# 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

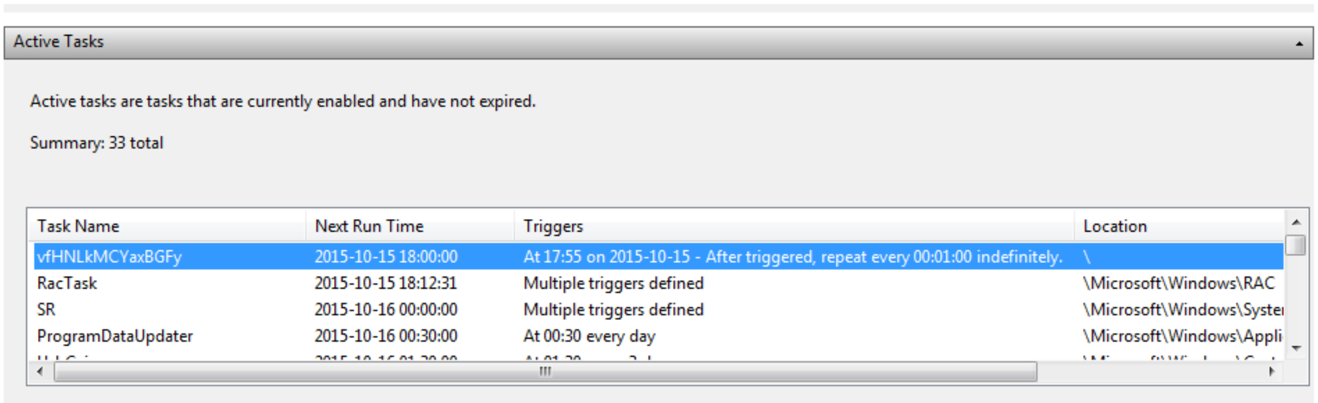
- ff3d706015b7b142ee0a8f0ad7ea2911 – **Dyreza** executable- a persistent botnet agent, carring DLLs with the core malicious activities
  - 5a0e393031eb2accc914c1c832993d0b – Dyreza DLL (32bit)
  - 91b62d1380b73baea53a50d02c88a5c6 – Dyreza DLL (64 bit)

## 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.



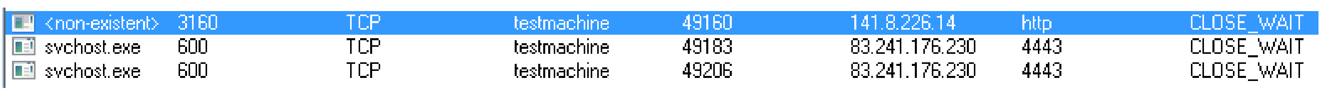
Active Tasks

Active tasks are tasks that are currently enabled and have not expired.

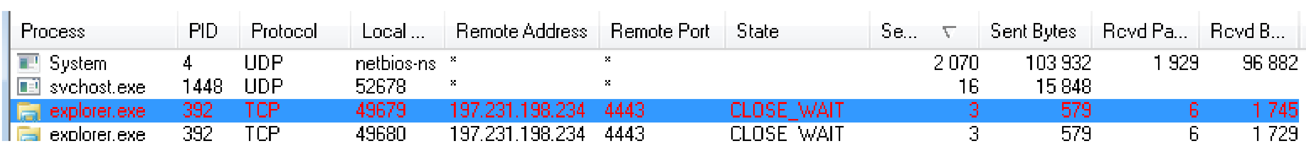
Summary: 33 total

| Task Name | Next Run Time | Triggers | Location |
|---|---|---|---|
| vfHNLkMCYaxBGFy | 2015-10-15 18:00:00 | At 17:55 on 2015-10-15 - After triggered, repeat every 00:01:00 indefinitely. | \ |
| RacTask | 2015-10-15 18:12:31 | Multiple triggers defined | \Microsoft\Windows\RAC |
| SR | 2015-10-16 00:00:00 | Multiple triggers defined | \Microsoft\Windows\Syste |
| ProgramDataUpdater | 2015-10-16 00:30:00 | At 00:30 every day | \Microsoft\Windows\Appli |

