

AZORult Trojan Serving Aurora Ransomware by MalActor Oktropys

By Vishal Thakur

Published: 2018-08-18 · Archived: 2026-04-05 15:16:07 UTC



This is a guest post from [Vishal Thakur](#), a Security Incident Handler, APAC CSIRT for Salesforce. In this article Thakur takes a deep dive 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.



Visit Advertiser website [GO TO PAGE](#)

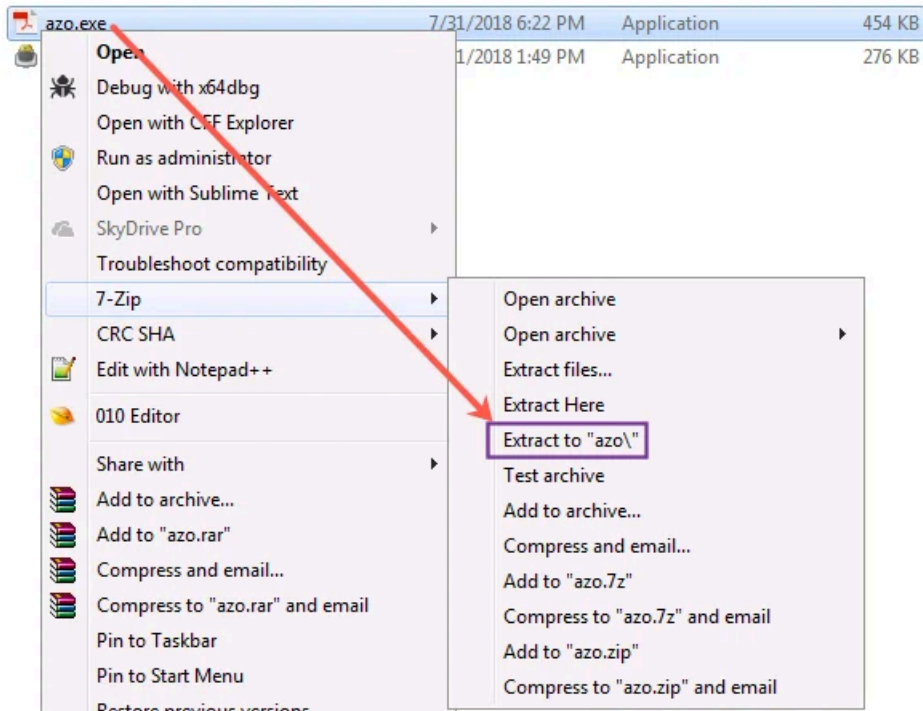
We also identified the MalActor “Oktropys” running the Aurora ransomware campaign in this case.

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:

Name	Date modified	Type	Size
\$1	7/31/2018 10:14 PM	File folder	
SPLUGINS\DIR	7/31/2018 10:14 PM	File folder	

Step into the folder \$1

Name	Date modified	Type	Size
1337	7/31/2018 10:14 PM	File folder	

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

Name	Date modified	Type	Size
AU3_EXE_2018-07-18_23-01.exe	7/18/2018 11:01 PM	Application	234 KB
Ransom_2018-07-18_23-06.exe	7/18/2018 11:06 PM	Application	430 KB

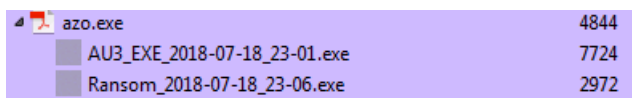
Now, instead of getting to the payloads directly, we'll follow the malware execution and see how it is using these embedded payloads.

