MAY 19, 2020 / VKZ2A

Netwalker Ransomware – From Static Reverse Engineering to Automatic Extraction

Author: Zero2Automated Course Team (preview from courses.zero2auto.com)

Netwalker ransomware has been around since at least 2019* and has recently been in the news from a TrendMicro report detailing it being leveraged embedded in a PowerShell script[1].

We will briefly go over how to recover the DLL files from the first script, it contains a large Base64 chunk of data that is base64 decoded and executed:

```
InVokE-ExPRESSIoN -COMmand $([StrinG]
([SySTEM.TexT.ENcOdInG]::ASCII.GETStRiNG([SysTEM.CoNVErT]:
:FRomBAsE64StRiNG("ICA <..snip..>
```

The next script layer then has an array of bytes which will be XOR decoded:

```
[bYTE[]]  $eFGCSjWKKPqLGPIYp  =
 @(0x67,0x67,0x1C,0x25,0x3e,0x33,0x22,0x1c,0x1A,0x1A,0x67,0x67,0x67,0x67,0x67,0x67,0x67,0x6
for ( $qTOJScZVUn = 0; $QTOjsCZvUn  -lt  $eFGCSjWKKPQLGpIYP.LeNgth;  $qtOJscZvUN++
   $EFGCSjWKKPqlGpIYP[$qtOJscZVUN] = $EFGCSJWkkPqLGPIYP[$QtOJscZvUN]  -BxOr 0x47
<..snip..>
```

Finally we are left with the main script layer that will load the DLL into memory:

```
[bYTE[]]    $eFGCSjWKKPqLGPlYp    =
 @(0x67,0x67,0x1C,0x25,0x3e,0x33,0x22,0x1c,0x1A,0x1A,0x67,0x67,0x67,0x67,0x67,0x67,0x67,0x6
for  (  $qTOJScZVUn =   0;
$QTOjsCZvUn    -lt   $eFGCSjWKKPQLGpIYP.LeNgth;    $qtOJscZvUN++  )  {
    $EFGCSjWKKPqlGpIYP[$qtOJscZVUN] = $EFGCSJWkkPqLGPlYP[$QtOJscZvUN]   -BxOr 0x47
<..snip..>
```

Finally we are left with the main script layer that will load the DLL into memory:

```
 [byte[]]    $ptFvKdtq
=   @(0xad,0xde,0x90,0x00,0x03,0x00,0x00,0x00,0x04,0x00,0x00,0x00,0xff,0xff,0x00,0x00,0xb8,0x
```

Netwalker DLL

The first thing the DLL does is resolve all of its needed functions.

Dynamic function resolution such as this normally involves some form of string hashing, in this case it is CRC32.

In the picture above you can see the hash value for FindResourceA being passed to a function that will resolve it and then store it in an array.

```
>>> hex(zlib.crc32('FindResourceA') % (1<<32))'0x3e006b7a'
```

After resolving there is a function responsible for simply getting the beginning address of the previous table. Shortly after resolving the needed functions the DLL gets its resource section.

A little ways down this data is passed off to another function which has a noticeable loop inside of it that is RC4 KSA[2]. If you are wanting to be able to identify some of these common routines I frequently recommend that people compile some files themselves and then take a look at them in a debugger or a disassembler. Routines I would recommend becoming familiar with will be included at the end of this blog.

So since the the resource section appears to be RC4 decrypted, let's take a look at the data.

```
>>> r = get_rsrc(pe)
>>> data = r[0][1]
>>> data[:100]"\x07\x00\x00\x00cZu-H!
<\x9b\xc5E\xder\x88&D\xf0\xd7{@\x00\xa9m\xa1\x1e\x8b\xac\xf7Z\xb4|\xc9&
[\x03\x1cY\xde\xf5\x8dA\x97C:8y\xb5&\x07\xb8q\xed\xe1^\xfd<\x98\x92\x95\x1a\xa6\xf7\xea
\x07\xea\x18v\xe1\x0f\xa1\xdcmR\x8c\x85'I\t\xc9\x12\x8e\xe5\x9d+\xc2\x7f\x00G\x01-Jp\xd48
"
```

The first 4 bytes appear to be a DWORD value stored as little endian, since I didn't see a reference to any hardcoded data being used as a RC4 key from the DLL and a small dword

value is the first thing in this resource section I'm going to assume it is some sort of a length value. Perhaps a key length?

```
>>> import struct
>>> from Crypto.Cipher import ARC4
>>> import struct
>>> struct.unpack_from('<I', data)(7,)
>>> key = data[4:4+7]
>>> rc4 = ARC4.new(key)
>>> t = rc4.decrypt(data[4+7:])
>>>
t[:100]'{"mpk":"+1KtL9ibbeqaChhoz4iEHeTtRtw8pNA5yC034\\/3klSA=","mode":0,"spsz":15360,"th
```

It's always nice when your hypothesis turns out to be correct on the first try, definitely isn't the norm though.

