ENCODERSPRO
An AKTU-Incubated CyberSecurity Company

PROJECT COMPLETION REPORT

Modular Botnet Simulation and Analysis

Submitted in partial fulfillment of the requirements
By

Team - अनिर्वचीय

1 . Abstract

This project presents the simulation of a modular botnet operating over a distributed, secure virtual network built using OpenVPN. The botnet consists of multiple Python based malware agents running on both Linux and Windows virtual machines, a centralized Flask based Command and Control (C2) dashboard hosted on a Kali Linux instance and a vulnerable Ubuntu based target server designed to test real world attack scenarios such as DDoS and scanning. Unlike traditional single system simulations, the project utilized OpenVPN to interconnect virtual machines hosted on different team members' systems, allowing for a realistic and scalable distributed botnet model. The simulation was performed entirely within a controlled, isolated environment, ensuring complete ethical compliance. Key focus areas included bot to C2 communication, task execution via modular payloads and limited network traffic capture using Wireshark. This project provides practical insights into malware dynamics and botnet behavior in real world conditions while maintaining strict containment boundaries.

2 . Introduction

A botnet is a network of devices infected with malware that are remotely controlled by a central system known as a Command and Control (C2) server. These networks are commonly used in cyberattacks such as Distributed Denial of Service (DDoS), data theft, credential harvesting and spam distribution. To combat and understand such threats, cybersecurity researchers often need to analyze how botnets function. However, experimenting with real malware in live environments pose serious legal, ethical and operational risks.

To address this challenge, our project focuses on safely simulating a modular botnet in a fully controlled environment. Unlike traditional simulations that operate on a single host or within one hypervisor, our setup used OpenVPN to build a distributed virtual lab. This allowed team members to host and control different parts of the botnet infrastructure bots, target servers and the C2 server from their individual systems, emulating how real world botnets span across devices and geographies.

Our project involved:

- Developing cross platform Python based malware bots for Linux and Windows.
- Building a web based C2 dashboard using Flask for command dispatch and result tracking.
- Implementing attack modules such as keylogging, port scanning and DDoS simulation.
- Capturing traffic using Wireshark for analysis of botnet communication patterns.

All activities were conducted in an ethically isolated environment and no component had access to public networks or uninvolved systems. The simulation offered valuable hands on experience in malware development, network based control and distributed system coordination key concepts in modern cybersecurity.

---

3 . System Overview

The modular botnet was designed and executed within a distributed virtual lab, created using OpenVPN. Unlike a traditional single machine or LAN only simulation, this approach allowed each team member to contribute separate virtual machines from their local systems, all interconnected over a private VPN. This method closely mimics real world botnet distribution and behavior while maintaining complete isolation from the public internet.

Overall Architecture

The botnet architecture consists of the following components:

- **Command & Control (C2) Server**
  A Flask based web dashboard running on a Kali Linux VM. It registers bots, issues commands and logs responses.
- **Linux Bots (x4)**
  Python based agents running on Debian/Ubuntu VMs. These bots poll the C2 for commands and execute attack modules.
- **Windows Bot (x1)**
  A Python agent modified for compatibility with Windows.
- **Target Server**
  An Ubuntu VM running a basic Flask web app. This server acts as a DDoS target for testing attack modules.

Virtual Lab Setup (via OpenVPN)

- An OpenVPN server was hosted on the team lead's Ubuntu host OS.
- All virtual machines including the C2 server, bots and target were configured as OpenVPN clients using .ovpn profiles with username/password authentication.
- Once connected, each VM received an internal VPN IP address (e.g., 10.9.0.x) allowing secure HTTP communication over the tunnel.
- This setup enabled each component to simulate botnet operations from separate physical locations while preserving traffic isolation and analysis capabilities.

Tools and Technologies Used

Component	**Technology**
Programming	Python 3
C2 Server	Flask (Kali Linux VM)
Bot Clients	Python3
Target Server	Flask Web App (Ubuntu VM)
Virtualization	VirtualBox, VMWare
VPN Networking	OpenVPN
Forensics / Capture	Wireshark

