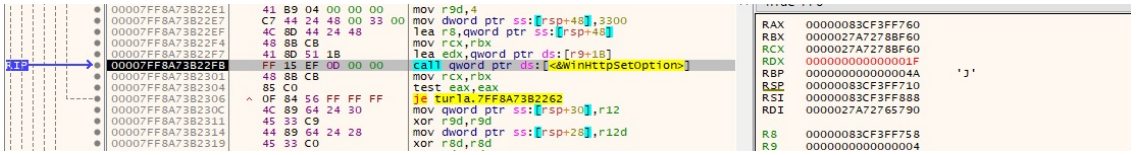


Figure 17

The malicious file sends the request to the C2 server using the WinHttpSendRequest routine:

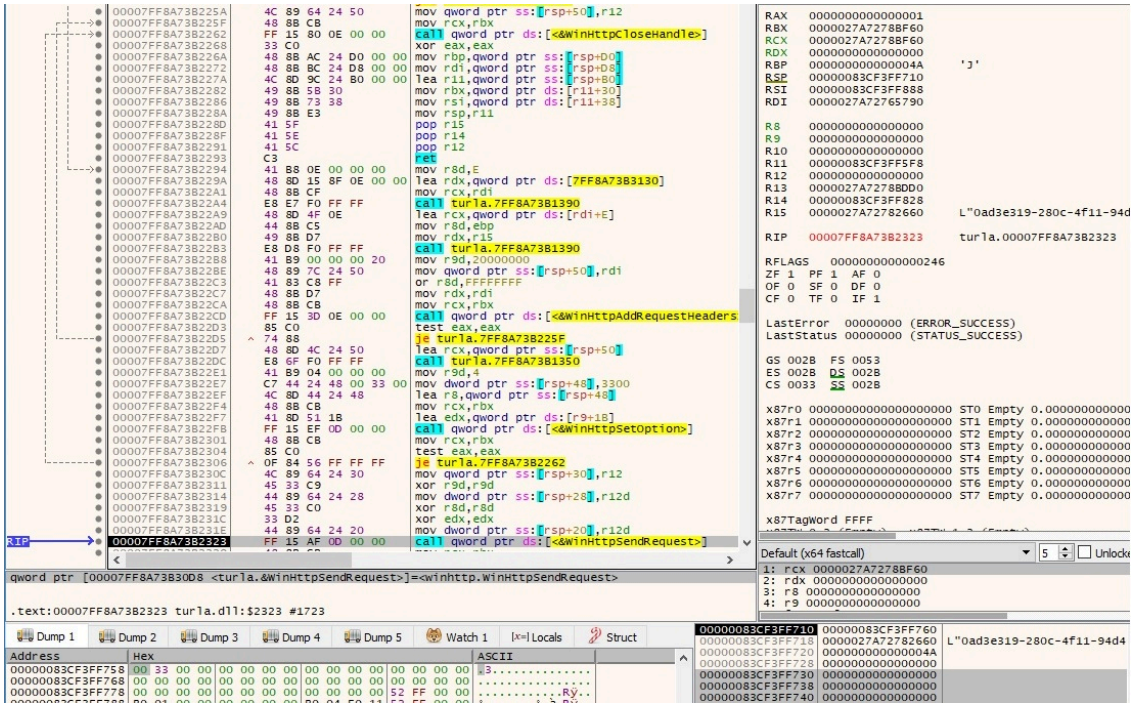


Figure 18

WinHttpReceiveResponse is used to receive the response to the GET request initiated above:

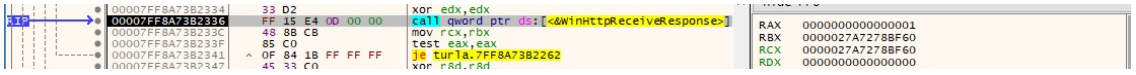


Figure 19

The binary obtains header information associated with the request by calling the WinHttpQueryHeaders API (0x26 = **WINHTTP_QUERY_TITLE**):

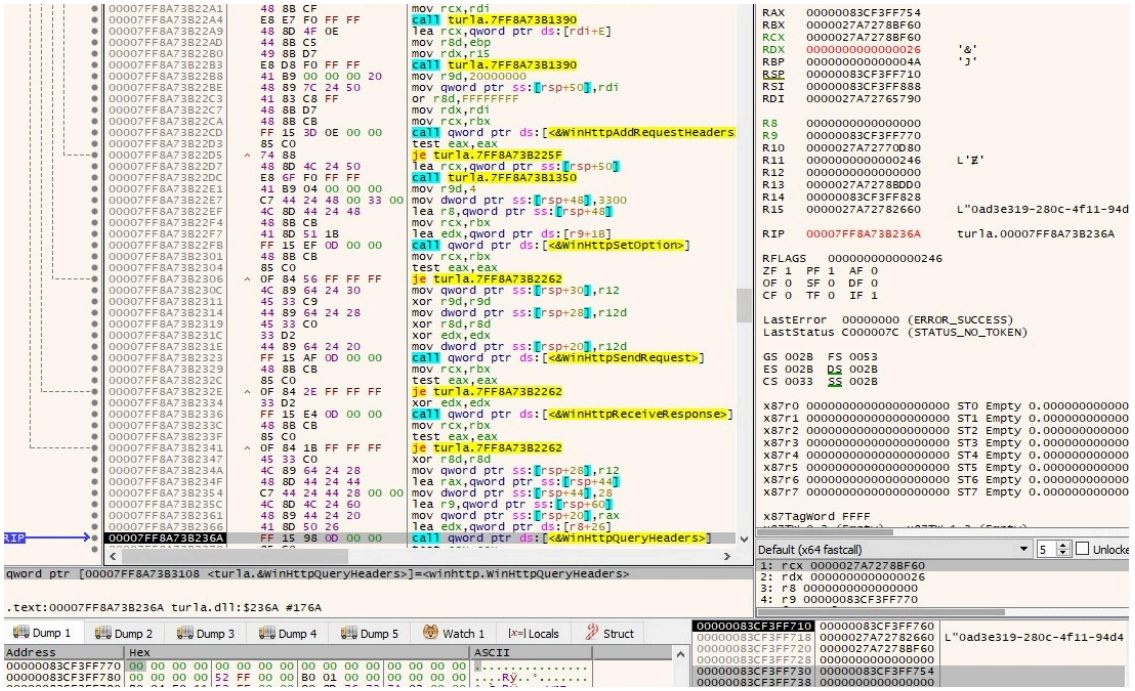


Figure 20

WinHttpQueryDataAvailable is utilized to extract the amount of data, in bytes, available to be read with WinHttpReadData:

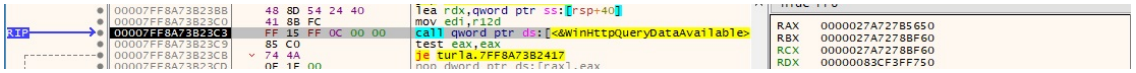


Figure 21

The response from the server is copied to a buffer via a call to WinHttpReadData:

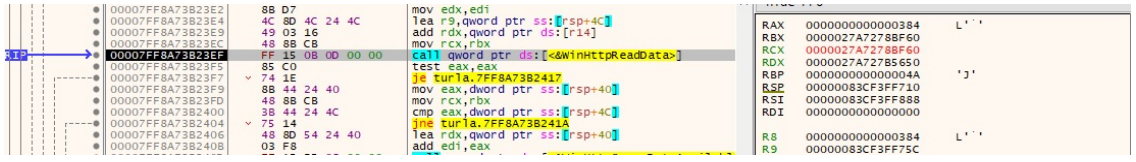


Figure 22

TinyTurla implements 12 different commands depending on the 1st byte received in the response. It uses a switch statement to execute a particular function:

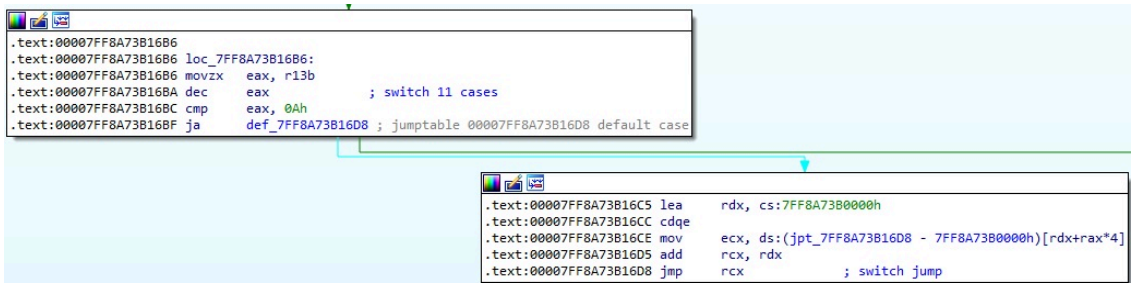


Figure 23

1st byte = 0x00 – Authentication

The backdoor compares the “Security” value with a string starting from the 2nd byte in the response:

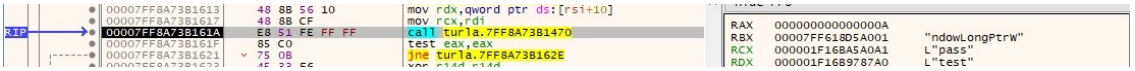


Figure 24

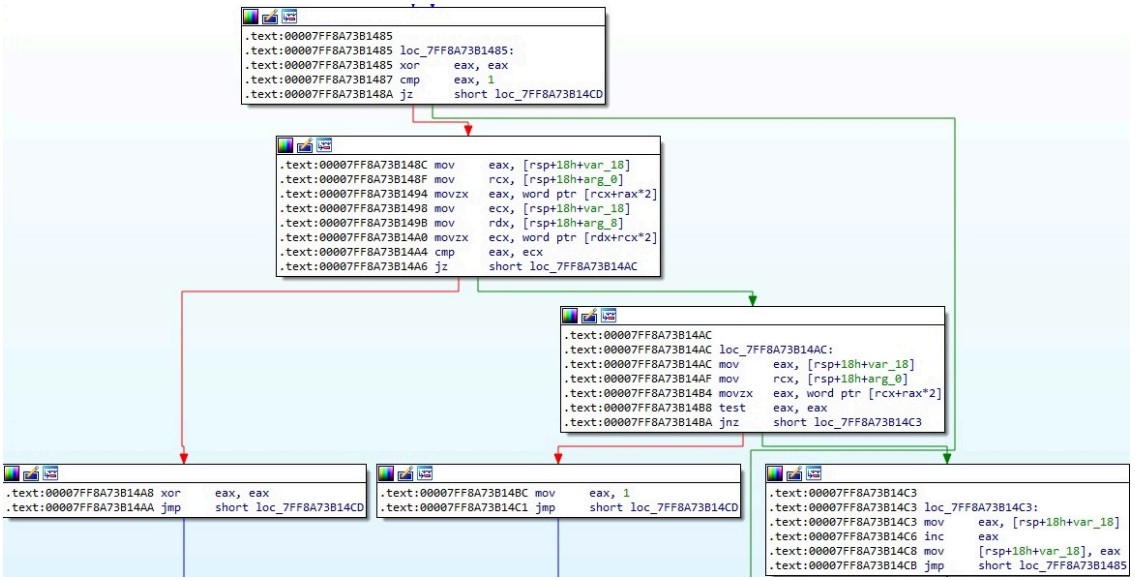


Figure 25

Whether the two strings are equal, the malware sends “00 00” to the C2 server. Otherwise, it sends “00 03”, indicating an unsuccessful “authentication”.

1st byte = 0x01 – create a process

The binary creates a process specified by the C2 server in the response (0x08000000 = **CREATE_NO_WINDOW**):

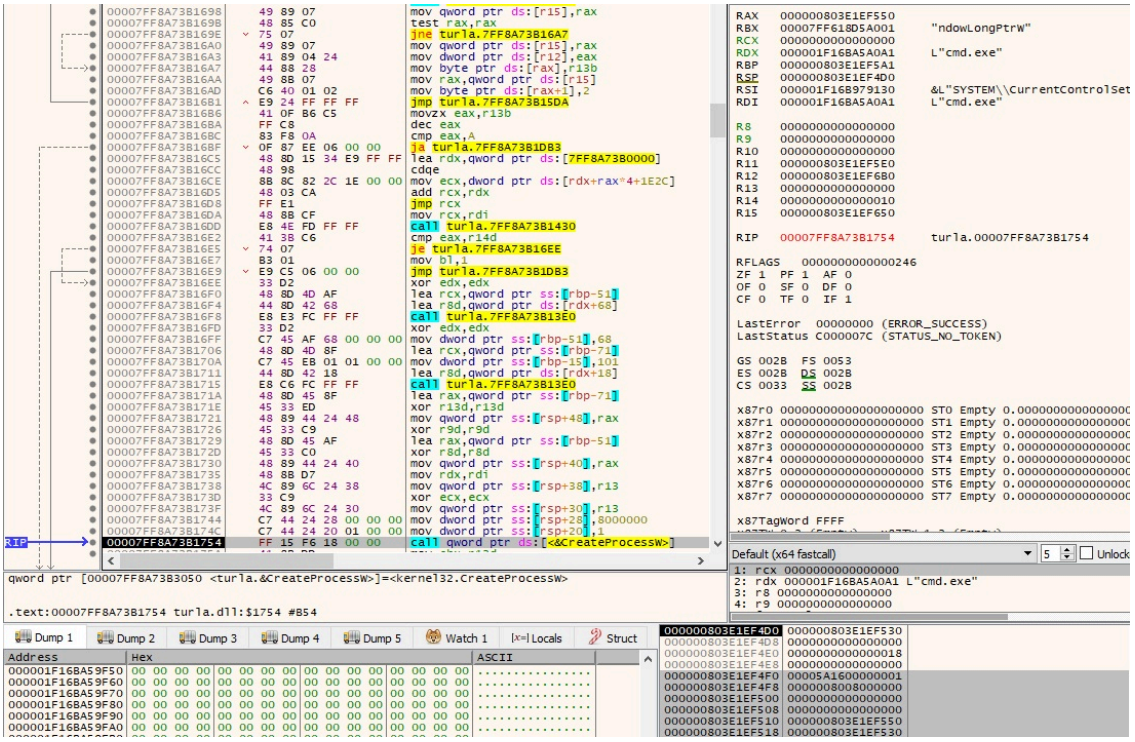


Figure 26

The WinHttpOpenRequest routine is used to create a POST request handle (0x800000 = **WINHTTP_FLAG_SECURE**):

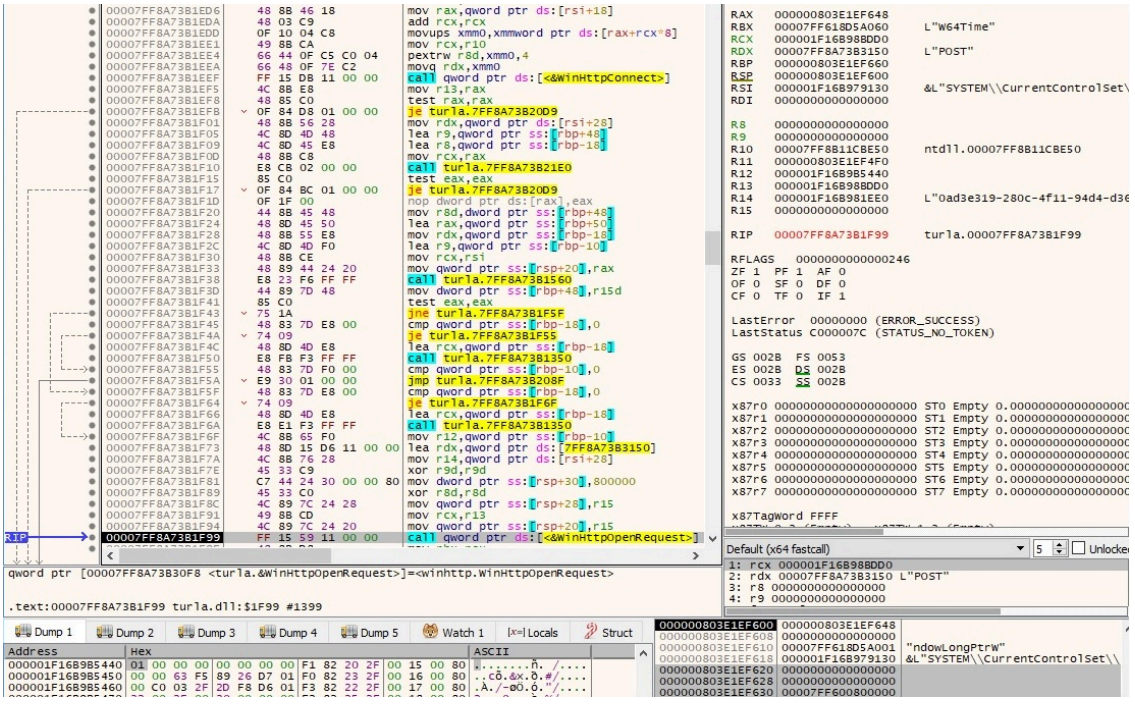


Figure 27

The backdoor adds an HTTP request header called “Title” that contains the Machine GUID to the request handle (0x20000000 = **HTTP_ADDREQ_FLAG_ADD**):

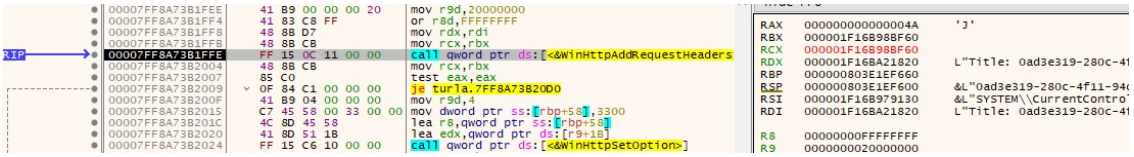


Figure 28

The security flags for the handle are set using WinHttpSetOption (0x1F = **WINHTTP_OPTION_SECURITY_FLAGS**, 0x3300 = **WinHttpRequestOption_SslErrorIgnoreFlags**):

