Project Overview

This project builds an **automated deception-based cyber defense model** that:

- Detects scanning behavior from potential attackers
- Diverts attackers to a **decoy clone system**
- Continues serving **legitimate users with no interruption**
- Uses **fake ports, proxy redirection, and service failover** logic
- Simulates real activity using **automated bots**

The result is a powerful illusion:

> **Attackers are misled into thinking they are compromising a real server** while all user-facing services are preserved and redirected safely.

Network Architecture

Component	IP Address	Role
Attacker (Arch Linux)	192.168.56.102	Scans/Exploits Server 1
Server 1 (Linux Mint)	192.168.56.103	Real service host + redirection/proxy logic
Decoy System (Linux Mint)	192.168.56.106	Clone of Server 1 with extra open ports
Server 2 (Linux Mint)	192.168.56.105	Standby server for failover response
Bots (Linux Mint)	Random IPs	Simulate user activity inside Decoy
Legit Users	Any IP	Accesses service via Server 1 or 2

Phase-by-Phase Breakdown

Phase 1: True + False Information Leakage

- **Server 1** answers port scans with true ports (e.g., 80) + **false ports** copied from Decoy (e.g., 21, 23).
- This adds uncertainty. The attacker doesn't know which system is real.

Phase 2: TCP SYN Scan Detection

- **Server 1** uses `Scapy` to **sniff for TCP SYN packets**.
- If more than 5 unique ports are scanned in < 5 seconds from one IP → Mark as **suspicious**

Phase 3: Proxy Redirection Based on IP

- Once flagged, attacker IPs are **routed to the Decoy**
- Server 1 becomes a middleman: forwards packets to/from Decoy
- Legit users continue getting real service from Server 1

Phase 4: Failover After Suspicion

- Once *any* attacker is detected:
    - Server 1 triggers Server 2's Apache2 via SSH
    - All **legit traffic gets redirected to Server 2**
    - Meanwhile attacker continues talking to Decoy

Phase 5: Bot Simulation

- Bots are executed from other machines (or local scripts)
- Simulate real users accessing web & FTP (port 80, 21)
- **Reduces attacker suspicion** of being in a trap

Web Server Simulation

Each server runs a simple HTML login portal:

Server 1 (Original Web Portal)

```
<!DOCTYPE html>
<html>
<head>
  <title>Secure Login Portal Server 1</title>
  <style>
    body { font-family: Arial; background: #f0f0f0; text-align: center; margin-top: 100px; }
    form { background: white; display: inline-block; padding: 20px; border-radius: 10px; }
    input { margin: 10px; padding: 10px; width: 200px; }
    button { padding: 10px 30px; }
  </style>
</head>
<body>
  <h2>Secure Login Portal - Server 1</h2>
  <form>
    <input type="text" name="username" placeholder="Username" required><br>
    <input type="password" name="password" placeholder="Password" required><br>
    <button type="submit">Login</button>
  </form>
  <p>Verifying credentials... please wait</p>
</body>
</html>
```

Decoy System (Trap Portal)

```
<!DOCTYPE html>
<html>
<head>
  <title>Secure Login Portal Decoy</title>
  <style>
    body { font-family: Arial; background: #f0f0f0; text-align: center; margin-top: 100px; }
    form { background: white; display: inline-block; padding: 20px; border-radius: 10px; }
    input { margin: 10px; padding: 10px; width: 200px; }
    button { padding: 10px 30px; }
  </style>
</head>
<body>
  <h2>Secure Login Portal - Decoy</h2>
  <form>
    <input type="text" name="username" placeholder="Username" required><br>
    <input type="password" name="password" placeholder="Password" required><br>
    <button type="submit">Login</button>
  </form>
```

```
  <p>Verifying credentials... please wait</p>
</body>
</html>
```

Server 2 (Backup Portal)

```
<!DOCTYPE html>
<html>
<head>
  <title>Secure Login Portal Server 2</title>
  <style>
    body { font-family: Arial; background: #f0f0f0; text-align: center; margin-top: 100px; }
    form { background: white; display: inline-block; padding: 20px; border-radius: 10px; }
    input { margin: 10px; padding: 10px; width: 200px; }
    button { padding: 10px 30px; }
  </style>
</head>
<body>
  <h2>Secure Login Portal - Server 2</h2>
  <form>
    <input type="text" name="username" placeholder="Username" required><br>
    <input type="password" name="password" placeholder="Password" required><br>
    <button type="submit">Login</button>
  </form>
  <p>Verifying credentials... please wait</p>
</body>
</html>
```

