Web Security and Malware Analysis

Answers for Assignment 10

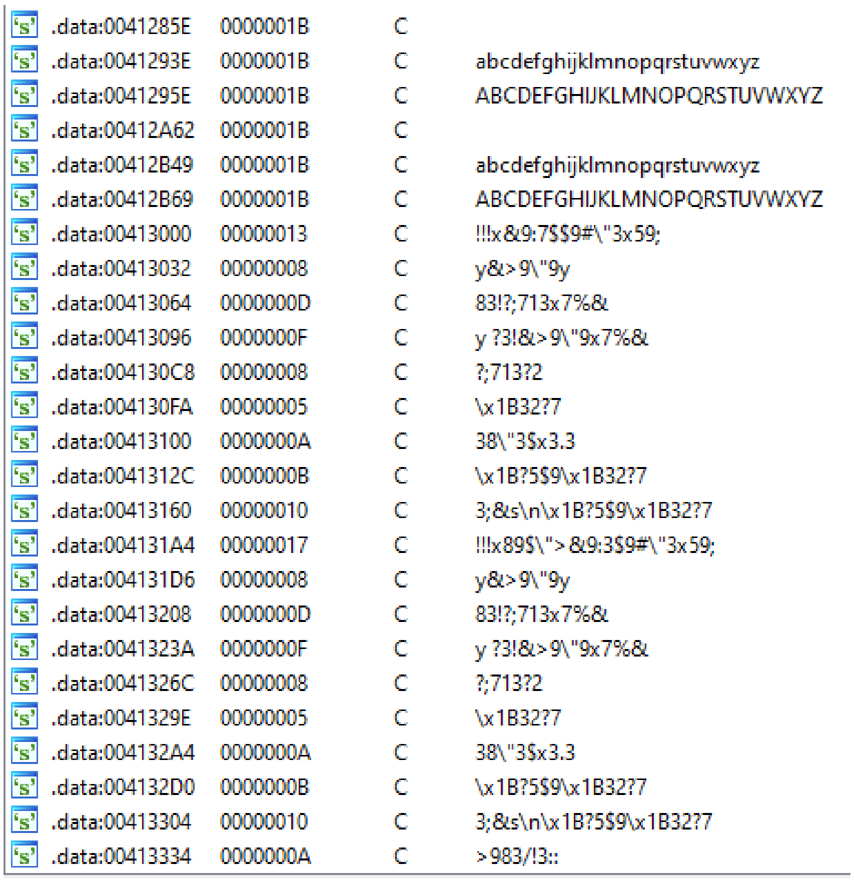
SILVIA LUCIA SANNA – 70/90/00053

Task 1 – FULL MALWARE ANALYSIS OF A REAL SAMPLE

In this task I have to analyse a real malware sample with static and dynamic analysis but the most important initial task to do is to add the extension ".exe" in the file name so that it can be analysed and executed.

The first thing I want to analyse is the PE structure to have an initial idea about the malware. Looking at DOS and File Header in the PE structure I know this malware is a Windows executable (in DOS header e_magic has value 5A4D) and can be executed in a 32 bit machine with Intel 386 (machine value 014C in File header), it is a normal executable (magic value in optional header is 010B) that can be run in Windows OS with version up to 5, this means it is for Windows 2000. Then, analysing optional header I can see sizes of code, data: there are 0xDC00 bytes of code, 0x8400 bytes of initialized data and no uninitialized data; the offset of this executable is 400000 (image base), so I can obtain the first address of code section (401000 obtained combining image base with BaseOfCode, in IDA you can see it in Text View the first instruction is .text401000) and data section (40F000 combining image base with BaseOfData, in IDA it is shown in .idata section). Another important thing from PE structure is "directories" where all imported libraries are listed, in this case they are kernel32.dll, advapi32.dll, shell32.dll, wininet.dll.

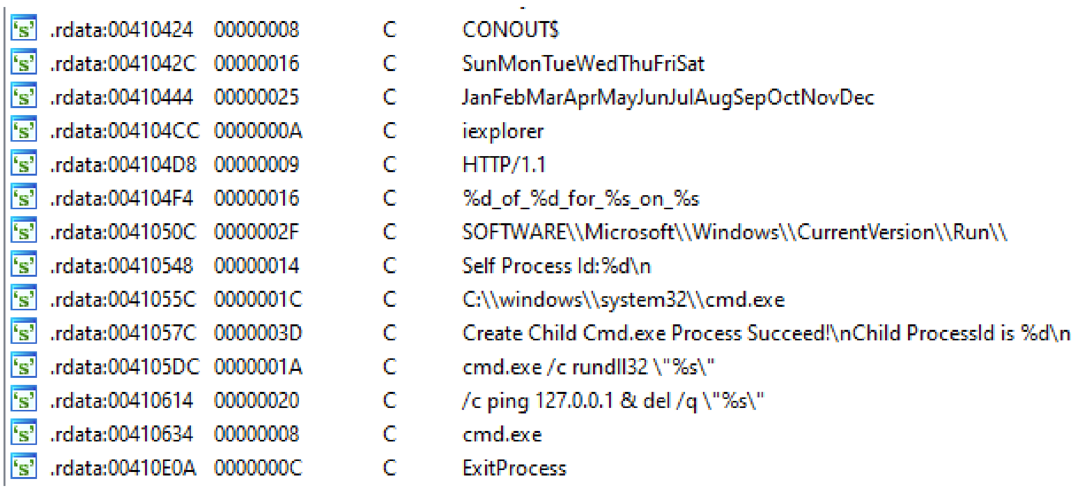
To understand better how this malware works, I can start some static analysis and as first step analyse the strings in Strings Window using IDA. Above all the .data section that can contain some interesting initialized strings, but as in the image below there are no human readable strings, except 2 which contains all English alphabet letters.



```
.data:0041285E   0000001B   C
.data:0041293E   0000001B   C   abcdefghijklmnopqrstuvwxyz
.data:0041295E   0000001B   C   ABCDEFGHIJKLMNOPQRSTUVWXYZ
.data:00412A62   0000001B   C
.data:00412B49   0000001B   C   abcdefghijklmnopqrstuvwxyz
.data:00412B69   0000001B   C   ABCDEFGHIJKLMNOPQRSTUVWXYZ
.data:00413000   00000013   C   !!!x&9:7$$9#\"3x59;
.data:00413032   00000008   C   y&>9\"9y
.data:00413064   0000000D   C   83!?;713x7%&
.data:00413096   0000000F   C   y ?3!&>9\"9x7%&
.data:004130C8   00000008   C   ?;713?2
.data:004130FA   00000005   C   \x1B32?7
.data:00413100   0000000A   C   38\"3$x3.3
.data:0041312C   0000000B   C   \x1B?5$9\x1B32?7
.data:00413160   00000010   C   3;&s\n\x1B?5$9\x1B32?7
.data:004131A4   00000017   C   !!!x89$\">&9:3$9#\"3x59;
.data:004131D6   00000008   C   y&>9\"9y
.data:00413208   0000000D   C   83!?;713x7%&
.data:0041323A   0000000F   C   y ?3!&>9\"9x7%&
.data:0041326C   00000008   C   ?;713?2
.data:0041329E   00000005   C   \x1B32?7
.data:004132A4   0000000A   C   38\"3$x3.3
.data:004132D0   0000000B   C   \x1B?5$9\x1B32?7
.data:00413304   00000010   C   3;&s\n\x1B?5$9\x1B32?7
.data:00413334   0000000A   C   >983/!3::
```