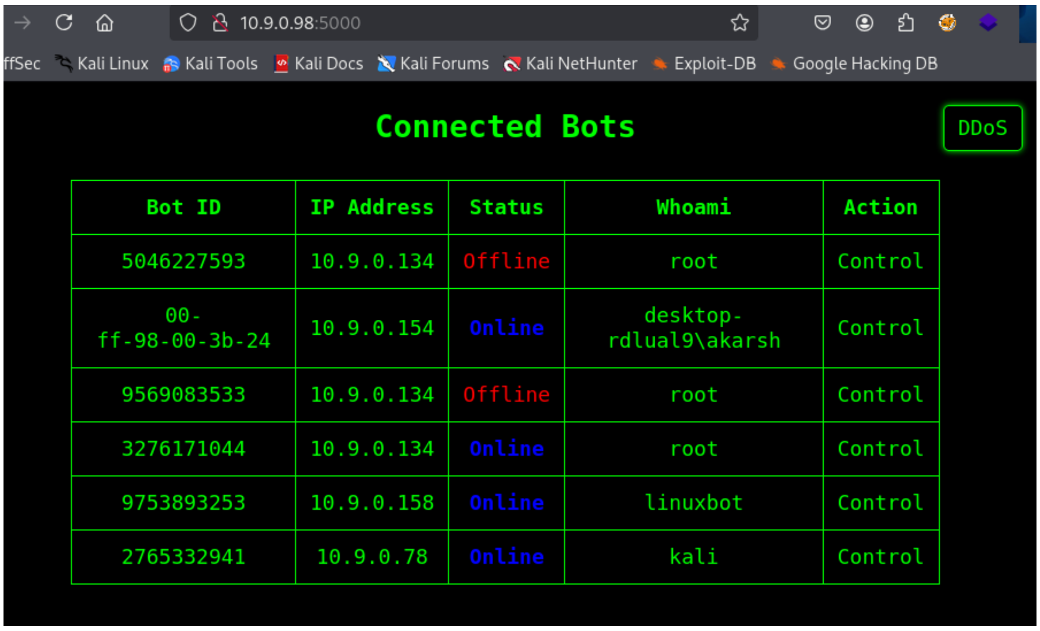
---

4 . Implementation Details

The botnet was implemented using a modular architecture with independently developed components for the Command and Control (C2) server, bots (both Linux and Windows) and a target server. Each module was containerized within its own virtual machine, connected securely through OpenVPN.

Command and Control (C2) Server

- **Platform**: Kali Linux VM
- **Tech Stack**: Python 3, Flask, HTML
- **Functionality**:
  - Hosts a web dashboard to view registered bots
  - Allows selection of predefined attack modules (e.g., whoami, port_scan, ddos)
  - Displays bot responses in real time
  - Logs bot IDs and timestamps
- **Communication Protocol**:
  - Bots poll via HTTP GET requests to /get_command/<BOT_ID>
  - C2 replies with pending commands
  - Bots send back results using HTTP POST to /report



Connected Bots

Bot ID	IP Address	Status	Whoami	Action
5046227593	10.9.0.134	Offline	root	Control
00-ff-98-00-3b-24	10.9.0.154	Online	desktop-rdlual9\akarsh	Control
9569083533	10.9.0.134	Offline	root	Control
3276171044	10.9.0.134	Online	root	Control
9753893253	10.9.0.158	Online	linuxbot	Control
2765332941	10.9.0.78	Online	kali	Control

Bots (Linux & Windows)

Common Features:

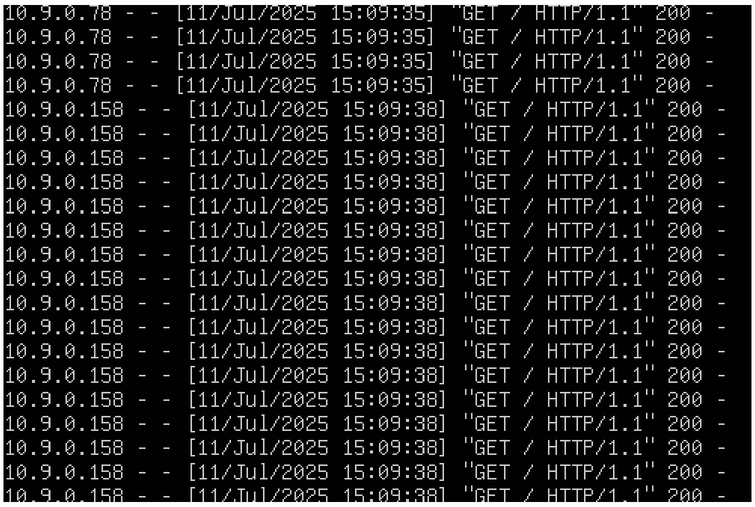
- Written in Python 3 using standard libraries: requests, subprocess, threading
- Use a persistent HTTP polling mechanism to receive commands
- Periodically poll the C2 (default every 5 seconds)

Modules:

- net_scan: scans network
- port_scan: performs TCP port scanning
- ddos: performs multi threaded HTTP GET flood
- keylogger: captures keyboard input using pynput
- stealer: harvests stored credential
- spyware: takes **screenshot** & **clipboard**
- bruteforce: performs bruteforce attacks on the given ip and port

