ISA 564: Lab 6 – Advanced Malware Answers

(by Sattyik Kundu)

I. Task 1 Answers:

Question 1.1 –Here's the entire *obfuscated* script for reference (a screenshot of the *deobfuscated* script will be shown later):

Lab6.1.ps1

```
1  $p = -join("S3SBwc3m9jbGFpbSB0a33GVlIERyYWdvbiBEa3XNjaXBsZSEgR3GVvYmZ1c2NhdG93yIG9mIG1hb3Hdhcm3Uh"-split "3")
2  $x = $env:comspec[4,15,25]-join''; $y = $env:ProgramFiles[-1,0]-join''
3  sv a ([string]((24,'7B',45,'4E',56,'3A') | foreach {([convert]::toint16(($_.tostring()),16) -as [char])}))
4  sv b ([string]((97,112,112,100,97,116,97,125) | foreach {([convert]::toint16(($_.tostring()),10) -as [char])}))
5  $c = ([char]65+[char]79+[char]99+[char]97+[char]65).replace("A",[char]76)
6  &$y -Path (($a+$c+$b-replace ' ' | & $x)+"\1.out") -Value $p
```

Here are the deobfuscated values I found for each variable:

$p = SSBwcm9jbGFpbSB0aGVlIERyYWdvbiBEaXNjaXBsZSEgRGVvYmZ1c2NhdG9yIG9mIG1hbHdhcmUh

$x = iex

$y = sC

$a = $ { E N V :

$b = a p p d a t a }

$c = LOcaL

Aliases expanded to their full command:

- In Lines 3 and 4, sv → 'Set-Variable'
- In Line 6, the call operator (&) in front of **$y** means to use the value of **$y** as a command. Thus, the **$y** value of "sC" will be treated as an alias that can then be expanded to the full command of 'Set-Content'.
- In Line 6, the call operator (&) in front of **$x** will *also* make the value of **$x** treated as a command. Then the **$x** value of "iex" will be treated as an alias which can then be expanded into the full command of 'Invoke-Expression'.

With all lines and variables replaced and deobfuscated, the true executed command of the last line is:

Set-Content -Path (({$ENV:LOcalappdata} | & iex) + "\1.out") -Value "SSBwcm9jbGFpbSB0aGVlIERyYWdvbiBEaXNjaXBsZSEgRGVvYmZ1c2NhdG9yIG9mIG1hbHdhcmUh"

OR (if the environment variable $ENV from *my* local computer is replaced with the full path)

Set-Content -Path "C:\Users\Sattyik Kundu\AppData\Local\1.out" -Value "SSBwcm9jbGFpbSB0aGVlIERyYWdvbiBEaXNjaXBsZSEgRGVvYmZ1c2NhdG9yIG9mIG1hbHdhcmUh"

Here is the screenshot showing *my* deobfuscated version of the script:

Lab6.1_deobfuscated.ps1

```
1  $p = "SSBwcm9jbGFpbSB0aGVlIERyYWdvbiBEaXNjaXBsZSEgRGVvYmZ1c2NhdG9yIG9mIG1hbHdhcmUh"
2  $x = "iex"; $y = "sC"
3  sv a "{ $ E N V :"
4  sv b "a p p d a t a }"
5  $c = "LOcaL"
6  &$y -Path (($a+$c+$b-replace ' ' | & $x)+"\1.out") -Value $p
```

All the above variables' values have been "deobfuscated"; and only their true values in the form of strings are shown (as defined earlier). Additionally, I left the command aliases as is without expanding them. Finally, I have left the last line unchanged so the deobfuscated variables be utilized.

Finally, using an online base64 decoder (www.base64decode.net), the decoded value of variable $p is:

I proclaim thee Dragon Disciple! Deobfuscator of malware!

Extra Credit – For Lab6.1.ps1, I'll explain how the deobfuscation works in each line of the script.