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



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| <non-existent> | 3160 | TCP | testmachine | 49160 | 141.8.226.14 | http | CLOSE_WAIT |
| svchost.exe | 600 | TCP | testmachine | 49183 | 83.241.176.230 | 4443 | CLOSE_WAIT |
| svchost.exe | 600 | TCP | testmachine | 49206 | 83.241.176.230 | 4443 | CLOSE_WAIT |



| 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/

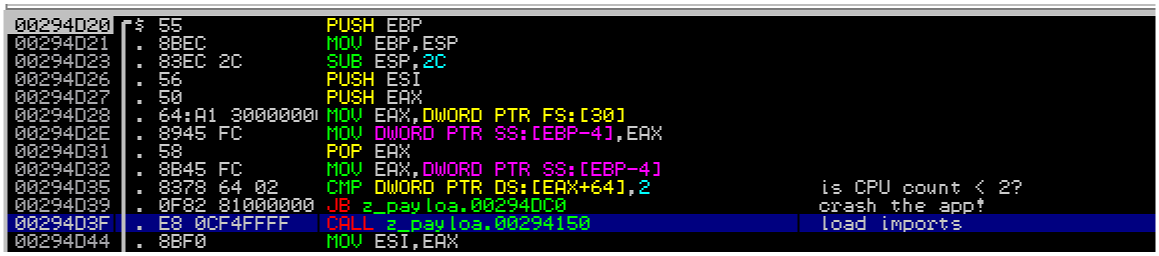
When we deploy any web browser, it directly injects the code into its process and deploys illegitimate connections.It is the way to keep in touch with the C&C, monitor user's activity and steal credentials.

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

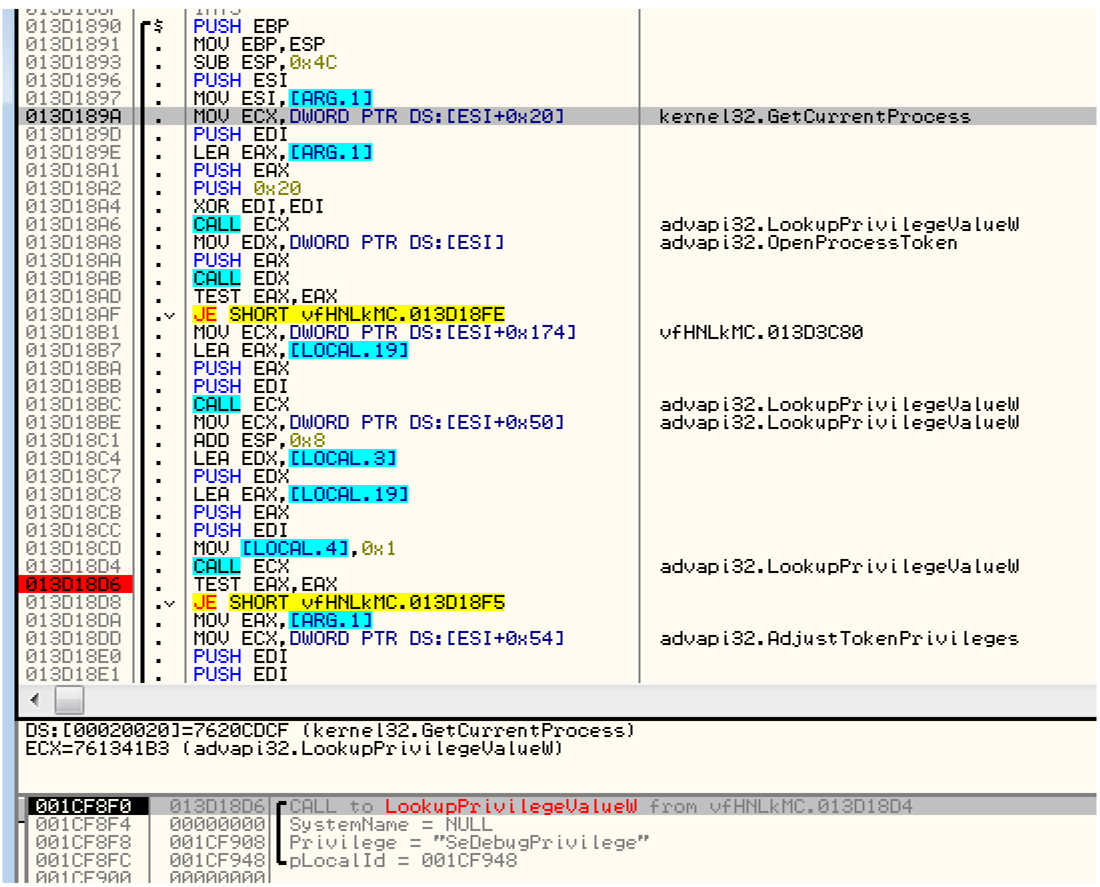
### Main executable

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.




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.