Figure 1: .data section inside String View on IDA

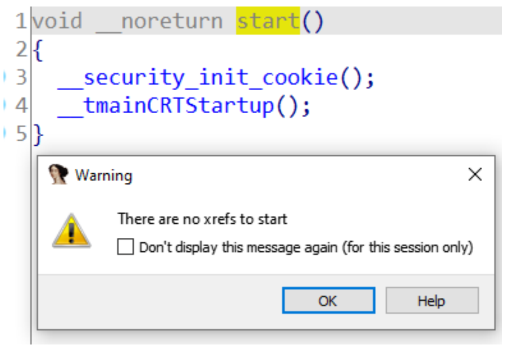
So I decided to look .rdata section in strings and I found two strings with days of the week and all the months; HTTP/1.1 and iexplorer which make me think about an Internet connection; "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run\\" which is a registry key that allows the program to run each time that there is a user login; "C:\\windows\\system32\\cmd.exe", "cmd.exe" where maybe the command prompt is run and in another string with the command prompt is executed also a ping to 127.0.0.1 (a request to localhost, which is the machine where this code is running, to know if the Internet connection is up or not) and this command contains also a string with "del /q \%s" which means that the %s (from this view I do not know the value) will be deleted in quiet mode; the string "Create Child Cmd.exe Process Succeed! \n Child ProcessId is %d \n" can be part of a printf printing the id of the process which is a child of process cmd.exe.



```
.rdata:00410424  00000008  C  CONOUT$
.rdata:0041042C  00000016  C  SunMonTueWedThuFriSat
.rdata:00410444  00000025  C  JanFebMarAprMayJunJulAugSepOctNovDec
.rdata:004104CC  0000000A  C  iexplorer
.rdata:004104D8  00000009  C  HTTP/1.1
.rdata:004104F4  00000016  C  %d_of_%d_for_%s_on_%s
.rdata:0041050C  0000002F  C  SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run\\
.rdata:00410548  00000014  C  Self Process Id:%d\n
.rdata:0041055C  0000001C  C  C:\\windows\\system32\\cmd.exe
.rdata:0041057C  0000003D  C  Create Child Cmd.exe Process Succeed!\nChild ProcessId is %d\n
.rdata:004105DC  0000001A  C  cmd.exe /c rundll32 \"%s\"
.rdata:00410614  00000020  C  /c ping 127.0.0.1 & del /q \"%s\"
.rdata:00410634  00000008  C  cmd.exe
.rdata:00410E0A  0000000C  C  ExitProcess
```

Figure 2: .rdata section inside Strings view containing the most interesting strings

To better understand how these strings (and maybe others) are used, I will perform a static analysis, which means to analyse the assembly and decompiled code trying to understand how it can work. Opening IDA, it immediately redirects to function WinMain that only has 2 functions. Clicking X key, I can see all X-refs, the functions where this instruction or function is called, and it is tmainCRTStartup, called by start function. As the name suggest, start can be the first function to be called, also because has no references (so is not called by any function).



```
void __noreturn start()
{
  __security_init_cookie();
  __tmainCRTStartup();
}
```

Warning

There are no xrefs to start

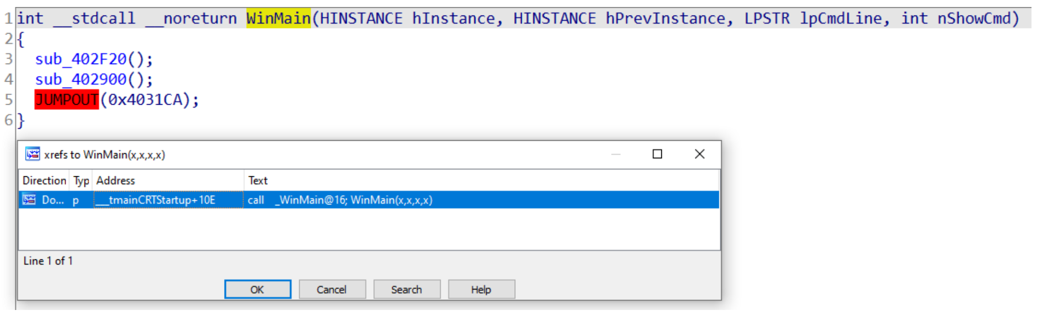
☐ Don't display this message again (for this session only)

Figure 3: start function decompiled with its xrefs

Function start calls only 2 functions:

- __security_init_cookie: set dword_41200C with two different values according to __security_cookie value.
- __tmainCRTStartup: initializes some values, some memory areas and then calls for WinMain function.

WinMain function only calls 2 functions:



```
int __stdcall __noreturn WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nShowCmd)
{
  sub_402F20();
  sub_402900();
  JUMPOUT(0x4031CA);
}
```

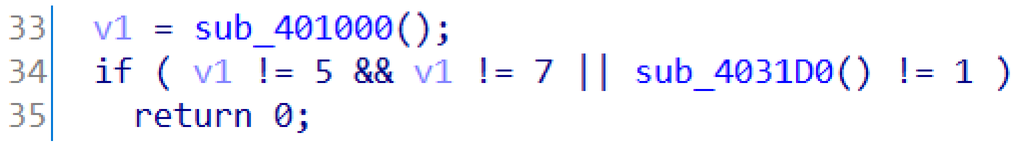
xrefs to WinMain(x,x,x,x)

Direction	Typ	Address	Text
Do...	p	__tmainCRTStartup+10E	call _WinMain@16; WinMain(x,x,x,x)

Line 1 of 1

Figure 4: function WinMain decompiled and its xrefs

- 402F20: the first thing made in this function is to check if the current user is also an admin, setting variable v15 to 1 or 0 if admin or not. Inside function 402B90, if the user is not an admin the OS version is checked by function 401000 (as found analysing PE structure, this should be major OS with version 5 and minor OS with version 0, in the debugging I will check it). According to the OS version and also the output of function 4031D0, function 402B90 will stop or continue doing other actions. Function 4031D0 checks if the security identifier (SID) created has same values of the access token related to the current process (the executable itself), returning 1 when the two SIDs are equals and -1 in case of any error. So, to continue the flow of function 402B90, the if condition at line 34 must be false and the only way to have 0 in that condition is that v1 has value 5 or 7 and the returned value of 4031D0 is 1 (the two SIDs are equals); to have v1 equal to 5 or 7, the major OS version must be 6 and minor version 1 but from PE header I know that major version is 5 and minor version is 0, so it will enter on this if and return 0, blocking the flow of function 402B90.

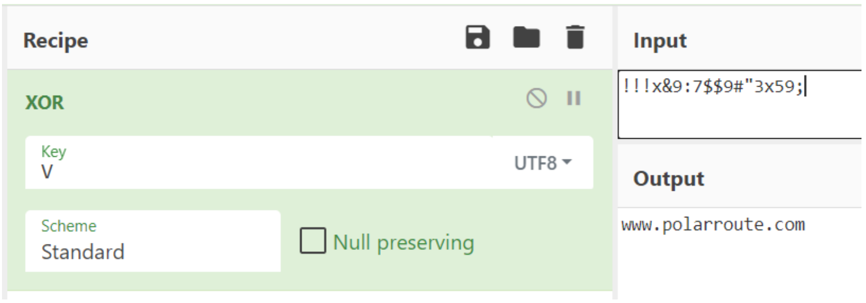


```
33  v1 = sub_401000();
34  if ( v1 != 5 && v1 != 7 || sub_4031D0() != 1 )
35    return 0;
```

Figure 5: if statement inside function 402B90 called by 402F20

In case that if condition is 0, function 402B90 looks for a DAT resource and copies its values (modifying them if different from “$” or ‘A’) in a file whose name depends on the time passed since the machine was started and located in Temp path (a Windows path where temporary files are

stored). If the file has been created without errors, it executes rundll32 for the application just created (the file containing resource's data). It will also save if the machine is Win32 or x64 according to the system directory path value (402AB0). I will have much more knowledge about it when I will debug it.
Coming back to 402F20, after the check of function 402B90, it will execute a xor between each letter of a specific memory area and value "V", saving in the same memory area the value of a domain: this can be the domain to which an Internet connection (seen in strings view) is made.



Key: V (UTF8)
Scheme: Standard
Null preserving: ☐

```
!!!x&9:7$$9#"3x59;
```

```
www.polarroute.com
```

Figure 6: ouput of xor encryption with the string found at memory area aX9793x59, computed using the online tool cyberchef

After this, expands the source string environment (I will know this value debugging) and according to the expanded string, creates a new path: if the expanded path is the path where this executable is running, will create a directory in the new path filling in a file which is the copy of this executable: briefly, it makes a copy of this file in another path.
If the user is Admin, will open key Run (seen previously in Strings view) in "local machine" mode, otherwise in "current user" mode, setting its value to a specific memory value.
After this it will check the value of a variable stored in memory (dword_4131A0) and if 0 will execute the application in expanded path.
As last thing of this function (call to 403360), it will open a shell pinging localhost (ping 127.0.0.1) and deleting dwItem1 which is the first dependent value of notification after asking current process id and current thread id, changing their priority.