For line 1 (as shown below):

```
$p = -join("S3SBwc3m9jbGFpbSBOa33GVlIERyYWdvbiBEa3XNjaXBsZSEgR3GVvYmZ1c2NhdG93yIG9mIG1hb3Hdhcm3Uh"-split "3")
```

Here, the "-split" operator first breaks the long string into substrings using 3 as a delimiter; then the "-join" operator recombines all the substrings into one whole string again; except all the 3s have been removed due to them being used as delimiters.

In line 2 (as shown below):

```
$x = $env:comspec[4,15,25]-join''; $y = $env:ProgramFiles[-1,0]-join''
```

In variable **$x**, the $env:comspec is an environment variable that yields the file path of the command prompt executable (cmd.exe). The cmd file path is:

```
PS C:\Users\Sattyik Kundu\Desktop> echo $env:comspec
C:\Windows\system32\cmd.exe
```

The cmd.exe file path consists of 27 characters with 'C' indexing as 0 while the 'e' at the end is indexed as 26. The "[4,15,25]" part of the **$x** consist of indexes of characters from the above file path; which respectively refer to characters 'i', 'e', and 'x'. Lastly, the "-join ' '" operator combines the characters so **$x**="iex".

Then in variable **$y**, the $env:ProgramFiles environment variable yields the file path of the 'ProgramFiles' folder in Windows. The file path is:

```
PS C:\Users\Sattyik Kundu\Desktop\Lab6> $env:ProgramFiles
C:\Program Files
```

The "[-1,0]" part of **$y** also refers to the indexes of characters in the above file path. The -1 index loops to the end of the file path and points at 's'; whilst 0 index points to 'C' at the beginning. Finally, the "-join ' '" operator combines both characters to make it so **$y**="sC".

In line 3 (as shown below):

```
sv a ([string]((24,'7B',45,'4E',56,'3A') | foreach {([convert]::toint16(($_.tostring()),16) -as [char])}))
```

First, 'sv' is an alias which refers to the command 'Set-Variable' (as shown earlier). The 'foreach' loop iterates through each hex string from [24,'7B',45,'4E',56,'3A'] and converts each of them to ASCII characters which are represented with a 16-bit output (this is due to the 'toint16()' function shown above in line 3 and later line 4).

To explain the significance of 16 bits, first note that each hex string consists of 2 hex characters (4-bits each) for a total of 8-bits (1 byte). An ASCII character is also made of 1 byte; but the 16-bit output space enables 2 bytes (i.e. 2 ASCII characters) as output. As earlier determined, the value of variable **$a** is:

```
PS C:\> ([string]((24,'7B',45,'4E',56,'3A') | foreach {([convert]::toint16(($_.tostring()),16) -as [char])}))
$ { E N V :
```

With 2 bytes as an output, each 'foreach' loop iteration outputs an ASCII value (converted from the corresponding hex string) into the 1st byte; and the leftover 2nd byte of the 16-bit output would consist of a blank space. Hence, this explains why each ASCII value is followed by a space (except for the last character since spaces are automatically trimmed at the end). The content of **a**="$ { E N V :".

In line 4 (as shown below):

```
sv b ([string]((97,112,112,100,97,116,97,125) | foreach {([convert]::toint16(($_.tostring()),10) -as [char])}))
```

Just like in Line 3, 'sv' is an alias which refers to the command 'Set-Variable'. This time, the 'foreach' loop iterates through *decimal* (instead of hex) values from the array [97,112,112,100,97,116,97,125] and converts each of them to a 16-bit output consisting of ASCII characters. As found earlier, variable **$b** is:

```
PS C:\> ([string]((97,112,112,100,97,116,97,125) | foreach {([convert]::toint16(($_.tostring()),10) -as [char])}))
a p p d a t a }
```

There are 256 decimal values (numbered 0…255) that can each represent an ASCII value. Hence, 1 byte (8 bits) can represent all possible decimal values that can represent an ASCII value. As stated for line 3, because the output consists of 2 bytes (from 16-bits), the 1st byte of the output is the ASCII character while the leftover 2nd

byte is an empty space. Thus, the output is as shown with spaces following each character (except for the last character since any trailing spaces are automatically trimmed). Finally, vairable **$b**="a p p d a t a }".