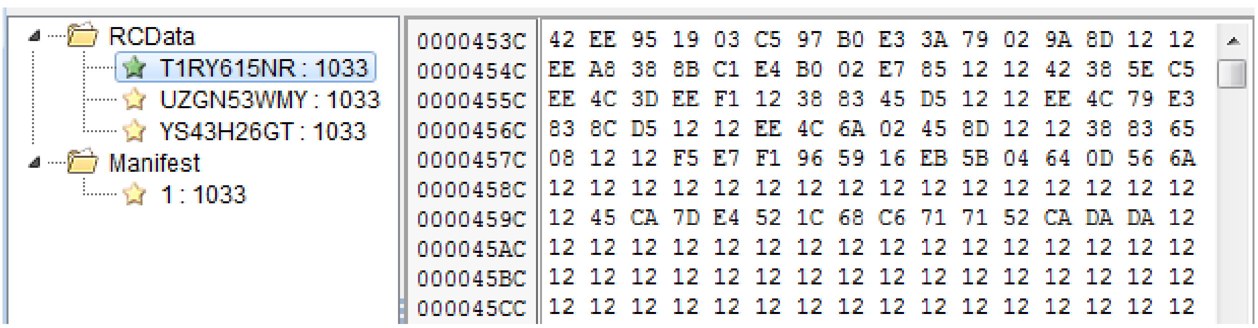
It checks, if it is deployed under debugger – using function *LookupPrivilegeValue* with argument *SeDebugPrivilege* – if it returns non-zero value, execution is terminated.



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.

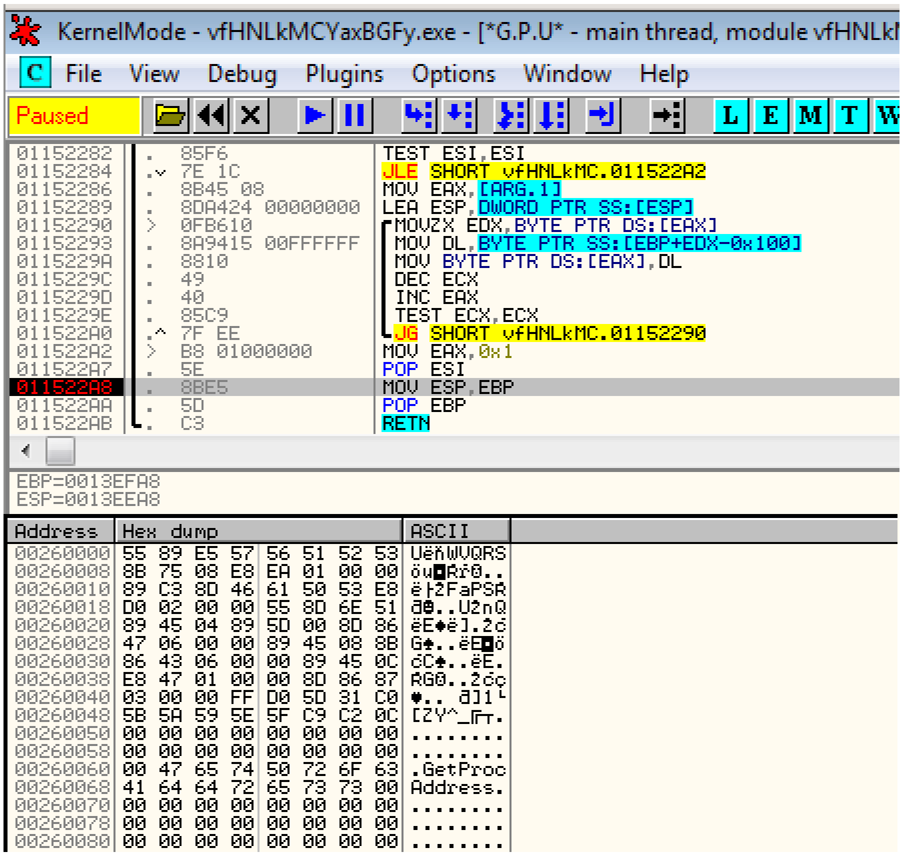
If it is deployed from the valid path and the initial argument passed validation, it performs another check – verifying if it is deployed for the first time. It is achieved by creating a specific Global mutex (it's name is a hash of Computer name and OS Version – fetched by functions: *GetComputerName*,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

```
013D3222  .    LEA EDX,[LOCAL.8]
013D3225  .    PUSH EDX                                  Arg2 = 770070B4
013D3226  .    PUSH ESI                                  Arg1 = 00020000
013D3227  .    CALL vfHNLkMC.013D1830                    find resource #1
013D322C  .    ADD ESP,0x18
013D322F  .    TEST EAX,EAX
013D3231  .v   JE SHORT vfHNLkMC.013D327C
013D3233  .    MOV EAX,DWORD PTR DS:[ESI+0x20]           kernel32.GetCurrentProcess
013D3236  .    PUSH vfHNLkMC.013D5040
013D323B  .    CALL EAX
013D323D  .    MOV ECX,DWORD PTR DS:[ESI+0x108]          kernel32.IsWow64Process
013D3243  .    PUSH EAX
013D3244  .    CALL ECX                                  vfHNLkMC.013D5040
013D3246  .    TEST EAX,EAX
013D3248  .v   JE SHORT vfHNLkMC.013D327C
013D324A  .    CMP DWORD PTR DS:[0x13D5040],0x0
013D3251  .    MOV EAX,DWORD PTR DS:[ESI+0x174]          vfHNLkMC.013D3C80
013D3257  .    LEA EDX,[LOCAL.8]
013D325A  .    PUSH EDX                                  ntdll.KiFastSystemCallRet
013D325B  .v   JE SHORT vfHNLkMC.013D3283
013D325D  .    PUSH 0x7
013D325F  >    CALL EAX
013D3261  .    PUSH vfHNLkMC.013D5044                    Arg4 = 013D5044
013D3266  .    PUSH vfHNLkMC.013D5038                    Arg3 = 013D5038
013D326B  .    LEA ECX,[LOCAL.8]
013D326E  .    PUSH ECX                                  Arg2 = 013D5040
013D326F  .    PUSH ESI                                  Arg1 = 00020000
013D3270  .    CALL vfHNLkMC.013D1830                    find resource #2
013D3275  .    ADD ESP,0x18
```