Now that we have decrypted the config we need to find more samples and see if we can decode more configs, I use VirusTotal sometimes and I understand it's not something everyone has access to so I will upload all the samples I end up going through onto MalwareBazaar.

For pivoting in order to find more samples I will normally turn to YARA and use a service that allows for scanning malware repositories, most of these services are either paid or private so you will normally obtain access to them as you progress into the world of malware research. VirusTotal has a convenient way to quickly search for hex strings using content searching so I pick a string that looks like it will remain static in this malware stub or template but will not give me a bunch of false positives. Sometimes this is harder than not and in this case the string I went with was actually my 3rd attempt.

Here we have two constant strings associated with SALSA20 or CHACHA20 encryption and following it is a dword value associated with hashing, kind of odd to see them back to back that way in the binary.

We can convert this to a hexlified string for doing VT content searching:

```
content:"{657870616e642033322d62797465206b657870616e642031362d62797465206b982f8a4
```

Searching in VirusTotal shows this to be a pretty good string for finding Netwalker related samples! Surprisingly enough it turns up lots of samples as well.

So now we need a decoder, or a way to automate decoding the config data from the files.

Let's build out a decoder, we need to:

- Read a file in
- Fix up MZ stomping
- Retrieve the resources
- Find the key length
- RC4 decode
- Print config

Let's setup for the first part for retrieving the resources:

*The full script can be found as part of the **Zero2Automated Advanced Malware Analysis Course***

Taking our script and the files from our content pivoting we will check the first 25 files and find that our script seems to work, we found a number of samples with new 'mpk' values but also one of the mailto samples instead of using onion domains:

{"mpk":"ESw9E\/AaTiQ9oZfLKNquj2dHV3TqTFU8c4zjg7HLHRc=","mode":0,"spsz":15360,"thr":150
Readme.txt",<..snip..>

{"mpk":"9WMeeC/lNogGtJPPRqQRzty2DMwgPRBWyU7mLzp992E=","mode":0,"thr":1500,"spsz":5
"crmask":".mailto[{mail1}].{id}","mail":["priparipri@tuta.io","praparapra@cock.li"],"lfile":"{ID}-Rea

So now we just need to decode more files, after decoding a larger data set I found that they all seemed to be using the same onion domains but managed to find a number of samples using email addresses as well:

2Hamlampampom@cock.liGalgalgalgalk@tutanota.comHariliuios@tutanota.comkavariusing@t

A quick web search of one of the emails, 'koktiktok@tuta.io', leads to a ransomed website immediately:

Going through the files that didn't decode out configs shows that we have found a number of decryptors that were uploaded to VirusTotal:

```
346fdff8d24cbb7ebd56f60933beca37a4437b5e1eb6e64f7ab21d48c862b5b744b5d24e5e8fd8e8
```

Also a DLL file that left the name embedded of 'Netwalker_dll.dll':

```
55870437ee97984ef461438777635b53eb52f8d83048ce3a825d95cabc2065f2
```

Mine the data

Now that we have a bunch of decoded configs we can do a bit of data mining as well.