Python Source Code

deception_proxy.py (Core Redirection System)

```
import socket
import threading
import time
import logging
import subprocess
import os
from scapy.all import sniff, IP, TCP
from collections import defaultdict

# ------------------ CONFIG ------------------
HONEYPOT_IP = "192.168.56.106"       # Decoy system
REAL_HOST_IP = "127.0.0.1"           # Web service on Server 1
REAL_HOST_PORT = 8080
SERVER2_IP = "192.168.56.105"        # Server 2 (Hot standby)
```

```
SERVER2_USER = "koku"
SERVER2_PORT = 80
FAILOVER_SCRIPT = "/home/koku/failover.sh"

MONITORED_PORTS = [21, 22, 80]
SYN_THRESHOLD = 5
TIME_WINDOW = 5
SUSPICION_TIMEOUT = 60
CHECK_INTERVAL = 5
WHITELIST = {"192.168.0.100"}        # Optional trusted IPs
LOG_FILE = "deception_proxy.log"

scan_tracker = defaultdict(list)
unique_ports_tracker = defaultdict(set)
suspicious_ips = set()
suspicion_timestamp = defaultdict(float)
failover_triggered = False

logging.basicConfig(
    filename=LOG_FILE,
    format='%(asctime)s - %(levelname)s - %(message)s',
    level=logging.INFO
)

# ------------------ SYN SCAN DETECTION ------------------
def detect_syn_scans():
    def callback(pkt):
        if IP in pkt and TCP in pkt and pkt[TCP].flags == 'S':
            src_ip = pkt[IP].src
            dst_port = pkt[TCP].dport
            now = time.time()

            if src_ip in WHITELIST:
                return

            scan_tracker[src_ip] = [t for t in scan_tracker[src_ip] if now - t < TIME_WINDOW]
            scan_tracker[src_ip].append(now)
            unique_ports_tracker[src_ip].add(dst_port)

            if len(unique_ports_tracker[src_ip]) >= SYN_THRESHOLD:
                if src_ip not in suspicious_ips:
                    suspicious_ips.add(src_ip)
                    suspicion_timestamp[src_ip] = now
                    logging.warning(f"[!] SYN scan detected from {src_ip} → redirecting to decoy")

    sniff(filter="tcp", prn=callback, store=False)

# ------------------ FAILOVER TRIGGER ------------------
```

```
def trigger_failover():
    global failover_triggered
    if failover_triggered:
        return
    failover_triggered = True
    logging.critical("[!] Failover triggered. Redirecting legit users to Server 2...")

    try:
        result = subprocess.run([
            "ssh", f"{SERVER2_USER}@{SERVER2_IP}", FAILOVER_SCRIPT
        ], capture_output=True, text=True)

        if result.returncode == 0:
            logging.info("[+] Server 2 Apache started via failover.sh.")
        else:
            logging.error(f"[!] SSH failover script error:\n{result.stderr}")

    except Exception as e:
        logging.error(f"[!] SSH connection failed: {e}")

# ------------------ EXPLOIT MONITOR ------------------
def monitor_attacks():
    while True:
        if suspicious_ips and not failover_triggered:
            trigger_failover()
        time.sleep(CHECK_INTERVAL)

# ------------------ TRAFFIC REDIRECTION ------------------
def forward(src, dst, direction, ip):
    try:
        while True:
            data = src.recv(4096)
            if not data:
                break
            logging.info("Forwarding (%s) [%s]: %d bytes", direction, ip, len(data))
            dst.sendall(data)
    except Exception as e:
        logging.warning("Pipe error (%s) [%s]: %s", direction, ip, e)
    finally:
        src.close()
        dst.close()

def get_target(ip, port):
    now = time.time()
    if ip in suspicious_ips:
        if now - suspicion_timestamp[ip] < SUSPICION_TIMEOUT:
            return (HONEYPOT_IP, port)
        else:
```

```
            suspicious_ips.discard(ip)
            scan_tracker[ip].clear()
            unique_ports_tracker[ip].clear()
            logging.info(f"[+] Cleared suspicion for {ip}")
    if failover_triggered and port == 80:
        return (SERVER2_IP, SERVER2_PORT)
    return (REAL_HOST_IP, REAL_HOST_PORT if port == 80 else port)

def handle_conn(client_sock, client_ip, port):
    target_ip, target_port = get_target(client_ip, port)
    try:
        server_sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        server_sock.connect((target_ip, target_port))
        threading.Thread(target=forward, args=(client_sock, server_sock, "Attacker → Target",
client_ip), daemon=True).start()
        threading.Thread(target=forward, args=(server_sock, client_sock, "Target → Attacker",
client_ip), daemon=True).start()
    except Exception as e:
        logging.error("[!] Connection error for %s: %s", client_ip, e)
        client_sock.close()

# ------------------ TCP PROXY LISTENERS ------------------
def start_listener(port):
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    sock.bind(("0.0.0.0", port))
    sock.listen(100)
    logging.info("[*] Listening on port %s", port)
    while True:
        try:
            client_sock, addr = sock.accept()
            logging.info("[+] Incoming from %s:%s on port %d", addr[0], addr[1], port)
            threading.Thread(target=handle_conn, args=(client_sock, addr[0], port),
daemon=True).start()
        except OSError as e:
            logging.error(f"[!] Socket accept failed: {e}")
            time.sleep(1)

# ------------------ RESTORE APACHE IF NEEDED ------------------
def restore_real_service():
    logging.info("[*] Restoring Apache on Server 1...")
    os.system("sudo systemctl start apache2")

import atexit
atexit.register(restore_real_service)

# ------------------ MAIN ------------------
print("[*] Deception Proxy Active with Decoy + SYN Detection + Failover")
```

```
logging.info("[*] Deception system running on monitored ports: %s", MONITORED_PORTS)

for p in MONITORED_PORTS:
    threading.Thread(target=start_listener, args=(p,), daemon=True).start()

threading.Thread(target=detect_syn_scans, daemon=True).start()
threading.Thread(target=monitor_attacks, daemon=True).start()

try:
    while True:
        time.sleep(1)
except KeyboardInterrupt:
    print("\n[!] Shutting down.")
    logging.info("[*] Deception proxy stopped.")
```