## Malicious DLL (core)

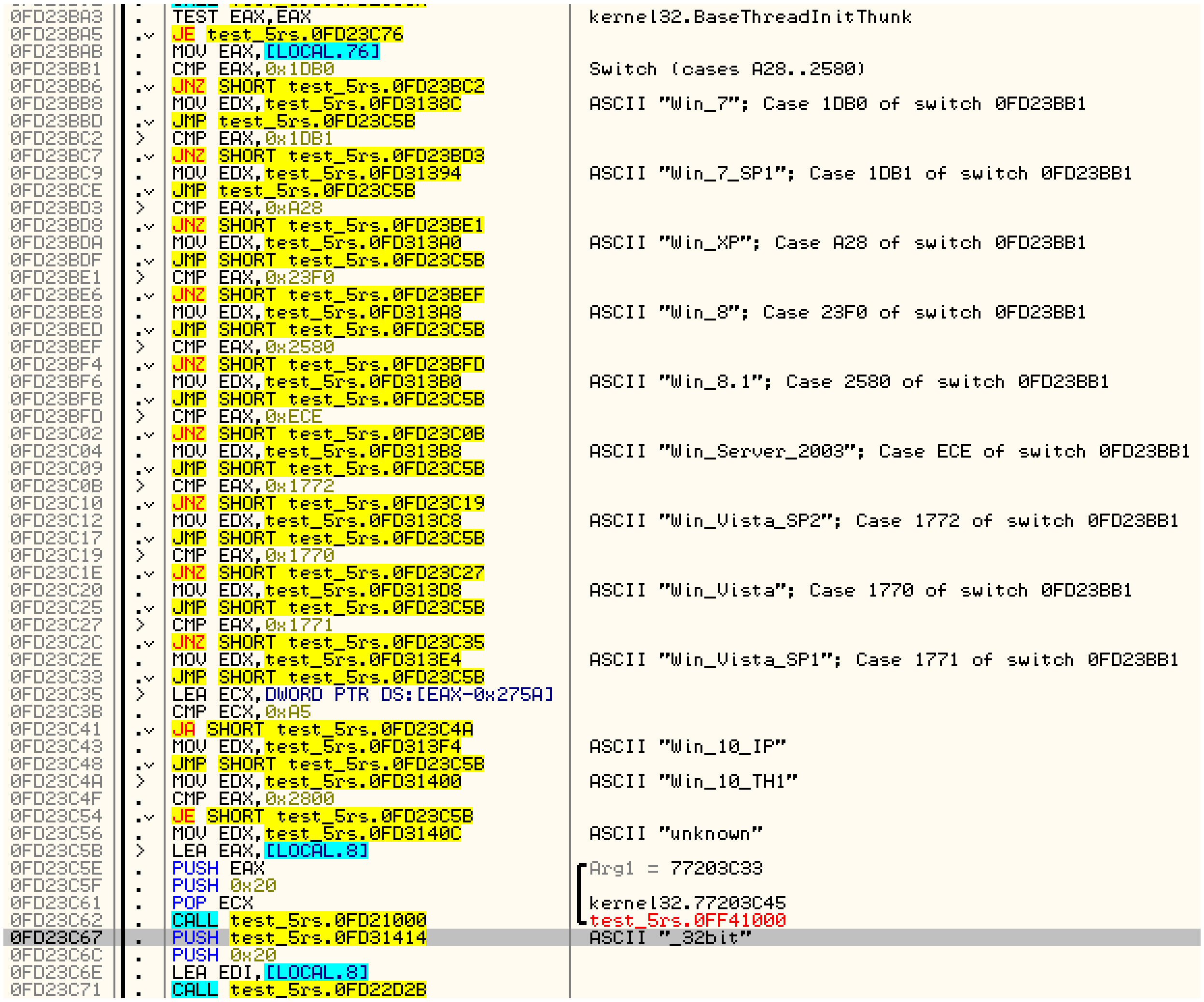
At this stage, functionality of the malware becomes pretty clear. The DLL does not contain much obfuscation – it has clear strings and a typical import table.

