
# A Technical Look At Dyreza

November 4, 2015 by hasherezade

Last updated: October 16, 2016

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

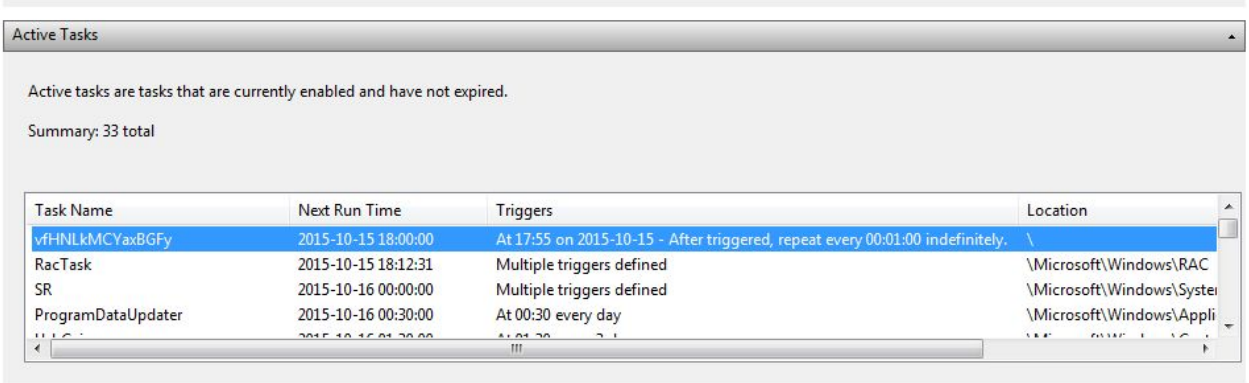
- ff3d706015b7b142ee0a8f0ad7ea2911 – **Dyreza** executable- a persistent botnet agent, carring DLLs with the core malicious activities
    - 5a0e393031eb2accc914c1c832993d0b – Dyreza DLL (32bit)
    - 91b62d1380b73baea53a50d02c88a5c6 – Dyreza DLL (64 bit)

## Behavioral analysis

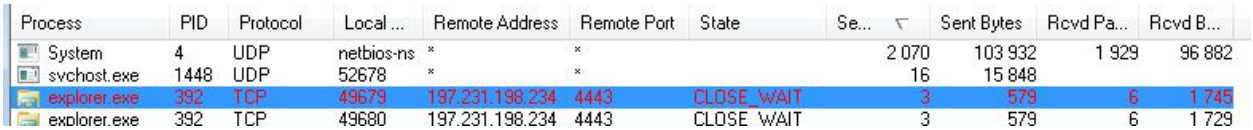
When Dyreza starts to infect the computer – it spreads like fire. Observing it in Process Explorer, we can see many new processes appearing and disappearing. As we can notice, it deploys ***explorer***, ***svchost***, ***taskeng***… All this is done in order to obfuscate the flow of execution, in hopes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




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


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/

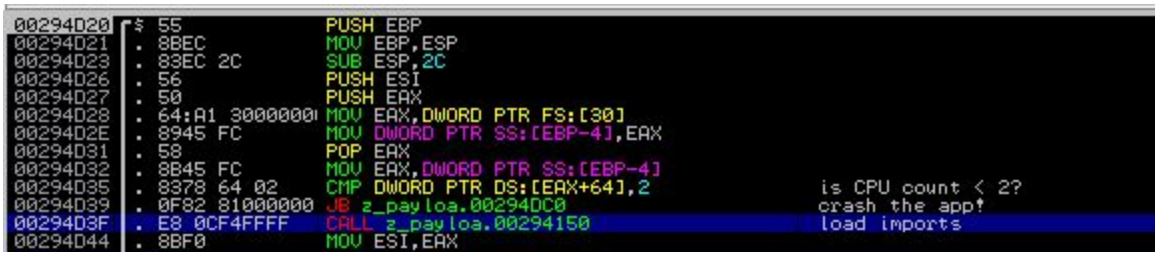
When we deploy any web browser, it directly injects the code into its process and deploys illegitimate connections.It is the way to keep in touch with the C&C, monitor user's activity and steal credentials.