In line 5 (as shown below):

```
$c = ([char]65+[char]79+[char]99+[char]97+[char]65).replace("A",[char]76)
```

For variable **$c**, the above are all decimal numbers that will be converted to an ASCII character and then concatenated. Before the ".replace()" function is applied, string is:

```
PS C:\> ([char]65+[char]79+[char]99+[char]97+[char]65)
AOcaA
```

Then when the ".replace("A", [char]76)" is applied, all the "A" letters are replaced with "L" since [char]76='L'. The final value of variable **$c** is $c="LOcaL".

Finally, in line 6 (as shown below):

```
&$y -Path (($a+$c+$b-replace ' ' | & $x)+"\1.out") -Value $p
```

First, the call operator (&) in front of **$y** makes it so the value of $y is treated as a PowerShell command. With $y="sC", "sC" will be treated as an alias in order to represent the PowerShell command 'Set-Content'. With this command, it can be determined that the goal of this line is to set the content of file '1.out' (in a specified file path) with the value of **$p**.

For the file path (which starts with "-Path"), The values of variables **$a**, **$c**, and **$b** are first concatenated. This will yield. Then, the subsequent "-replace ' ' " operation will remove all spaces within the concatenated string.

```
PS C:\Windows\system32> $a+$c+$b
$ { E N V :LOcaLa p p d a t a }

PS C:\Windows\system32> $a+$c+$b-replace ' '
${ENV:LOcaLappdata}
```

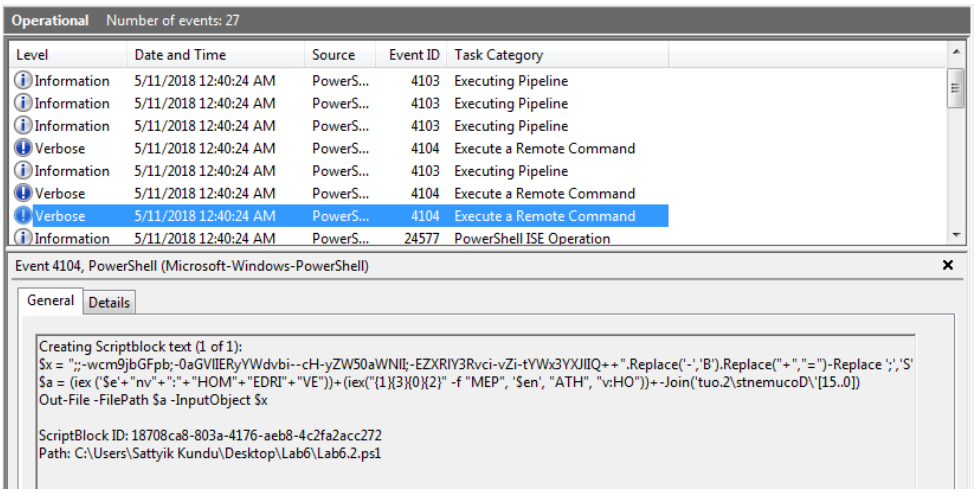
In the next operation consisting of "| & $x", the call operator (&) on variable **$x** means that its value of "iex" will be treated as an alias for the command 'Invoke-Expression'. The '|' symbol will pipeline the concatenated string (without spaces) as a parameter for the 'Invoke-Expression' command. This will output the full file path needed to put the file '1.out' into; which is:

```
PS C:\Windows\system32> ($a+$c+$b-replace ' ' | & $x)+"\1.out"
C:\Users\Sattyik Kundu\AppData\Local\1.out
```

In my local machine, 'Sattyik Kundu' is the folder under 'Users'. In other local machines, the name of the folder under 'Users' will vary. When the file path is fully constructed (as shown above) for file '1.out', the 'Set-Content' command (from alias 'sC') will set the content of file '1.out' with the value of variable **$p**. If '1.out' doesn't already exist, the file will automatically be created at the file path.