- 402900: this function too calls other different functions. First of all, it will check the value of a stored variable (dword_4131A0) and if 1 will sleep for some milliseconds (but looking at the memory, its value is 0).
  After this it calls 401420 with input 0 that stores the values of current file (the executable that I am analysing) in an array changing them if are not “V” (call to function 401150 that checks if each char is different from letter “V” and changes it with the result between xor of this letter and “V”); it will check also the Volume Serial Number, will create an http string and opens Internet Explorer.
  Then, there is a forever loop inside which there is a loop whose condition is the return value of function 402340 (of course to know its returned value it must be executed).
  Function 402340 will check the OS version (call to function 401000), calls the function to know system directory path and find a file inside it (402AB0), then converts the system time value to a file time structure. After this, it sets variable v1 with different values according if the user is an admin, if the OS version is greater or equal than 4, and in this case, also according to the equality between SID and access token related to this process. In the last part of this function, it gets the PC username and if it is able to retrieve it, calls 401780 to convert it in MultiByte from WideChar. The return value of function 402340 depends on the returned value of function 4018E0 that performs an Internet connection to a specific domain stored in memory (which is the one generated before with the XOR operation) and a POST request to a URL (stored in dword_414560) generated according to time passed since machine was started: if the POST request is ok the function returns 1 otherwise 0, closing connection in both cases.
  So, if function 402340 returns 1 the program will proceed with its flow, setting variable v2 according to the returned value of 401B80. Function 401B80 connects to the polarroute.com domain but looking for a jpg (stored in dword_414460) file with parameters depending on the elapsed time since restart. It will read the content of this page changing some of its values if they are not equals to “Y”

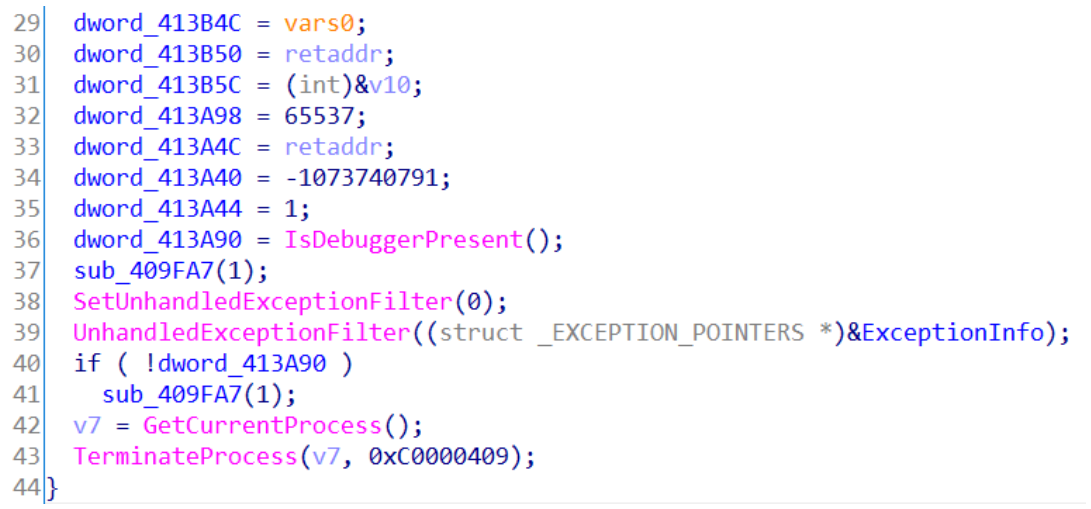
(as always it sets them as the XOR result between the current value and "Y" letter, performed by function 4018C0). Inside this function there is also the call to 401AA0 that will read the HTML file in a specific page (stored in dword_414660), returning 0 or the pointer to memory area where read bytes are stored, if the read failed or not.
Coming back to function 402900, according to v2 value, different actions will be performed:

- CASE 1: creates a pipe, sets start-up and process info to execute command line with these parameters, read the data from the pipe storing them in an array and changes them to WideChar type, after this, it will make the POST request to polarroute website (4018E0).
- CASE 2: creates a file containing data of temp path and then executes it.
- CASE 3: creates a file whose name is determined with the conversion of some multibyte in widechar, allocates it virtually, converts it to multibyte, saves only the last part of the filename (after the last "\\") and will make the usual POST request.
- CASE 4: executes the application v2+24.
- CASE 5: changes the ServerName if a condition is valid.
- CASE 6: if the specified condition is true, will delete the value of key Run for the current user, close the key and call 403360 that will open a shell pinging localhost and deleting dwItem1. This case is the only condition to stop the loop execution.
- CASE 7: modifies the path of the current process and makes the POST request.
- CASE 8: creates 2 pipes and makes them communicate, sets start-up and process info and creates a child of process cmd.exe and then makes the POST request.
- DEFAULT: sleeps for different milliseconds.

If 402340 returns 0, the program will try to 402340 to return 1 for 3 times (it sleeps for some time between one trial and the next), that if one of them is successful will come back to the switch cases, otherwise will check the value of some dwords and Sleep for enough time.

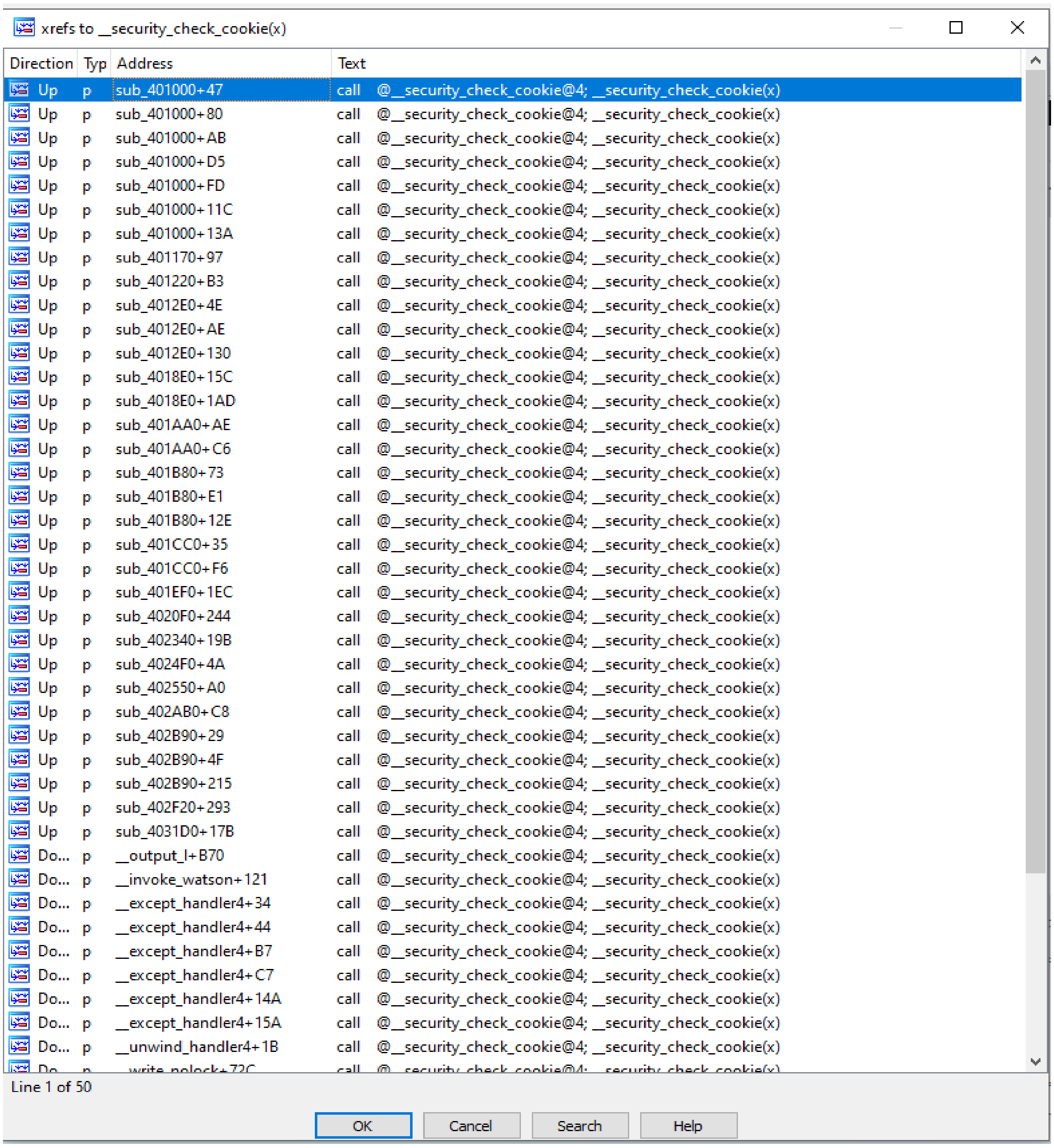
- JUMPOUT: this is not an instruction, but something interpreted by IDA because maybe there is an instruction inside a function that jumps out from the function itself.