bot_decoy_simulator.py (Bot Generator)

```
import threading
import time
import random
import requests
import socket

# ------------------------
# CONFIGURATION
# ------------------------

DECOY_HTTP = "http://192.168.56.106"
DECOY_IP = "192.168.56.106"
BOTS = 5
HEADERS = {
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)"
}
PORTS = {
    "http": 80,
    "ftp": 21
}

# ------------------------
# BOT ACTIONS
# ------------------------

def access_http(bot_name):
    try:
        response = requests.get(DECOY_HTTP, headers=HEADERS, timeout=3)
        print(f"[{bot_name}] HTTP {response.status_code}")
    except Exception as e:
```

```
        print(f"[{bot_name}] HTTP error: {e}")

def access_ftp(bot_name):
    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.settimeout(3)
        s.connect((DECOY_IP, PORTS["ftp"]))
        print(f"[{bot_name}] Connected to FTP port")
        s.close()
    except Exception as e:
        print(f"[{bot_name}] FTP error: {e}")

# ------------------------
# BOT BEHAVIOR
# ------------------------

def bot_loop(bot_id):
    bot_name = f"Bot-{bot_id}"
    while True:
        choice = random.choice(["http", "ftp"])
        if choice == "http":
            access_http(bot_name)
        else:
            access_ftp(bot_name)
        time.sleep(random.randint(5, 15))  # Human-like delay

# ------------------------
# START BOTS
# ------------------------

if __name__ == "__main__":
    for i in range(1, BOTS + 1):
        t = threading.Thread(target=bot_loop, args=(i,), daemon=True)
        t.start()

    while True:
        time.sleep(10)
```

Step-by-Step Deployment Instructions

On Server 1 (192.168.56.103)

```
sudo apt update && sudo apt install apache2 openssh-server
python3 python3-pip -y
```

Create web service:

```
sudo nano /var/www/html/index.html
# Paste: Secure Login Portal - Server 1
sudo systemctl enable apache2
sudo systemctl start apache2
```

Allow necessary ports:

```
sudo ufw allow 21
sudo ufw allow 22
sudo ufw allow 80
```

Create and activate Python virtual environment:

```
python3 -m venv myenv
source myenv/bin/activate
pip install scapy paramiko
```

Give passwordless sudo for Apache on Server 2: (At New Tab)

```
ssh koku@192.168.56.105
sudo visudo
# Add this line:
koku ALL=(ALL) NOPASSWD: /bin/systemctl start apache2
```

Copy SSH key to Server 2: (At New Tab)

```
ssh-copy-id koku@192.168.56.105
```

Save the Deception script:

nano deception_proxy.py #Copy past the deception script

Run Deception Proxy:

```
sudo myenv/bin/python3 deception_proxy.py
```

On Decoy (192.168.56.106)

Open the decoy fake ports

Run Real/Vulnerable Services on Linux Mint (Decoy)