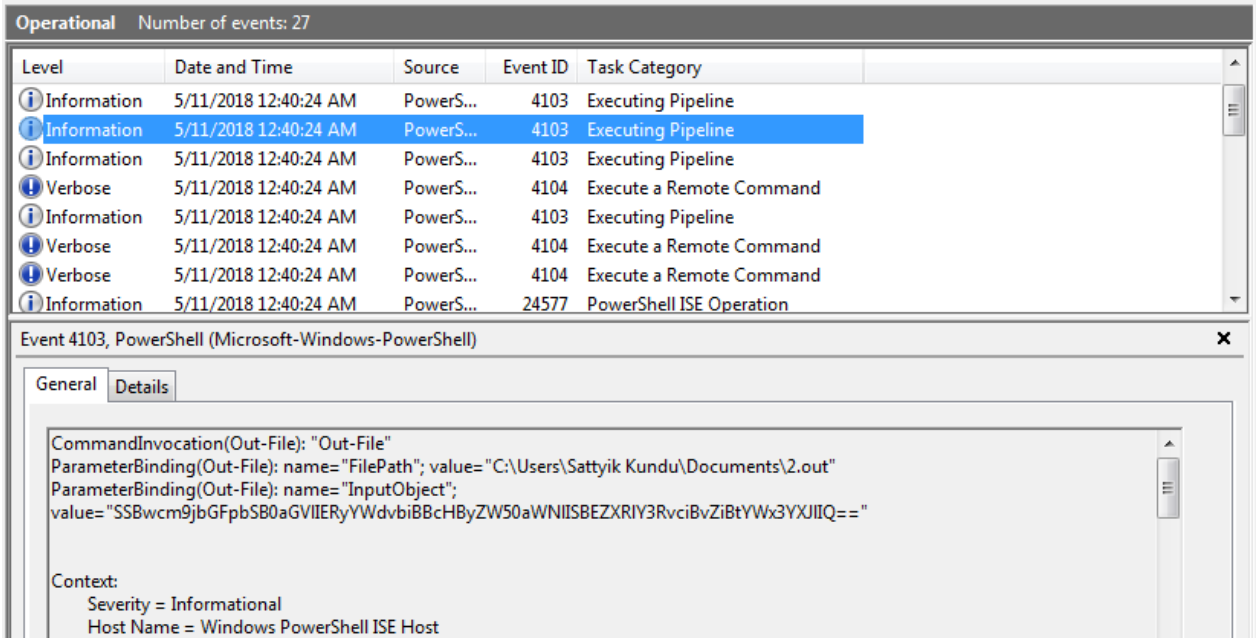
II. Task 2 Answers:

Question 2.1 – First, here is the 4104 event (Execute a Remote Command) that contains the entire script block code:



As expected, the above 4014 event shows the entire script block code; but it is currently obfuscated.

The 4103 event (Executing Pipeline) with the parameters for the command invocation of Out-File:

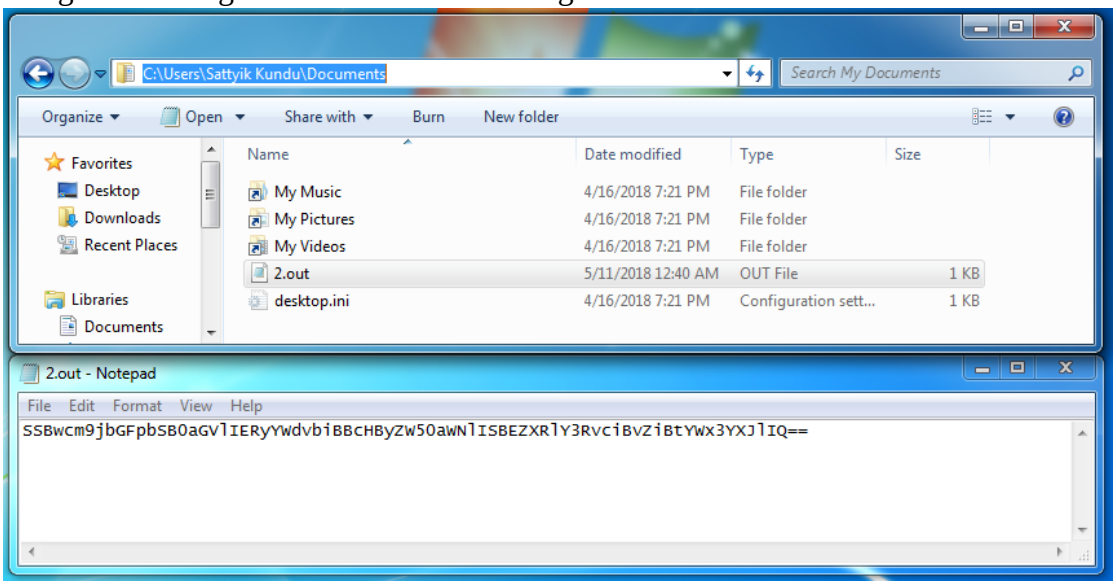


Above shows the CommandInvocation of the 'Out-File' as required of the 4103 event. The Out-File command has two parameters of "FilePath" and "InputObject" as shown above. The Out-File command writes the "InputObject" into the file listed in "FilePath".

The "FilePath" value listed above is deobfuscated value of variable **$a** listed in the previous screenshot. Now that it's deobfuscated, it is shown that the real file path is "C:\Users\Sattyik Kundu\Document\2.out".

Also, the "InputObject" value listed above is the deobfuscated value of variable **$x** listed in the previous screenshot. The obfuscated variable **$x** consisted of a function that made character replacements to an initial long string. The unobfuscated value of variable **$x**, after all character replacements are made, is: "SSBwcm9jbGFpbSB0aGVlIERyYWdvbiBBcHByZW50aWNlISBEZXRlY3RvciBvZiBtYWx3YXJlIQ==".

As stated before, the Out-File command will basically write the above base64 encoded string into file *2.out* from file path "C:\Users\Sattyik Kundu\Document\". Below shows that file 2.out has been successfully created along with having the base64 encoded string written in:



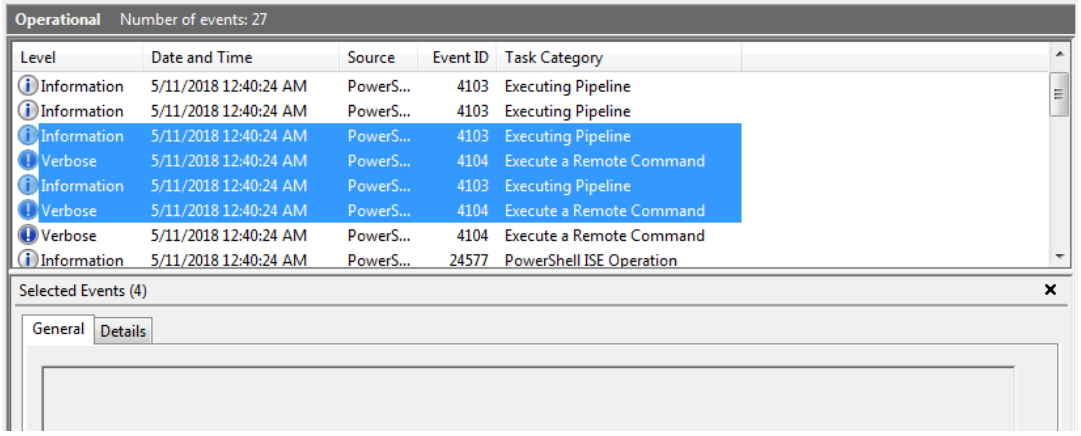
Question 2.2 – The revealed file path in the 4103 event log from Question 2.1 didn't reveal how the path was constructed. To help explain the path construction, the obfuscated variable **$a** as shown in the Lab6.2.ps1 script needs to examined as well as various 4104 and 4013 event logs.

