VIT
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

SCHOOL OF COMPUTER SCIENCE ENGINEERING AND INFORMATION SYSTEMS

Winter Semester 2023-2024

Programme Name and Branch: B. Tech (IT)

Course Name: Information Security Management

Faculty Name: MOHANA PRIYA P

Project Title: Dynamic Server-Side Redirection

Team Details:

OMKAR (21BIT0530)

PRATHAM RAJ SINHA(21BIT0092)

ADITI PATRA(21BIT0125)

MALAY SAHU(21BIT0594)

Abstract:

In the realm of network security management, safeguarding mainframe systems is paramount due to their critical role in housing sensitive data and supporting essential operations within organizations. Traditional security measures often focus on perimeter defines and static redirection techniques, which may not adequately adapt to evolving threats or dynamic network conditions. Therefore, there is a pressing need for dynamic server-side redirection solutions tailored specifically to mainframe protection. By implementing such solutions, organizations can enhance their ability to detect and mitigate various security risks, including unauthorized access attempts, data breaches, and other malicious activities, thereby fortifying the overall security posture of their mainframe environments. Additionally, dynamic redirection can offer flexibility in responding to emerging threats and optimizing network performance, further underscoring its importance in modern network security strategies.

Introduction:

In the realm of network security management, safeguarding mainframe systems is paramount due to their critical role in housing sensitive data and supporting essential operations within organizations. However, traditional security measures often fall short in adapting to evolving threats and dynamic network conditions. To address this gap, dynamic server-side redirection solutions emerge as a vital approach tailored specifically to bolster mainframe security. By enabling real-time detection and mitigation of security risks such as unauthorized access attempts and data breaches, these solutions offer a flexible and proactive defense mechanism. This project explores the implementation of dynamic redirection technologies aimed at fortifying the security posture of mainframe environments, providing insights and recommendations for organizations seeking to enhance their security strategies amidst the ever-evolving cybersecurity landscape.

Scope of the Project:

The scope of the project encompasses the design, development, implementation, and maintenance of a dynamic server-side redirection system tailored for mainframe protection within network security management. This includes conducting a comprehensive assessment of existing mainframe security measures and identifying vulnerabilities and shortcomings. The project involves the creation of a robust architecture capable of dynamically redirecting network traffic to safeguard mainframe resources against various threats, such as unauthorized access attempts, malware, and data breaches. Additionally, the scope encompasses the integration of real-time monitoring and analytics capabilities to detect and respond to security incidents promptly. Furthermore, the project will entail thorough testing, validation, and optimization of the redirection system to ensure its effectiveness and reliability in diverse network environments. Ongoing maintenance and updates to adapt to evolving security threats and technological advancements are also part of the project scope. Finally, documentation and training materials will be provided to support the successful deployment and operation of the dynamic server-side redirection solution within network security management frameworks.

Experimental Setup/Project KIT:

- **Laptop**
- **Python (programming language)**
- **Internet (Hotspot)**

Architecture:

The architecture of Dynamic Server-Side Redirection is characterized by a structured arrangement of components working cohesively to manage incoming requests and efficiently direct traffic to appropriate destination URLs.

At the core of the architecture lies the Redirection Engine, responsible for processing incoming requests and determining the optimal redirection strategy based on predefined rules or dynamic conditions. The Redirection Engine interfaces with a Database or Configuration Repository, housing redirection rules and settings, facilitating centralized management and flexible rule modification.

Additionally, the architecture encompasses several modules:

1. Request Parser: Responsible for parsing incoming request data to extract relevant information such as user agent, referrer, and URL parameters.

2. Transformation Module: Applies transformation rules to URLs, enabling dynamic modification based on contextual factors such as user attributes or device type.

3. Logging Module: Records redirection events for analytical purposes, capturing data such as timestamp, source URL, destination URL, and user agent information.

4. Caching Layer: Optionally stores frequently accessed redirection mappings to enhance performance by reducing the need for repeated rule evaluation.

The modular architecture ensures scalability, flexibility, and maintainability, facilitating seamless integration with existing web infrastructures and enabling customization to suit specific application requirements.