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
apischema.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
rpctr4.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
SortDefault.nls	0x17b0000	2.81 MB
sspici.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


```
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.



azo.exe	4844
AU3_EXE_2018-07-18_23-01.exe	7724
Ransom_2018-07-18_23-06.exe	2972

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
crt.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 SpisCli.GetUserNameExA to wininet.75403151
0011E6D0 /00000002
0011E6D4 75634D38 ; ASCII "PC\Administrator"
0011E6D8 0011E704
0011E6DC 00000000
0011E6E0 00000104
0011E6E4 75634D38 ; ASCII "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\17521679265312186441325254.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\17521679265312186441325254.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](#)

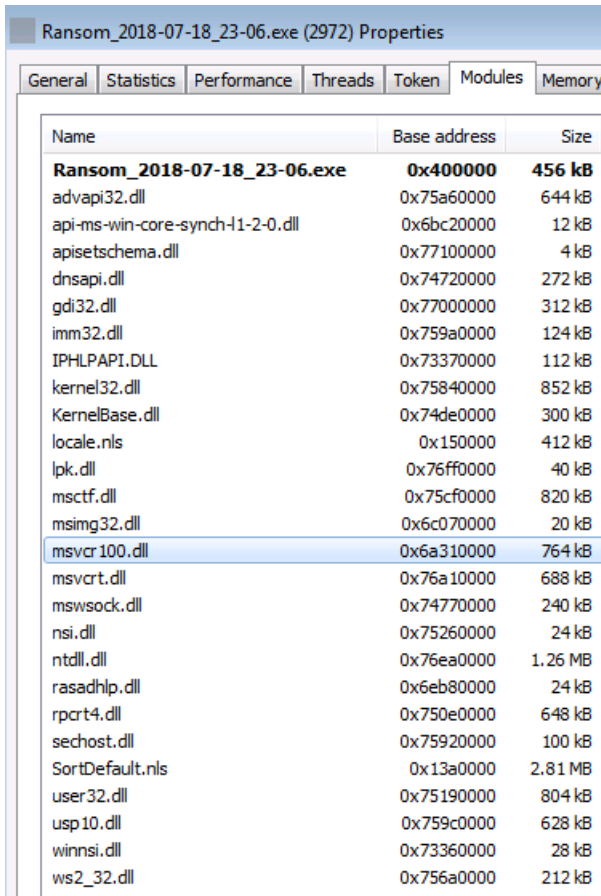
This function is called on to get the [cryptographic service provider](#) (CSP).

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 get the IOC from them.

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



Name	Base address	Size
Ransom_2018-07-18_23-06.exe	0x400000	456 kB
advapi32.dll	0x75a60000	644 kB
api-ms-win-core-synch-l1-2-0.dll	0x6bc20000	12 kB
apisetschema.dll	0x77100000	4 kB
dnsapi.dll	0x74720000	272 kB
gdi32.dll	0x77000000	312 kB
imm32.dll	0x759a0000	124 kB
IPHLPAPI.DLL	0x73370000	112 kB
kernel32.dll	0x75840000	852 kB
KernelBase.dll	0x74de0000	300 kB
locale.nls	0x150000	412 kB
lpk.dll	0x76ff0000	40 kB
msctf.dll	0x75cf0000	820 kB
msimg32.dll	0x6c070000	20 kB
msvcr100.dll	0x6a310000	764 kB
msvcrt.dll	0x76a10000	688 kB
mswsock.dll	0x74770000	240 kB
nsi.dll	0x75260000	24 kB
ntdll.dll	0x76ea0000	1.26 MB
rasadhlp.dll	0x6eb80000	24 kB
rpcrt4.dll	0x750e0000	648 kB
sechost.dll	0x75920000	100 kB
SortDefault.nls	0x13a0000	2.81 MB
user32.dll	0x75190000	804 kB
usp10.dll	0x759c0000	628 kB
winnsi.dll	0x73360000	28 kB
ws2_32.dll	0x756a0000	212 kB

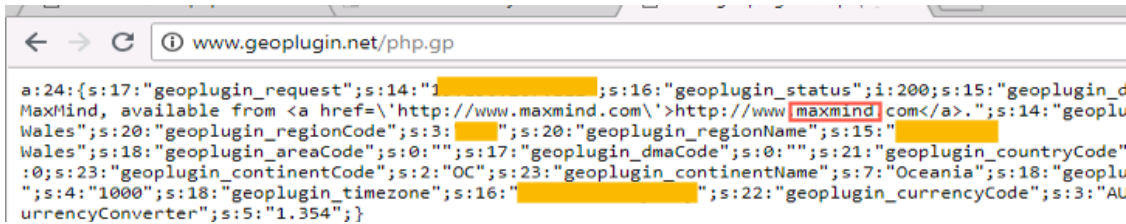
This ransomware is geo-targeted or at least it has that functionality built into it. To perform geolocation it attempts to connect to a geo-location site and get the location of the victim computer. Here's the call that is made for this purpose:

```
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:

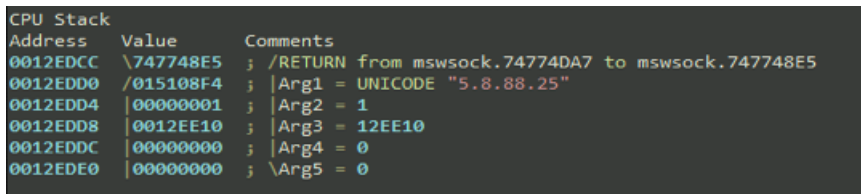
```
CPU Stack
Address Value Comments
0012EFC4 [756AE5A4 ; /RETURN from WS2_32.WSASocketW to WS2_32.WSASocketA+42
0012EFC8 00000002 ; |Arg1 = 2
0012EFC8 00000001 ; |Arg2 = 1
0012EFD0 00000006 ; |Arg3 = 6
0012EFD4 00000000 ; |Arg4 = 0
0012EFD8 00000000 ; |Arg5 = 0
0012EFD8 00000001 ; \Arg6 = 1
0012EFE0 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:

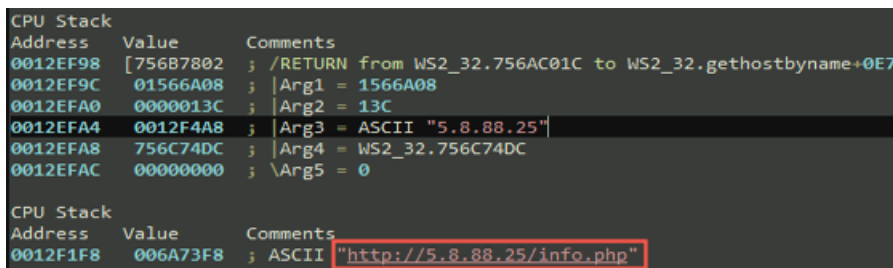


At this time it looks like the MalActor is avoiding infections in Russia based on the geo-result from the above functionality.