I also noticed that when I clicked on a function to decompile it, almost all of them contained an assembly part that was not decompiled: the call to function "__security_check_cookie". This function will call __report_gsfailure if the input variable is different to ___security_cookie variable stored in memory. Function __report_gsfailure will check if the debugger is present or not (sets a variable in memory, dword_414810 to 0 if no debugger), sets some exception (proceeding with normal execution of the exception, passing this exception to the debugger if it is debugged) and terminate the current process. Maybe this is an anti-debugging technique but even if the process is debugged, as the exception is set to 0, it will continue to the normal flow. To be a real anti-debugging technique, the exception should be set to 1 to terminate the program or to 0xffffffff to change its flow.



```
  dword_413B4C = vars0;
  dword_413B50 = retaddr;
  dword_413B5C = (int)&v10;
  dword_413A98 = 65537;
  dword_413A4C = retaddr;
  dword_413A40 = -1073740791;
  dword_413A44 = 1;
  dword_413A90 = IsDebuggerPresent();
  sub_409FA7(1);
  SetUnhandledExceptionFilter(0);
  UnhandledExceptionFilter((struct _EXCEPTION_POINTERS *)&ExceptionInfo);
  if ( !dword_413A90 )
    sub_409FA7(1);
  v7 = GetCurrentProcess();
  TerminateProcess(v7, 0xC0000409);
}
```

Figure 7: last section of function security_check_cookie, this part contains the debugger check and terminate process previously described



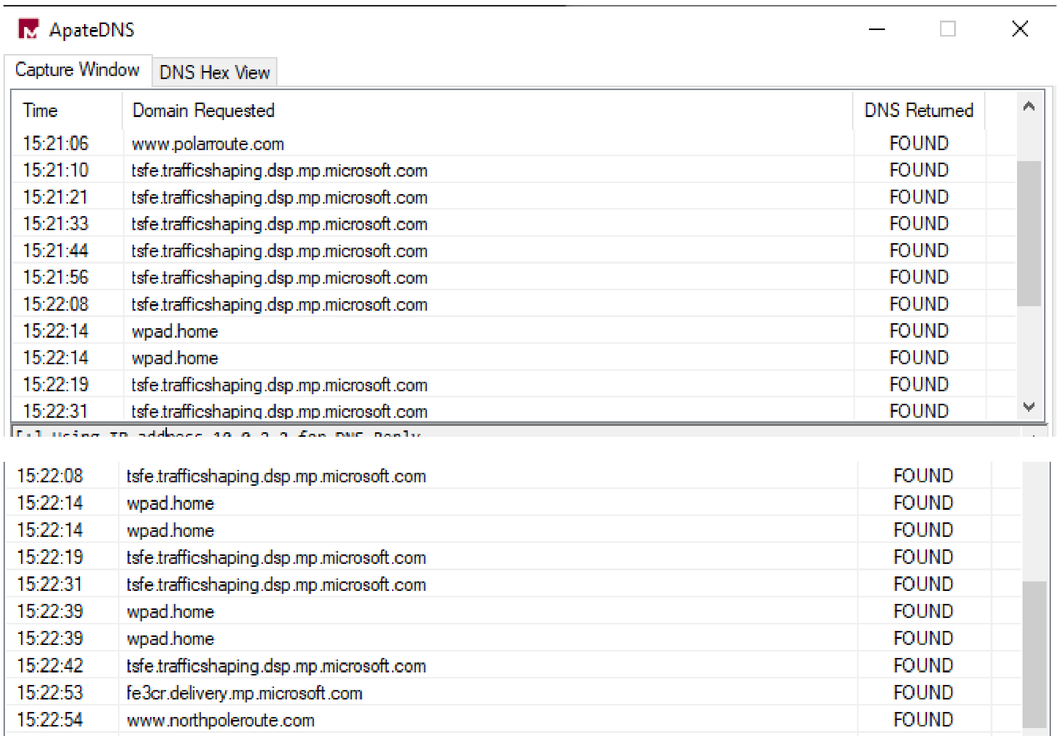
xrefs to __security_check_cookie(x)

Direction	Typ	Address	Text
Up	p	sub_401000+47	call @__security_check_cookie@4; __security_check_cookie(x)
Up	p	sub_401000+80	call @__security_check_cookie@4; __security_check_cookie(x)
Up	p	sub_401000+AB	call @__security_check_cookie@4; __security_check_cookie(x)
Up	p	sub_401000+D5	call @__security_check_cookie@4; __security_check_cookie(x)
Up	p	sub_401000+FD	call @__security_check_cookie@4; __security_check_cookie(x)
Up	p	sub_401000+11C	call @__security_check_cookie@4; __security_check_cookie(x)
Up	p	sub_401000+13A	call @__security_check_cookie@4; __security_check_cookie(x)
Up	p	sub_401170+97	call @__security_check_cookie@4; __security_check_cookie(x)
Up	p	sub_401220+B3	call @__security_check_cookie@4; __security_check_cookie(x)
Up	p	sub_4012E0+4E	call @__security_check_cookie@4; __security_check_cookie(x)
Up	p	sub_4012E0+AE	call @__security_check_cookie@4; __security_check_cookie(x)
Up	p	sub_4012E0+130	call @__security_check_cookie@4; __security_check_cookie(x)
Up	p	sub_4018E0+15C	call @__security_check_cookie@4; __security_check_cookie(x)
Up	p	sub_4018E0+1AD	call @__security_check_cookie@4; __security_check_cookie(x)
Up	p	sub_401AA0+AE	call @__security_check_cookie@4; __security_check_cookie(x)
Up	p	sub_401AA0+C6	call @__security_check_cookie@4; __security_check_cookie(x)
Up	p	sub_401B80+73	call @__security_check_cookie@4; __security_check_cookie(x)
Up	p	sub_401B80+E1	call @__security_check_cookie@4; __security_check_cookie(x)
Up	p	sub_401B80+12E	call @__security_check_cookie@4; __security_check_cookie(x)
Up	p	sub_401CC0+35	call @__security_check_cookie@4; __security_check_cookie(x)
Up	p	sub_401CC0+F6	call @__security_check_cookie@4; __security_check_cookie(x)
Up	p	sub_401EF0+1EC	call @__security_check_cookie@4; __security_check_cookie(x)
Up	p	sub_4020F0+244	call @__security_check_cookie@4; __security_check_cookie(x)
Up	p	sub_402340+19B	call @__security_check_cookie@4; __security_check_cookie(x)
Up	p	sub_4024F0+4A	call @__security_check_cookie@4; __security_check_cookie(x)
Up	p	sub_402550+A0	call @__security_check_cookie@4; __security_check_cookie(x)
Up	p	sub_402AB0+C8	call @__security_check_cookie@4; __security_check_cookie(x)
Up	p	sub_402B90+29	call @__security_check_cookie@4; __security_check_cookie(x)
Up	p	sub_402B90+4F	call @__security_check_cookie@4; __security_check_cookie(x)
Up	p	sub_402B90+215	call @__security_check_cookie@4; __security_check_cookie(x)
Up	p	sub_402F20+293	call @__security_check_cookie@4; __security_check_cookie(x)
Up	p	sub_4031D0+17B	call @__security_check_cookie@4; __security_check_cookie(x)
Do...	p	__output_l+B70	call @__security_check_cookie@4; __security_check_cookie(x)
Do...	p	__invoke_watson+121	call @__security_check_cookie@4; __security_check_cookie(x)
Do...	p	__except_handler4+34	call @__security_check_cookie@4; __security_check_cookie(x)
Do...	p	__except_handler4+44	call @__security_check_cookie@4; __security_check_cookie(x)
Do...	p	__except_handler4+B7	call @__security_check_cookie@4; __security_check_cookie(x)
Do...	p	__except_handler4+C7	call @__security_check_cookie@4; __security_check_cookie(x)
Do...	p	__except_handler4+14A	call @__security_check_cookie@4; __security_check_cookie(x)
Do...	p	__except_handler4+15A	call @__security_check_cookie@4; __security_check_cookie(x)
Do...	p	__unwind_handler4+1B	call @__security_check_cookie@4; __security_check_cookie(x)
Do...	p	__write_nolock+72C	call @__security_check_cookie@4; __security_check_cookie(x)

