Sign in

fortra / impacket Public

Notifications Fork 3.6k Star 13.5k

Code Issues 196 Pull requests 151 Actions Projects Security Insights

impacket / examples / dcomexec.py

Executable File · 657 lines (574 loc) · 28.2 KB

Code Blame Raw

```
#!/usr/bin/env python
# Impacket - Collection of Python classes for working with network protocols.
#
# SECUREAUTH LABS. Copyright (C) 2021 SecureAuth Corporation. 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 psexec but executing commands through DCOM.
#   You can select different objects to be used to execute the commands.
#   Currently supported objects are:
#     1. MMC20.Application (49B2791A-B1AE-4C90-9B8E-E860BA07F889) - Tested Windows 7, Windows 10, S
#     2. ShellWindows (9BA05972-F6A8-11CF-A442-00A0C90A8F39) - Tested Windows 7, Windows 10, Server
#     3. ShellBrowserWindow (C08AFD90-F2A1-11D1-8455-00A0C91F3880) - Tested Windows 10, Server 2012
#
#   Drawback is it needs DCOM, hence, I have to be able to access
#   DCOM ports at the target machine.
#
#   Original discovery by Matt Nelson (@enigma0x3):
#   https://enigma0x3.net/2017/01/05/lateral-movement-using-the-mmc20-application-com-object/
#   https://enigma0x3.net/2017/01/23/lateral-movement-via-dcom-round-2/
#
# Author:
#   beto (@agsolino)
#   Marcello (@byt3bl33d3r)
#
# Reference for:
#  DCOM
#
# ToDo:
#   [ ] Kerberos auth not working, invalid_checksum is thrown. Most probably sequence numbers out
#       getInterface() method
#

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

from six import PY2, PY3
from impacket import version
from impacket.dcerpc.v5.dcom.oaut import IID_IDispatch, string_to_bin, IDispatch, DISPPARAMS, DISPA
    VARIANT, VARENUM, DISPATCH_METHOD
from impacket.dcerpc.v5.dcomrt import DCOMConnection, COMVERSION
from impacket.dcerpc.v5.dcomrt import OBJREF, FLAGS_OBJREF_CUSTOM, OBJREF_CUSTOM, OBJREF_HANDLER, \
    OBJREF_EXTENDED, OBJREF_STANDARD, FLAGS_OBJREF_HANDLER, FLAGS_OBJREF_STANDARD, FLAGS_OBJREF_EXT
    IRemUnknown2, INTERFACE
from impacket.dcerpc.v5.dtypes import NULL
from impacket.examples import logger
from impacket.examples.utils import parse_target
from impacket.smbconnection import SMBConnection, SMB_DIALECT, SMB2_DIALECT_002, SMB2_DIALECT_21
from impacket.krb5.keytab import Keytab

OUTPUT_FILENAME = '__' + str(time.time())[:5]
CODEC = sys.stdout.encoding

class DCOMEXEC:
    def __init__(self, command='', username='', password='', domain='', hashes=None, aesKey=None, s
                 noOutput=False, doKerberos=False, kdcHost=None, dcomObject=None, shell_type=None):
        self.__command = command
        self.__username = username
        self.__password = password
        self.__domain = domain
        self.__lmhash = ''
        self.__nthash = ''
        self.__aesKey = aesKey
        self.__share = share
        self.__noOutput = noOutput
        self.__doKerberos = doKerberos
        self.__kdcHost = kdcHost
        self.__dcomObject = dcomObject
        self.__shell_type = shell_type
        self.shell = None
        if hashes is not None:
            self.__lmhash, self.__nthash = hashes.split(':')

    def getInterface(self, interface, resp):
        # Now let's parse the answer and build an Interface instance
        objRefType = OBJREF(b''.join(resp))['flags']
        objRef = None
        if objRefType == FLAGS_OBJREF_CUSTOM:
            objRef = OBJREF_CUSTOM(b''.join(resp))
        elif objRefType == FLAGS_OBJREF_HANDLER:
            objRef = OBJREF_HANDLER(b''.join(resp))
        elif objRefType == FLAGS_OBJREF_STANDARD:
            objRef = OBJREF_STANDARD(b''.join(resp))
        elif objRefType == FLAGS_OBJREF_EXTENDED:
            objRef = OBJREF_EXTENDED(b''.join(resp))
        else:
            logging.error("Unknown OBJREF Type! 0x%x" % objRefType)

        return IRemUnknown2(
            INTERFACE(interface.get_cinstance(), None, interface.get_ipidRemUnknown(), objRef['std'
                      oxid=objRef['std']['oxid'], oid=objRef['std']['oxid'],
                      target=interface.get_target()))

    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, self.__lmhash,
            else:
                smbConnection.kerberosLogin(self.__username, self.__password, self.__domain, self._
                                            self.__nthash, self.__aesKey, kdcHost=self.__kdcHost)

            dialect = smbConnection.getDialect()
            if dialect == SMB_DIALECT:
                logging.info("SMBv1 dialect used")
            elif dialect == SMB2_DIALECT_002:
                logging.info("SMBv2.0 dialect used")
```

```
                        logging.info('SMBv2.0 dialect used')
```

```
                                                                            '(128 or 256 bits)')
    group.add_argument('-dc-ip', action='store',metavar = "ip address",  help='IP Address of the dc
                       'ommited it use the domain part (FQDN) specified in the target parameter')
    group.add_argument('-A', action="store", metavar = "authfile", help="smbclient/mount.cifs-style
                                                                        "See smbclient man page's -
    group.add_argument('-keytab', action="store", help='Read keys for SPN from keytab file')

    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:
                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. \"5.7\"")
                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=%s' % (repr

        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_pass is Fals
            from getpass import getpass
            password = getpass("Password:")

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

        executer = DCOMEXEC(' '.join(options.command), username, password, domain, options.hashes,
                            options.share, options.nooutput, options.k, options.dc_ip, options.obje
        executer.run(address, options.silentcommand)
    except (Exception, KeyboardInterrupt) as e:
        if logging.getLogger().level == logging.DEBUG:
            import traceback
            traceback.print_exc()
        logging.error(str(e))
    sys.exit(0)
```