We can use some bash magic to dump out the counts for 'mpk' values:

```
  1 0OQuf4tSKXtXERycfxzsWvcWPE2xayXH1rBpyAwyhVw=   1
1B+hUSlP+KcLFtZn5q0ND6WzFyQPIHxtM1vv1zlm4TE=   4
+1KtL9ibbeqaChhoz4iEHeTtRtw8pNA5yC034\/3klSA=   1
2ofyizDC65TzCiTW6SHYlP0mHF8VHklx5\/FFlECTOWE=   1
3ZVt8Qrdiimm/8OxCtmqUBZhZJ6DxqheQQ0wf9dIvU4=   1
7gkgNrgVvnSdnUifXmoRclyQU1tmmH9uLXZQ+mrJq1Y=   1
8kQrJzFg\/OugByxLk1JyT1cstH0gAQa6EmbRTJYlPyA=   1
9m56vLlZOlxhDtqMpZBLLOB21rKNaqnfMZmAICgbHB8=   1
9qQgCpjzg9Gg4R4QntJaXdyqbeK7tBKkyatX6U809B8=   1
9rekeP1eSTG+1alJX084XEnZkNZbMI2i7f7xMARuYG0=   1
9WMeeC/lNogGtJPPRqQRzty2DMwgPRBWyU7mLzp992E=   1
AopcQZuT8iGQsDBOQ5vum9rCosprWR4HAtyoDLBe2jk=   1
B1zaW+rgS4zOpn9MVHL0SXpAgmPx9992cw/zrjTStUQ=   1
bd9dpX2CspQ65wRiTP1TjN1cqwk8dSdJehbmYTNMxEY=   1
cePAiKEieIEsWYzINEJe7\/M7yXm+i6y\/Y+fjupgPdhc=   1
cT9cjcSYKMZMSxEObt0H0djsPtEVOW6Pfy0IU20uCEc=   1
EsvRQAFnerbfU8Z+27XyeJD82IPjl7PL62KSSzYOVUM=   2
ESw9E\/AaTiQ9oZfLKNquj2dHV3TqTFU8c4zjg7HLHRc=   1
EXgCIpyclJzspm07Loi9L5uOcxC+VZ/NjxWfOn7UqVE=   1
fXe6IoRs263eE42xnNF\/L5vYmC81qAS5F08zqHShBnE=   1
i3XpDekYmaDR5/C8gssEfc4BdgWSoKrU/RLtK10V4x8=   1
lrZwVI2HoRerwgd99xkmdIL3g0tfJOU1GoXMIPq5gkA=   1
izndhmDBfqQMxnbJX3rwRXETOO7hXuPRUrw91jUeuCE=   4
JgQzhhhjdTmwJuFlSWQ99doIV6sNRQWfNdqTbxulxCc=   1
```

```
JgRipXGeK7D6diXrdsBTdT\/o4ps0NS+6uUQ2K+lrmns=   1
JXrdFkPUAEVVROz8EN45wkOUGupEphGPQz4GoMLXwAk=   1
jZOEXL4Vyq413gNYUVSUuiJyvFvII63YSv3LMTm6YSQ=   1
kRmTz3GjStQDqVIO35VbP2DX5HzhQKd5bZJ5fkuQp2o=   1
LCiJ44QQcMh5P1jJtpV7XaJoMezvQc0D8bxdK97oWHM=   1
lxd4WwFEl80nASzXFosZNWY0+M3\/ljFwqXGcQ7C3Jmg=   1
O7iv5p+9M2xSENzUqbH2q4hEPiIShKQON7UQJZ\/qhxs=   1
pLcj6adbJHIyhEWwUd+Wht4pAXQMi4MhYXb1DmSE5Uc=   1
p+V4T581H2IJPowJP7b9Laa4P0cAfDgyREsFNhpETDw=   2
qGFBYhyYzmg5jkJh0giXpwDfUdbopaCdj4Zcxsdab0c=   1
RQfrkb+zP4wEDw4UerO5Xqwm/5R3zHSuTHD12cRuBTM=   4
srj15Yy9UFP\/jvwiLyCiEWXRb\/8RwnuHqGtdJCQzWk0=   1
sVMyZQe7xbAMYsKdX7B\/RmV92sc+Qd9YL7EmRVCZDnI=   1
TPeyvEhWusttXBNAkbii\/6tocN5IWcOl+O2JCMyXtWA=   2
uHbQk+xn3zyr9HAJIbR6wDKpszDZC2Dri\/D8cdZCPBU=   2
uMDlH\/VVLunARbKVQ4kKse+8Mdxbk00Pf+NSkoDASW0=   1
U+P1u\/vlAgiFK\/NnScZbuLtNbohxu\/65\/XiNMHzV82E=   1
usVG0ZPeGZRvG5clAh81RbrG7Whq9b3dtQ2tmoYf2iI=   1
vWdriuVFhHBNk87xEzAlkZNfVRnPIln1PsCx8s/+Uj8=   1
w2nejMQFYibyk97OISK8vcRcoUeJz2c024P98feVuXs=   2
XfwHXZw5ITJ/PB4tTJaJCa9cbigLKwS1hlNyMvT3CnQ=   1
XQixjtCbv3GHOcOGAAnHAO2rJFTbq8jr4j6bkajSWjU=   1
YisAjypwt6EuV8PeHBflq\/sQwAWgo+Y6CI9NKu3Fu10=   1
z2qgNTBTz6Ff6XQROdTvmEiBJn+b+HV3+Gw97n0rZxk=   1
zxbaxrewir3mRzNnoTP5NYvdhQNmFKwczCHzih9Kig0=
```