Line 1 of 50

Figure 8: portion of xrefs about security_check_cookie where almost the majority of them are function of WinMain

To better understand how this works, I will perform some dynamic analysis. Using ApateDNS I confirm there are connections to domain “www.polarroute.com” but also to “www.northpoleroute.com” (this one can be the changed ServerName, in the switch case 5) and as usual other windows connections. I tried to browse to those websites, but I found nothing interesting neither suspected.



ApateDNS

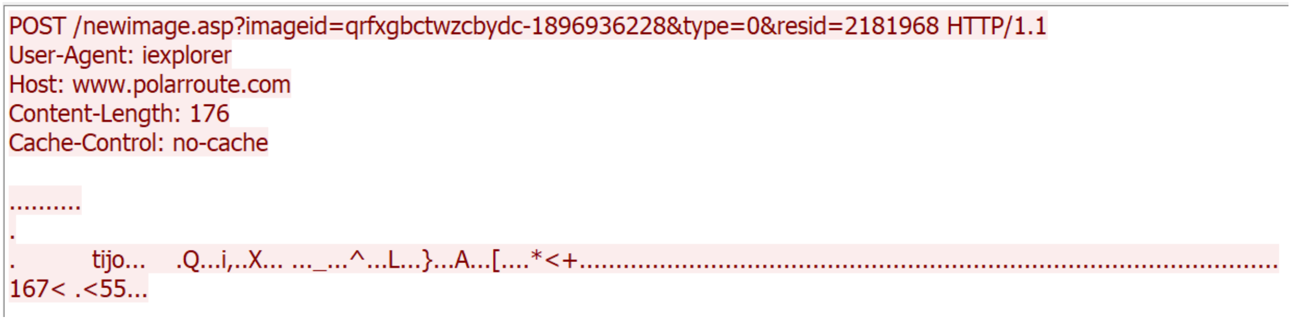
Capture Window | DNS Hex View

Time	Domain Requested	DNS Returned
15:21:06	www.polarroute.com	FOUND
15:21:10	tsfe.trafficshaping.dsp.mp.microsoft.com	FOUND
15:21:21	tsfe.trafficshaping.dsp.mp.microsoft.com	FOUND
15:21:33	tsfe.trafficshaping.dsp.mp.microsoft.com	FOUND
15:21:44	tsfe.trafficshaping.dsp.mp.microsoft.com	FOUND
15:21:56	tsfe.trafficshaping.dsp.mp.microsoft.com	FOUND
15:22:08	tsfe.trafficshaping.dsp.mp.microsoft.com	FOUND
15:22:14	wpad.home	FOUND
15:22:14	wpad.home	FOUND
15:22:19	tsfe.trafficshaping.dsp.mp.microsoft.com	FOUND
15:22:31	tsfe.trafficshaping.dsp.mp.microsoft.com	FOUND

15:22:08	tsfe.trafficshaping.dsp.mp.microsoft.com	FOUND
15:22:14	wpad.home	FOUND
15:22:14	wpad.home	FOUND
15:22:19	tsfe.trafficshaping.dsp.mp.microsoft.com	FOUND
15:22:31	tsfe.trafficshaping.dsp.mp.microsoft.com	FOUND
15:22:39	wpad.home	FOUND
15:22:39	wpad.home	FOUND
15:22:42	tsfe.trafficshaping.dsp.mp.microsoft.com	FOUND
15:22:53	fe3cr.delivery.mp.microsoft.com	FOUND
15:22:54	www.northpoleroute.com	FOUND

Figure 9: ApateDNS output where a connection to www.polarroute.com and www.northpoleroute.com is made

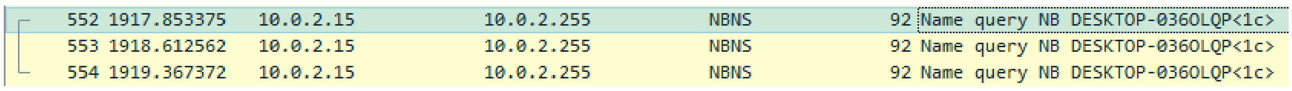
The other attempt is to run the malware and with Wireshark capture the network traffic. During capture there were different POST requests to the same URL but changing only the last part which I think is the part depending on elapsed time since restart.



```
POST /newimage.asp?imageid=qrfxgbctwzcbydc-1896936228&type=0&resid=2181968 HTTP/1.1
User-Agent: iexplorer
Host: www.polarroute.com
Content-Length: 176
Cache-Control: no-cache

..........
.
.        tijo...    .Q...i,..X... ..._...^...L...}...A...[....*<+...........................................................................................
167< .<55...
```

Figure 10: content of the POST request, all of them are equals but the different part is resid which depends on elapsed time since restart

