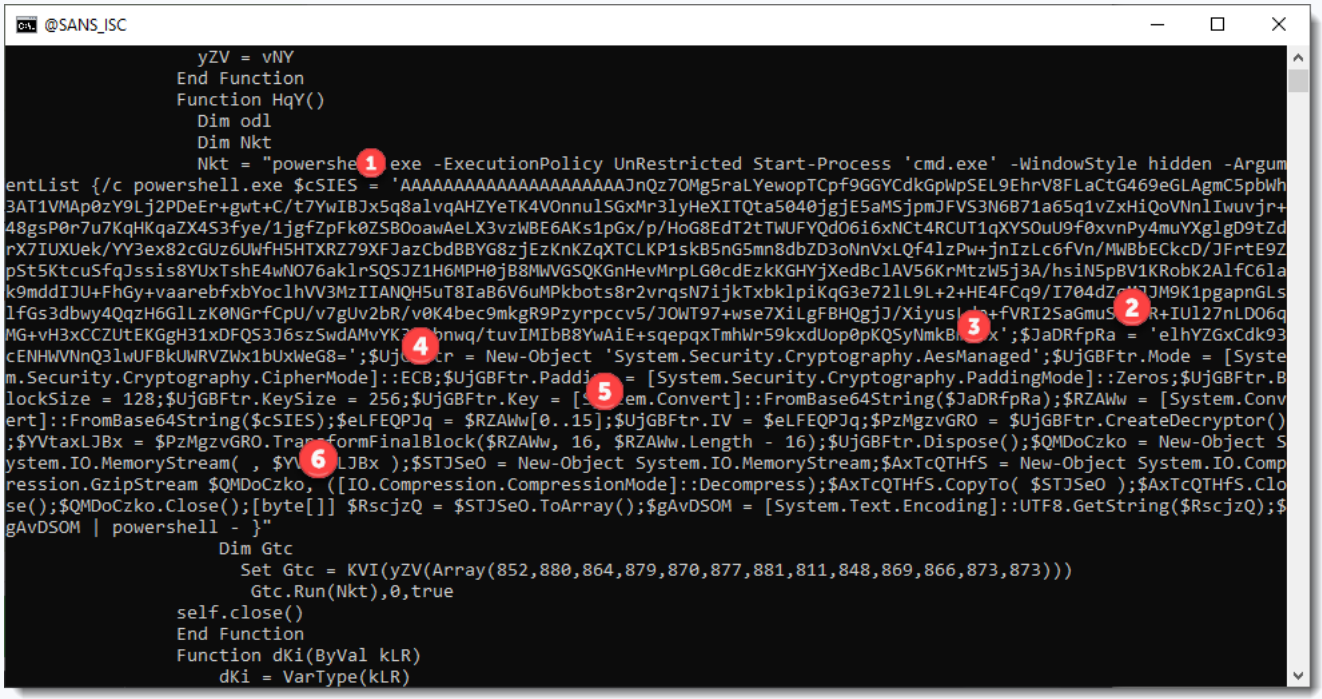
Another Malicious HTA File Analysis - Part 2

Published: 2023-04-10
Last Updated: 2023-04-10 08:13:31 UTC
by Didier Stevens (Version: 1)

0 comment(s)

The first part in this series can be found here. In the first part, we ended with a decoded PowerShell script. We will now start to decrypt the payload found inside this PowerShell script:



1: right under number 1, you can find a variable that contains the BASE64 encoded ciphertext.

2: right under number 2, you can find a variable that contains the BASE64 encoded key.

3: right under number 3, you can see that this is AES encryption

4: right under number 4, you can see that ECB mode is used

5: right under number 5, you can see that the first 16 bytes of the BASE64-decoded payload, is the initialisation vector (although ECB mode does not use an IV)

6: right under number 6, you can see GZip decompression classes & methods.

Thus, to obtain the decoded payload, we need to BASE64-decode it, decrypt it and decompress it.