Target Server

- **Platform**: Ubuntu VM
- **Service**: Flask based lightweight web server
- **Purpose**:
  - Acts as a victim for testing bot based attacks
  - Handles DDoS flood (spammed HTTP GET requests)
  - Displays received requests on terminal or logs
- Monitored using:
  - Flask access logs
  - Wireshark captures to observe HTTP flood traffic



```
10.9.0.78 - - [11/Jul/2025 15:09:35] "GET / HTTP/1.1" 200 -
10.9.0.78 - - [11/Jul/2025 15:09:35] "GET / HTTP/1.1" 200 -
10.9.0.78 - - [11/Jul/2025 15:09:35] "GET / HTTP/1.1" 200 -
10.9.0.78 - - [11/Jul/2025 15:09:35] "GET / HTTP/1.1" 200 -
10.9.0.158 - - [11/Jul/2025 15:09:38] "GET / HTTP/1.1" 200 -
10.9.0.158 - - [11/Jul/2025 15:09:38] "GET / HTTP/1.1" 200 -
10.9.0.158 - - [11/Jul/2025 15:09:38] "GET / HTTP/1.1" 200 -
10.9.0.158 - - [11/Jul/2025 15:09:38] "GET / HTTP/1.1" 200 -
10.9.0.158 - - [11/Jul/2025 15:09:38] "GET / HTTP/1.1" 200 -
10.9.0.158 - - [11/Jul/2025 15:09:38] "GET / HTTP/1.1" 200 -
10.9.0.158 - - [11/Jul/2025 15:09:38] "GET / HTTP/1.1" 200 -
10.9.0.158 - - [11/Jul/2025 15:09:38] "GET / HTTP/1.1" 200 -
10.9.0.158 - - [11/Jul/2025 15:09:38] "GET / HTTP/1.1" 200 -
10.9.0.158 - - [11/Jul/2025 15:09:38] "GET / HTTP/1.1" 200 -
10.9.0.158 - - [11/Jul/2025 15:09:38] "GET / HTTP/1.1" 200 -
10.9.0.158 - - [11/Jul/2025 15:09:38] "GET / HTTP/1.1" 200 -
10.9.0.158 - - [11/Jul/2025 15:09:38] "GET / HTTP/1.1" 200 -
10.9.0.158 - - [11/Jul/2025 15:09:38] "GET / HTTP/1.1" 200 -
10.9.0.158 - - [11/Jul/2025 15:09:38] "GET / HTTP/1.1" 200 -
10.9.0.158 - - [11/Jul/2025 15:09:38] "GET / HTTP/1.1" 200 -
10.9.0.158 - - [11/Jul/2025 15:09:38] "GET / HTTP/1.1" 200 -
```

*DDOS logs on the target server

Sample Flow Summary

1. Bot starts, sends GET request to http://<C2 IP>:5000/get_command/<BOT_ID>
2. C2 replies with a command string (port_scan 10.9.0.x)
3. Bot executes the task and POSTs the result to /report
4. Dashboard updates with timestamp and response

---

5 . Analysis and Observations

This section outlines how the distributed botnet simulation was designed, interconnected and made operational using OpenVPN, followed by the observable behaviors during test runs. The majority of effort in this phase was spent on building a reliable and modular infrastructure, ensuring that the components communicated accurately across systems hosted on different machines.

System Integration & Workflow

The project was developed across multiple team members’ systems, each contributing one or more virtual machines. The infrastructure was designed to mimic a real world distributed botnet environment while remaining completely safe and offline.

VPN Based Connectivity

- An **OpenVPN server** was hosted on the team lead’s Ubuntu machine.
- Each participant connected their VM (bot, target, or C2) using .ovpn profiles with username/password based authentication.
- Once connected, all machines received IPs in the 10.9.0.0/24 range and could directly communicate with each other.
- This setup made it possible to **simulate a geographically distributed botnet**, even though the machines were running locally across different laptops.

Component Linking Flow

1. Bots boot up and automatically connect to the OpenVPN network.
2. On startup, each bot begins polling the C2 server at http://<c2 vpn ip>:5000/get_command/<BOT_ID> using HTTP GET.
3. The C2 Flask dashboard, running on a Kali Linux VM, listens for incoming bot registrations and provides an admin interface to send commands.
4. Once a command is issued, it is served to the requesting bot in plaintext.
5. Bots execute the command and return the result via a POST request to /report.
6. Target server (Ubuntu VM) sits idle until it becomes the recipient of DDoS flood or scanning attempts triggered by the bots.
7. The C2 dashboard updates live with the results as they arrive, displaying each bot's response alongside its assigned ID.

All communication happens through the VPN network, keeping the real host networks isolated.