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



Connection Initiation



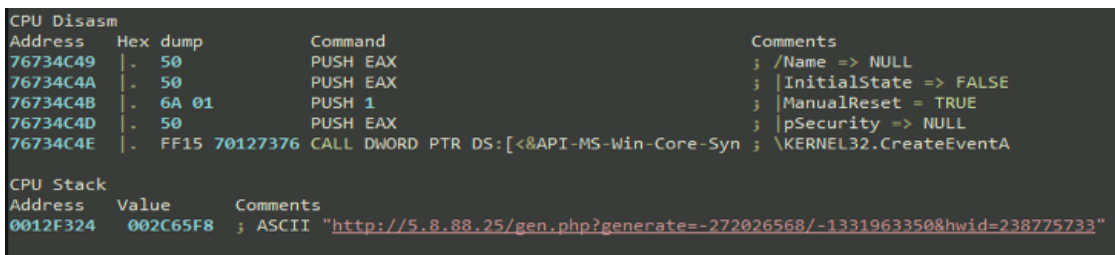
C2 Connection

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.

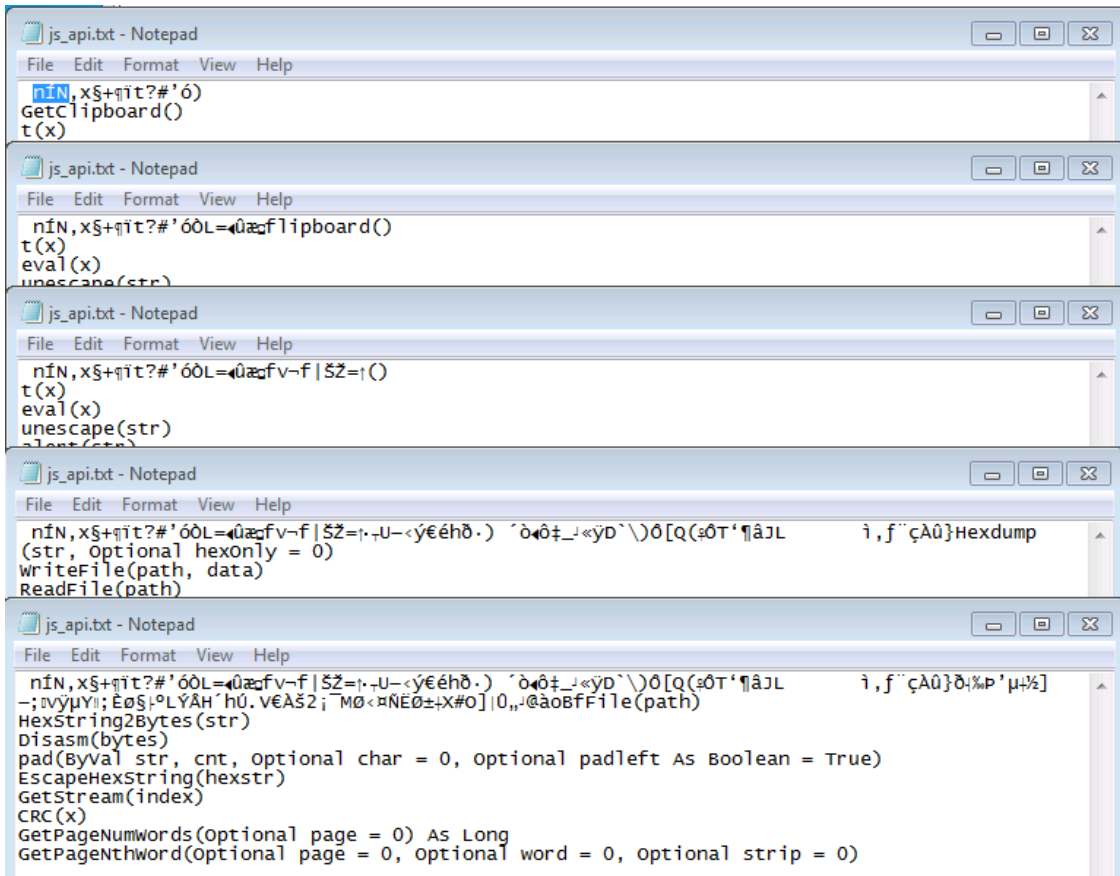
This malware uses ws2_32.dll for all networking operations. Look at the image below to see how the connection is constructed:

First the event is created:

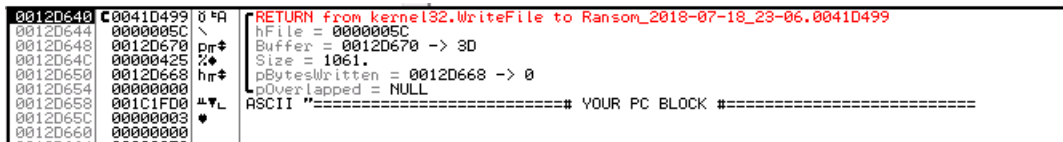


The next step is to load it in memory:

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:

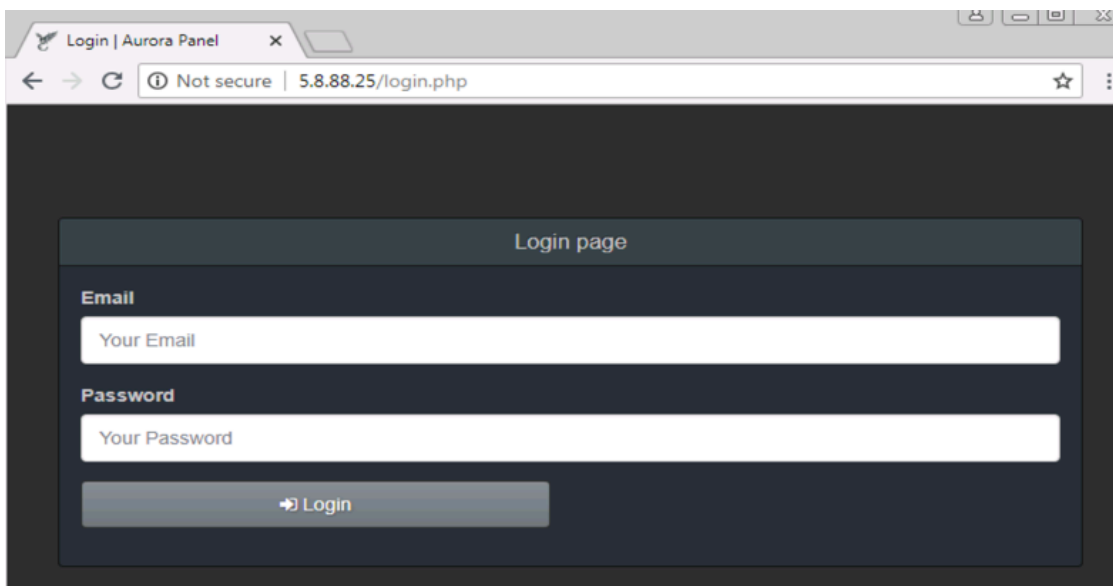


The ransom being asked by this MalActor is \$150. Here's the ransom note:

```
===== 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
```

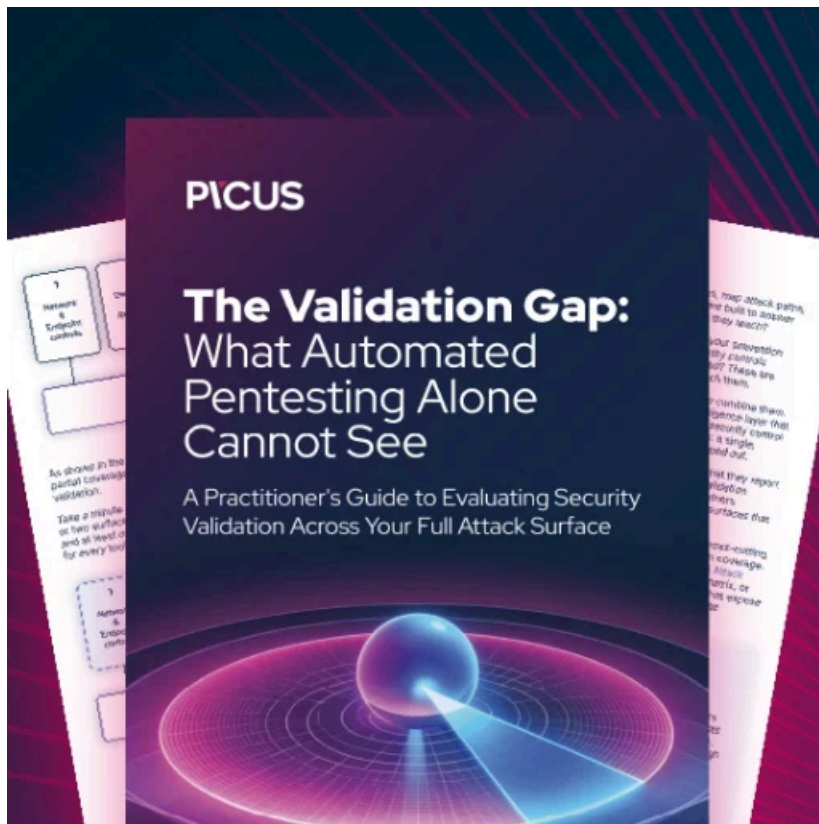
```
hxp://lulaaura[.]top/index.php?-?stealer
```

HASHES

```
Main Dropper: 09ffaa1523fbdceb7c0e6fa2be7221c161b5499dd45fc5dd4c210425fb333427
```

```
Stealer: 5151d9245858f3e28fa45f696421a49307436808d3ec18ff9e36f7876b0696d3
```

```
Ransomware: 41d35a960b3f28b1a729cdae920573de3ccefef7fdd3bbdb9d3ce729b6aa5277
```



[Automated Pentesting Covers Only 1 of 6 Surfaces.](#)

Automated pentesting proves the path exists. BAS proves whether your controls stop it. Most teams run one without the other.

This whitepaper maps six validation surfaces, shows where coverage ends, and provides practitioners with three diagnostic questions for any tool evaluation.

Source: <https://www.bleepingcomputer.com/news/security/azorult-trojan-serving-aurora-ransomware-by-malactor-oktrops/>