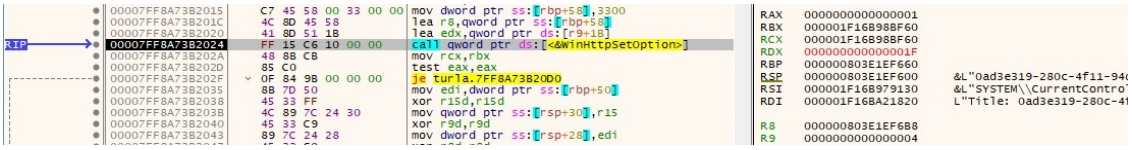


Figure 29

The malicious process sends the POST request to the C2 server by calling the WinHttpSendRequest API:

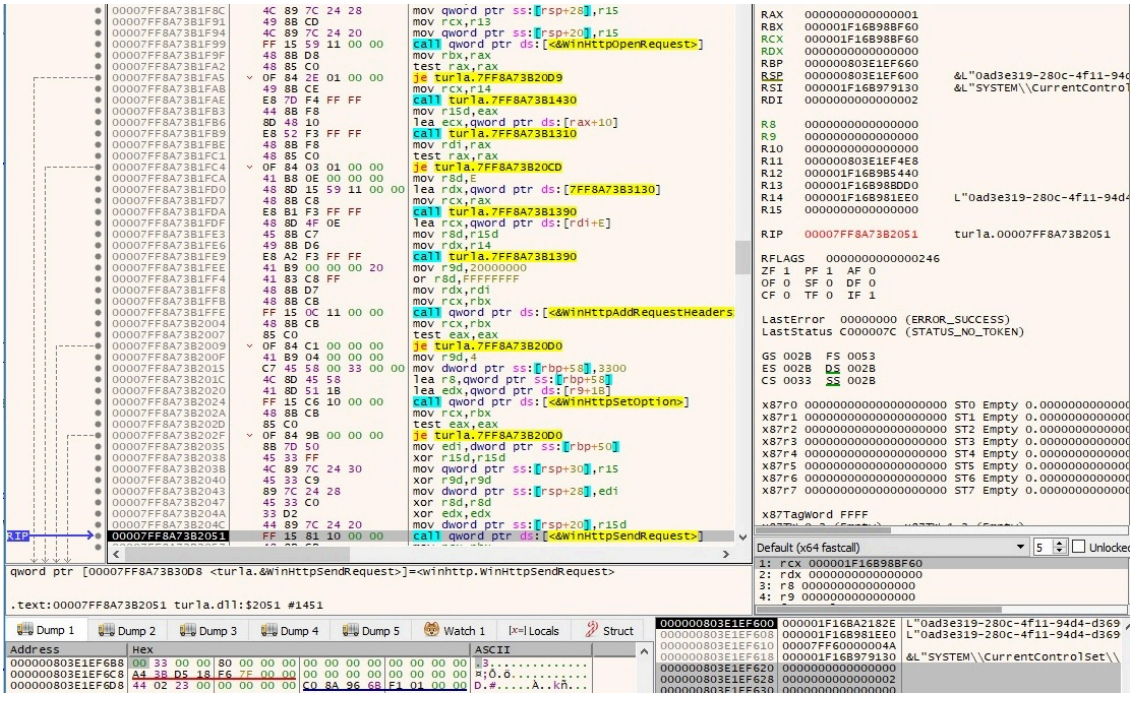


Figure 30

A confirmation message “01 00” is sent to the C2 server using WinHttpWriteData:

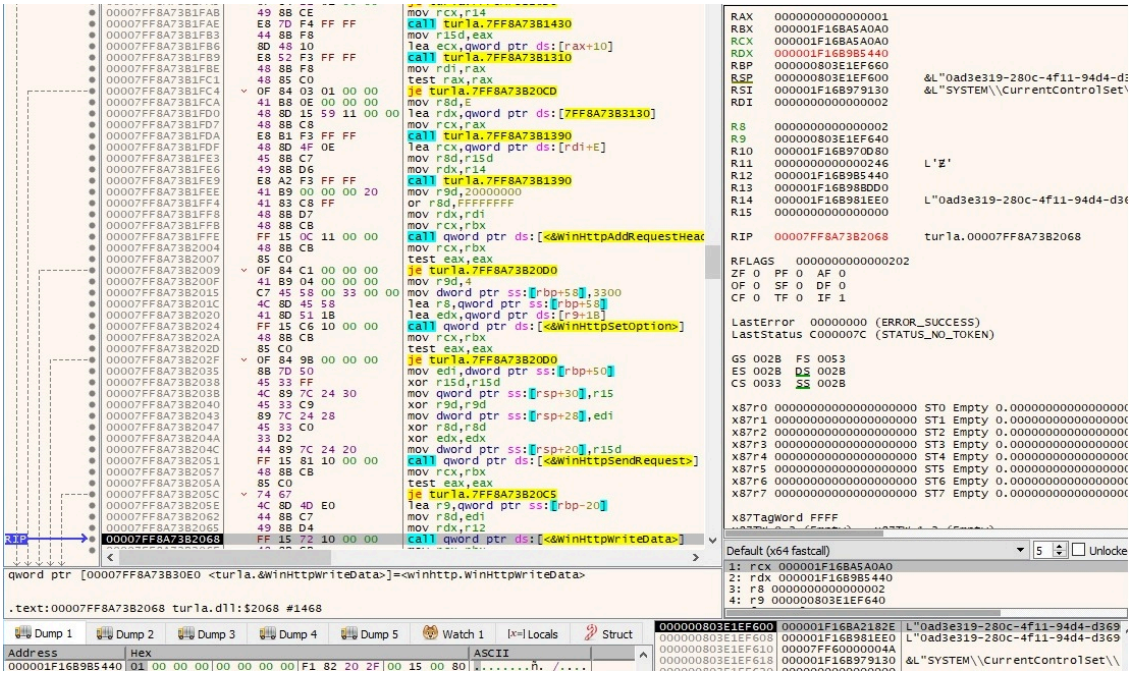


Figure 31

WinHttpReceiveResponse is utilized to halt the process until it receives the response to the HTTP request:

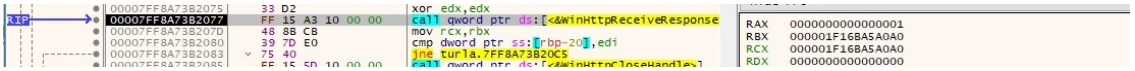


Figure 32

The backdoor sleeps for “TimeShort” milliseconds and waits for further instructions:

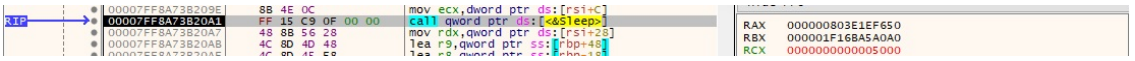


Figure 33

1st byte = 0x02 – create a process and exfiltrate its output

The malicious file creates an anonymous pipe and returns handles to the read/write ends of the pipe:

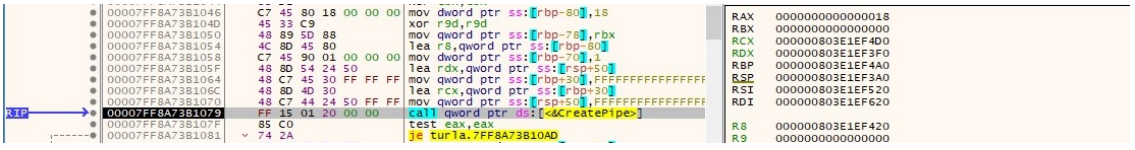


Figure 34

The write handle is set to be inherited by calling the SetHandleInformation routine (0x1 = **HANDLE_FLAG_INHERIT**):

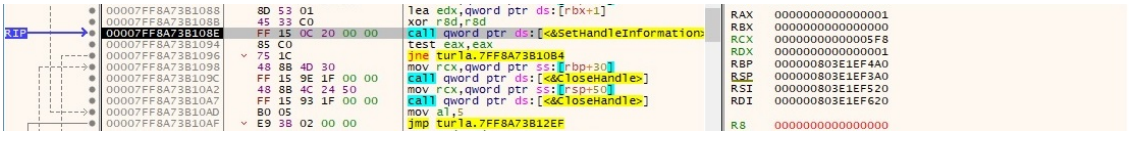


Figure 35

A second anonymous pipe is created via a function call to CreatePipe:

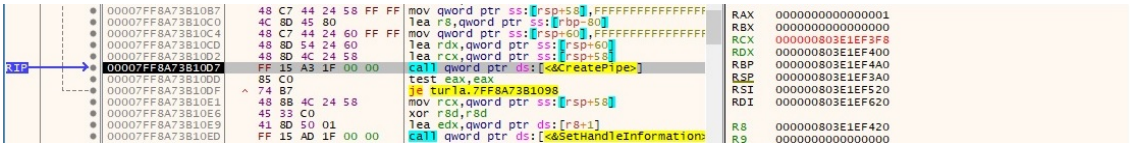


Figure 36

The read handle is set to be inherited by calling the SetHandleInformation routine (0x1 = **HANDLE_FLAG_INHERIT**):