Command Execution Pipeline

Each bot is designed to be modular and stateless. The pipeline for each interaction looks like this:

1. **Bot polls** /get_command/<BOT_ID> → GET
2. **C2 returns command** (e.g., ddos 10.8.0.12)
3. **Bot runs appropriate module** (e.g., spawns threads for flooding target)
4. **Bot posts result back** → /report → POST
5. **C2 logs and displays** the result on the admin panel

This approach allowed multi threaded actions without persistent socket connections or reverse shells, making it more secure for educational use and more scalable for adding bots on the fly.

Modular Design Benefits

Each functionality (keylogger, scanner, info, ddos, etc.) was treated as a plug and play module. New features could be added without affecting existing logic by:

- Creating a new module (e.g., brute_force.py)
- Adding a conditional match in the bot polling logic to call it when received

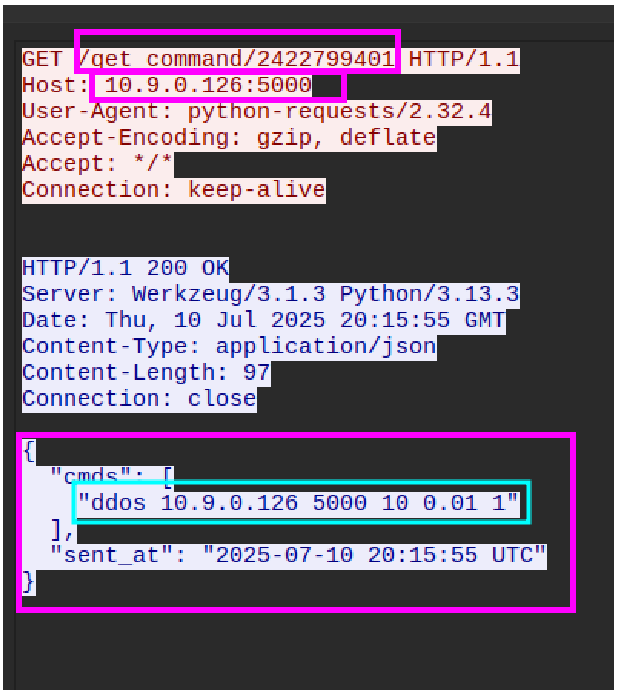
This made it easy to test and debug modules independently before integrating them into a live botnet.

Traffic Observation

While packet level analysis was limited, the system's observable behavior via Wireshark confirmed its functional correctness:

- Polling traffic was visible as frequent HTTP GETs to /get_command/<BOT_ID> with predictable timing (~10s intervals)
- Command replies were visible in raw HTTP response payload
- DDoS traffic was seen as bursts of HTTP GET requests from multiple VPN IPs (bots) hitting the target server.

Standard filters like http.request, ip.addr == <bot vpn ip> and tcp.port == 5000 were used to isolate traffic.



```
GET /get_command/2422799401 HTTP/1.1
Host: 10.9.0.126:5000
User-Agent: python-requests/2.32.4
Accept-Encoding: gzip, deflate
Accept: */*
Connection: keep-alive


HTTP/1.1 200 OK
Server: Werkzeug/3.1.3 Python/3.13.3
Date: Thu, 10 Jul 2025 20:15:55 GMT
Content-Type: application/json
Content-Length: 97
Connection: close

{
  "cmds": [
    "ddos 10.9.0.126 5000 10 0.01 1"
  ],
  "sent_at": "2025-07-10 20:15:55 UTC"
}
```

Summary

Rather than emphasizing packet forensics, the focus of this project was on **constructing and simulating a realistic botnet infrastructure**, using modular design principles and distributed network architecture. The use of OpenVPN made the system both realistic and safe and the modular architecture ensured scalability and clarity. Basic traffic analysis was performed to validate system behavior and confirm that botnet actions were being carried out over the network as expected.

---

6 . Challenges Faced

Despite the successful completion of the project, the team encountered several practical challenges during development, testing and integration. These challenges reflect both technical limitations and the realities of collaborative student driven projects.

1. Team Coordination & Workflow

One of the biggest challenges was maintaining smooth coordination across six team members. Tasks like code integration, virtual machine setup, testing suffered due to:

- Overlapping responsibilities and lack of strict task ownership
- Delays in delivering shared components (e.g., updated bots)
- Communication gaps during critical phases like testing and network setup

These issues were compounded when multiple team members were unavailable during key stages of debugging and reconfiguration.