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

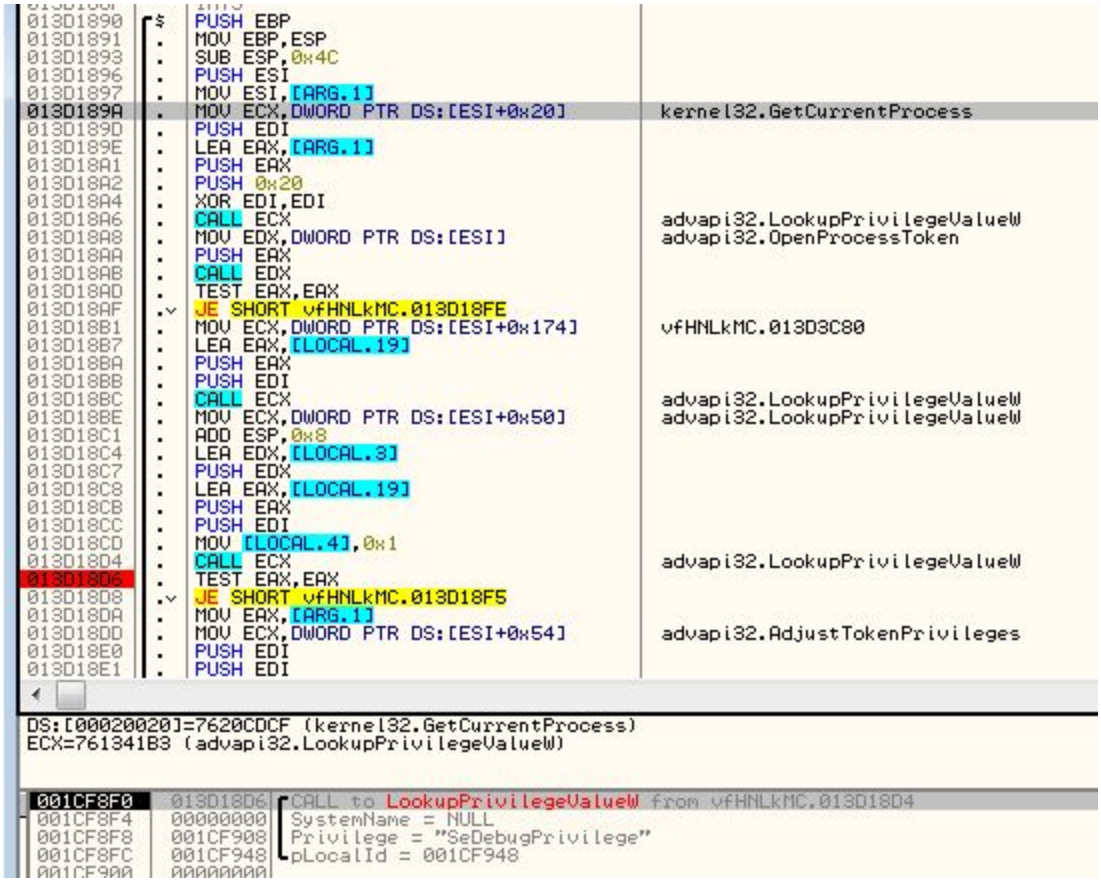
### Main executable

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.

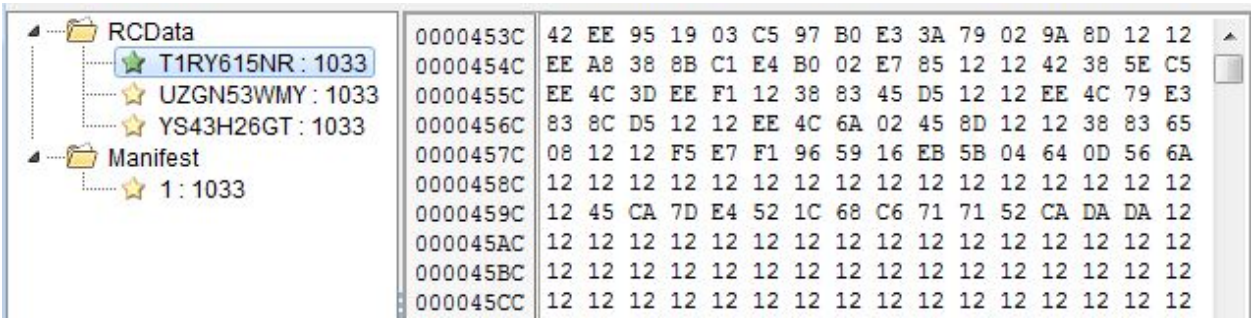
It checks, if it is deployed under debugger – using function ***LookupPrivilegeValue*** with argument ***SeDebugPrivilege*** – if it returns non-zero value, execution is terminated.




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.