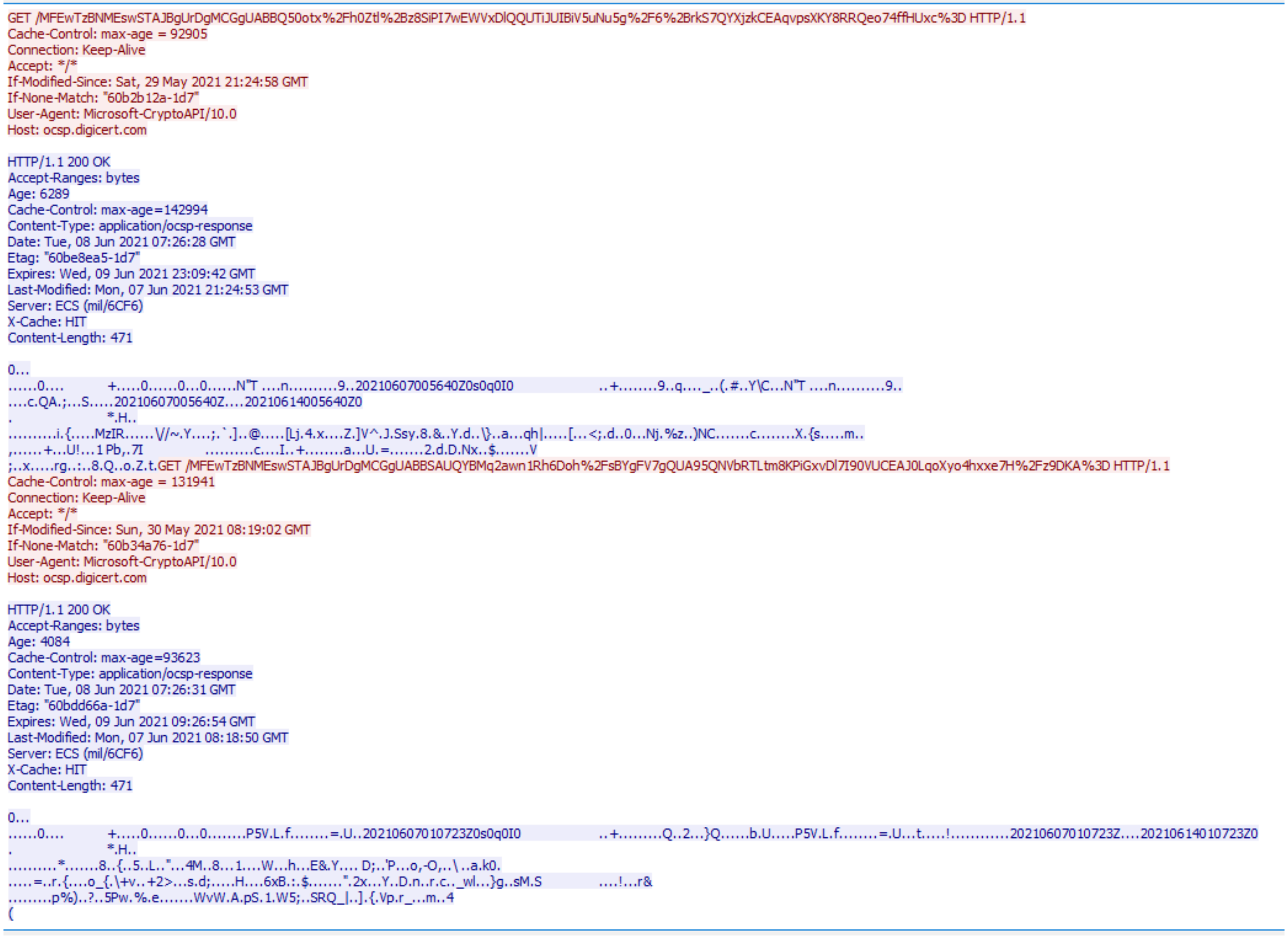
There are also some requests to protocol NBNS which is the part where asks for computer or user name.



```
552 1917.853375    10.0.2.15          10.0.2.255          NBNS          92 Name query NB DESKTOP-036OLQP<1c>
553 1918.612562    10.0.2.15          10.0.2.255          NBNS          92 Name query NB DESKTOP-036OLQP<1c>
554 1919.367372    10.0.2.15          10.0.2.255          NBNS          92 Name query NB DESKTOP-036OLQP<1c>
```

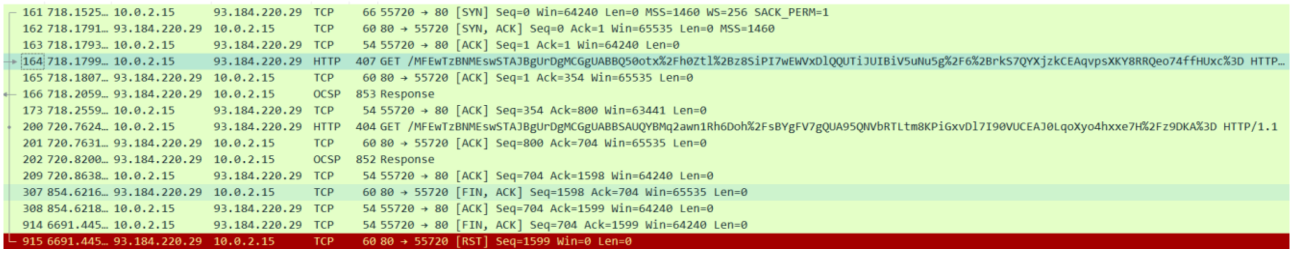
Figure 11: connection to get the computer name

I found also some GET requests which can be the URL made by the functions reading HTML files.



```
GET /MFEwTzBNMEswSTAJBgUrDgMCGgUABBQ50otx%2Fh0Ztl%2Bz8SiPI7wEWVxDlQQUTiJUIBiV5uNu5g%2F6%2BrkS7QYXjzkCEAqvpsXKY8RRQeo74ffHUxc%3D HTTP/1.1
Cache-Control: max-age = 92905
Connection: Keep-Alive
Accept: */*
If-Modified-Since: Sat, 29 May 2021 21:24:58 GMT
If-None-Match: "60b2b12a-1d7"
User-Agent: Microsoft-CryptoAPI/10.0
Host: ocsp.digicert.com

HTTP/1.1 200 OK
Accept-Ranges: bytes
Age: 6289
Cache-Control: max-age=142994
Content-Type: application/ocsp-response
Date: Tue, 08 Jun 2021 07:26:28 GMT
Etag: "60be8ea5-1d7"
Expires: Wed, 09 Jun 2021 23:09:42 GMT
Last-Modified: Mon, 07 Jun 2021 21:24:53 GMT
Server: ECS (mil/6CF6)
X-Cache: HIT
Content-Length: 471

0...
......0....          +.....0......0...0......N"T ....n..........9..20210607005640Z0s0q0I0              ..+........9..q...._..(.#..Y\C...N"T ....n..........9..
....c.QA.;...S.....20210607005640Z....20210614005640Z0
.                    *.H..
..........i.{.....MzIR......\/~.Y....;.`.]..@.....[Lj.4.x....Z.]V^.J.Ssy.8.&..Y.d..\}..a...qh|.....[...<;.d..0...Nj.%z..)NC.......c........X.{s.....m..
,......+...U!...1 Pb,.7I             ..........c....I..+........a...U.=.......2.d.D.Nx..$.......V
;..x.....rg..:..8.Q..o.Z.t.GET /MFEwTzBNMEswSTAJBgUrDgMCGgUABBSAUQYBMq2awn1Rh6Doh%2FsBYgFV7gQUA95QNVbRTLtm8KPiGxvDl7I90VUCEAJ0LqoXyo4hxxe7H%2Fz9DKA%3D HTTP/1.1
Cache-Control: max-age = 131941
Connection: Keep-Alive
Accept: */*
If-Modified-Since: Sun, 30 May 2021 08:19:02 GMT
If-None-Match: "60b34a76-1d7"
User-Agent: Microsoft-CryptoAPI/10.0
Host: ocsp.digicert.com

HTTP/1.1 200 OK
Accept-Ranges: bytes
Age: 4084
Cache-Control: max-age=93623
Content-Type: application/ocsp-response
Date: Tue, 08 Jun 2021 07:26:31 GMT
Etag: "60bdd66a-1d7"
Expires: Wed, 09 Jun 2021 09:26:54 GMT
Last-Modified: Mon, 07 Jun 2021 08:18:50 GMT
Server: ECS (mil/6CF6)
X-Cache: HIT
Content-Length: 471

0...
......0....          +.....0......0...0........P5V.L.f........=.U..20210607010723Z0s0q0I0            ..+.........Q..2...}Q......b.U.....P5V.L.f........=.U...t.....!............20210607010723Z....20210614010723Z0
.                    *.H..
..........*.......8..{..5..L.."...4M..8...1....W...h...E&.Y.... D;..'P...o,-O,..\..a.k0.
.....=..r.{....o_{.\+v..+2>...s.d;.....H....6xB.:.$.......".2x...Y..D.n..r.c.._wl...}g..sM.S            ....!...r&
.........p%)..?..5Pw.%.e.......WvW.A.pS.1.W5;..SRQ_|..].{.Vp.r_...m..4
(
```