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

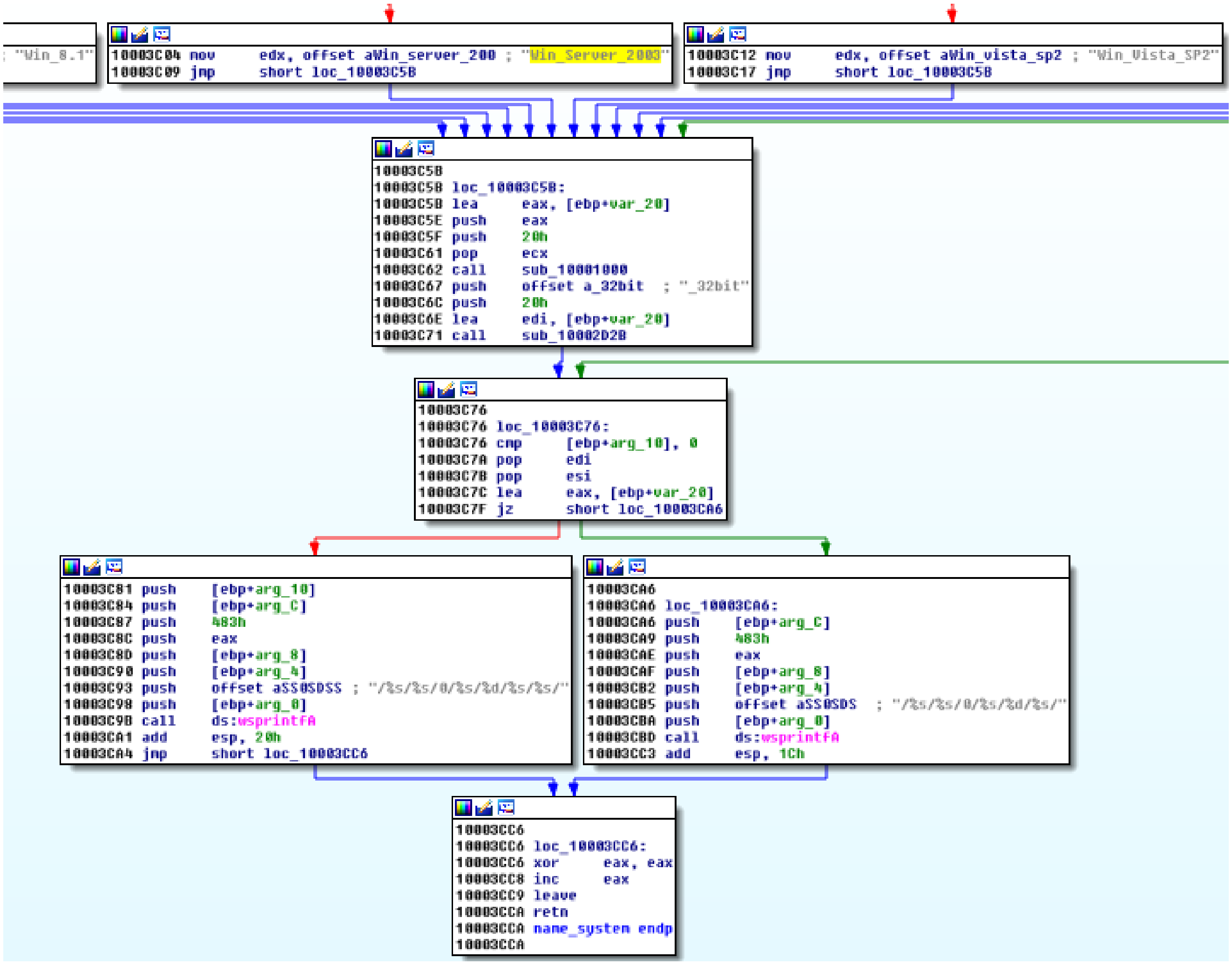
| Address | Length | Type | String |
|---|---|---|---|
| .rdata:10011804 | 00000034 | C | \r\n--%s\r\nContent-Disposition: form-data; name=\"%s\"\r\n |
| .rdata:10011838 | 0000000F | C | Content-Type: |
| .rdata:10011848 | 00000005 | C | \r\n\r\n |
| .rdata:10011850 | 00000009 | C | \r\n--%s-- |
| .rdata:1001185C | 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%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X |
| .rdata:10012264 | 0000000E | C | %s_W%d%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 | httprdc |
| .rdata:10012300 | 0000000A | C | 0.0.0.0:0 |
| .rdata:1001230C | 0000000B | C | respparser |
| .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.