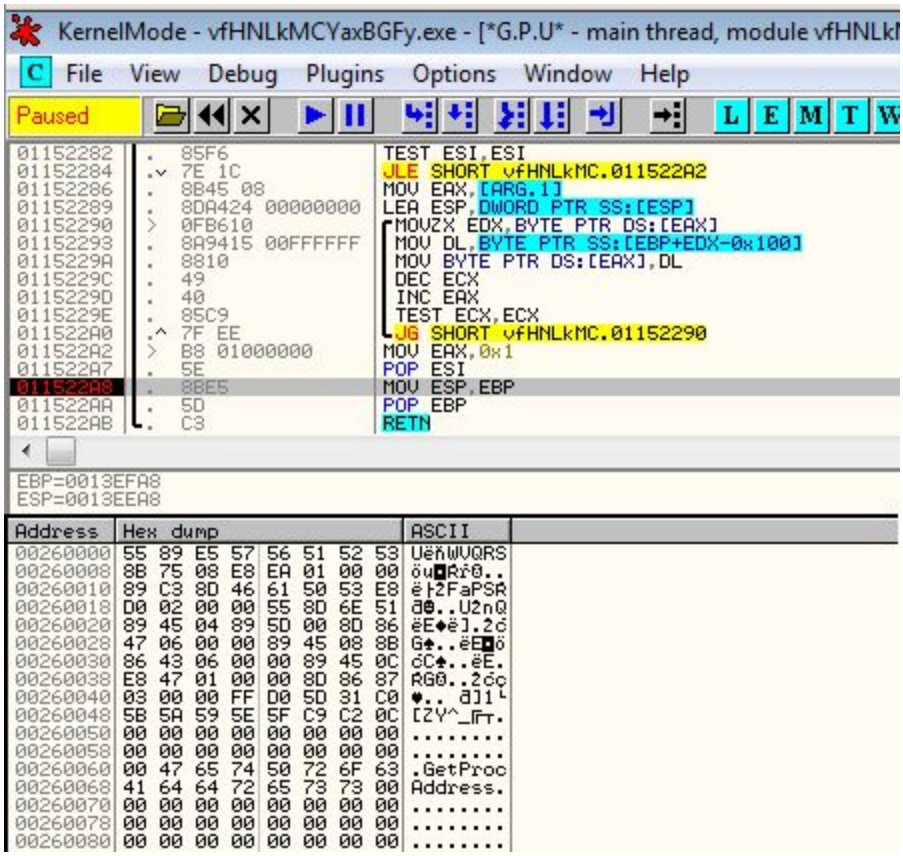
If it is deployed from the valid path and the initial argument passed validation, it performs another check – verifying if it is deployed for the first time. It is achieved by creating a specific Global mutex (it's name is a hash of Computer name and OS Version – fetched by functions: ***GetComputerName***, ***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 ï encrypted 32 bit code, UZGN53WMY ï the key, YS45H26GT ï encrypted 64bit code***

Unpacking:



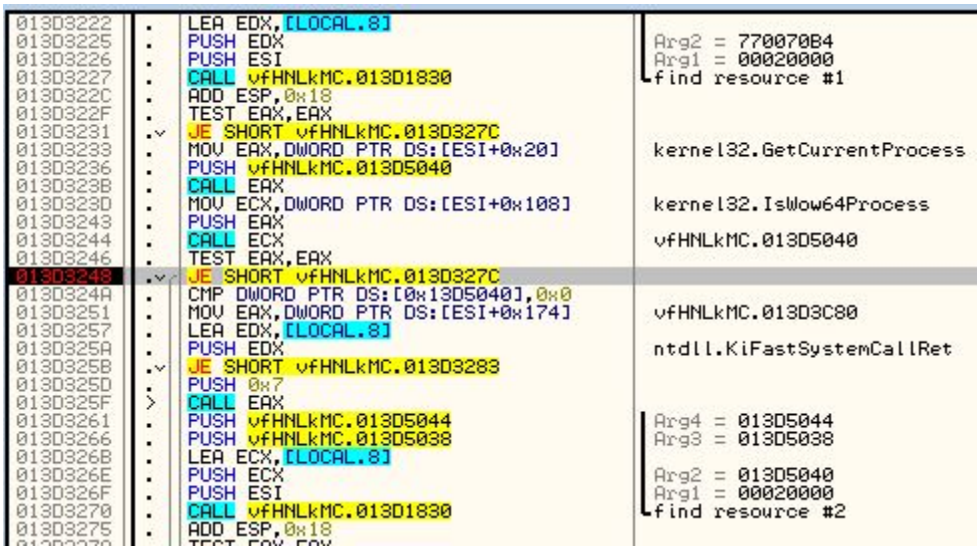

The unpacking algorithm is pretty simple – ***key_data*** contains values and ***data*** – list of indexes of the values in ***key_data.*** We process the list of indexes and read the corresponding values:

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

This script decrypts dumped resources:

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

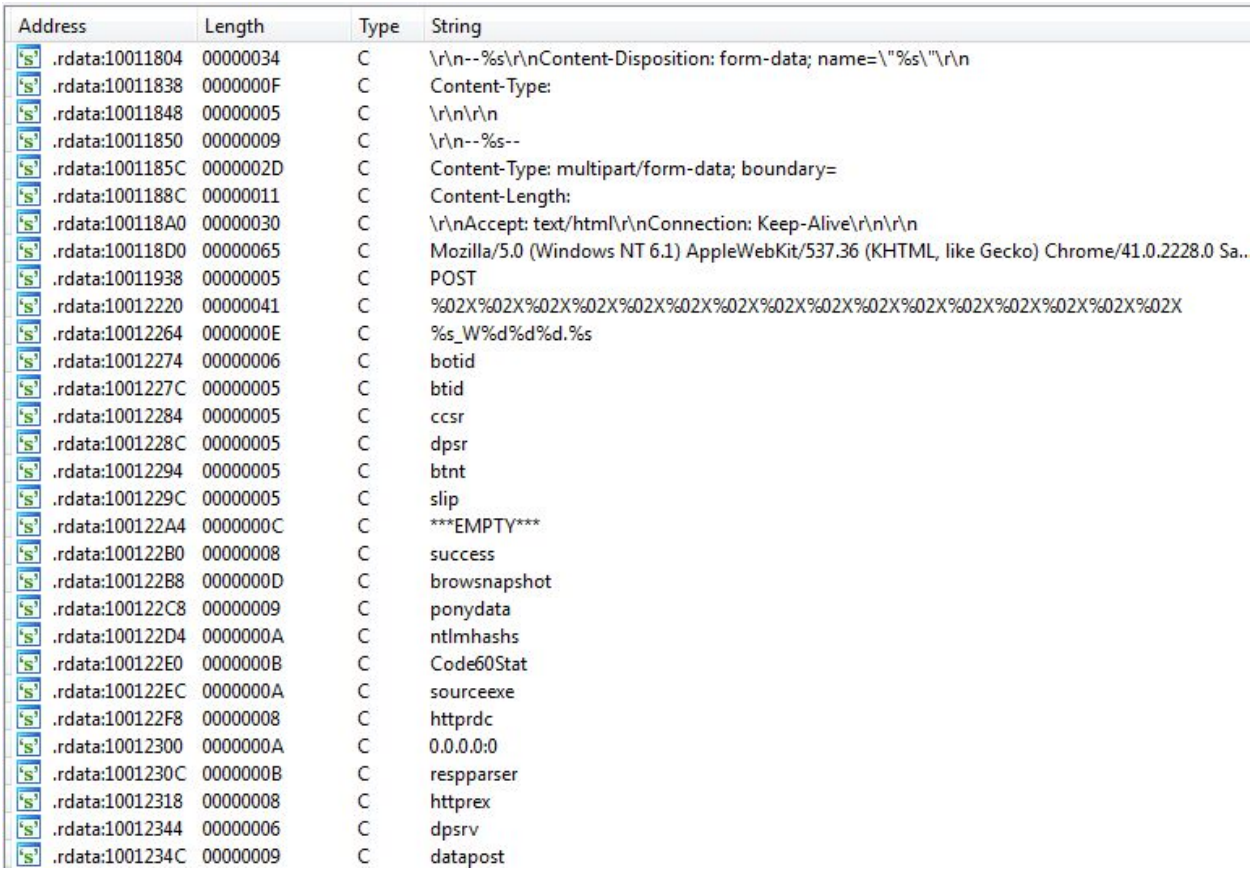
The revealed content contains a shellcode to be injected and a a DLL with malicious functions (32 or 64 bit appropriately). The main sample chooses which one to unpack and deploy, by checking if it is running via WOW64 (emulation for 32 bit on 64 bit machine) – calling function ***IsWow64Process***.




## 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.

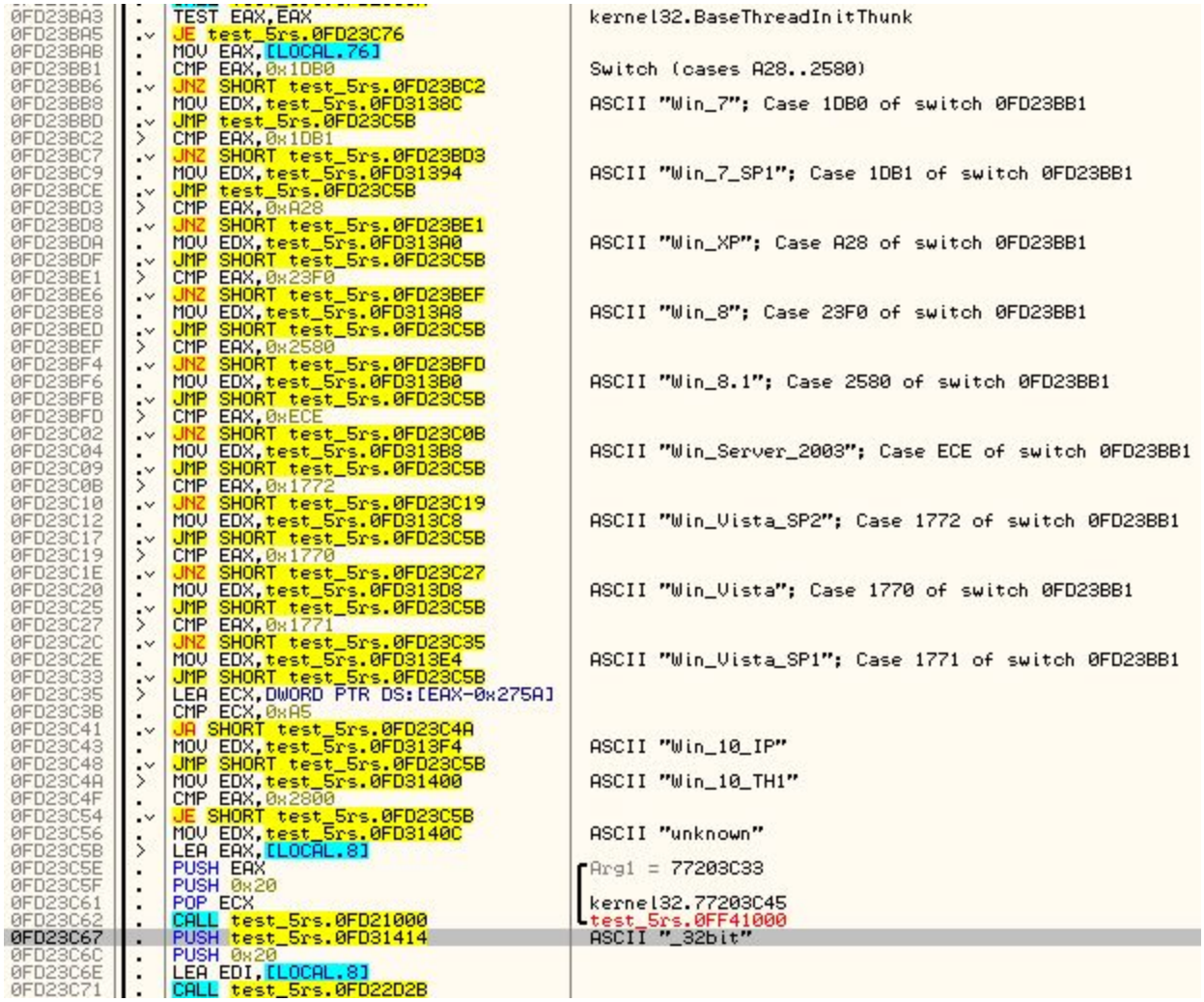
We can see the strings that are used for communication with the C&C:



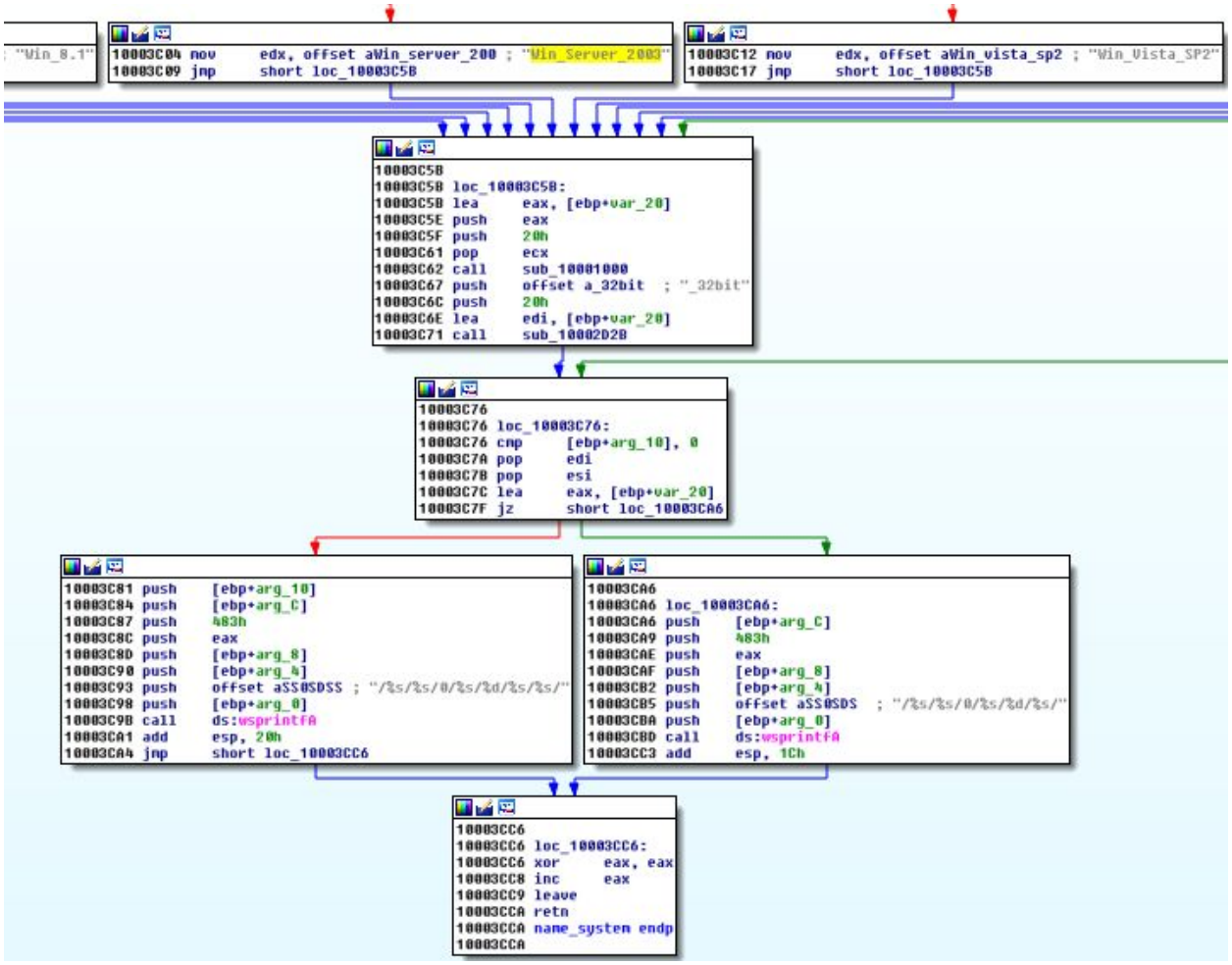
| Address | Length | Type | String |
|---|---|---|---|
| .rdata:10011804 | 00000034 | C | \r\n--%s\r\nContent-Disposition: form-data; name=\"%s\"\r\n |
| .rdata:10011838 | 0000000F | C | Content-Type: |
| .rdata:10011848 | 00000005 | C | \r\n\r\n |
| .rdata:10011850 | 00000009 | C | \r\n--%s-- |
| .rdata:1001185C | 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%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.