2. Shift in Architecture (Late Redesign)

The initial plan was to simulate all components (C2, bots, target) using multiple VMs on a single powerful host. However, due to:

- Member's unavailability
- System resource limitations on individual laptops
- Poor VM performance under load
- The need for flexible collaboration across machines

the architecture was redesigned just **one day before submission** to use a distributed VPN model via **OpenVPN**. This required the team to quickly:

- Reconfigure all virtual machines for VPN access
- Adapt bot code to use dynamic VPN assigned Ips
- Test the full communication flow in a compressed timeframe

While this late change created pressure, it ultimately made the simulation more realistic and scalable.

3. Bot Stability & Functional Gaps

Although the malware bots were modular and functional to a basic degree, they were not fully stable across all environments:

- Some modules (e.g., stealer, spyware) did not run reliably in all Vms
- Certain commands failed silently due to threading bugs or platform mismatches
- The bots were not designed to evade antivirus software, though no interference occurred during testing due to the offline setup

These limitations were accepted, as the core objective was to simulate the architecture and flow of a botnet rather than to build fully robust malware.

4. VM Performance Issues

Running multiple virtual machines simultaneously on limited hardware (often with only 8–16 GB of RAM) led to:

- Freezing, lag and unresponsive GUIs
- Difficulty launching multiple bots or modules in parallel
- Delays in testing during high resource phases (e.g., DDoS tests)

5. Testing Limitations

- Limited time for full Wireshark based analysis and forensics
- Log outputs were captured but not deeply processed or visualized

Summary

Despite these challenges, the team successfully demonstrated:

- Cross platform botnet communication
- VPN based distributed simulation
- Modular command execution from a central C2 server

7 . Key Learnings

Throughout the course of this project, the team gained significant hands on experience across multiple domains of cybersecurity, software development and network simulation. While the botnet was simulated in a constrained and ethical environment, it exposed the team to many real world concepts and challenges.

Technical Learnings

- **Botnet Architecture**
  We learned how real world botnets are structured with a centralized C2 server, distributed bots and a modular control protocol and replicated this using our own Python based implementation.
- **Python Networking and HTTP Polling**
  The project improved our understanding of client server communication, especially using HTTP for polling, command execution and result transmission in a secure (VPN based) environment.
- **C2 Dashboard Development**
  We developed and deployed a Flask based dashboard to monitor bots, issue commands and handle POST/GET traffic, simulating real C2 functionality.
- **Modular Malware Design**
  By isolating each functionality (DDoS, keylogger, port scan, etc.) into modules, we learned how malware can be extended and maintained flexibly.
- **Virtual Networking and VPN Configuration**
  We gained practical experience in setting up and troubleshooting OpenVPN for private, secure communication across multiple systems a setup rarely taught in traditional coursework.
- **Cross Platform Development Challenges**
  Developing for both Linux and Windows introduced us to OS specific challenges like path handling and differences in permission models.

Cybersecurity and Analysis Concepts

- **Basic Traffic Analysis using Wireshark**
  Even though deep analysis was not performed, we successfully captured botnet related HTTP traffic and learned to identify key patterns such as polling frequency, payload structure and DDoS signatures.
- **Importance of Containment**
  Working with simulated malware required extreme caution. We learned the importance of isolation (using VMs and host only/VPN networks) and maintaining a strict ethical boundary.
- **Threat Simulation vs. Real Malware**
  This project helped distinguish the difference between emulating malware behavior for learning versus deploying real malicious tools a valuable perspective for ethical hacking.

Teamwork and Project Management

- **Working in a Distributed Environment**
  By splitting VM hosting and bot development among multiple members and integrating over VPN, we experienced real world style collaborative infrastructure building.
- **Adaptability Under Constraints**
  Facing time pressure, resource issues and late architecture changes, the team adapted quickly to redesign the environment and still meet core goals.
- **Version Control and Integration Lessons**
  While not using Git from the start created friction, it also taught us the importance of centralized code repositories and early sync in multi member projects.

---

8 . Conclusion and Future Work

Conclusion

This project successfully simulated a modular botnet using a distributed virtual lab architecture built on OpenVPN. By combining Python based bots, a centralized Flask Command and Control (C2) dashboard and a separate target server, the team recreated realistic malware behavior including polling, command execution and simulated attacks such as DDoS and scanning within a fully isolated, ethically contained environment.

Though some modules and components were not fully production stable, the project demonstrated all core functionality of a basic botnet infrastructure. It also highlighted how such systems can be developed, coordinated and tested collaboratively, even across geographically distributed machines.

The late stage switch from local VirtualBox networking to a VPN based setup was challenging but ultimately made the simulation more realistic and educational. The team gained valuable exposure to systems integration, malware simulation and network level behavior of botnets all under strict containment and responsible usage.

