Files

f4b848f

Go to file

- .github
- examples
  - Get-GPPPassword.py
  - GetADUsers.py
  - GetNPUsers.py
  - GetUserSPNs.py
  - addcomputer.py
  - atexec.py
  - dcomexec.py
  - dpapi.py
  - esentutl.py
  - exchanger.py
  - findDelegation.py
  - getArch.py
  - getPac.py
  - getST.py
  - getTGT.py
  - goldenPac.py
  - karmaSMB.py
  - keylistattack.py
  - kintercept.py
  - lookupsid.py
  - machine_role.py
  - mimikatz.py
  - mqtt_check.py
  - mssqlclient.py
  - mssqlinstance.py
  - netview.py
  - nmapAnswerMachine.py
  - ntfs-read.py
  - ntlmrelayx.py
  - ping.py
  - ping6.py
  - psexec.py
  - raiseChild.py
  - rbcd.py

impacket / examples / **wmiexec.py**

gabrielg5 Update file banners to reflect Fortra ownership 8799a1a · 2 years ago History

Code Blame Executable File · 473 lines (402 loc) · 19.1 KB Raw

```python
#!/usr/bin/env python
# Impacket - Collection of Python classes for working with network protocols.
#
# Copyright (C) 2022 Fortra. All rights reserved.
#
# This software is provided under a slightly modified version
# of the Apache Software License. See the accompanying LICENSE file
# for more information.
#
# Description:
#   A similar approach to smbexec but executing commands through WMI.
#   Main advantage here is it runs under the user (has to be Admin)
#   account, not SYSTEM, plus, it doesn't generate noisy messages
#   in the event log that smbexec.py does when creating a service.
#   Drawback is it needs DCOM, hence, I have to be able to access
#   DCOM ports at the target machine.
#
# Author:
#   beto (@agsolino)
#
# Reference for:
#   DCOM
#

from __future__ import division
from __future__ import print_function
import sys
import os
import cmd
import argparse
import time
import logging
import ntpath
from base64 import b64encode

from impacket.examples import logger
from impacket.examples.utils import parse_target
from impacket import version
from impacket.smbconnection import SMBConnection, SMB_DIALECT, SMB2_DIALECT_002, SMB2_D
from impacket.dcerpc.v5.dcomrt import DCOMConnection, COMVERSION
from impacket.dcerpc.v5.dcom import wmi
from impacket.dcerpc.v5.dtypes import NULL
from impacket.krb5.keytab import Keytab
from six import PY2

OUTPUT_FILENAME = '__' + str(time.time())
CODEC = sys.stdout.encoding


class WMIEXEC:
    def __init__(self, command='', username='', password='', domain='', hashes=None, ae
                 noOutput=False, doKerberos=False, kdcHost=None, shell_type=None):
        self.__command = command
        self.__username = username
        self.__password = password
        self.__domain = domain
        self.__lmhash = ''
```

- rdp_check.py
- reg.py
- registry-read.py
- rpcdump.py
- rpcmap.py
- sambaPipe.py

```
        self.__lmhash =
        self.__nthash = ''
        self.__aesKey = aesKey
        self.__share = share
        self.__noOutput = noOutput
        self.__doKerberos = doKerberos
        self.__kdcHost = kdcHost
        self.__shell_type = shell_type
        self.shell = None
        if hashes is not None:
            self.__lmhash, self.__nthash = hashes.split(':')

    def run(self, addr, silentCommand=False):
        if self.__noOutput is False and silentCommand is False:
            smbConnection = SMBConnection(addr, addr)
            if self.__doKerberos is False:
                smbConnection.login(self.__username, self.__password, self.__domain, se
            else:
                smbConnection.kerberosLogin(self.__username, self.__password, self.__do
                                            self.__nthash, self.__aesKey, kdcHost=self.

            dialect = smbConnection.getDialect()
            if dialect == SMB_DIALECT:
                logging.info("SMBv1 dialect used")
            elif dialect == SMB2_DIALECT_002:
                logging.info("SMBv2.0 dialect used")
            elif dialect == SMB2_DIALECT_21:
                logging.info("SMBv2.1 dialect used")
            else:
                logging.info("SMBv3.0 dialect used")
        else:
            smbConnection = None

        dcom = DCOMConnection(addr, self.__username, self.__password, self.__domain, se
                              self.__aesKey, oxidResolver=True, doKerberos=self.__doKer
        try:
            iInterface = dcom.CoCreateInstanceEx(wmi.CLSID_WbemLevel1Login, wmi.IID_IWb
            iWbemLevel1Login = wmi.IWbemLevel1Login(iInterface)
            iWbemServices = iWbemLevel1Login.NTLMLogin('//./root/cimv2', NULL, NULL)
            iWbemLevel1Login.RemRelease()

            win32Process, _ = iWbemServices.GetObject('Win32_Process')

            self.shell = RemoteShell(self.__share, win32Process, smbConnection, self.__
            if self.__command != ' ':
                self.shell.onecmd(self.__command)
            else:
                self.shell.cmdloop()
        except  (Exception, KeyboardInterrupt) as e:
            if logging.getLogger().level == logging.DEBUG:
                import traceback
                traceback.print_exc()
            logging.error(str(e))
            if smbConnection is not None:
                smbConnection.logoff()
            dcom.disconnect()
            sys.stdout.flush()
            sys.exit(1)

        if smbConnection is not None:
            smbConnection.logoff()
        dcom.disconnect()
```

```
    if len(sys.argv) == 1:
        parser.print_help()
        sys.exit(1)

    options = parser.parse_args()

    # Init the example's logger theme
    logger.init(options.ts)

    if options.codec is not None:
        CODEC = options.codec
    else:
        if CODEC is None:
            CODEC = 'utf-8'

    if ' '.join(options.command) == ' ' and options.nooutput is True:
        logging.error("-nooutput switch and interactive shell not supported")
        sys.exit(1)
    if options.silentcommand and options.command == ' ':
        logging.error("-silentcommand switch and interactive shell not supported")
        sys.exit(1)

    if options.debug is True:
        logging.getLogger().setLevel(logging.DEBUG)
        # Print the Library's installation path
        logging.debug(version.getInstallationPath())
    else:
        logging.getLogger().setLevel(logging.INFO)

    if options.com_version is not None:
        try:
```

```python
            try:
                major_version, minor_version = options.com_version.split('.')
                COMVERSION.set_default_version(int(major_version), int(minor_version))
            except Exception:
                logging.error("Wrong COMVERSION format, use dot separated integers e.g. \"5
                sys.exit(1)

        domain, username, password, address = parse_target(options.target)

        try:
            if options.A is not None:
                (domain, username, password) = load_smbclient_auth_file(options.A)
                logging.debug('loaded smbclient auth file: domain=%s, username=%s, password
                repr(domain), repr(username), repr(password)))

            if domain is None:
                domain = ''

            if options.keytab is not None:
                Keytab.loadKeysFromKeytab(options.keytab, username, domain, options)
                options.k = True

            if password == '' and username != '' and options.hashes is None and options.no_
                from getpass import getpass

                password = getpass("Password:")

            if options.aesKey is not None:
                options.k = True

            executer = WMIEXEC(' '.join(options.command), username, password, domain, optio
                               options.share, options.nooutput, options.k, options.dc_ip, o
            executer.run(address, options.silentcommand)
        except KeyboardInterrupt as e:
            logging.error(str(e))
        except Exception as e:
            if logging.getLogger().level == logging.DEBUG:
                import traceback

                traceback.print_exc()
            logging.error(str(e))
            sys.exit(1)

        sys.exit(0)
```