Modules:

- Host: This module represents the mainframe system or systems that are being protected. It encompasses the hardware, software, and data housed within the mainframe environment.
- Server: The server module functions as the central component responsible for implementing dynamic server-side redirection. It manages the redirection of incoming network traffic to different destinations based on predefined security policies and threat detection mechanisms.
- Dummy Host: The dummy host module serves as a decoy or diversionary target within the mainframe environment. It is designed to mimic legitimate services or resources, diverting potential attackers away from critical assets and providing an additional layer of defense.
- Attacker: The attacker module represents external or internal threats targeting the mainframe environment. This could include malicious actors attempting unauthorized access, data breaches, or other nefarious activities aimed at compromising the security and integrity of the mainframe systems.

These modules interact dynamically within the mainframe security framework to detect and respond to security threats in real-time, thereby enhancing the overall security posture of the mainframe environment. The server module orchestrates the redirection of network traffic, while the dummy host module serves as a deterrent to potential attackers. The host module represents the assets being protected, while the attacker module represents the potential threats against those assets.

Novelty:

- Dynamic Redirection: Unlike traditional security measures that rely on static redirection techniques, this project introduces dynamic server-side redirection, allowing for real-time adaptation to evolving threats and network conditions. This dynamic approach enables more effective detection and mitigation of security risks.
- Tailored for Mainframe Security: While dynamic redirection techniques are increasingly used in network security, this project specifically tailors these techniques to the unique requirements and challenges of securing mainframe environments. Mainframes often house sensitive data and support critical operations, making them high-value targets for attackers. By focusing on mainframe security, this project addresses a specific and pressing need within the cybersecurity landscape.
- Decoy Infrastructure (Dummy Host): Introducing a dummy host module as part of the security strategy adds an innovative layer of defense. By creating decoy targets within the mainframe environment, potential attackers are misled and diverted away from critical assets, reducing the risk of successful breaches or compromises.
- Real-time Threat Response: The project emphasizes real-time threat detection and response, enabled by dynamic redirection capabilities. By continuously monitoring network traffic and security events, the system can quickly identify and mitigate security threats as they arise, minimizing the potential impact on mainframe operations and data integrity.
- Integration and Adaptability: The project highlights the importance of seamless integration with existing mainframe systems and security infrastructure. By ensuring compatibility and adaptability, organizations can deploy and maintain the solution effectively within their existing environments, maximizing its effectiveness in enhancing mainframe security.

Overall, the novelty of this project lies in its innovative approach to addressing the specific challenges of mainframe security through dynamic server-side redirection, tailored defense mechanisms, and real-time threat response capabilities.

CODE IMPLEMENTATION:

server.py:

```
import sys
import socket
import socket

def redirect_connection(client_socket):
    # Implement redirection logic here
    redirectionIP = input('Enter Redirection Address')
    redirected_address = redirectionIP  # Example redirected address
    redirected_port = 12346  # Example redirected port
    redirected_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    redirected_socket.connect((redirected_address, redirected_port))
    print("Connection redirected to:", redirected_address)
    data = redirected_socket.recv(4096).decode()
    client_socket.send(data.encode())


def main():

    host = socket.gethostbyname(socket.gethostname())
    port = 12340
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind((host, port))
    server_socket.listen(5)
    mainframeIP = input("Enter the MainFrame IP Address ->")
    print("Server listening on", host, "port", port)

    while True:
        client_socket, addr = server_socket.accept()
        print("Connection from", addr)
        data = client_socket.recv(1024).decode()

        if (data == mainframeIP):
            redirect_connection(client_socket)
        else:
            print("Bypass unsuccessful")




if __name__ == "__main__":

    main()
```

Dummy_Host:

```
import socket
import subprocess

def main():
    host = socket.gethostbyname(socket.gethostname())
    port = 12346
    host_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    host_socket.bind((host, port))
    host_socket.listen(5)
    print("Host listening on", host, "port", port)

    client_socket, addr = host_socket.accept()
    print("Connection from", addr)
    client_socket.send(host.encode())
    subprocess.run(['python', 'Reciever.py'])


def start_program_with_ip(ip_address):
    try:
        # Replace 'program_to_start.py' with the name of your Python program
        # that you want to start
        subprocess.Popen(['python', 'Reciever.py', ip_address])
        print("Started program with IP:", ip_address)
    except Exception as e:
        print("Error starting program:", e)

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

Mainframe.py:

```
import socket


def main():
    host = socket.gethostbyname(socket.gethostname())
    port = 12346
    host_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    host_socket.bind((host, port))
    host_socket.listen(5)
    print("Host listening on", host, "port", port)

    client_socket, addr = host_socket.accept()
    print("Connection from", addr)
    client_socket.send("Welcome to the host!".encode())

    client_socket.close()


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

Reciever.py:

```
import tkinter as tk
import socket
import threading
import sys

class ReceiverApp:
    def __init__(self, master):
        self.master = master
        self.master.title("Dummy Host Receiver")

        self.chat_label = tk.Label(master, text="Chat:")
        self.chat_label.pack()

        self.chat_text = tk.Text(master, height=20, width=50)
        self.chat_text.pack()
        self.chat_text.config(state=tk.DISABLED)

        self.message_label = tk.Label(master, text="Message:")
        self.message_label.pack()

        self.message_entry = tk.Entry(master, width=50)
        self.message_entry.pack()

        self.send_button = tk.Button(master, text="Send", command=self.send_message)
        self.send_button.pack()

        self.bye_button = tk.Button(master, text="Bye", command=self.close_connection)
        self.bye_button.pack()

        self.server_ip = socket.gethostbyname(socket.gethostname())  # Change this to the
receiver's IP address
        self.server_port = 12345

        self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.socket.bind((self.server_ip, self.server_port))
        self.socket.listen(1)

        self.client_socket, self.client_address = self.socket.accept()

        receive_thread = threading.Thread(target=self.receive_messages)
        receive_thread.start()

    def receive_messages(self):
        while True:
            try:
                message = self.client_socket.recv(1024).decode('utf-8')
                self.chat_text.config(state=tk.NORMAL)
                self.chat_text.insert(tk.END, message + "\n")
                self.chat_text.config(state=tk.DISABLED)
                self.chat_text.see(tk.END)
            except ConnectionAbortedError:
                break
```

```
    def send_message(self):
        message = self.message_entry.get()
        self.client_socket.send(message.encode('utf-8'))
        self.message_entry.delete(0, tk.END)

    def close_connection(self):
        self.client_socket.close()
        sys.exit()

def start_receiver():
    root = tk.Tk()
    app = ReceiverApp(root)
    root.mainloop()

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

Sender.py:

```
import tkinter as tk
import socket
import threading
import sys

class SenderApp:
    def __init__(self, master):

        ip_address = sys.argv[1]
        self.master = master
        self.master.title("Sender")

        self.chat_label = tk.Label(master, text="Chat:")
        self.chat_label.pack()

        self.chat_text = tk.Text(master, height=20, width=50)
        self.chat_text.pack()
        self.chat_text.config(state=tk.DISABLED)

        self.message_label = tk.Label(master, text="Message:")
        self.message_label.pack()

        self.message_entry = tk.Entry(master, width=50)
        self.message_entry.pack()

        self.send_button = tk.Button(master, text="Send", command=self.send_message)
        self.send_button.pack()