Future Work

To enhance the system’s depth, security realism and analysis capabilities, the following improvements are proposed:

- **Encrypted Communication:**
  Upgrade bot to C2 traffic to use HTTPS or AES encrypted payloads to simulate stealthier malware.
- **Bot Hardening and Stealth:**
  Implement randomized user agents, obfuscation, or sandbox detection to improve realism.

- **Real Time Dashboard Enhancements:**
  Add dynamic graphs and logs for command response time, polling frequency and bot health.
- **Lightweight Bot Deployment:**
  Replace full VMs with Docker or LXC containers to simulate 10–20 bots at scale.
- **Advanced Traffic Analysis:**
  Use tools like Suricata or Zeek to generate alerts and anomaly patterns for traffic generated by the simulated botnet.
- **Machine Learning Integration:**
  Train a simple ML model to detect botnet behavior using features from captured .pcap files.
- **Propagation Simulation:**
  Implement safe, local network based replication (e.g., scanning and infecting other simulated machines within VPN).

---

9 . Appendix

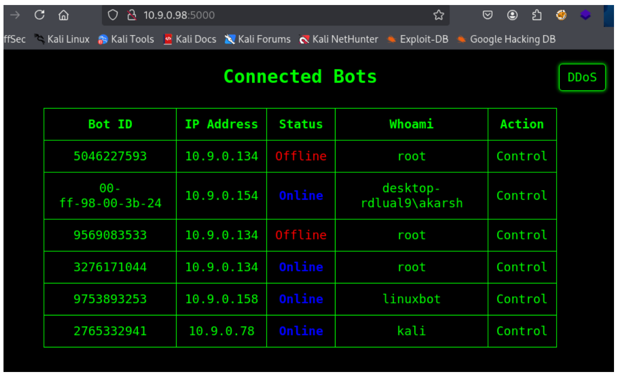
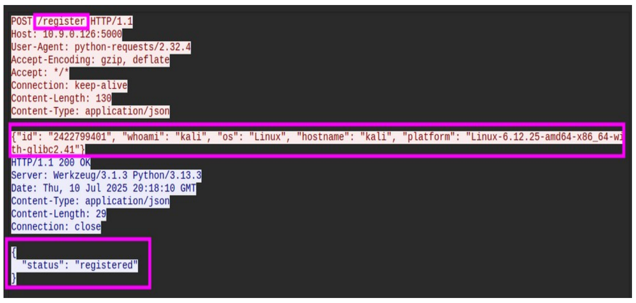
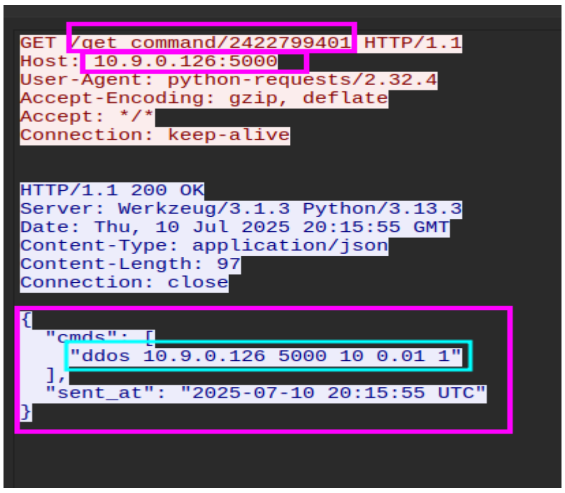
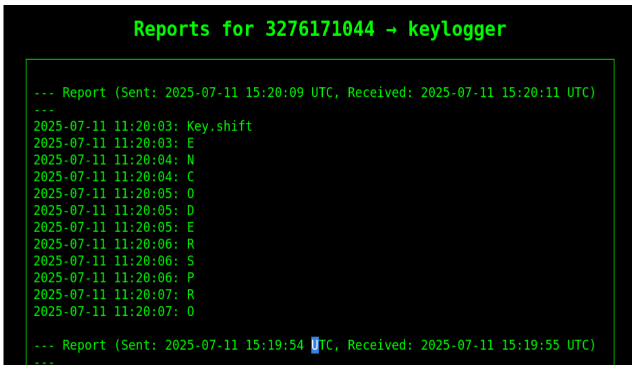
This section contains supporting materials, references, logs and contribution details that supplement the main content of the project.

A. Sample Bot Modules

Module	**Description**
net_scan	Scans the network of the target host
port_scan	Scans for open ports
ddos	Starts distributed DoS attack from selected bots
keylogger	Captures keys
stealer	Harvests stored credentials
spyware	Captures screenshots & and clipboard.
bruteforce	Performs bruteforce attacks on the given ip and port

B. Screenshot Index

*The screenshots are taken while different phases of testing.