Let's start with base64dump.py, my tool to extract & decode BASE64 strings:

```
@SANS_ISC
@SANS_ISC C:\Demo>zipdump.py -D 2023-03-24-21-40-33.hta.Loader.6781a85bf0dd90e3ba1390143b17c08244f410dc165fa61bf7d6dacb4
a4c8656.hta.zip | python-per-line.py --split ":" --regex "^a.+=(.+) - &H(.+)$" -j "" "chr(int(oMatch.groups()[0]) - int(
oMatch.groups()[1], 16))" | base64dump.py
ID  Size    Encoded          Decoded          md5 decoded                      Item
--  ----    -------          -------          -----------                      ----
 1:       8 Function         ....*'           6bf5e7e1b6e4e2eabcaa6e0cb385f778
 2:       4 7000             .M4              68c7c74422898cc0568e29826b5807d3
 3:       4 1204             .m8              bc87d8c307d2428fe5601a624ed2b5fb
 4:       4 Then             N..              6d6e4b4289e9e958d0a666af5c4e7e53
 5:       4 Each             ..!              06a9c6900baa0cc413ed91bf1f62263c
 6:       4 Next             5.m              e8f993902d4cf87be7fb493829cded0a
 7:       8 Function         ....*'           6bf5e7e1b6e4e2eabcaa6e0cb385f778
 8:       8 Function         ....*'           6bf5e7e1b6e4e2eabcaa6e0cb385f778
 9:      12 UnRestricted     Rt^...r..        886b522a42c5a2394ddce7de8dc66f20
10:      12 ArgumentList     .......+-        c2908633b9574b1823ff0fa8d8644aa2
11:     896 AAAAAAAAAAAAAAAA ................ 9fcf303bbfb37c2bbdfcdad36842a4fc
12:       8 JaDRfpRa         %..~.Z           efa6a5e1ec0da8c2aa4985e05a96ed20
13:      44 elhYZGxCdk93cENH zXXdlBvOwpCGYSgC 8782afe0415e9dafcf5434ded922c4aa
14:       8 Security         I...+r           6df45e1ac702c5ae1513041339a6b59c
15:      12 Cryptography     .....+j.r        61249ed3cc269a6f0f77cbe8792884ef
16:       4 Mode             2.^              6803512fd5f186567a244c9bf68f451e
17:       8 Security         I...+r           6df45e1ac702c5ae1513041339a6b59c
18:      12 Cryptography     .....+j.r        61249ed3cc269a6f0f77cbe8792884ef
19:       8 Security         I...+r           6df45e1ac702c5ae1513041339a6b59c
20:      12 Cryptography     .....+j.r        61249ed3cc269a6f0f77cbe8792884ef
21:      16 FromBase64String ..&.......).     4cfff9a87d891e1961d358c98991e469
22:       8 JaDRfpRa         %..~.Z           efa6a5e1ec0da8c2aa4985e05a96ed20
23:      16 FromBase64String ..&.......).     4cfff9a87d891e1961d358c98991e469
24:       8 eLFEQPJq         x.D@.j           961c48134354d6d4dcc0764596225ddc
25:       8 eLFEQPJq         x.D@.j           961c48134354d6d4dcc0764596225ddc
```

With option n, we can specify a minimum length for BASE64 strigns, so that we only have the long strings that interest us (ciphertext and key):

```
@SANS_ISC
@SANS_ISC C:\Demo>zipdump.py -D 2023-03-24-21-40-33.hta.Loader.6781a85bf0dd90e3ba1390143b17c08244f410dc165fa61bf7d6dacb4
a4c8656.hta.zip | python-per-line.py --split ":" --regex "^a.+=(.+) - &H(.+)$" -j "" "chr(int(oMatch.groups()[0]) - int(
oMatch.groups()[1], 16))" | base64dump.py -n 16
ID  Size    Encoded          Decoded          md5 decoded                      Item
--  ----    -------          -------          -----------                      ----
 1:     896 AAAAAAAAAAAAAAAA ................ 9fcf303bbfb37c2bbdfcdad36842a4fc
 2:      44 elhYZGxCdk93cENH zXXdlBvOwpCGYSgC 8782afe0415e9dafcf5434ded922c4aa

@SANS_ISC C:\Demo>
```