Here is the obfuscated variable **$a** (highlighted) which is a collection of added functions:

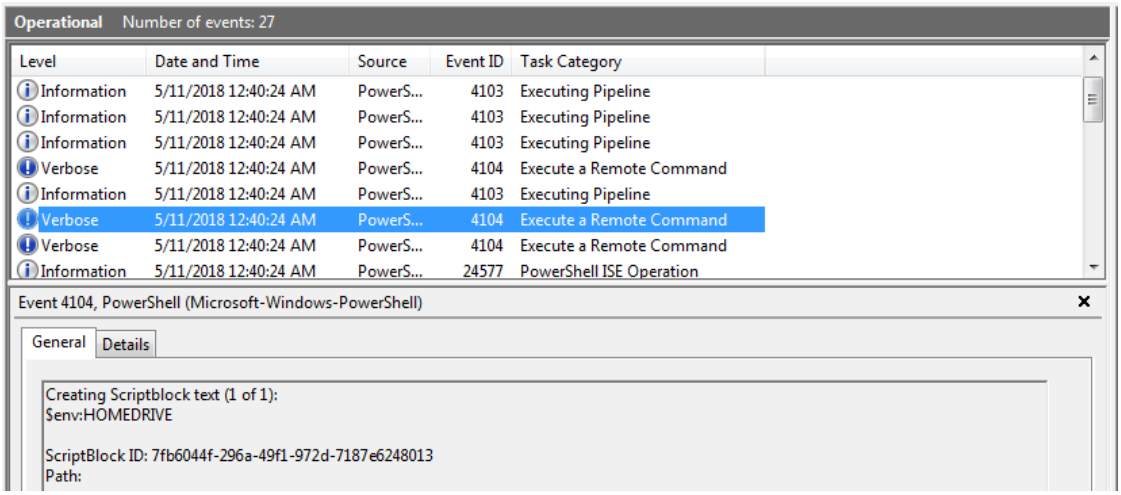
Lab6.2.ps1

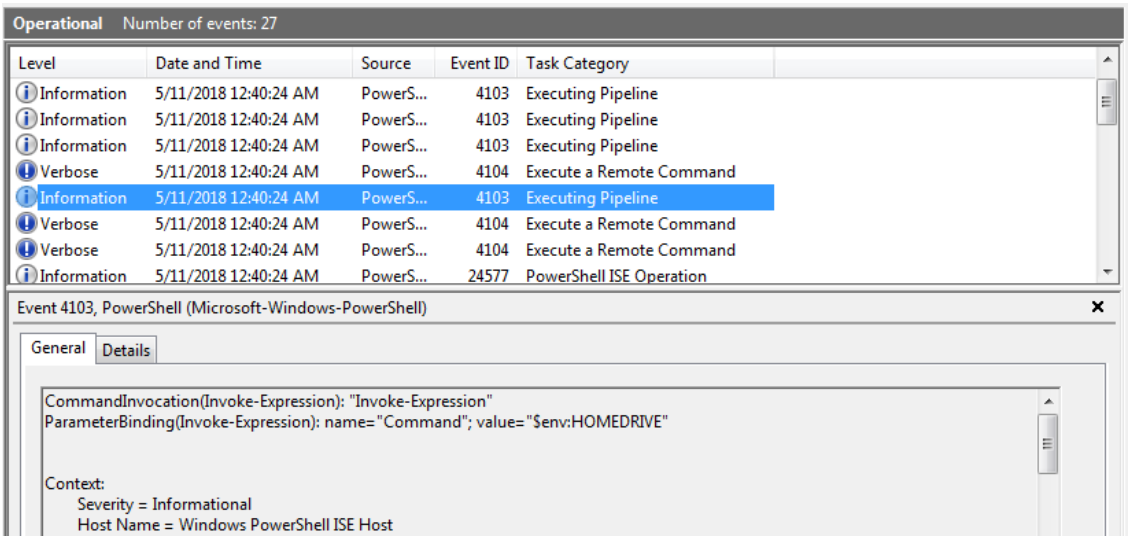
```
1  $x = ";;-wcm9jbGFpb;-0aGVlIERyYWdvbi--cH-yZW50aWNlI;-EZXRlY3Rvci-vZi-tYWx3YXJlIQ++".Replace('-','B').Replace("+","=")-Replace ';',
2  $a = (iex ('$e'+"nv"+":"+"HOM"+"EDRI"+"VE"))+(iex("{1}{3}{0}{2}" -f "MEP", '$en', "ATH", "v:HO"))+-Join('tuo.2\stnemucoD\'[15..0])
3  Out-File -FilePath $a -InputObject $x
```