Figure 37

The malware creates a process mentioned by the C2 server in the response (0x08000000 = **CREATE_NO_WINDOW**):

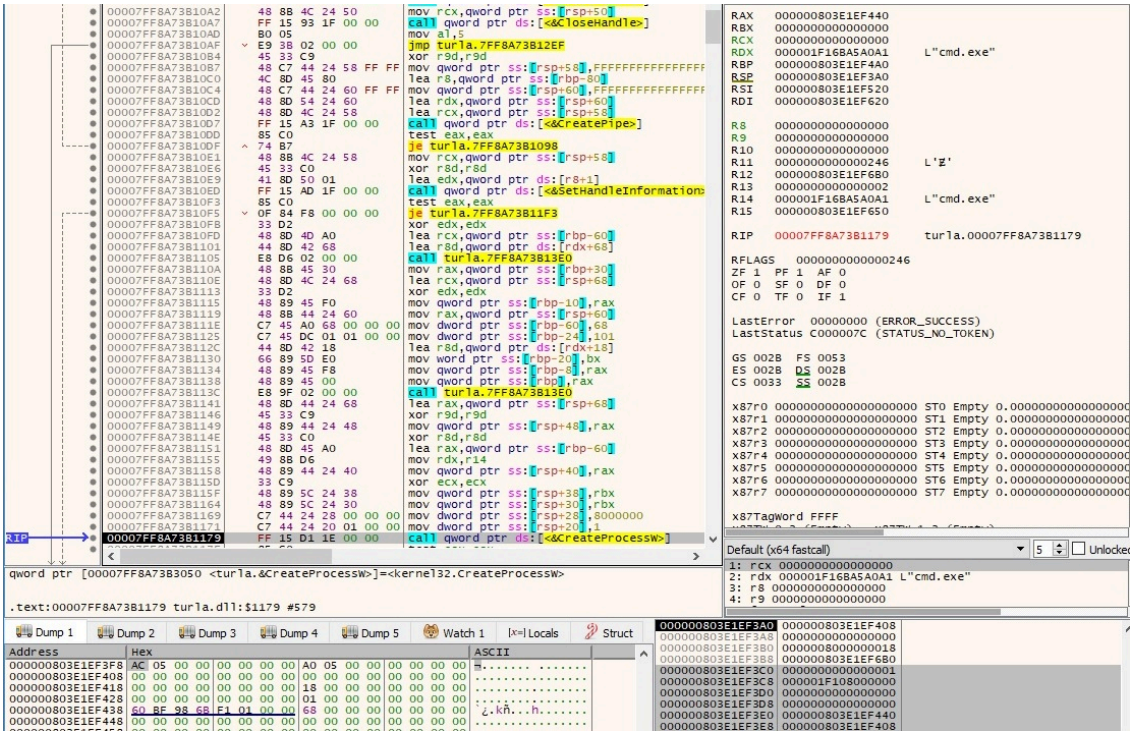


Figure 38

WaitForSingleObject is used to wait until the above process is in the signaled state or 0xEA60 = 60000ms = 60 seconds have elapsed:

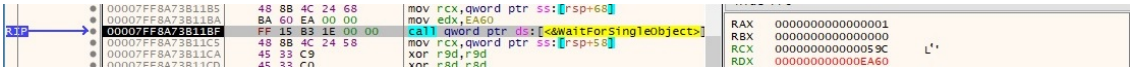


Figure 39

The output of the created process is copied from the anonymous pipe into a buffer by calling the PeekNamedPipe function:

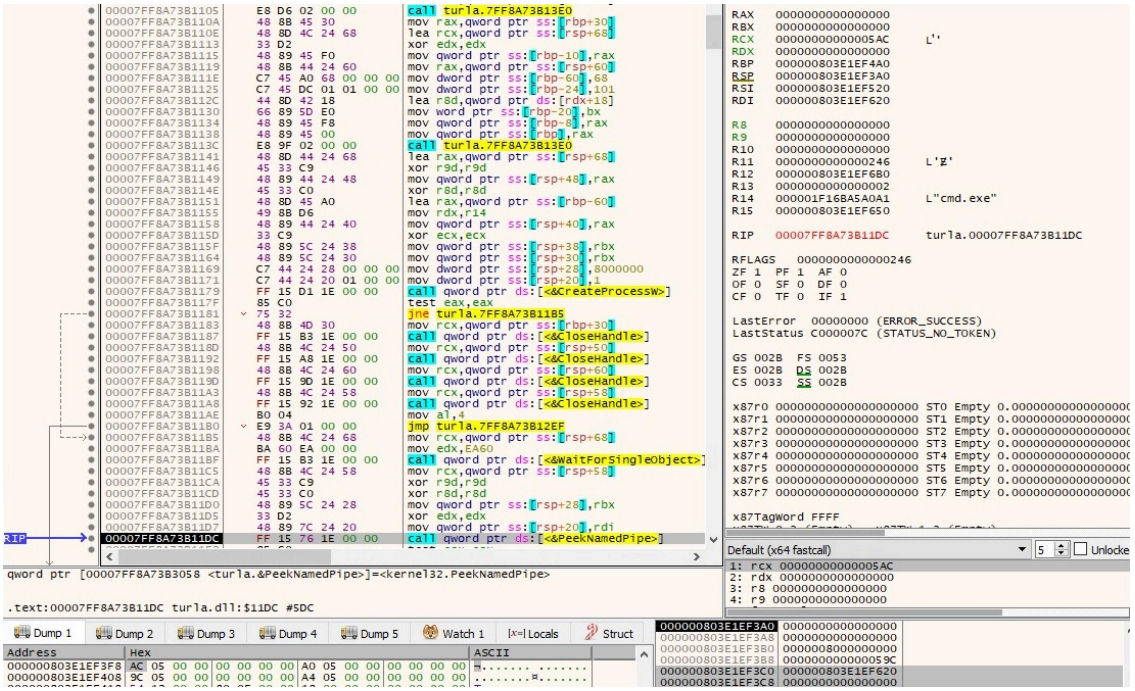


Figure 40

The process reads data from the pipe using ReadFile:

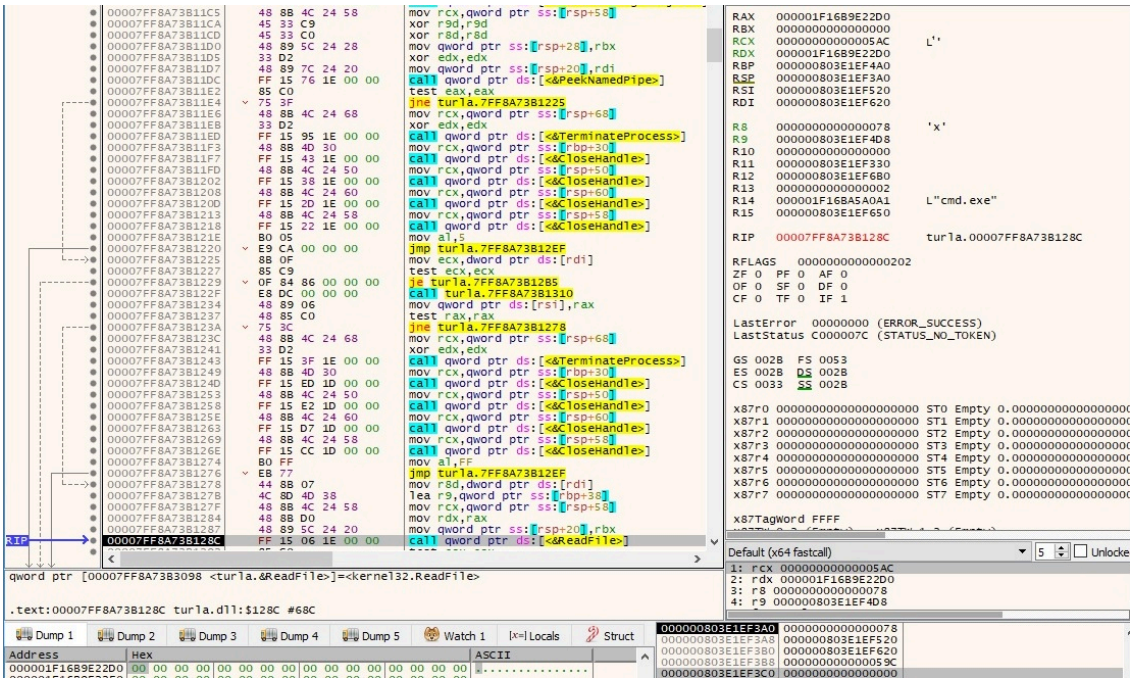
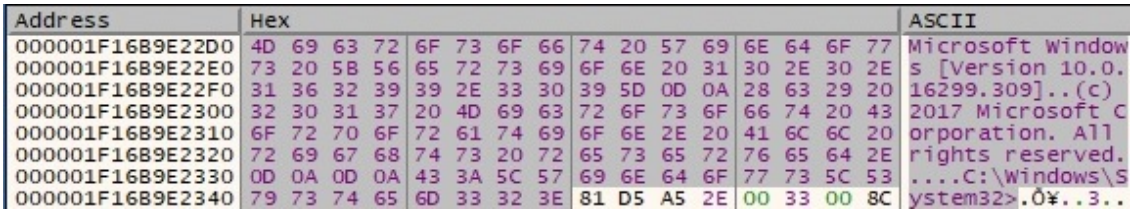