Figure 12: content of GET request to another malicious domain



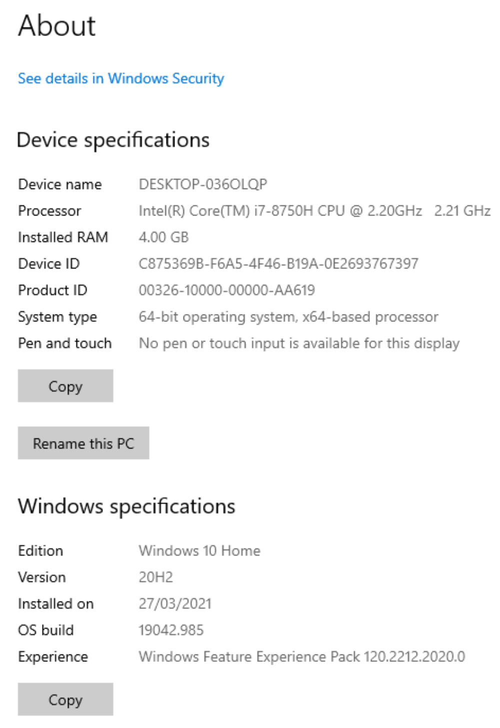
```
┌ 161 718.1525… 10.0.2.15     93.184.220.29 TCP    66 55720 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
│ 162 718.1791… 93.184.220.29 10.0.2.15     TCP    60 80 → 55720 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
│ 163 718.1793… 10.0.2.15     93.184.220.29 TCP    54 55720 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
→ 164 718.1799… 10.0.2.15     93.184.220.29 HTTP  407 GET /MFEwTzBNMEswSTAJBgUrDgMCGgUABBQ50otx%2Fh0Ztl%2Bz8SiPI7wEWVxDlQQUTiJUIBiV5uNu5g%2F6%2BrkS7QYXjzkCEAqvpsXKY8RRQeo74ffHUxc%3D HTTP…
│ 165 718.1807… 93.184.220.29 10.0.2.15     TCP    60 80 → 55720 [ACK] Seq=1 Ack=354 Win=65535 Len=0
← 166 718.2059… 93.184.220.29 10.0.2.15     OCSP  853 Response
│ 173 718.2559… 10.0.2.15     93.184.220.29 TCP    54 55720 → 80 [ACK] Seq=354 Ack=800 Win=63441 Len=0
│ 200 720.7624… 10.0.2.15     93.184.220.29 HTTP  404 GET /MFEwTzBNMEswSTAJBgUrDgMCGgUABBSAUQYBMq2awn1Rh6Doh%2FsBYgFV7gQUA95QNVbRTLtm8KPiGxvDl7I90VUCEAJ0LqoXyo4hxxe7H%2Fz9DKA%3D HTTP/1.1
│ 201 720.7631… 93.184.220.29 10.0.2.15     TCP    60 80 → 55720 [ACK] Seq=800 Ack=704 Win=65535 Len=0
│ 202 720.8200… 93.184.220.29 10.0.2.15     OCSP  852 Response
│ 209 720.8638… 10.0.2.15     93.184.220.29 TCP    54 55720 → 80 [ACK] Seq=704 Ack=1598 Win=64240 Len=0
│ 307 854.6216… 93.184.220.29 10.0.2.15     TCP    60 80 → 55720 [FIN, ACK] Seq=1598 Ack=704 Win=65535 Len=0
│ 308 854.6218… 10.0.2.15     93.184.220.29 TCP    54 55720 → 80 [ACK] Seq=704 Ack=1599 Win=64240 Len=0
│ 914 6691.445… 10.0.2.15     93.184.220.29 TCP    54 55720 → 80 [FIN, ACK] Seq=704 Ack=1599 Win=64240 Len=0
└ 915 6691.445… 93.184.220.29 10.0.2.15     TCP    60 80 → 55720 [RST] Seq=1599 Win=0 Len=0
```

Figure 13: different GET requests to a base64 URL of a malicious domain are made

As the domain is no more www.polarroute.com or www.nothpoleroute.com I thought that this request is not made by the program and so I browsed its host ocsp.digicert.com which is classified as a malicious website where some spam activity is made. I tried to decode this base64 URL but found nothing interesting. I totally exclude it is made by other processes as the malware is the only one program running on this virtual machine.

After the execution the malware was no more in its initial path, I tried the command to locate it into the machine (there should be some copies of it) but I was not able to find it. I noticed also that after the removal the internet requests continued. The only part that I found about program deletion is the last part of function 403360 where the shell was invoked to ping the localhost and deleted something. This something can be the process itself but maybe performing a localhost, this could mean to like install itself and continue performing its actions even without finding it.

As there are different parts that I was not able to understand, I tried to perform some debug.


About

See details in Windows Security

Device specifications

Device name	DESKTOP-036OLQP
Processor	Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz 2.21 GHz
Installed RAM	4.00 GB
Device ID	C875369B-F6A5-4F46-B19A-0E2693767397
Product ID	00326-10000-00000-AA619
System type	64-bit operating system, x64-based processor
Pen and touch	No pen or touch input is available for this display

Windows specifications

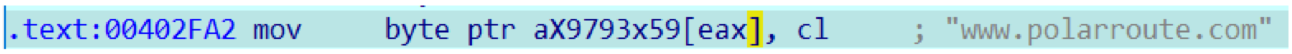
Edition	Windows 10 Home
Version	20H2
Installed on	27/03/2021
OS build	19042.985
Experience	Windows Feature Experience Pack 120.2212.2020.0

Figure 14: device info as computer name and OS version

When debugging I put a breakpoint in the WinMain at the call of both functions 402F20 and 402900. I stepped into the first function and then inside 402B90 where there is the output of 401000 regarding the OS version, that is 8, so this means that my OS is Windows 8 or Windows Server 2021 (majorOS=6 and minorOS=2) even if looking at system settings, it is a Windows10 Home edition (this is a little bit strange).

Following the debugger flow, the if condition at line 34 is verified, so it will enter on it and return 0 without executing the other instructions: so 402B90, if the OS version is not the required one, will return 0, otherwise will continue its flow without running by command line the dll on the file containing the resource data.

After that, coming back to 402F20 is executed the part regarding the XOR values in memory with letter ‘V’, which, as predicted, returns the domain value of “www.polarroute.com”. But as the for loop was big enough, to continue the debugging flow I selected in the assembly the instruction where the ExpandEnvironmentStringsA is called, I set the IP to it and jumped there.



```
.text:00402FA2 mov     byte ptr aX9793x59[eax], cl     ; "www.polarroute.com"
```

Figure 15: the debugger confirmed the correct output of XOR

After the environment string was expanded and as it was successful, the program created a new path of which I saw only the first value “C” (it is a pointer to a memory location). In this path a directory is created, and a file is filled in and then, as the user is not an admin, the key Run for current user is created with value MicroMedia. This key is created after the call to function 402E70 that opens MediaCenter.exe (“C:\\Users\\User\\AppData\\Local\\Temp\\MicroMedia\\MediaCenter.exe”) in which writes the content of the current file previously read. This means that every time the current user logs on the key is run, so the MediaCenter.exe is run but it contains the data of this current program and as MediaCenter.exe is a Windows software used to play video, I think that this behavior is very similar to a Trojan Horse.

Then, in the last lines of this function, the value of dword_4131A0 is checked and as it is 0, the if condition is true and so the file at Dst path is run but I was not able to retrieve its value.

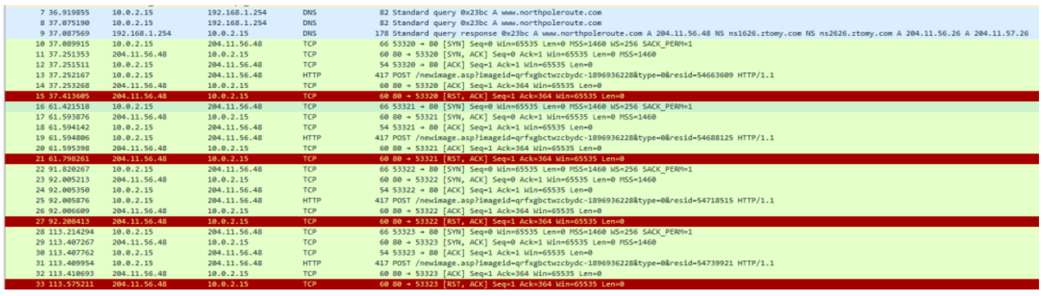
At the end, function 403360 is called and analysing it I discovered that the deleted file was the running program. Of course, I tried to continue debugging coming back to WinMain function where I wanted to analyse 402900 but as the file was deleted there were no possibilities to call this function. In order to proceed I unzipped the original file and I debugged it starting from 402900, with the awareness that some parts will not work as 402F20 was not called and here some variables are set to be used also by 402900.