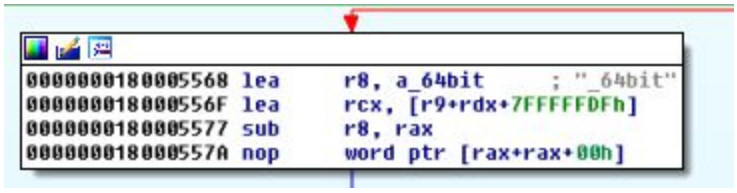
The agent identifies the system:



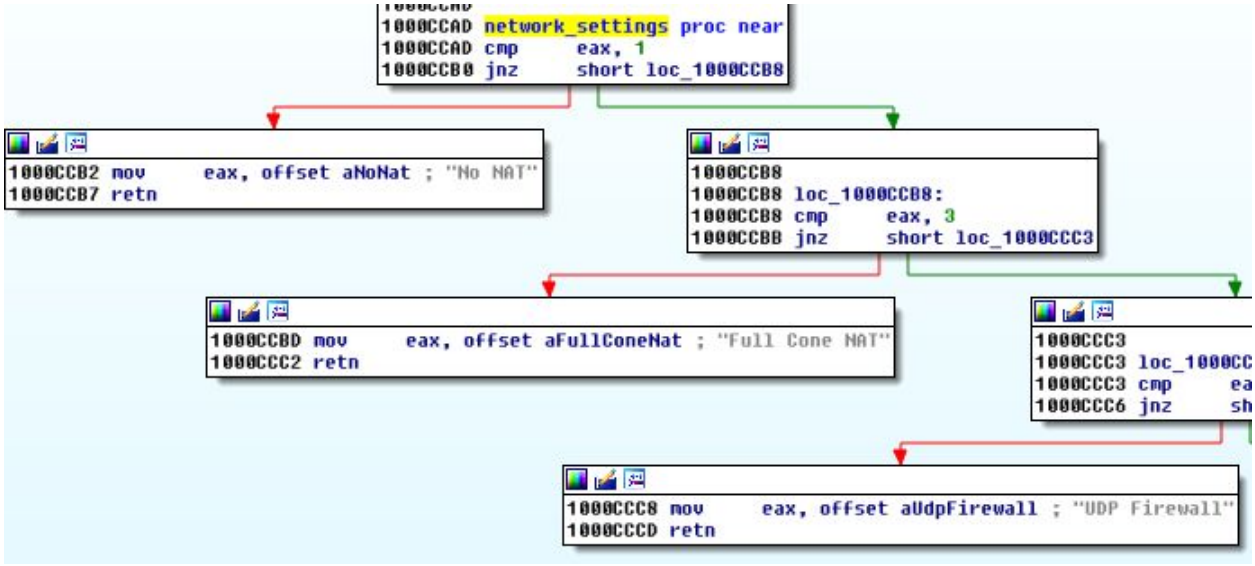

and then – include this data in information sent to the C&C:




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



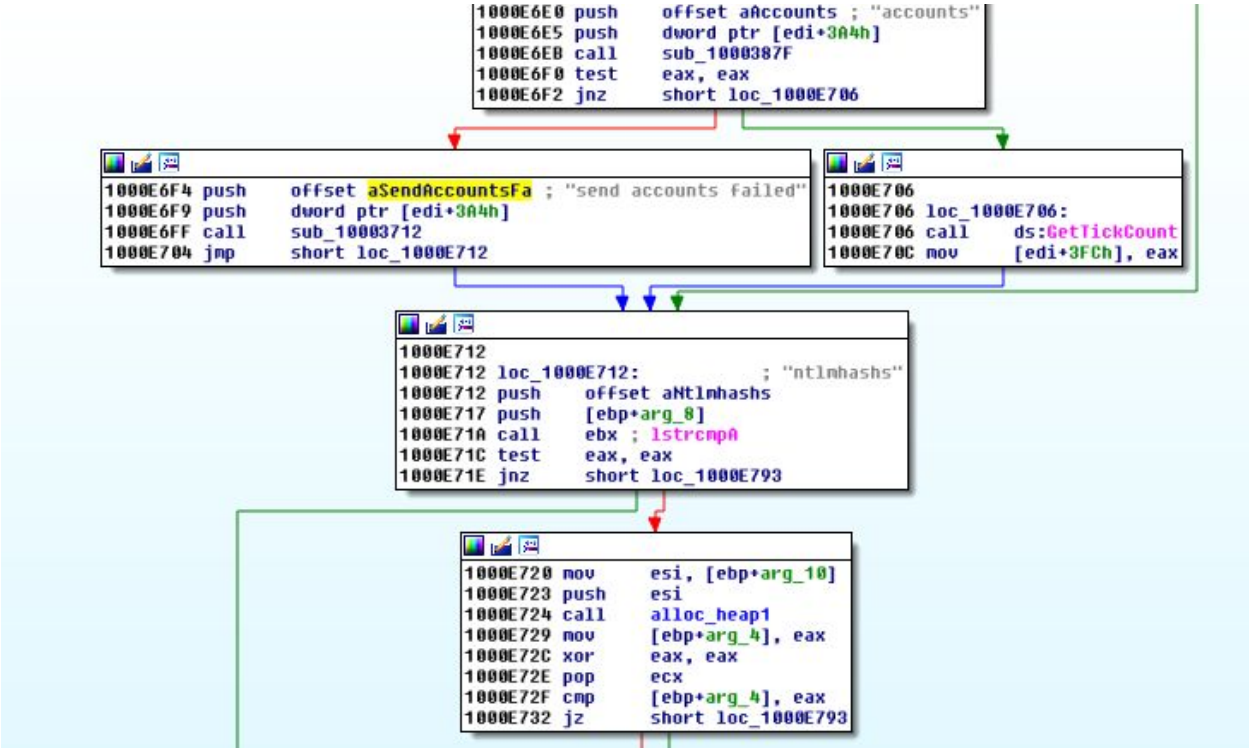

Also, network settings are examined (to verify and inform the C&C whether the client can establish back connection – command : AUTOBACKCONN)




It targets following browsers:

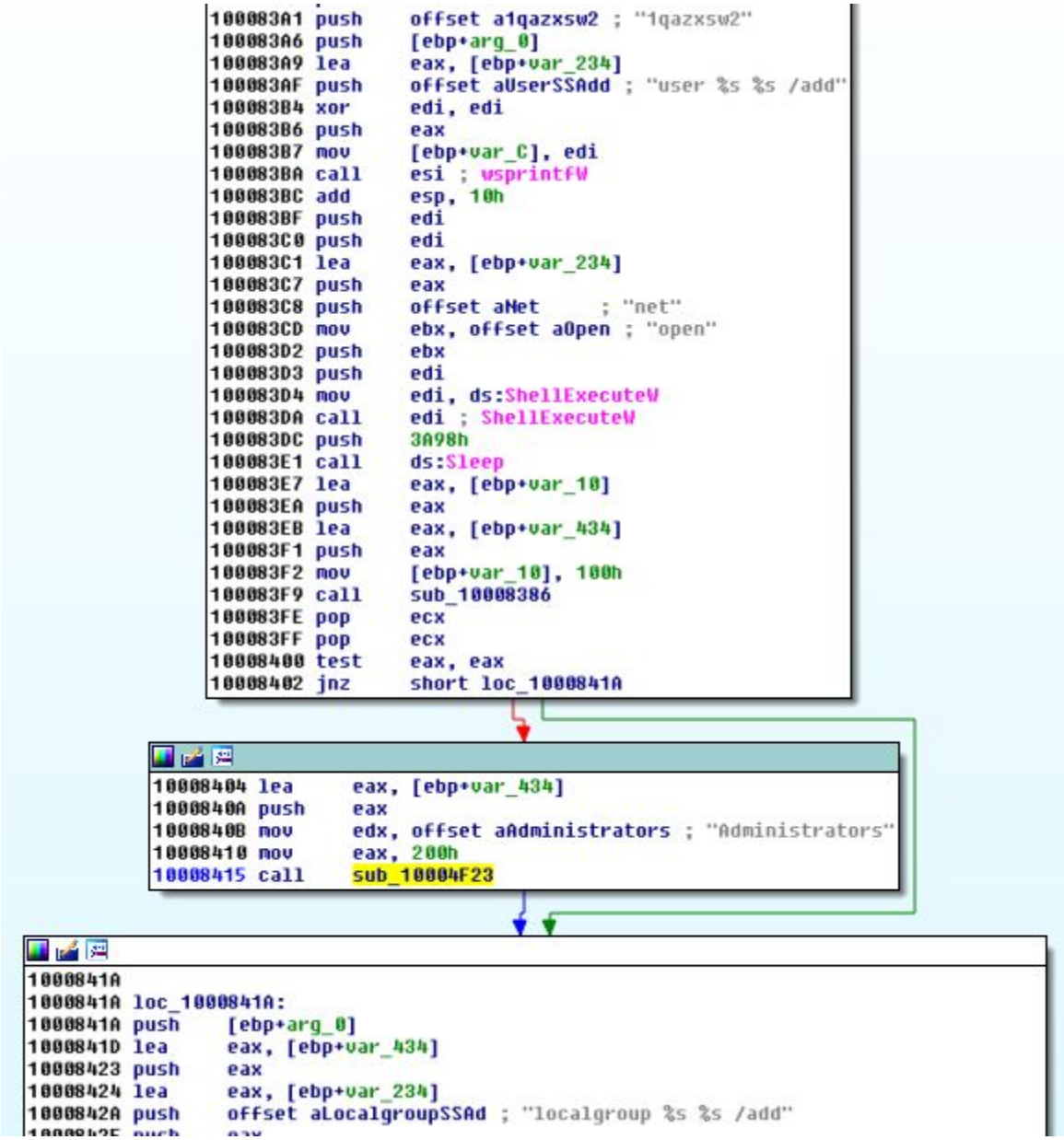
```
F9C024F  .  8B35 04129C0F   MOV ESI,DWORD PTR DS:[<&SHLWAPI.StrStrIW]  shlwapi.StrStrIW
F9C0255  .  68 442D9C0F     PUSH test_5rs.0F9C2D44                     Pattern = "chrome.exe"
F9C025A  .  57              PUSH EDI                                   String = NULL
F9C025B  .  C745 FC 01000000 MOV [LOCAL.1],0x1
F9C0262  .  FFD6            CALL ESI                                   StrStrIW
F9C0264  .  85C0            TEST EAX,EAX
F9C0266  .v 75 24           JNZ SHORT test_5rs.0F9C028C
F9C0268  .  68 5C2D9C0F     PUSH test_5rs.0F9C2D5C                     Pattern = "firefox.exe"
F9C026D  .  57              PUSH EDI                                   String = NULL
F9C026E  .  FFD6            CALL ESI                                   StrStrIW
F9C0270  .  85C0            TEST EAX,EAX
F9C0272  .v 75 18           JNZ SHORT test_5rs.0F9C028C
F9C0274  .  68 742D9C0F     PUSH test_5rs.0F9C2D74                     Pattern = "iexplore.exe"
F9C0279  .  57              PUSH EDI                                   String = NULL
F9C027A  .  FFD6            CALL ESI                                   StrStrIW
F9C027C  .  85C0            TEST EAX,EAX
F9C027E  .v 75 0C           JNZ SHORT test_5rs.0F9C028C
F9C0280  .  68 902D9C0F     PUSH test_5rs.0F9C2D90                     Pattern = "microsoftedge"
F9C0285  .  57              PUSH EDI                                   String = NULL
F9C0286  .  FFD6            CALL ESI                                   StrStrIW
```