Also, here are the 4104 and 4013 event logs that need to be examined in order to help detail path construction:



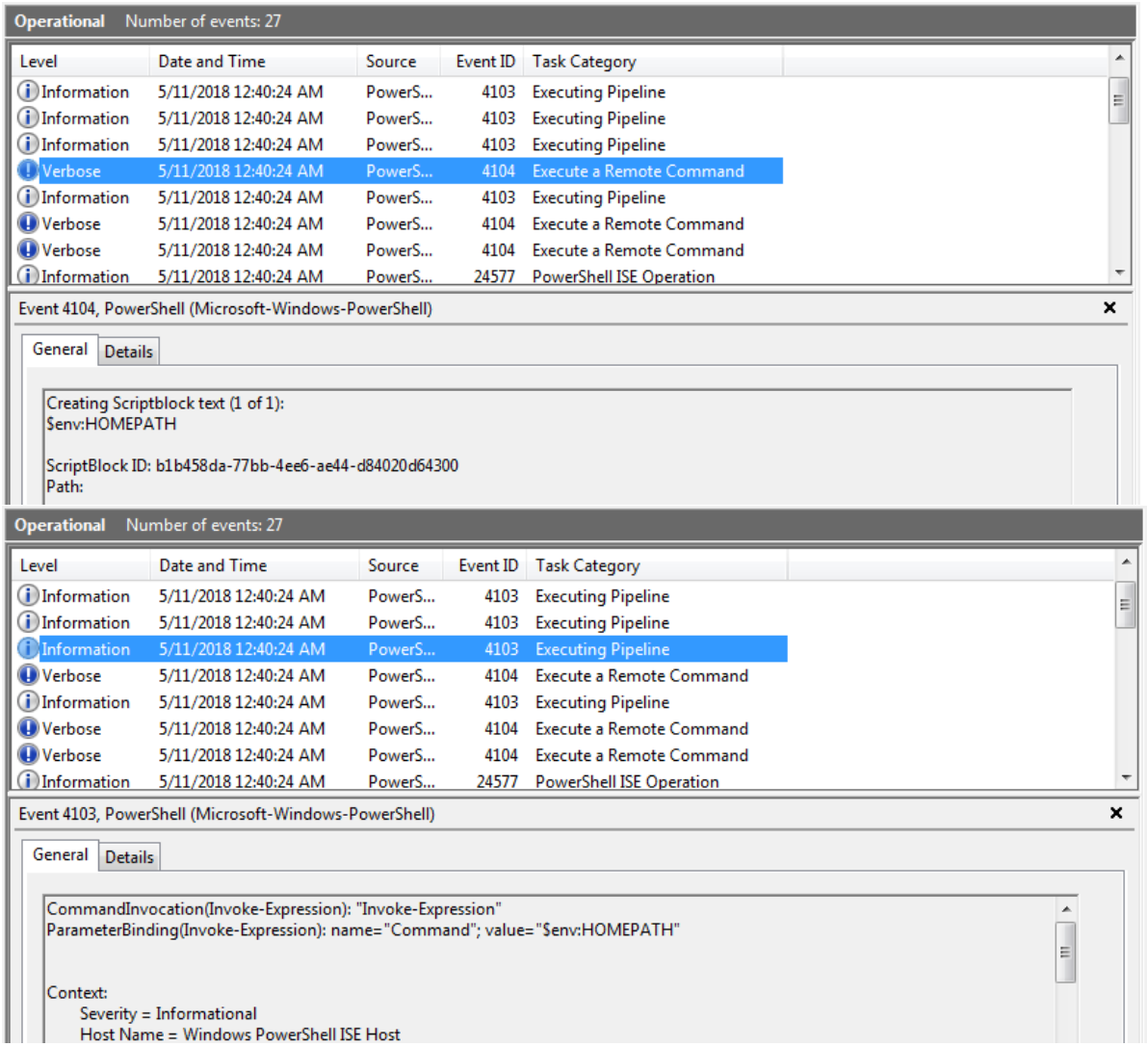
First, I'll start with the bottom 2 logs from the above highlighted logs:





As shown above, these 1st two 4104 and 4103 event logs deal with getting the environment variable of **$env:HOMEDRIVE**. The above 2nd screenshot (4103 event) shows that the "Invoke-Expression" command uses $env:HOMEDRIVE as a parameter to get the path segment value from $env:HOMEDRIVE.

Next, here are the screenshots of the top 2 logs of the earlier shown four highlighted event logs:



Looking closely, these last two event logs deal with the variable **$env:HOMEPATH** (not to be confused with $env:HOMEDRIVE). Similar to before, the 2nd screenshot shows that the "Invoke-Expression" command uses the $env:HOMEPATH as a parameter to get the path segment value from $env:HOMEPATH.

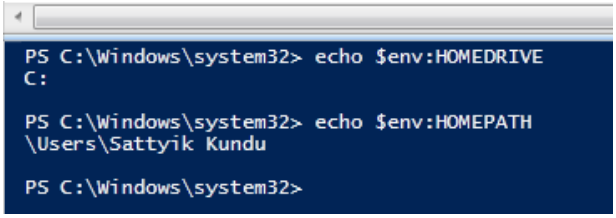
During the deobfuscation of variable **$a**, the path segment values of $env:HOMEDRIVE and $env:HOMEPATH are used to *respectively* replace the two terms in $a that contain "iex" (found in line 2).

Lab6.2.ps1

```
1  $x = ";;-wcm9jbGFpb;-0aGVlIERyYWdvbi--cH-yZW50aWNlI;-EZXRlY3Rvci-vZi-tYWx3YXJlIQ++".Replace('-','B').Replace("+","=")-Replace ';',
2  $a = (iex ('$e'+"nv"+":"+"HOM"+"EDRI"+"VE"))+(iex("{1}{3}{0}{2}" -f "MEP", '$en', "ATH", "v:HO"))+-Join('tuo.2\stnemucoD\'[15..0])
3  Out-File -FilePath $a -InputObject $x
```

The last term in $a, which starts with "+-Join", is basically '\Documents\2.out' written backwards; it is appended after the previous two terms which contain "iex"; which are again represented by variables $env:HOMEDRIVE and $env:HOMEPATH respectively.

To determine what $env:HOMEDRIVE and $env:HOMEPATH actually are, the PowerShell cmd can be used:



At this point, it is known that **$a** = **$env:HOMEDRIVE** + **$env:HOMEPATH** + '**\Document\2.out**'. With the variables replaced, $a = 'C:' + '\User\Sattyik Kundu' + '\Documents\2.out'.

The full constructed file path of **$a** is "C:\User\Sattyik Kundu\Documents\2.out" which is EXACTLY as found in Question2.1.

Finally, there is the base64 string from variable **$x** that was written into the 2.out file. Using an online base64 decoder (www.base64decode.net), the decoded string is:
I proclaim thee Dragon Apprentice! Detector of malware!