The ‘lend’ value usually contains the ransom message so we can dump out the values and see how often it has changed. When doing this I noticed that most of the samples contained a generic message but there were lots of samples that had slightly custom messages including victim names.

c21ecd18f0bbb28112240013ad42dad5c01d20927791239ada5b61e1c6f5f010

```
Hello, O2MICRO.
Your files are encrypted by Netwalker.All encrypted files for this computer has extension: .
{id}
```

3ba905e1cda7307163d4c8fe3fd03c2fbce7eda030522084e33d0604c204630e

```
Hi University of Seattle,
Your files are encrypted.All encrypted files for this computer has extension: .{id}
```

0d7ee7ce88e790ad66aa53589f5a2638207bc3adf2eb4f8a813fd52b5b22ba27

```
Hi Stellar,
Your files are encrypted.All encrypted files for this computer has extension: .{id}
```

b2d68a79a621c3f9e46f9df52ed19b8fec22c3cf5f4e3d8630a2bc68fd43d2ee

```
Hi InventUsPower,
Your files are encrypted by Netwalker.All encrypted files for this computer has extension: .
{id}
```

29aef790399029029e0443455d72a8b928854a0706f2e211ae7a03bba0e3d4f4

```
Hi Bolloré,
Your files are encrypted.All encrypted files for this computer has extension: .{id}
```

Also a version that warns of releasing stolen data:

26dfa8512e892dc8397c4ccbbe10efbcf85029bc2ad7b6b6fe17d26f946a01bb

```
Hi!Your files are encrypted.All encrypted files for this computer has extension: .{id}
–If for some reason you read this text before the encryption ended,this can be understood by the fact that the computer slows down,and your heart rate has increased due to the ability to turn it off,then we recommend that you move away from the computer and accept that you have been compromised.Rebooting/shutdown will cause you to lose files without the possibility of recovery.
–Our  encryption algorithms are very strong and your files are very well protected,the only way to get your files back is to cooperate with us and get the decrypter program.
Do not try to recover your files without a decrypter program, you may damage them and
```

then they will be impossible to recover.
For us this is just business and to prove to you our seriousness, we will decrypt you one file for free.Just open our website, upload the encrypted file and get the decrypted file for free.
Additionally, your data may have been stolen and if you do not cooperate with us, it will become publicly available on our blog.
—
Steps to get access on our website:
1.Download and install tor-browser: **https://torproject.org/**
2.Open our website: {onion1}If the website is not available, open another one: {onion2}
3.Put your personal code in the input form:
{code}

Routines

This is not an exhaustive list by any means but is listed in the most common to least common order and are probably the algorithms I find most often inside of malware.

Encoding:

- Base64

Encryption:

- RC4
- XTEA
- AES
- RC6

Hashing:

- CRC32
- MD5
- SHA256
- SHA512

Compression:

- APLIB
- FLATE
- GZIP
- ZLIB
- LZ

References

1: https://blog.trendmicro.com/trendlabs-security-intelligence/netwalker-fileless-ransomware-injected-via-reflective-loading/

2: https://en.wikipedia.org/wiki/RC4

Like Loading...

Posted in **Malware Analysis**, **Ransomware**, **Reverse Engineering**, **Uncategorized**, **zero2auto** / **3 Comments**

NEXT POST
DEALING WITH OBFUSCATED MACROS, STATICALLY – NANOCORE

3 thoughts on “Netwalker Ransomware – From Static Reverse Engineering to Automatic Extraction”

Pingback: Shadow news

Pingback: Netwalker Ransomware – From Static Reverse Engineering to Automatic Extraction – Zero2Automated Course Blog – Library 8: Operazione Dyn-O-Mite!

Pingback: Zero2Auto – Netwalker detailed walkthrough

Leave a comment