Update System First

```
sudo apt update && sudo apt upgrade -y
```

Open Port 80: Apache Web Server (HTTP)

```
sudo apt install apache2 -y
sudo systemctl enable apache2
sudo systemctl start apache2
```

Test:

```
curl <http://localhost>
```

Open Port 22: SSH

```
sudo apt install openssh-server -y
sudo systemctl enable ssh
sudo systemctl start ssh
```

To confirm:

```
sudo systemctl status ssh
```

Open Port 21: FTP Server (vsftpd)

```
sudo apt install vsftpd -y
sudo systemctl enable vsftpd
sudo systemctl start vsftpd
```

Disable Firewall or Allow Ports

If `ufw` is enabled:

```
sudo ufw allow 21
sudo ufw allow 22
sudo ufw allow 80
sudo ufw enable
```

Confirm Open Ports

On the decoy:

```
sudo ss -tuln
```

From attacker machine:

```
nmap -sS <decoy-ip>
```

- **Host a Fake Web Service:**

```
sudo systemctl enable apache2
sudo systemctl start apache2
sudo nano /var/www/html/index.html
# Use: "Secure Login Portal - Decoy"
```

- **Exploit trigger simulation:**

Create a file that attacker will trigger:

```
sudo touch /tmp/rooted
```

You can delete it to simulate non-exploited state.

Step-by-Step: Inject Spoof Aliases Inline in `.bashrc`

Step 1: Open the `.bashrc` file

```
nano ~/.bashrc
```

Step 2: Scroll to the bottom of the file, and paste these lines:

```
# Deception aliases (victim identity masking)
alias ip='echo "inet 192.168.56.103/24 brd 192.168.56.255 scope global eth0"'
alias ifconfig='echo -e "eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>
mtu 1500\\n    inet 192.168.56.103  netmask 255.255.255.0  broadcast
192.168.56.255\\n    ether 08:00:27:de:ad:be  txqueuelen 1000 (Ethernet)"'
alias hostname='echo victim-linux'
alias uname='echo "Linux victim-linux 5.15.0-kali #1 SMP"'
alias whoami='echo root'
alias id='echo uid=0(root) gid=0(root) groups=0(root)'
```

These aliases override common fingerprinting commands.

Step 3: Apply Changes Immediately

```
source ~/.bashrc
```

Now the spoofed environment is live.

On Server 2 (192.168.56.105)

- **Host Web Service:**

```
sudo apt install apache2 -y
sudo systemctl enable apache2
sudo systemctl start apache2
sudo nano /var/www/html/index.html
# Use: "Secure Login Portal - Server 2"
```

Apache will be OFF initially. Server 1 will trigger its launch after detecting an attacker.

On Bot Machine (or Decoy itself)

```
sudo apt update
sudo apt install python3 python3-pip -y
python3 -m venv myenv
source myenv/bin/activate
pip install requests
```

Run the bot script:

```
python3 bot_decoy_simulator.py
```

On Attacker Machine (192.168.56.102)

Before Scan:

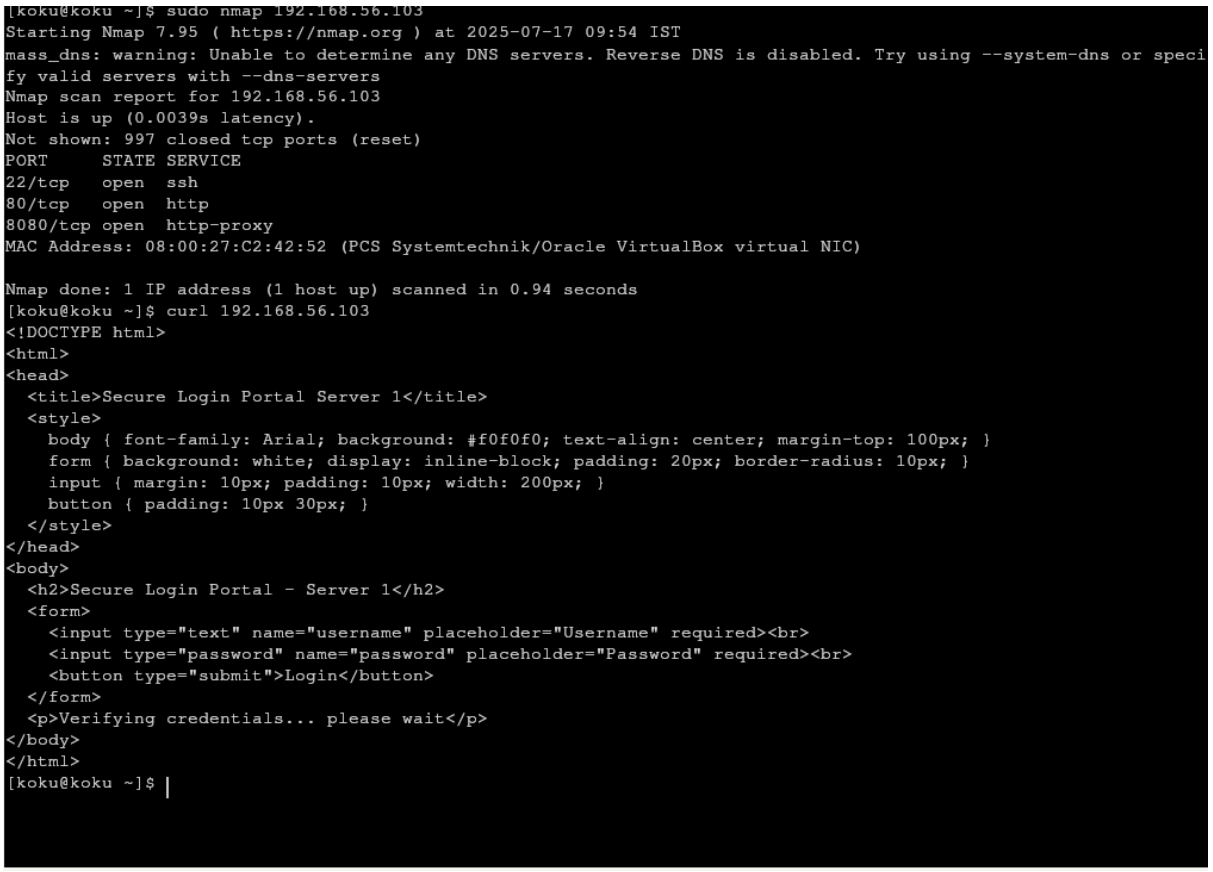
Open a web browser and visit:

http://192.168.56.103

You can access the real web service.

Or run the following command in the terminal:

curl 192.168.56.103



To simulate a SYN scan, run:

```
nmap 192.168.56.103
```

After Scan:

Open a web browser and visit:

```
http://192.168.56.103
```

You will now access the fake (decoy) web service.

Or run the following command in the terminal:

```
curl 192.168.56.103
```

```
Starting Nmap 7.95 ( https://nmap.org ) at 2025-07-19 01:50 IST
mass_dns: warning: Unable to determine any DNS servers. Reverse DNS is disabled. Try
using --system-dns or specify valid servers with --dns-servers
Nmap scan report for 192.168.56.103
Host is up (0.0031s latency).
Not shown: 996 closed tcp ports (reset)
PORT     STATE SERVICE
21/tcp   open  ftp
22/tcp   open  ssh
80/tcp   open  http
8080/tcp open  http-proxy
MAC Address: 08:00:27:C2:42:52 (PCS Systemtechnik/Oracle VirtualBox virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 0.72 seconds
[koku@koku ~]$ curl 192.168.56.103
<!DOCTYPE html>
<html>
<head>
  <title>Secure Login Portal Decoy</title>
  <style>
    body { font-family: Arial; background: #f0f0f0; text-align: center; margin-top: 1
00px; }
    form { background: white; display: inline-block; padding: 20px; border-radius: 10
px; }
    input { margin: 10px; padding: 10px; width: 200px; }
    button { padding: 10px 30px; }
  </style>
</head>
<body>
  <h2>Secure Login Portal - Decoy</h2>
  <form>
    <input type="text" name="username" placeholder="Username" required><br>
    <input type="password" name="password" placeholder="Password" required><br>
    <button type="submit">Login</button>
  </form>
  <p>Verifying credentials... please wait</p>
</body>
</html>

[koku@koku ~]$
```

After all implementations:

- Run the Python Deception Proxy:

```
sudo myenv/bin/python3 deception_proxy.py
```

Execute this script on Server 1.

- After starting the proxy, initiate the scan from the attacker machine.

- Once the scan begins, go to Server 1. It will prompt for the password of Server 2. Enter the password to enable redirecting all real services to Server 2.

Testing / Validation Checklist

Action	Expected Result
User opens Server 1	Sees "Secure Login Portal - Server 1"
Attacker scans Server 1	Gets ports: 80 (real) + 21, 23 (fake)
Attacker flagged	Redirected to Decoy
User reopens Server 1	Gets redirected to Server 2 portal