The first thing checked at 402900 is the dword_4131A0, which is 0, and so the if condition is not verified and the call to Sleep function is not done. Instead, function 401420 stores in dword_414760 the value of the hostname computed previously, then gets the computer name which is “DESK” as seen in Figure 14 and then function 401220 is called which generates the parameter of the URL used in the POST request and retrieves also the Volume Serial Number. Then, following the execution flow in 401420, 3 different URLs are computed and stored in memory, then Internet Explorer is opened and is not closed, as if it remains listening waiting for a request and a space in memory is reserved for variable lpBuffer.

As 401420 ended its execution, the program flow came back to 402900 when the while(1) instruction has been read and immediately after the function 402340 is called, whose output will be used for another while loop.

Function 402340 sets the values of array v7 (contains time values) and variable v1 (according to user admin and OS version and also with respect to the equality between current thread and SID), then converts the current username in multibyte type. The last step of this function is the call to function 4018E0 that will make a POST request to the previously set URL. The return of function 402340 is 0, it is based on return value of 4018E0 that is 0 when is not able to connect or to send the POST request.

So, the nested while loop in 402900 is not verified and so the 401B80 is not performed and also the switch cases. The program will jump to the for loop repeated 3 times where it still tries to perform the connection. This step does not correspond to Wireshark output where different POST requests are made and above all they are more than 3 before making a sort of pause in the requests which can be the consequence of a Sleep call.



```
 7 36.919855   10.0.2.15      192.168.1.254  DNS    82 Standard query 0x23bc A www.northpoleroute.com
 8 37.075190   10.0.2.15      192.168.1.254  DNS    82 Standard query 0x23bc A www.northpoleroute.com
 9 37.087569   192.168.1.254  10.0.2.15      DNS   178 Standard query response 0x23bc A www.northpoleroute.com A 204.11.56.48 NS ns1626.ztomy.com NS ns2626.ztomy.com A 204.11.56.26 A 204.11.57.26
10 37.089915   10.0.2.15      204.11.56.48   TCP    66 53320 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM=1
11 37.251353   204.11.56.48   10.0.2.15      TCP    60 80 → 53320 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
12 37.251511   10.0.2.15      204.11.56.48   TCP    54 53320 → 80 [ACK] Seq=1 Ack=1 Win=65535 Len=0
13 37.252167   10.0.2.15      204.11.56.48   HTTP  417 POST /newimage.asp?imageid=qrfxgbctwzcbydc-1896936228&type=0&resid=54663609 HTTP/1.1
14 37.253268   204.11.56.48   10.0.2.15      TCP    60 80 → 53320 [ACK] Seq=1 Ack=364 Win=65535 Len=0
15 37.413605   204.11.56.48   10.0.2.15      TCP    60 80 → 53320 [RST, ACK] Seq=1 Ack=364 Win=65535 Len=0
16 61.421518   10.0.2.15      204.11.56.48   TCP    66 53321 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM=1
17 61.593876   204.11.56.48   10.0.2.15      TCP    60 80 → 53321 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
18 61.594142   10.0.2.15      204.11.56.48   TCP    54 53321 → 80 [ACK] Seq=1 Ack=1 Win=65535 Len=0
19 61.594806   10.0.2.15      204.11.56.48   HTTP  417 POST /newimage.asp?imageid=qrfxgbctwzcbydc-1896936228&type=0&resid=54688125 HTTP/1.1
20 61.595398   204.11.56.48   10.0.2.15      TCP    60 80 → 53321 [ACK] Seq=1 Ack=364 Win=65535 Len=0
21 61.798261   204.11.56.48   10.0.2.15      TCP    60 80 → 53321 [RST, ACK] Seq=1 Ack=364 Win=65535 Len=0
22 91.820267   10.0.2.15      204.11.56.48   TCP    66 53322 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM=1
23 92.005213   204.11.56.48   10.0.2.15      TCP    60 80 → 53322 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
24 92.005350   10.0.2.15      204.11.56.48   TCP    54 53322 → 80 [ACK] Seq=1 Ack=1 Win=65535 Len=0
25 92.005876   10.0.2.15      204.11.56.48   HTTP  417 POST /newimage.asp?imageid=qrfxgbctwzcbydc-1896936228&type=0&resid=54718515 HTTP/1.1
26 92.006609   204.11.56.48   10.0.2.15      TCP    60 80 → 53322 [ACK] Seq=1 Ack=364 Win=65535 Len=0
27 92.208413   204.11.56.48   10.0.2.15      TCP    60 80 → 53322 [RST, ACK] Seq=1 Ack=364 Win=65535 Len=0
28 113.214294  10.0.2.15      204.11.56.48   TCP    66 53323 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM=1
29 113.407267  204.11.56.48   10.0.2.15      TCP    60 80 → 53323 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
30 113.407762  10.0.2.15      204.11.56.48   TCP    54 53323 → 80 [ACK] Seq=1 Ack=1 Win=65535 Len=0
31 113.409954  10.0.2.15      204.11.56.48   HTTP  417 POST /newimage.asp?imageid=qrfxgbctwzcbydc-1896936228&type=0&resid=54739921 HTTP/1.1
32 113.410693  204.11.56.48   10.0.2.15      TCP    60 80 → 53323 [ACK] Seq=1 Ack=364 Win=65535 Len=0
33 113.575211  204.11.56.48   10.0.2.15      TCP    60 80 → 53323 [RST, ACK] Seq=1 Ack=364 Win=65535 Len=0
```

Figure 16: 4 different POST requests made one after the other to same domain, the only difference is the elapsed time

What I do not understand is why the domain has changed in www.northpoleroute.com (as shown in DNS request), maybe is the last line performed in function 401420 where lpBuffer is set according to a specific memory area and this value is used to make some connections as a server name.

After the for loop the dword_4147A0 is set and the program will sleep for 1 hour, after which it will continue with the while(1) loop.

The fact that function 4018E0 returns 0 maybe is because it is not able to retrieve the value of dword_414560 from memory even if it was set inside 401420 before executing this part. If this function would be able to make the POST request and so return 1, the program would have run 401B80. In function 401B80 there is the connection to URL set in dword_414460 but when debugging also this function returns 0 because of the internet handle that is not able to retrieve. The previous functions where an Internet handle is created are 401420 and 4018E0, as this one is the last one called and because of the error mentioned before, it closes the Internet connection, so function 401B80 finds internet closed and even has the URL is not able to open it.

I think that the only way to execute 401B80 and the switch cases is to like patch 402340, above all the Interne connection in 4018E0.

The requests I found on Wireshark are only the ones made by 4018E0 inside 402340, thanks to the while(1) loop of function 402900. As the switch case is not executed, so case 6 will not run (the only condition to break the while(1) loop), this program briefly does only some things: when executed it retrieves some information (OS version, process ID, tokens, SID), copies inside file MediaCenter.exe the data inside this executable and runs it. After that, pings the localhost and deletes this executable. As in Wireshark I have the POST requests, it has to find a way to call function 402900 to send them and in my opinion there are two ways: as MediaCenter.exe has been executed and has the same code of this executable, the flow can be managed by this program; the other solution is that even it was deleted, with the ping command it created a sort of installation (maybe loaded the whole program in memory) and even if it deletes, it has this way to continue running. After the deletion, 3 different URLs are stored in memory and some other info are retrieved like VSN, computer name and user name; then some posts requests are made and as shown in Wireshark they have like the same URL but the changing argument is the last one which depends on the elapsed time since restart. The program continues sending these requests and checking if the Internet connection is established to send a GET request in a specific webpage and according to the content of the read page, different actions will be performed. The program will stop only when the Run key containing the value of MediaCenter.exe is deleted and another ping to localhost with consecutive deletion of current executable is performed. As I just guessed, the flow of the program is managed by MediaCenter.exe.

So, analysing this behaviours I can say that this malware is a Trojan Horse because it deletes itself (no trace about it remains), it copies itself inside a program that plays video but also performs its tasks, sends different requests to different webpages maybe to send data of the user.

The last thing I want to point is that function __security_chec_cookie while debugging and analysing the assembly code, not the decompiled code, was never called and so never executed that part: in the debugging I was not able to understand when that code should be executed. This function can also be a sort of undebugged function but as it is called almost every time a function ends, this could be used to set cookies inside those pages and keep the connection opened.

To confirm the malware type, I put it inside virustotal and it was 62/70 a Trojan.