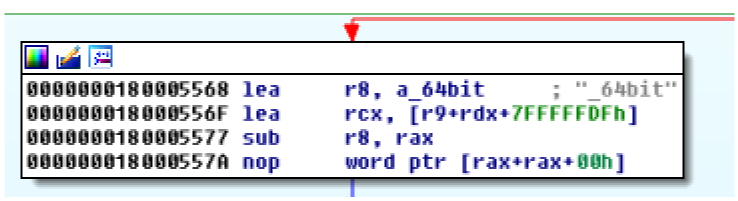
The agent identifies the system:



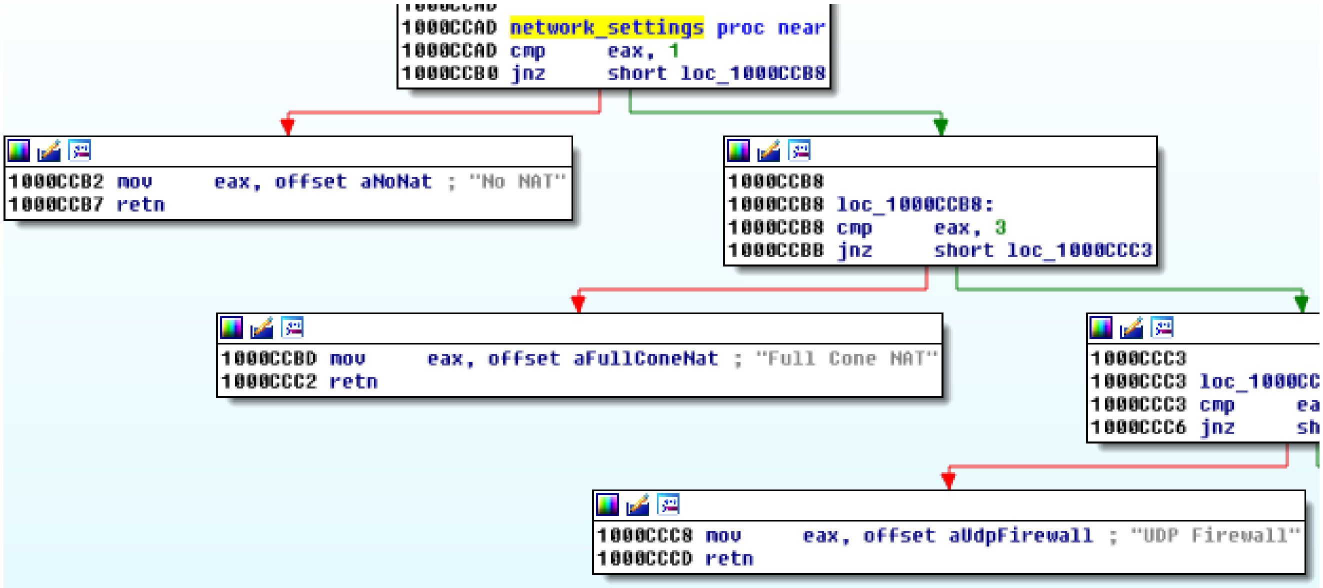

and then – include this data in information sent to the C&C:



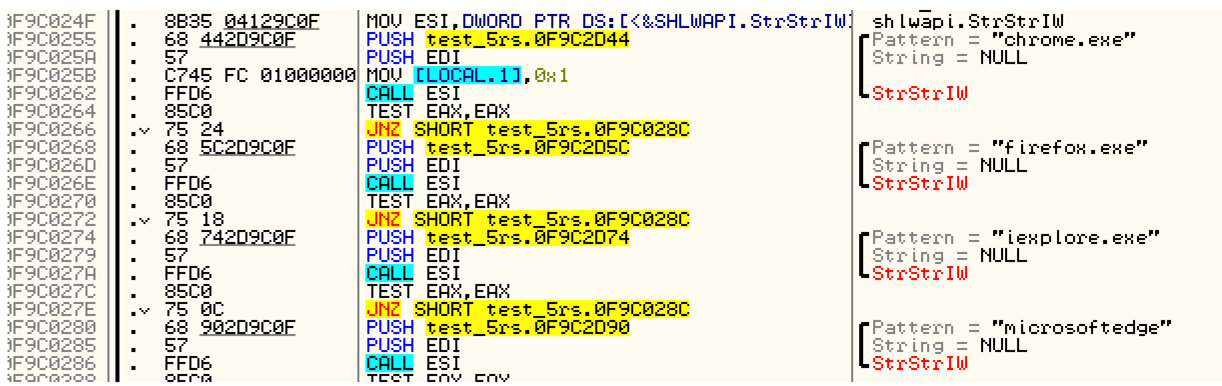

Similar procedure is present in the 64 bit version of the DLL, only the hardcoded string "_32bit" is substituted by "_64bit":




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



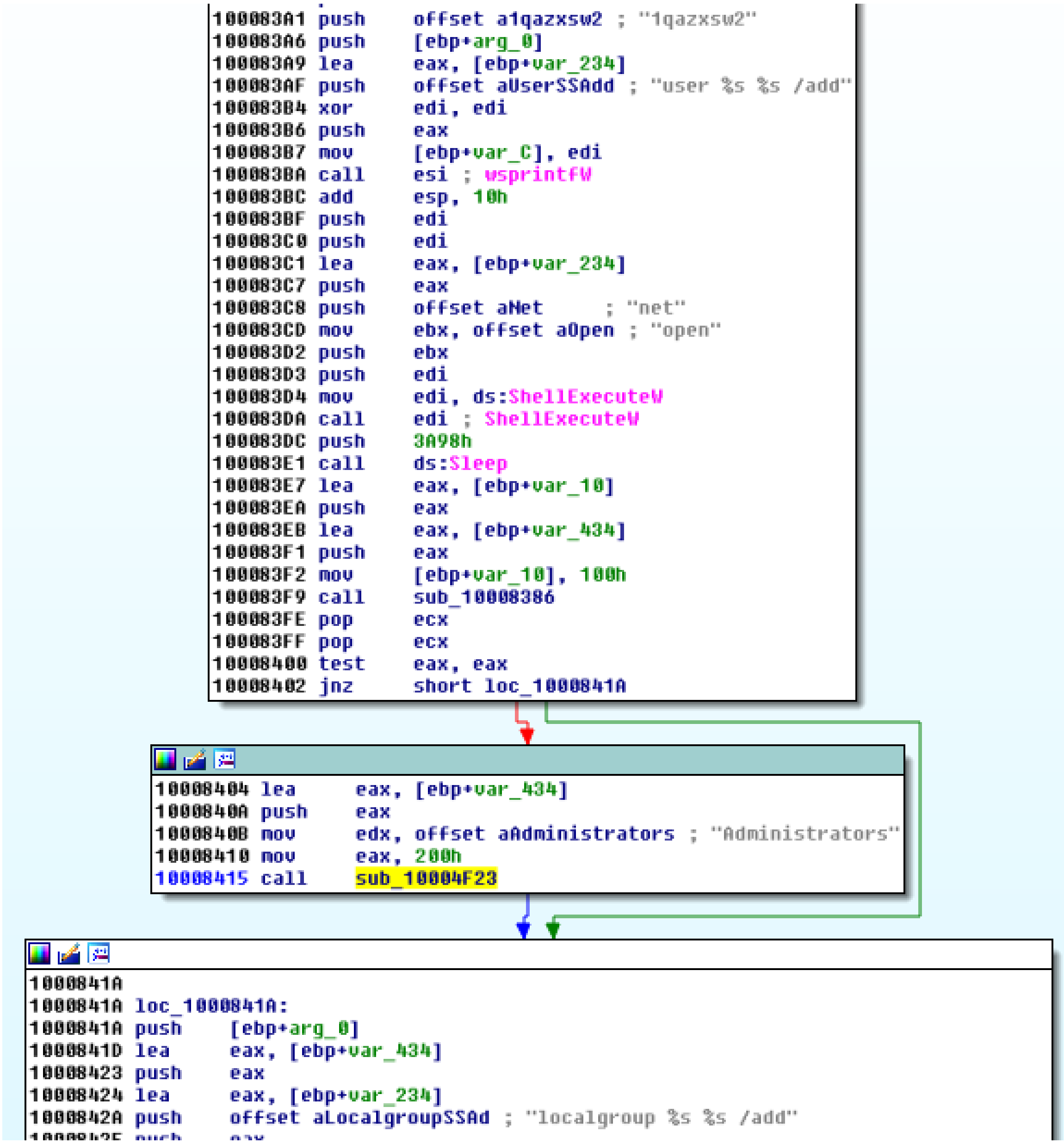

It targets following browsers:



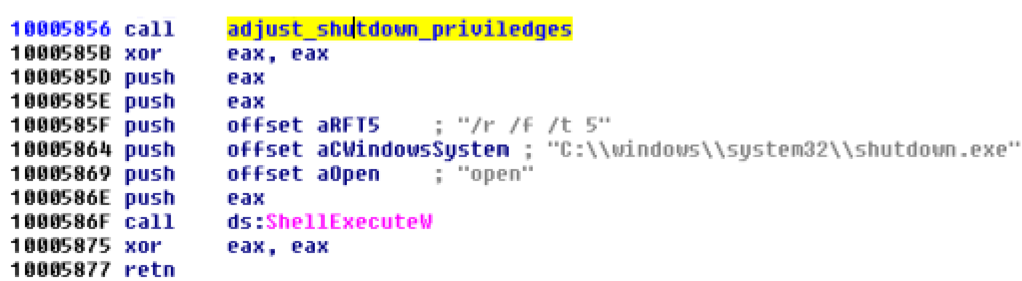

Below – attempt to send stolen account credentials:


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:




*Trying to add a user with administrative privileges*

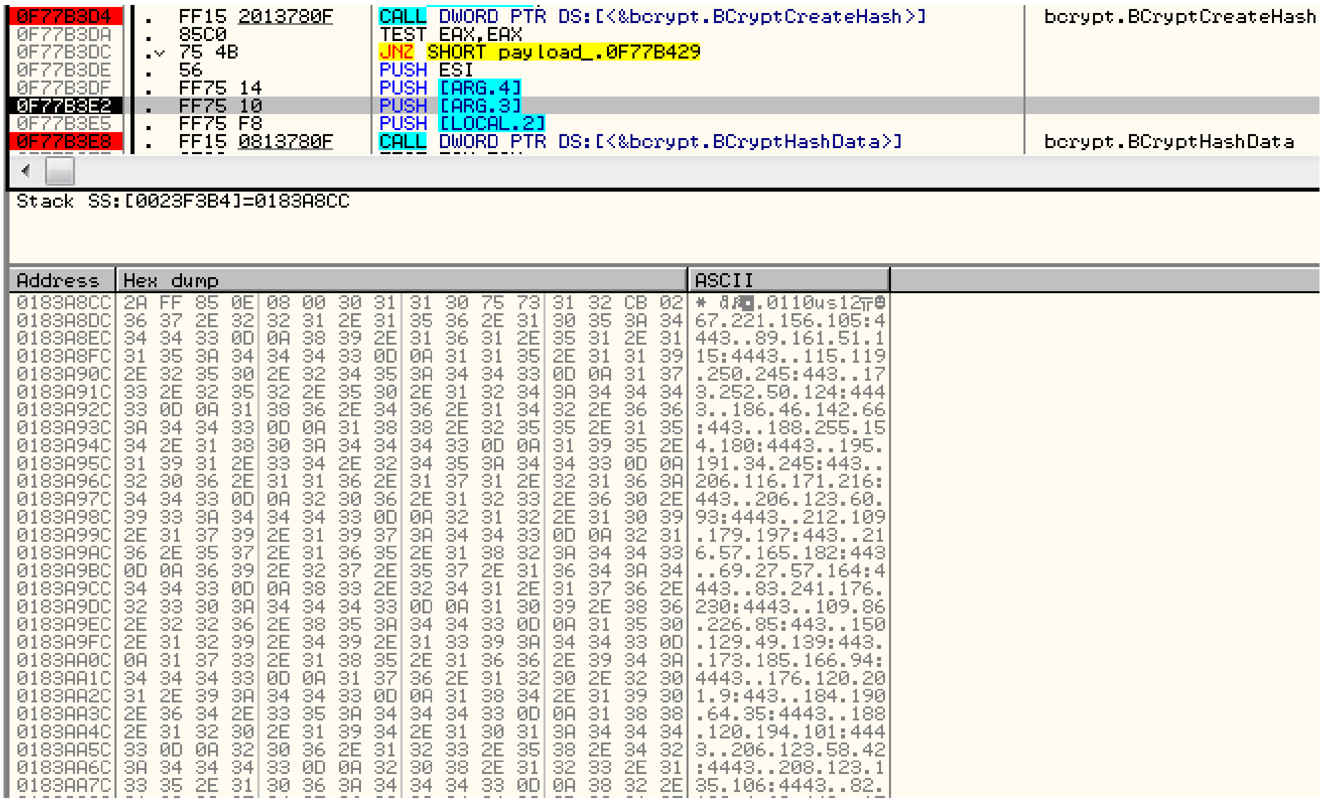



*Shutdown system on command (AUTOKILLOS)*

## 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:




(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