Screenshot	Description
	C2 Dashboard
	Wireshark Analysis 1. Here the bot has polled the C2 server for registration included the data like username, os info, platform. 2. Server replies with the status, in this case ‘registered’.
	Wireshark Analysis 1. The infected bot polls server for commands/modules to run. 2. The server sends the order to run ddos module with the data as ‘target_ip’, ‘target_port’, ‘no_of_packets’, ‘delay’, ‘threads’. (Here the ip is same as the c2 server, because the screenshot was taken while the testing phase.)
	Report for keylogger module.

10.9.0.98:5000

Connected Bots

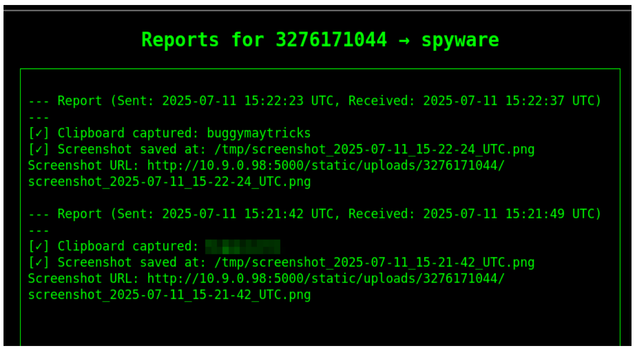
Bot ID	IP Address	Status	Whoami	Action
5046227593	10.9.0.134	Offline	root	Control
00-ff-98-00-3b-24	10.9.0.154	Online	desktop-rdlual9\akarsh	Control
9569083533	10.9.0.134	Offline	root	Control
3276171044	10.9.0.134	Online	root	Control
9753893253	10.9.0.158	Online	linuxbot	Control
2765332941	10.9.0.78	Online	kali	Control

```
POST /register HTTP/1.1
Host: 10.9.0.126:5000
User-Agent: python-requests/2.32.4
Accept-Encoding: gzip, deflate
Accept: */*
Connection: keep-alive
Content-Length: 130
Content-Type: application/json

{"id": "2422799401", "whoami": "kali", "os": "Linux", "hostname": "kali", "platform": "Linux-6.12.25-amd64-x86_64-wi
th-glibc2.41"}
HTTP/1.1 200 OK
Server: Werkzeug/3.1.3 Python/3.13.3
Date: Thu, 10 Jul 2025 20:18:10 GMT
Content-Type: application/json
Content-Length: 29
Connection: close

{
  "status": "registered"
}
```

```
GET /get_command/2422799401 HTTP/1.1
Host: 10.9.0.126:5000
User-Agent: python-requests/2.32.4
Accept-Encoding: gzip, deflate
Accept: */*
Connection: keep-alive

HTTP/1.1 200 OK
Server: Werkzeug/3.1.3 Python/3.13.3
Date: Thu, 10 Jul 2025 20:15:55 GMT
Content-Type: application/json
Content-Length: 97
Connection: close

{
  "cmds": [
    "ddos 10.9.0.126 5000 10 0.01 1"
  ],
  "sent_at": "2025-07-10 20:15:55 UTC"
}
```

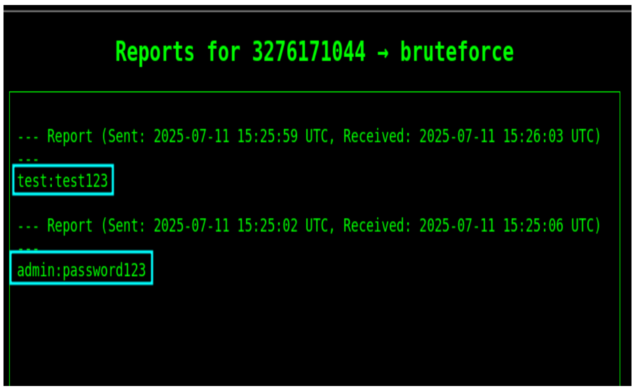
Reports for 3276171044 → keylogger

```
--- Report (Sent: 2025-07-11 15:20:09 UTC, Received: 2025-07-11 15:20:11 UTC)
---
2025-07-11 11:20:03: Key.shift
2025-07-11 11:20:03: E
2025-07-11 11:20:04: N
2025-07-11 11:20:04: C
2025-07-11 11:20:05: O
2025-07-11 11:20:05: D
2025-07-11 11:20:05: E
2025-07-11 11:20:06: R
2025-07-11 11:20:06: S
2025-07-11 11:20:06: P
2025-07-11 11:20:07: R
2025-07-11 11:20:07: O

--- Report (Sent: 2025-07-11 15:19:54 UTC, Received: 2025-07-11 15:19:55 UTC)
---
```