Figure 41



Address	Hex	ASCII
000001F16B9E22D0	4D 69 63 72 6F 73 6F 66 74 20 57 69 6E 64 6F 77	Microsoft Window
000001F16B9E22E0	73 20 5B 56 65 72 73 69 6F 6E 20 31 30 2E 30 2E	s [Version 10.0.
000001F16B9E22F0	31 36 32 39 39 2E 33 30 39 5D 0D 0A 28 63 29 20	16299.309]..(c)
000001F16B9E2300	32 30 31 37 20 4D 69 63 72 6F 73 6F 66 74 20 43	2017 Microsoft C
000001F16B9E2310	6F 72 70 6F 72 61 74 69 6F 6E 2E 20 41 6C 6C 20	orporation. All
000001F16B9E2320	72 69 67 68 74 73 20 72 65 73 65 72 76 65 64 2E	rights reserved.
000001F16B9E2330	0D 0A 0D 0A 43 3A 5C 57 69 6E 64 6F 77 73 5C 53	....C:\Windows\S
000001F16B9E2340	79 73 74 65 6D 33 32 3E 81 D5 A5 2E 00 33 00 8C	ystem32>.Õ¥..3..

Figure 42

The backdoor kills the process created above using the TerminateProcess routine:

Figure 43

The execution flow of creating a POST request (WinHttpOpenRequest -> WinHttpAddRequestHeaders -> WinHttpSetOption -> WinHttpSendRequest) is repeated and will not be detailed again. The process output is exfiltrated to the CnC server:

Figure 44

1st byte = 0x03 – create and populate a file

The backdoor creates a file specified by the C2 server using CreateFileW (0x40000000 = **GENERIC_WRITE**, 0x2 = **CREATE_ALWAYS**, 0x80 = **FILE_ATTRIBUTE_NORMAL**):

Figure 45

The WriteFile API is utilized to populate the file with data received from the server:

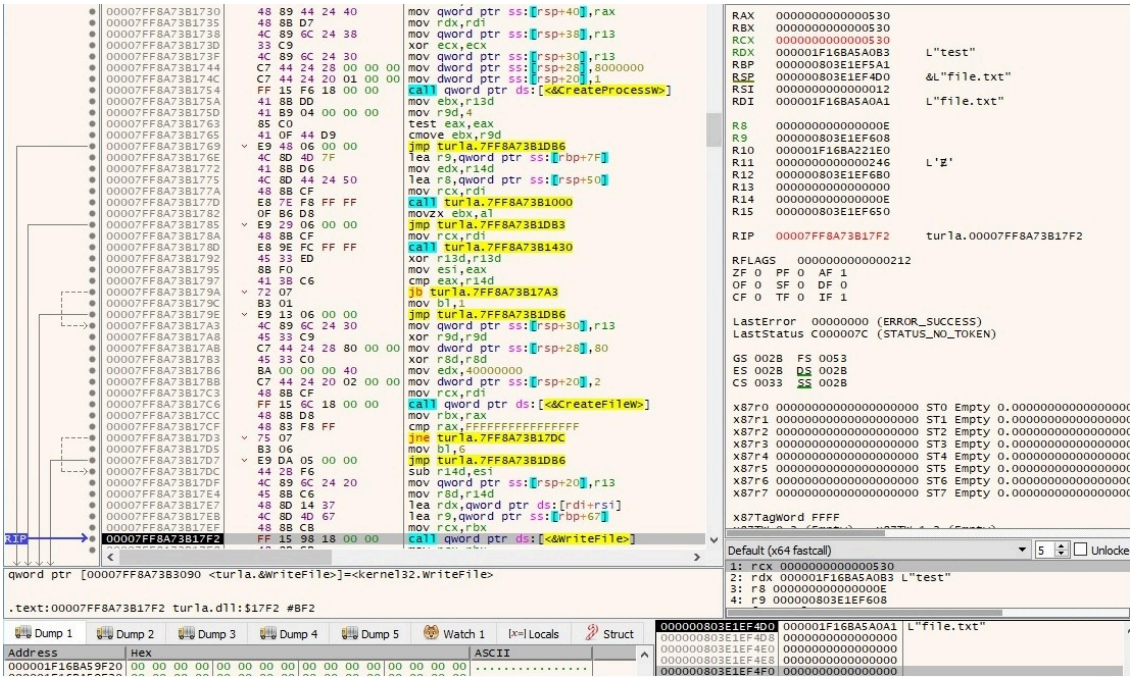


Figure 46

A confirmation message “03 00” is sent to the C2 server.

1st byte = 0x04 – exfiltrate a file to the C2 server

The process opens a file nominated by the server using CreateFileW (0x80000000 = **GENERIC_READ**, 0x3 = **OPEN_EXISTING**, 0x80 = **FILE_ATTRIBUTE_NORMAL**):

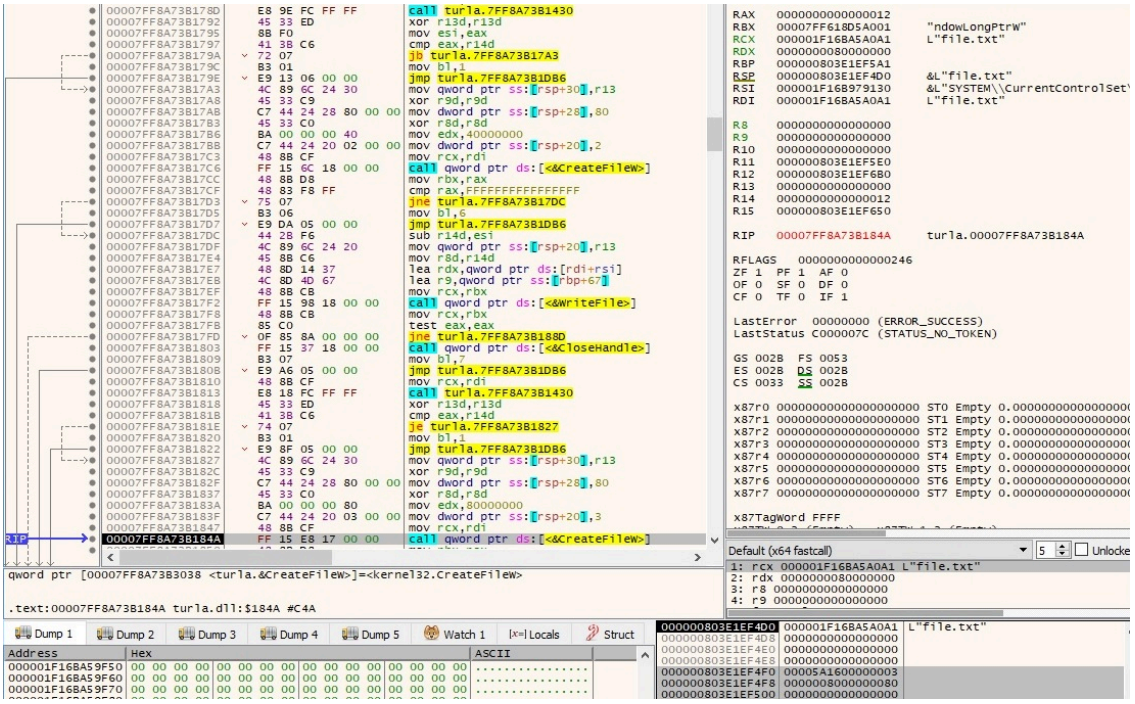


Figure 47

The size of the file is retrieved by calling the GetFileSize routine:

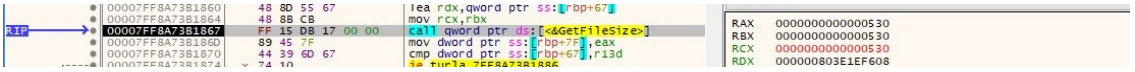


Figure 48

The file content is copied to a buffer via a function call to ReadFile:

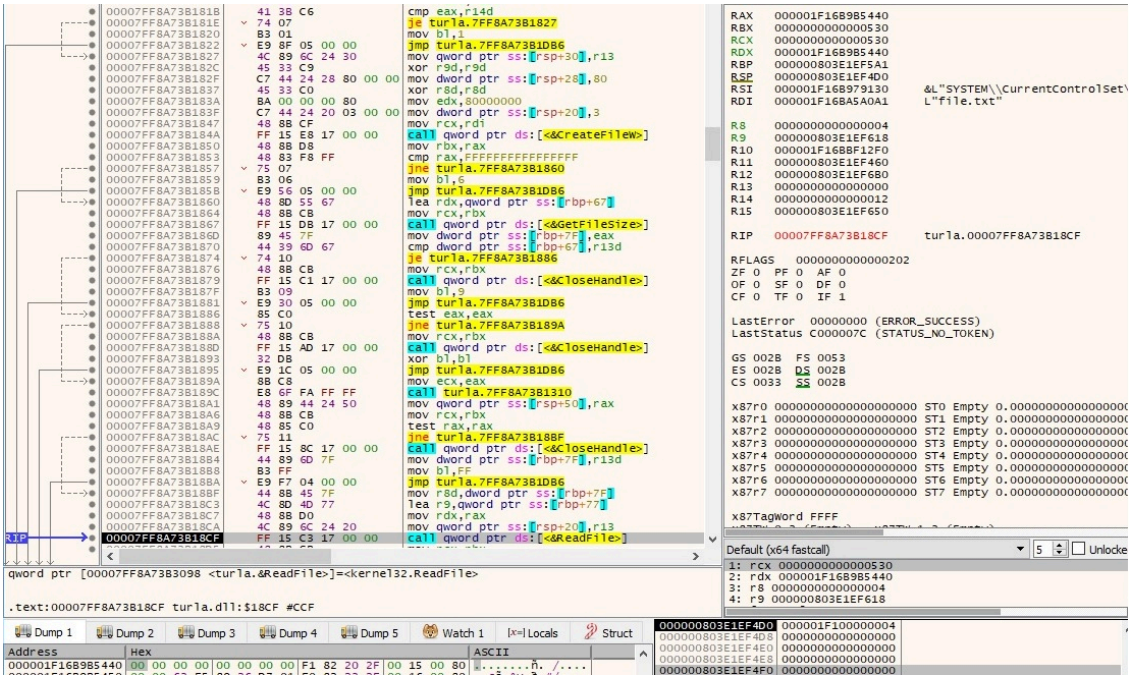


Figure 49

The content extracted above is transmitted to the CnC server:

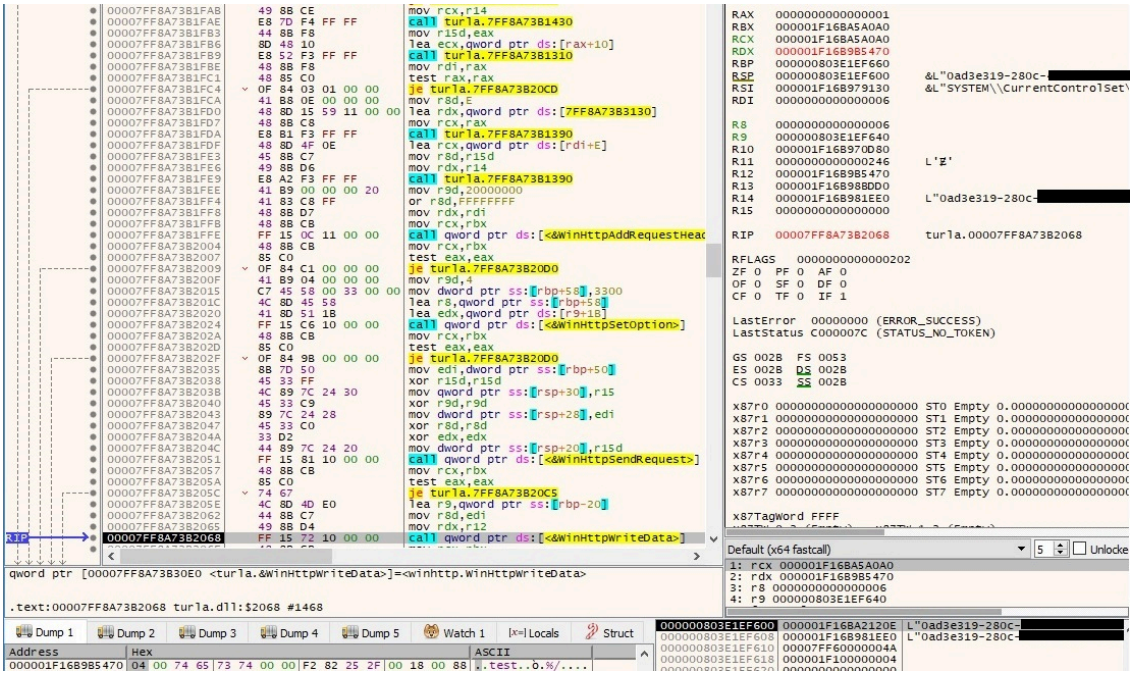


Figure 50

1st byte = 0x05 – spawn a new process

The malicious process creates an anonymous pipe using the CreatePipe API:

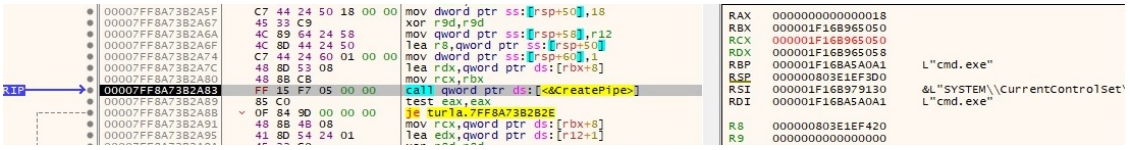


Figure 51

The write handle is set to be inherited by calling the SetHandleInformation routine (0x1 = **HANDLE_FLAG_INHERIT**):

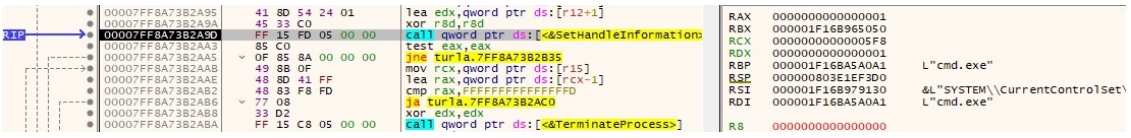


Figure 52

A second anonymous pipe is created by the malware:

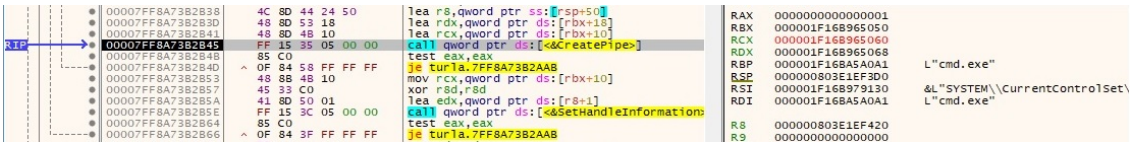


Figure 53

The read handle is set to be inherited by calling the SetHandleInformation routine (0x1 = **HANDLE_FLAG_INHERIT**):

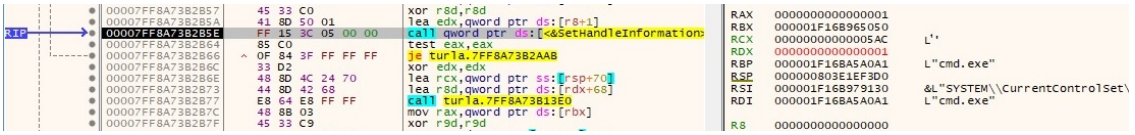


Figure 54

CreateProcessW is used to create a process specified by the C2 server (0x08000000 = **CREATE_NO_WINDOW**):

Figure 55

A confirmation message “05 00” is sent to the C2 server.

1st byte = 0x06 – kill a process

The binary kills the process spawned in the above command by calling TerminateProcess:

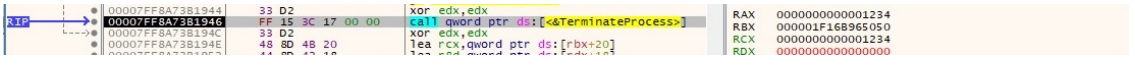


Figure 56

A confirmation message “06 00” is sent to the C2 server.

1st byte = 0x07 – read/write to a pipe

The WriteFile API is utilized to write data transmitted by the CnC server to a pipe created earlier:

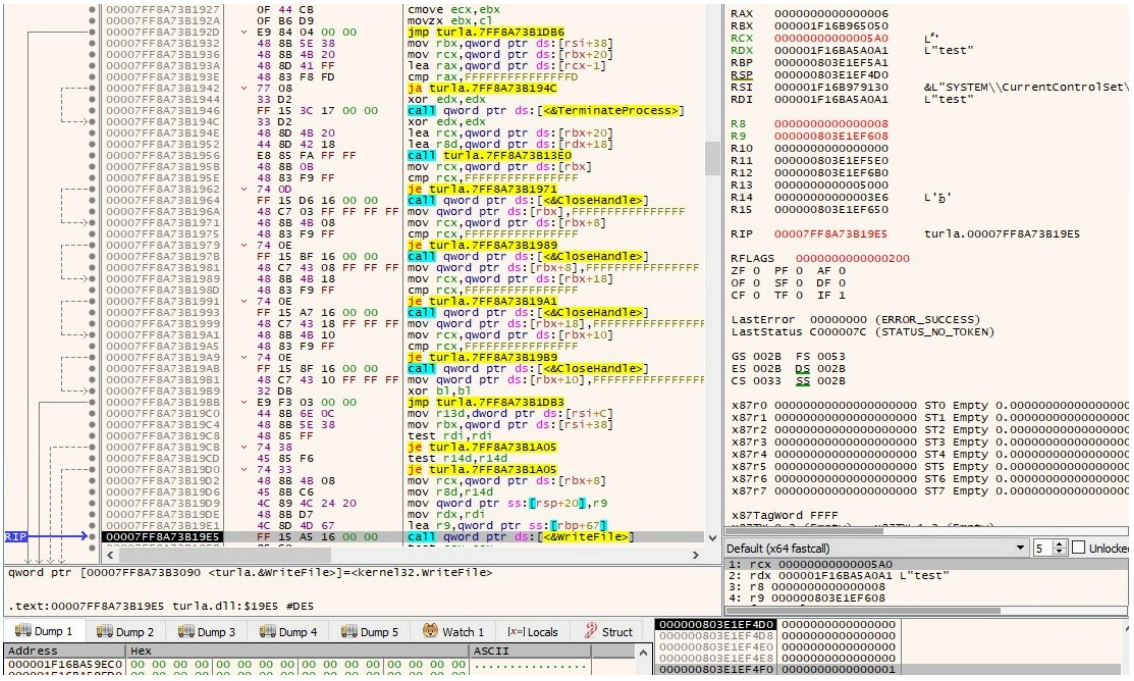


Figure 57

The process reads data that is available through the pipe using the PeekNamedPipe and ReadFile APIs:

```
.text:00007FF8A73B1A05
.text:00007FF8A73B1A05 loc_7FF8A73B1A05:       ; dwMilliseconds
.text:00007FF8A73B1A05 mov     ecx, r13d
.text:00007FF8A73B1A08 call    cs:Sleep
.text:00007FF8A73B1A0E mov     rcx, [rbx+10h]  ; hNamedPipe
.text:00007FF8A73B1A12 lea     rax, [rbp+4Fh+nNumberOfBytesToRead]
.text:00007FF8A73B1A16 xor     r13d, r13d
.text:00007FF8A73B1A19 xor     r9d, r9d        ; lpBytesRead
.text:00007FF8A73B1A1C mov     qword ptr [rsp+120h+dwCreationFlags], r13 ; lpBytesLeftThisMessage
.text:00007FF8A73B1A21 xor     r8d, r8d        ; nBufferSize
.text:00007FF8A73B1A24 xor     edx, edx        ; lpBuffer
.text:00007FF8A73B1A26 mov     qword ptr [rsp+120h+bInheritHandles], rax ; lpTotalBytesAvail
.text:00007FF8A73B1A2B call    cs:PeekNamedPipe
.text:00007FF8A73B1A31 test    eax, eax
.text:00007FF8A73B1A33 jnz     short loc_7FF8A73B1A48
```

Figure 58

```
.text:00007FF8A73B1A7F
.text:00007FF8A73B1A7F loc_7FF8A73B1A7F:       ; nNumberOfBytesToRead
.text:00007FF8A73B1A7F mov     r8d, [rbp+4Fh+nNumberOfBytesToRead]
.text:00007FF8A73B1A83 lea     r9, [rbp+4Fh+NumberOfBytesRead] ; lpNumberOfBytesRead
.text:00007FF8A73B1A87 mov     rcx, [rbx+10h]  ; hFile
.text:00007FF8A73B1A8B mov     rdx, rax        ; lpBuffer
.text:00007FF8A73B1A8E mov     qword ptr [rsp+120h+bInheritHandles], r13 ; lpOverlapped
.text:00007FF8A73B1A93 call    cs:ReadFile
.text:00007FF8A73B1A99 test    eax, eax
.text:00007FF8A73B1A9B jz      short loc_7FF8A73B1A5E
```

Figure 59

The pipe content extracted above is exfiltrated to the C2 server.

1st byte = 0x08 – modify the “TimeLong” registry value

The malware opens the “SYSTEM\CurrentControlSet\Services\W64Time\Parameters” registry key by calling the RegOpenKeyExW routine (0x80000002 = **HKEY_LOCAL_MACHINE**, 0x20006 = **KEY_WRITE**):

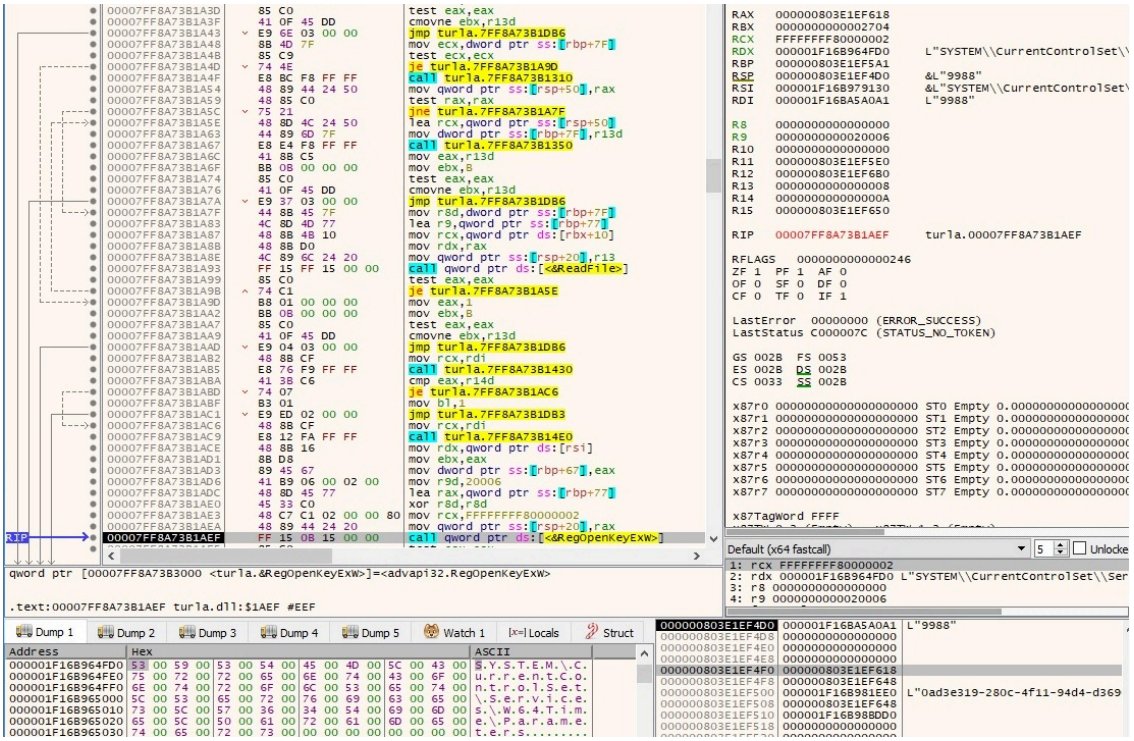


Figure 60

The “TimeLong” value is modified to a number sent by the C2 server:

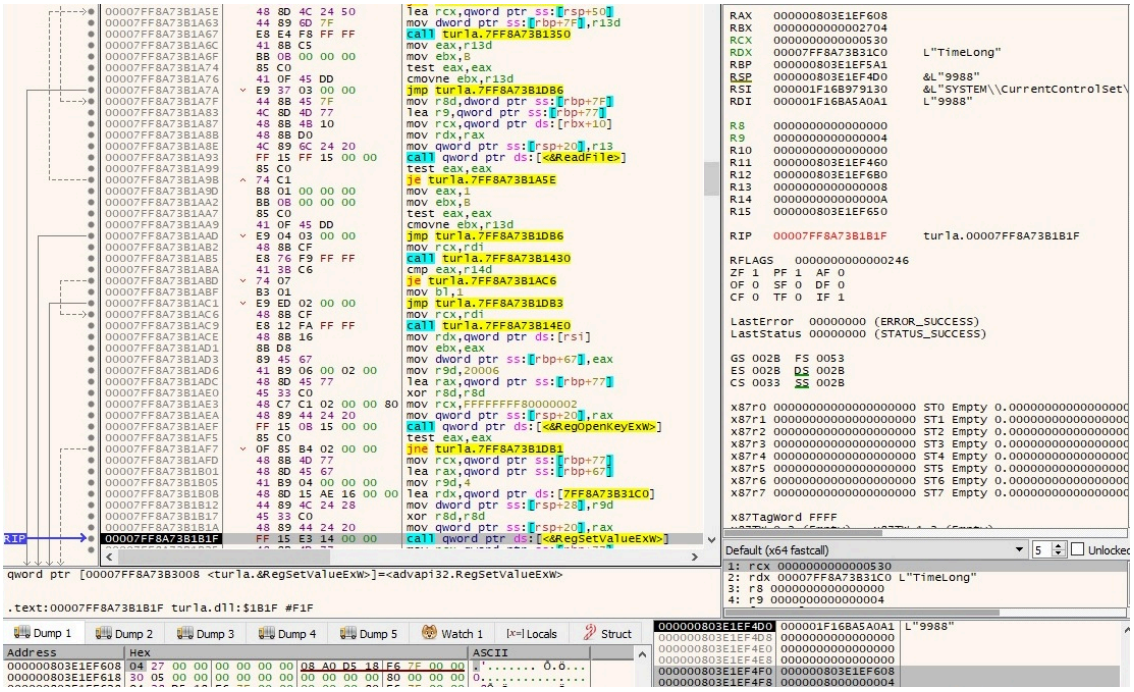


Figure 61

A confirmation message “08 00” is sent to the C2 server.