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***

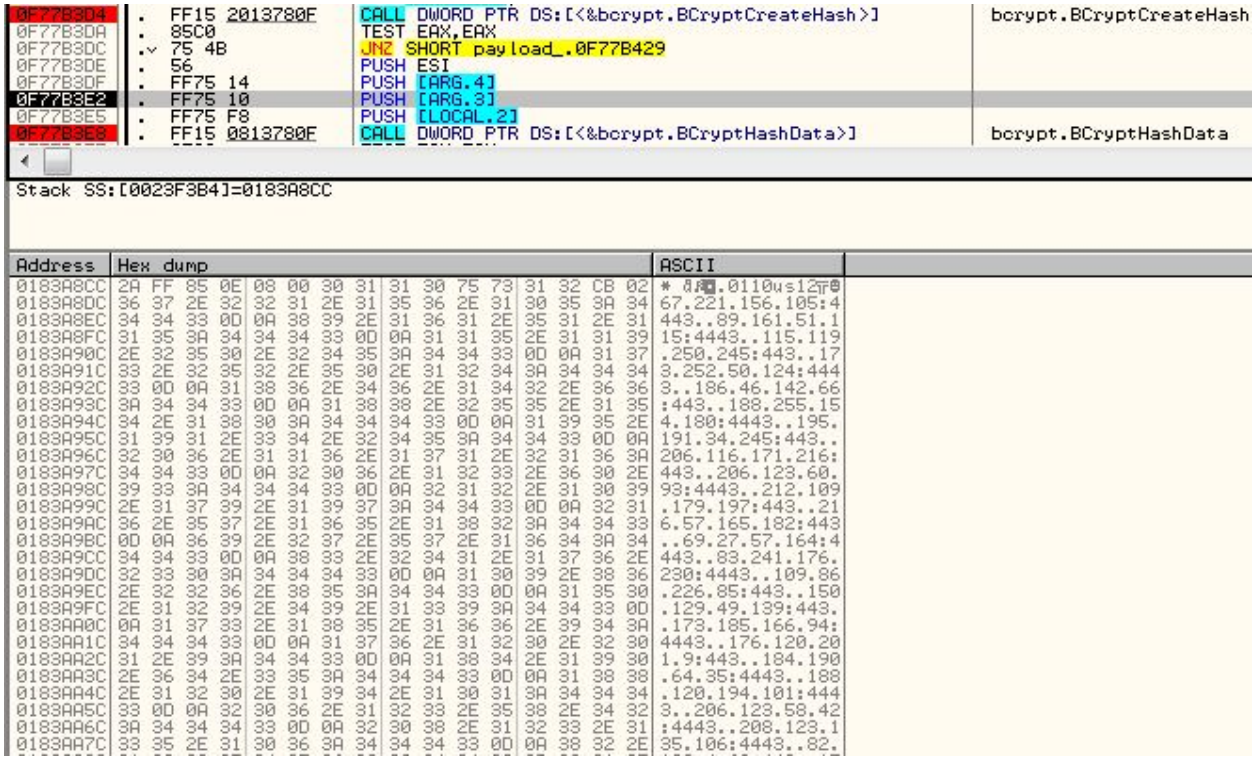
```
10005856 call    adjust_shutdown_priviledges
1000585B xor     eax, eax
1000585D push    eax
1000585E push    eax
1000585F push    offset aRFT5     ; "/r /f /t 5"
10005864 push    offset aCWindowsSystem ; "C:\\windows\\system32\\shutdown.exe"
10005869 push    offset aOpen     ; "open"
1000586E push    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:




(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