        self.bye_button = tk.Button(master, text="Bye", command=self.close_connection)
        self.bye_button.pack()
```

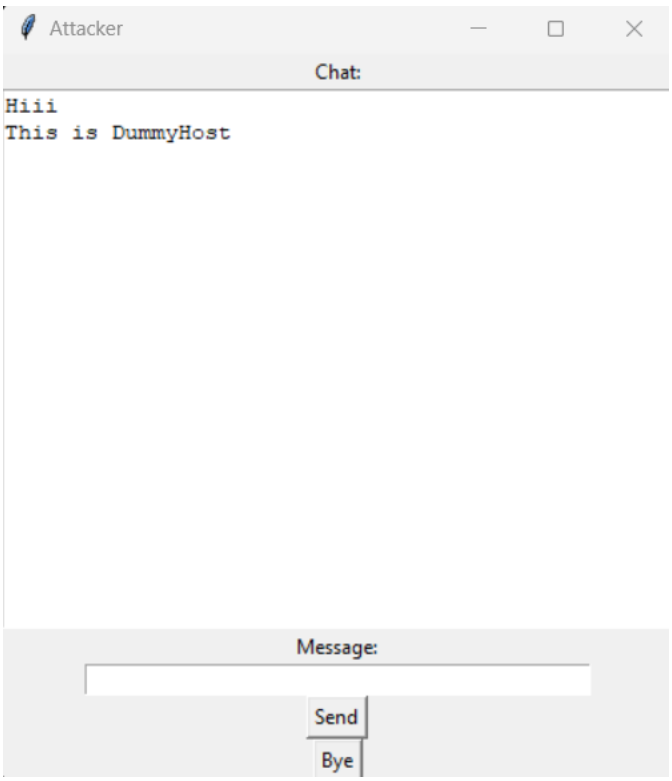
```
        self.server_ip = ip_address  # Change this to the receiver's IP address
        self.server_port = 12345

        self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.socket.connect((self.server_ip, self.server_port))

        receive_thread = threading.Thread(target=self.receive_messages)
        receive_thread.start()

    def send_message(self):
        message = self.message_entry.get()
        self.socket.send(message.encode('utf-8'))
        self.message_entry.delete(0, tk.END)

    def receive_messages(self):
        while True:
            try:
                message = self.socket.recv(1024).decode('utf-8')
                self.chat_text.config(state=tk.NORMAL)
                self.chat_text.insert(tk.END, message + "\n")
                self.chat_text.config(state=tk.DISABLED)
                self.chat_text.see(tk.END)
            except ConnectionAbortedError:
                break

    def close_connection(self):
        self.socket.close()
        sys.exit()

def start_sender():
    root = tk.Tk()
    app = SenderApp(root)
    root.mainloop()

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

attacker.py:

```
import subprocess
import socket
import subprocess

server_ip = input('Enter Server IP Address')

def scan_local_network():
    local_ips = []

    scanningIP = server_ip.split(".")[2]
    IP = "192.168."+scanningIP+".0/24"

    try:
        nmap_output = subprocess.check_output(
```

```
            ["nmap", "-sn", IP])
        nmap_output = nmap_output.decode("utf-8")

        # Parse nmap output to extract available IP addresses
        lines = nmap_output.split("\n")
        for line in lines:
            if "Nmap scan report" in line:
                ip = line.split()[-1]
                local_ips.append(ip)
    except Exception as e:
        print("Error:", e)

    return local_ips


def main():
    print("Scanning local network for available IP addresses...")
    available_ips = scan_local_network()
    print("Available IP addresses on the local network:", available_ips)

    if not available_ips:
        print("No available IP addresses found on the local network.")
        return

    server_port = 12340
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client_socket.connect((server_ip, server_port))  # Server IP Address
    print("Connected to server")

    ip_to_send = input('Enter IP Address ')
    client_socket.sendall(ip_to_send.encode())

    ip_address = client_socket.recv(1024).decode()
    print("Connection Successful!!")
    subprocess.run(['python', 'Sender.py',ip_address])



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

TESTING:



SCREENSHOTS:

First the server.py program is executed:

```
D:\Sem 6\ISM\Project2>python server.py
Enter the MainFrame IP Address ->192.168.145.250
Server listening on 192.168.145.250 port 12340
```

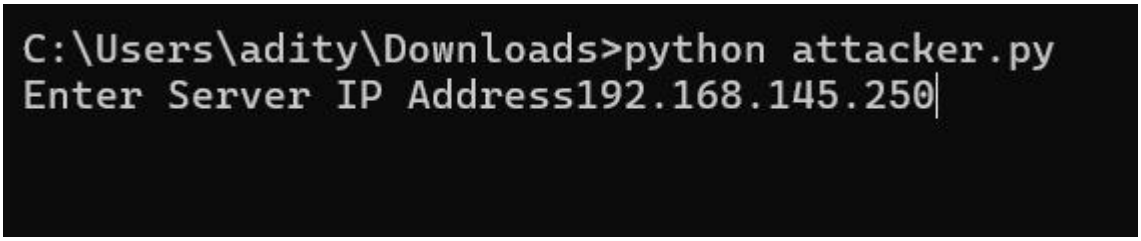
Then the mainframe.py program is executed:

```
D:\Sem 6\ISM\Project2>python Mainframe.py
Host listening on 172.17.27.24 port 12346
```

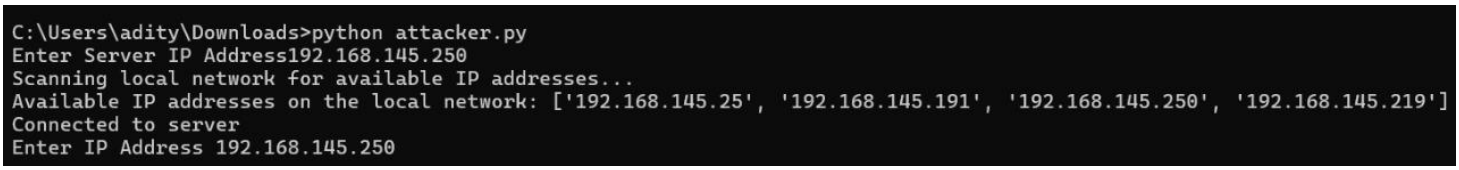
Then the Dummy_Host.py program is executed:

```
Microsoft Windows [Version 10.0.22621.3296]
(c) Microsoft Corporation. All rights reserved.

E:\ismlabda>python Dummy_Host.py
Host listening on 192.168.145.191 port 12346
```

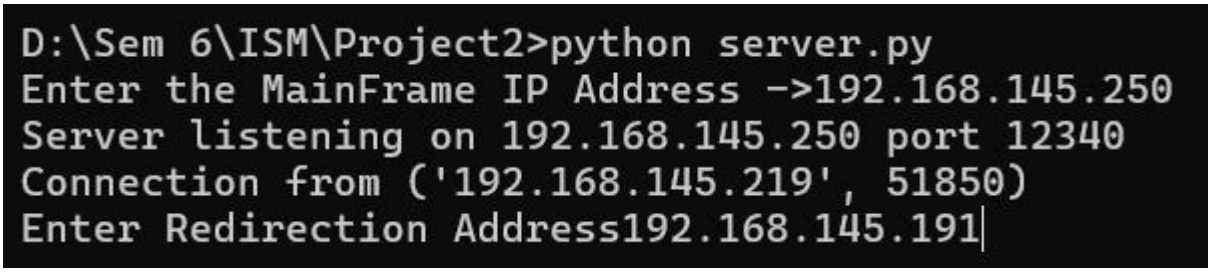
Now attacker.py is executed (IP Address of server is given as input):



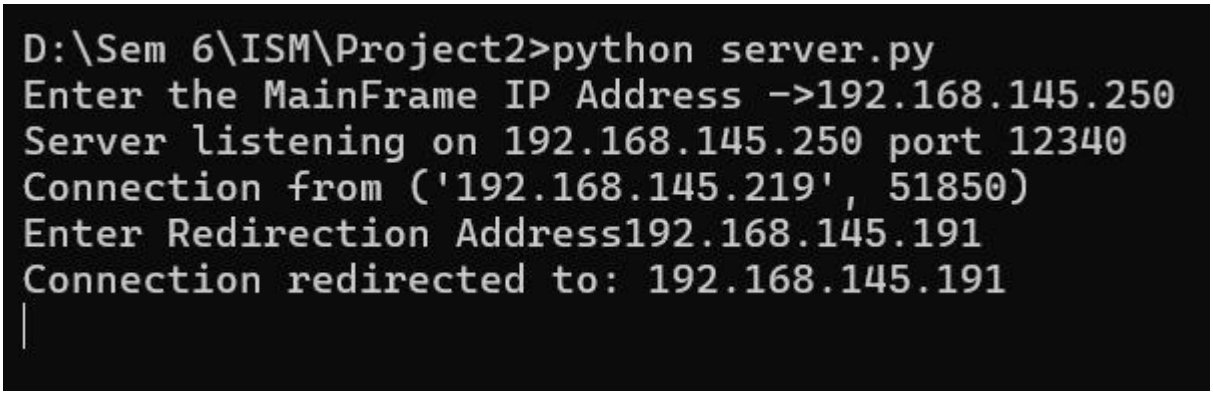
Connection is established and IP Address of Mainframe PC is given as input to establish connection:



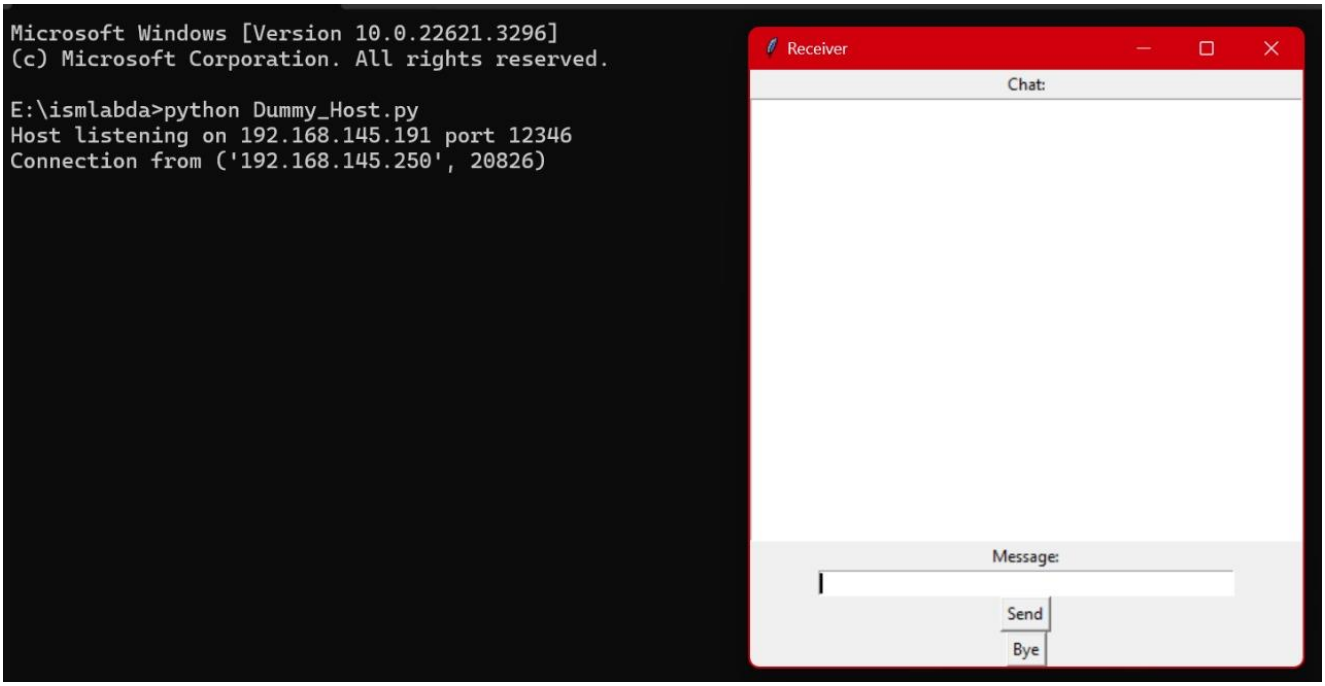
This connection is received by the server and the redirection address (dummy host) is given as input:



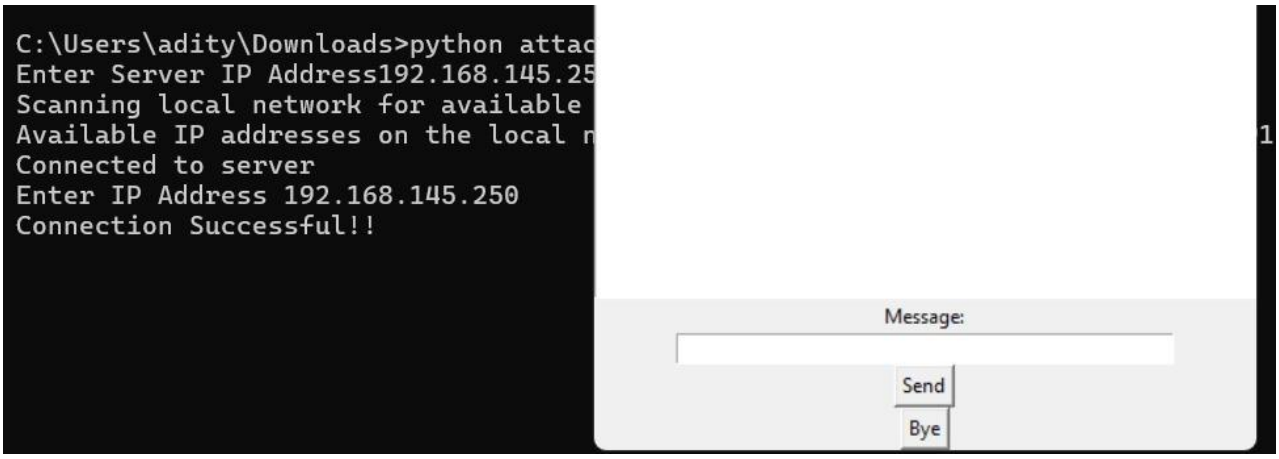
After successful redirection:



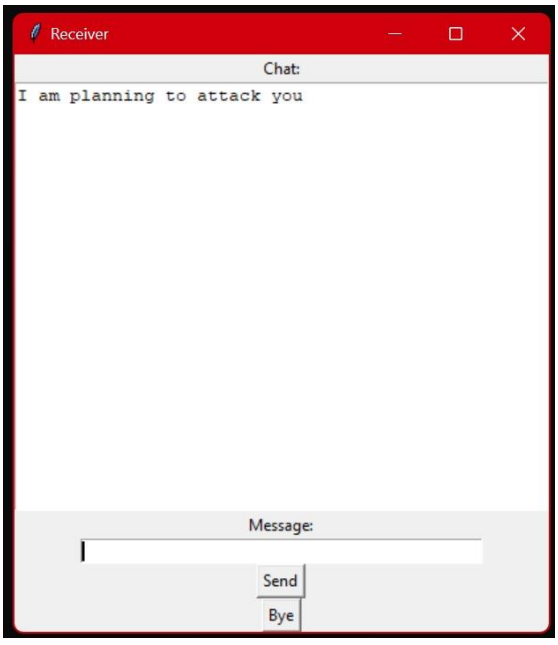
At Receivers (Dummy host) side:

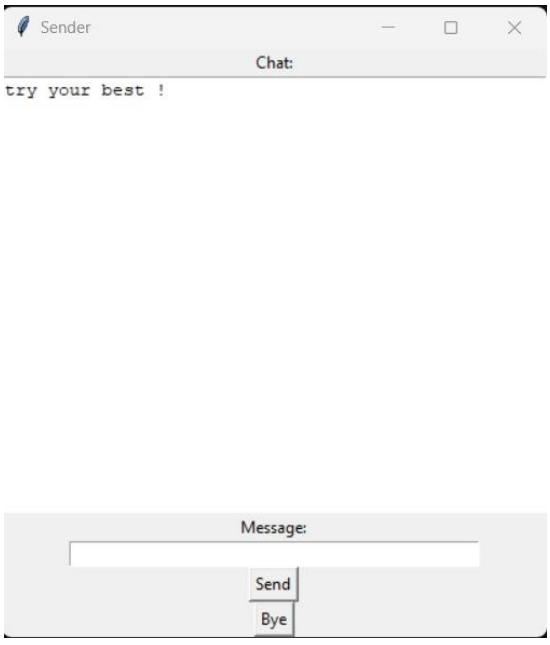


At Attacker's side:

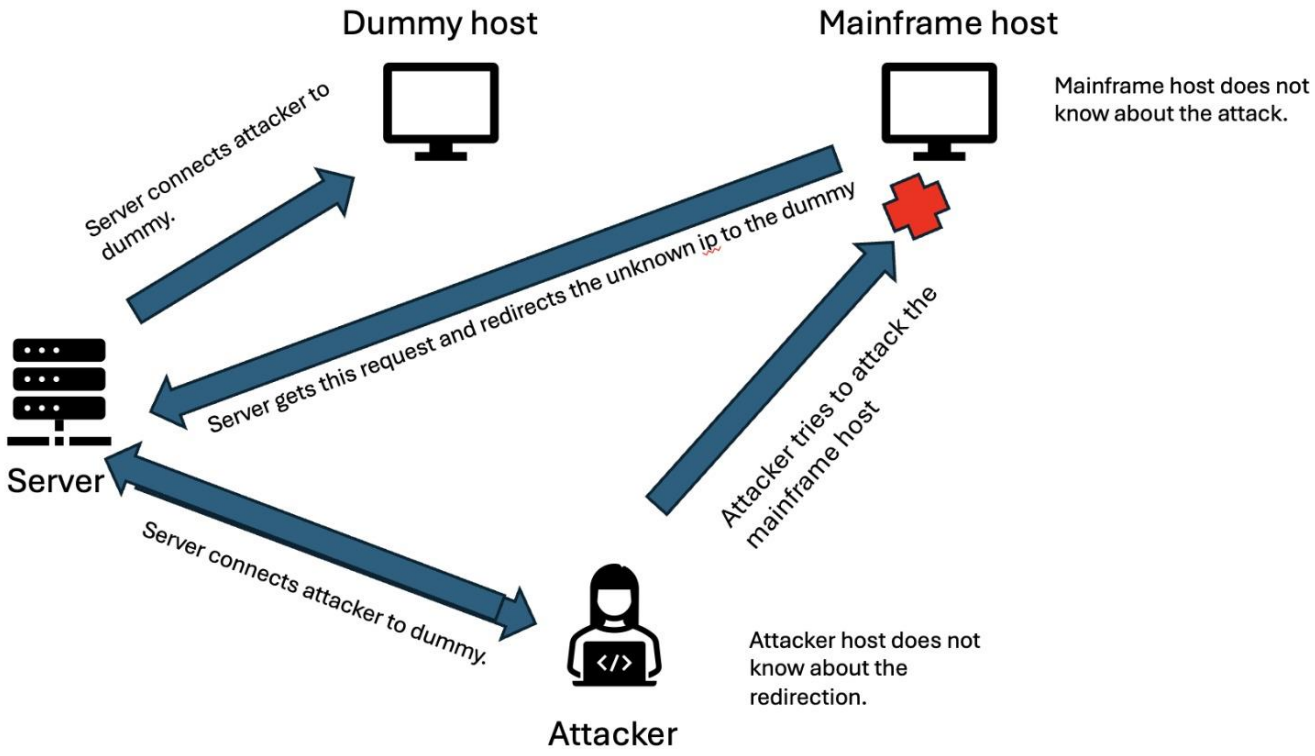


Chat starts between Attacker and Receiver:





SYSTEM ARCHITECTURE:



CONCLUSION:

The outcome of implementing the dynamic server-side redirection solution for mainframe protection in network security management is a significantly strengthened security posture and enhanced resilience against various cyber threats. By dynamically rerouting network traffic based on real-time analysis and predefined security policies, the system effectively mitigates risks associated with unauthorized access attempts, malware infiltration, and data breaches targeted at mainframe systems. This proactive approach to security reduces the likelihood of successful attacks and minimizes the potential impact on organizational operations and sensitive data assets. Additionally, the integration of comprehensive monitoring, reporting, and analytics capabilities empowers administrators to swiftly detect and respond to security incidents, further bolstering the overall resilience of the network infrastructure. Moreover, the scalable and resilient architecture ensures that the solution can adapt to evolving threats and accommodate increasing traffic loads without compromising performance or availability. Ultimately, the outcome is a more robust and adaptive network security management framework that safeguards mainframe environments against a wide range of cyber threats effectively.