1st byte = 0x09 – modify the “TimeShort” registry value

This command is similar to the one from above. The “TimeShort” value is modified accordingly:

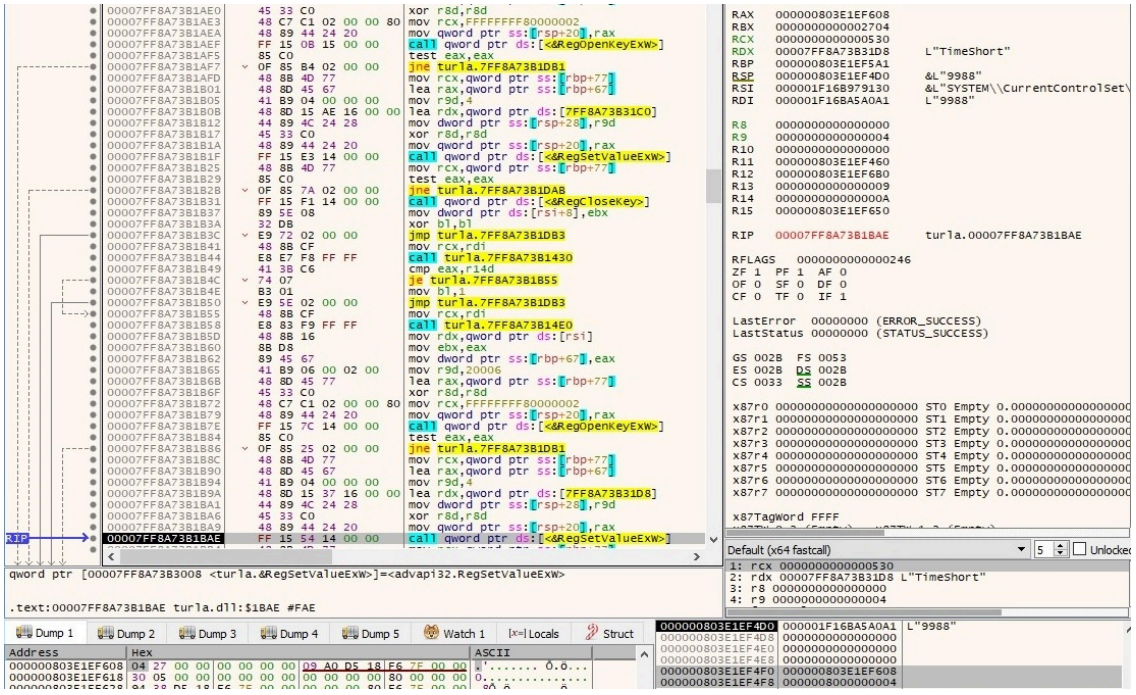


Figure 62

A confirmation message “09 00” is sent to the C2 server.

1st byte = 0x0A – modify the “Security” registry value

This command is similar to the one from above. The “Security” value used in the authentication process is changed by the backdoor:

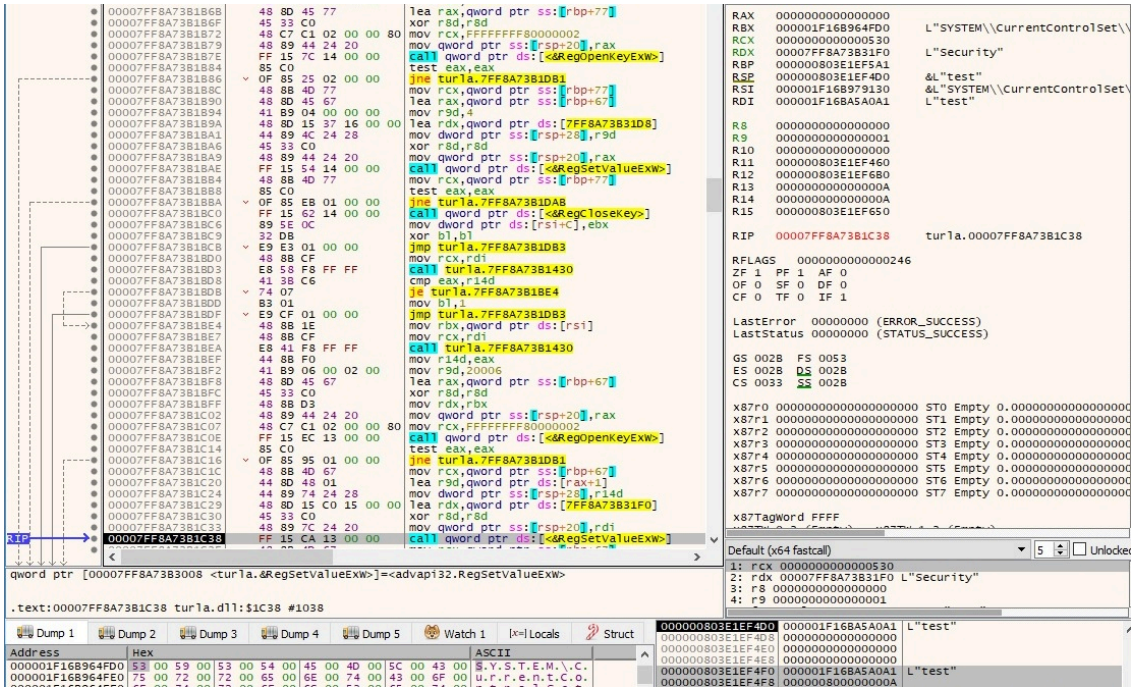


Figure 63

A confirmation message “0A 00” is sent to the C2 server.

1st byte = 0x0B – modify the “Hosts” registry value

This command is similar to the one from above. The “Hosts” value that contains the list of C2 servers is changed by the malware:

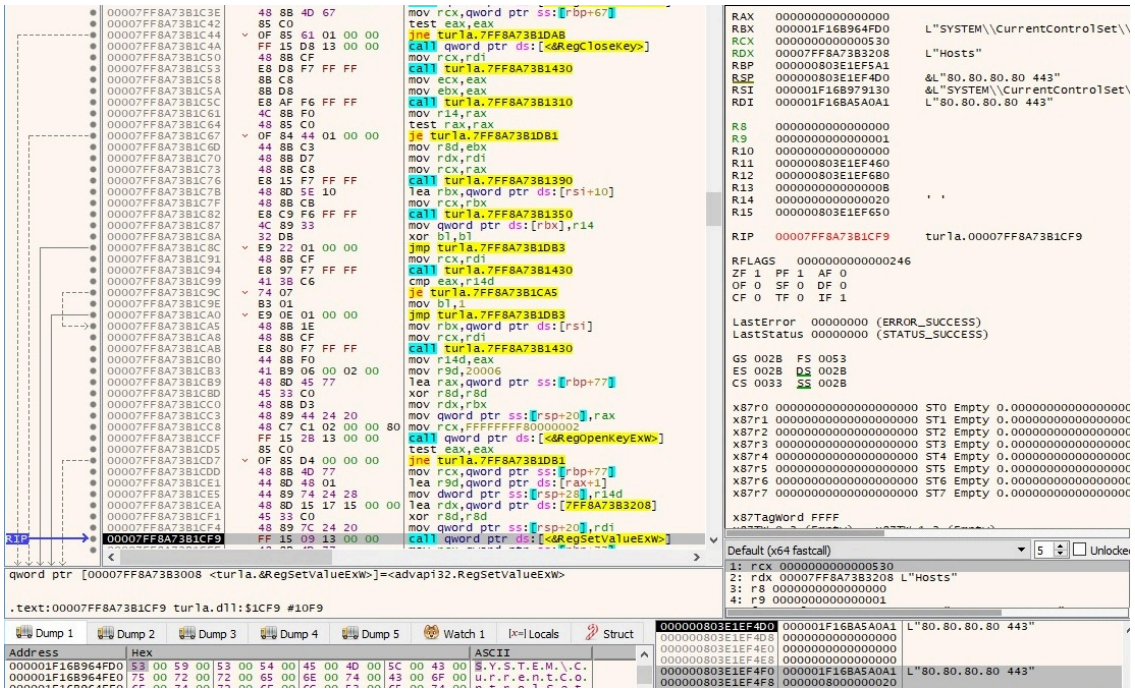


Figure 64

CommandLineToArgvW is utilized to retrieve an array of pointers to the C2 server(s):

Figure 65

A confirmation message “0B 00” is sent to the C2 server.

References

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

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

VirusTotal:
https://www.virustotal.com/gui/file/030cbd1a51f8583ccfc3fa38a28a5550dc1c84c05d6c0f5eb887d13dedf1da01

MalwareBazaar:
https://bazaar.abuse.ch/sample/030cbd1a51f8583ccfc3fa38a28a5550dc1c84c05d6c0f5eb887d13dedf1da01/

Talos article: https://blog.talosintelligence.com/2021/09/tinyturla.html

---

Source: https://cybergeeks.tech/a-step-by-step-analysis-of-the-russian-apt-turla-backdoor-called-tinyturla/