And then, we use option --jsonoutput to produce JSON output that contains the complete extracted BASE64 strings. This JSON format can be processed by other tools I develop.

```
@SANS_ISC
@SANS_ISC C:\Demo>zipdump.py -D 2023-03-24-21-40-33.hta.Loader.6781a85bf0dd90e3ba1390143b17c08244f410dc165fa61bf7d6dacb4
a4c8656.hta.zip | python-per-line.py --split ":" --regex "^a.+=(.+) - &H(.+)$" -j "" "chr(int(oMatch.groups()[0]) - int(
oMatch.groups()[1], 16))" | base64dump.py -n 16 --jsonoutput
{"version": 2, "id": "didierstevens.com", "type": "content", "fields": ["id", "name", "content"], "items": [{"id": 1, "n
ame": "AAAAAAAAAAAAAAAA", "content": "AAAAAAAAAAAAAAAAAAAAAJnQz7OMg5raLYewopTCpf9GGYCdkGpWpSEL9EhrV8FLaCtG469eGLAgmC5pbW
h3AT1VMAp0zY9Lj2PDeEr+gwt+C/t7YwIBJx5q8alvqAHZYeTK4VOnnulSGxMr3lyHeXITQta5040jgjE5aMSjpmJFVS3N6B71a65q1vZxHiQoVNnlIwuvjr
+48gsP0r7u7KqHKqaZX4S3fye/1jgfZpFk0ZSBOoawAeLX3vzWBE6AKs1pGx/p/HoG8EdT2tTWUFYQdO6i6xNCt4RCUT1qXYSOuU9f0xvnPy4muYXglgD9tZ
drX7IUXUek/YY3ex82cGUz6UWfH5HTXRZ79XFJazCbdBBYG8zjEzKnKZqXTCLKP1skB5nG5mn8dbZD3oNnVxLQf4lzPw+jnIzLc6fVn/MWBbECkcD/JFrtE9
ZpSt5KtcuSfqJssis8YUxTshE4wNO76aklrSQSJZ1H6MPH0jB8MWVGSQKGnHevMrpLG0cdEzkKGHYjXedBclAV56KrMtzW5j3A/hsiN5pBV1KRobK2AlfC6l
ak9mddIJU+FhGy+vaarebfxbYoclhVV3MzIIANQH5uT8IaB6V6uMPkbots8r2vrqsN7ijkTxbklpiKqG3e72ll9L+2+HE4FCq9/I704dZqMJJM9K1pgapnGL
slfGs3dbwy4QqzH6GlLzK0NGrfCpU/v7gUv2bR/v0K4bec9mkgR9Pzyrpccv5/JOWT97+wse7XiLgFBHQgjJ/XiyusLdp+fVRI2SaGmuSx/eR+IUl27nLDO6
qMG+vH3xCCZUtEKGgH31xDFQS3J6szSwdAMvYK3wpbnwq/tuvIMIbB8YwAiE+sqepqxTmhWr59kxdUop0pKQSyNmkBm98x"}, {"id": 2, "name": "elh
YZGxCdk93cENH", "content": "elhYZGxCdk93cENHWVNnQ3lwUFBkUWRVZWx1bUxWeG8="}]}

@SANS_ISC C:\Demo>
```

Here specifically, we will use myjson-transform.py. That is a tool that reads JSON data produced by my tools, and then transforms that data with a user provided Python function.

The script containing this function we will use is the following:

```
from Crypto.Cipher import AES
import gzip

def Transform(items, options):
    ciphertext = items[0]['content']
    key        = items[1]['content']

    iv = ciphertext[:AES.block_size] # unused
    oAES = AES.new(key, AES.MODE_ECB)
    cleartext = oAES.decrypt(ciphertext[AES.block_size:])
    transformed = gzip.decompress(cleartext)
```

```
    items[0]['content'] = transformed
    return transformed
```

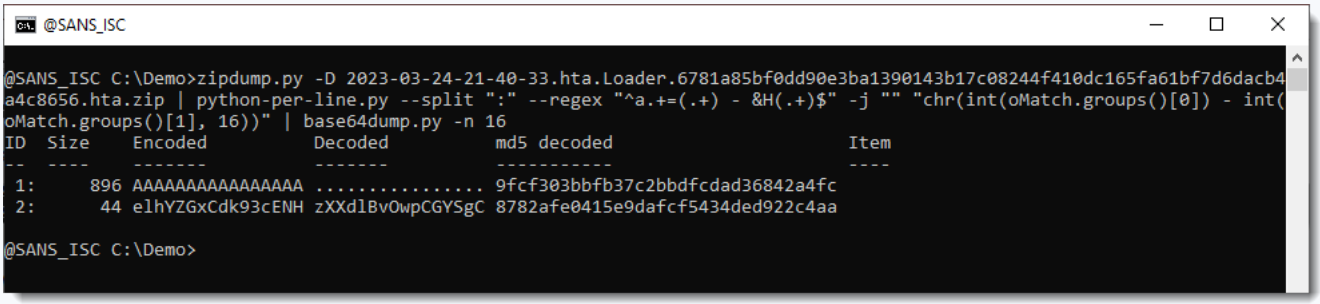
This script defines a function Transform, that is called by my tool myjson-transform.py.

As parameters, it receives a list (items) will the decoded JSON data and it received the options passed on to my tool.

The first 2 lines of the script import modules necessary for AES decryption and GZip decomrpession.

The ciphertext extracted and BASE64-decode by tool base64dump.py from the PowerShell script, is the first item in the list.

The encryption key is the second item in the list:



Each item is a dictionary, and the decoded data can be found under key 'content'.

We assign this to variables ciphertext and key.

We also extract the initialisation vector from the first 16 bytes of the ciphertext into variable iv, but this will not be used as the malware developers use ECB mode, and that mode works without an IV.

Then we create an AES object (oAES) with the key and ECB mode.

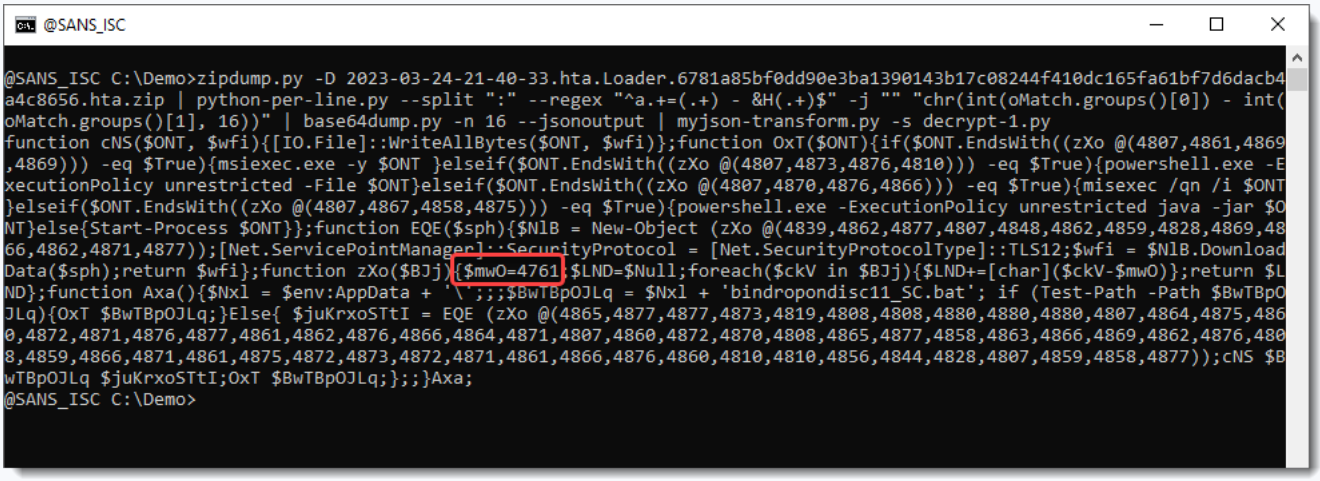
We call method decrypt of object oAES, giving it the ciphertext (excluding the first 16 bytes, e.g., the IV): we store the decrypted data into variable cleartext.

Then we decompress the cleartext and store it into variable transformed.

Finally, we store the decoded payload (variable transformed) back into the items list and we make it the return value of our Transform function.

This summarizes how function Transform works.

We can let myjson-transform.py use this function as follows:

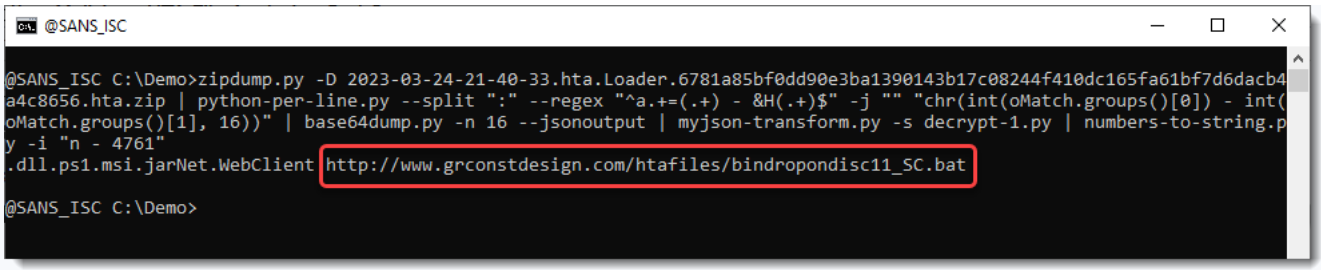


The decrypted payload is another PowerShell script ...

Notice that this PowerShell script contains a series of numbers (4800+), and a single number in the same range: 4761.

As explained in diary entry "Extra: "String Obfuscation: Character Pair Reversal"", this is an encoded payload that can be decoded with my tool numbers-to-string.py.

Like this:



We see some extensions, a .Net object, and a URL.

The file obtained from this URL, is a .bat file and can be found on MalwareBazaar too.

This is the complete command to extract the URL from the HTA file:

```
zipdump.py -D 2023-03-24-21-40-33.hta.Loader.6781a85bf0dd90e3ba1390143b17c08244f410dc165fa61bf7d6dacb4a4c8656.hta.zip | python-per-line.py --split
```

Didier Stevens
Senior handler
Microsoft MVP
blog.DidierStevens.com