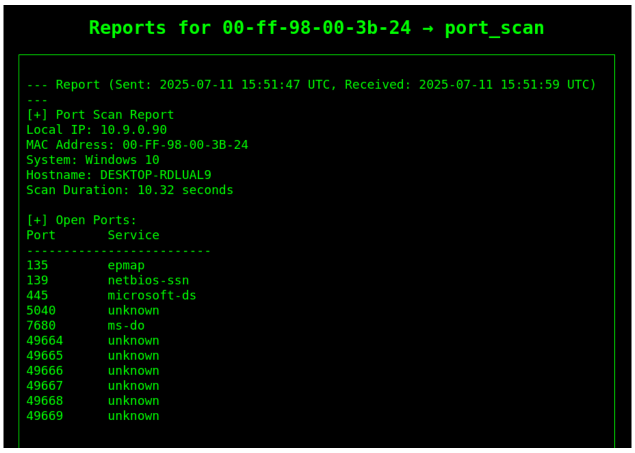
```
Reports for 3276171044 → spyware

--- Report (Sent: 2025-07-11 15:22:23 UTC, Received: 2025-07-11 15:22:37 UTC)
---
[✓] Clipboard captured: buggymaytricks
[✓] Screenshot saved at: /tmp/screenshot_2025-07-11_15-22-24_UTC.png
Screenshot URL: http://10.9.0.98:5000/static/uploads/3276171044/
screenshot_2025-07-11_15-22-24_UTC.png

--- Report (Sent: 2025-07-11 15:21:42 UTC, Received: 2025-07-11 15:21:49 UTC)
---
[✓] Clipboard captured: [redacted]
[✓] Screenshot saved at: /tmp/screenshot_2025-07-11_15-21-42_UTC.png
Screenshot URL: http://10.9.0.98:5000/static/uploads/3276171044/
screenshot_2025-07-11_15-21-42_UTC.png
```

Report for spyware module



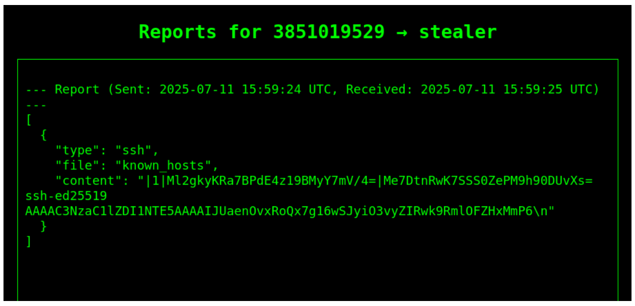
```
Reports for 3276171044 → bruteforce

--- Report (Sent: 2025-07-11 15:25:59 UTC, Received: 2025-07-11 15:26:03 UTC)
---
test:test123

--- Report (Sent: 2025-07-11 15:25:02 UTC, Received: 2025-07-11 15:25:06 UTC)
---
admin:password123
```

Report for bruteforce module on the custom target website.



```
Reports for 00-ff-98-00-3b-24 → port_scan

--- Report (Sent: 2025-07-11 15:51:47 UTC, Received: 2025-07-11 15:51:59 UTC)
---
[+] Port Scan Report
Local IP: 10.9.0.90
MAC Address: 00-FF-98-00-3B-24
System: Windows 10
Hostname: DESKTOP-RDLUAL9
Scan Duration: 10.32 seconds

[+] Open Ports:
Port       Service
-------------------------
135        epmap
139        netbios-ssn
445        microsoft-ds
5040       unknown
7680       ms-do
49664      unknown
49665      unknown
49666      unknown
49667      unknown
49668      unknown
49669      unknown
```

Report for the port_scan module



```
Reports for 3851019529 → stealer

--- Report (Sent: 2025-07-11 15:59:24 UTC, Received: 2025-07-11 15:59:25 UTC)
---
[
  {
    "type": "ssh",
    "file": "known_hosts",
    "content": "|1|Ml2gkyKRa7BPdE4z19BMyY7mV/4=|Me7DtnRwK7SSS0ZePM9h90DUvXs=
ssh-ed25519
AAAAC3NzaC1lZDI1NTE5AAAAIJUaenOvxRoQx7g16wSJyiO3vyZIRwk9RmlOFZHxMmP6\n"
  }
]
```

Report for the stealer module.

C. Team Contributions

Team Member	**Responsibilities/Contributions**
Pranav More (Lead)	• Team Collaboration • Stealer Module • Network Scan Module • VM Setup • OpenVPN Setup • Finalizing Documents • Testing • Github Repo Management • Wire Shark Analysis • Project Completion Report
Hemanth Kumar	• Port Scan Module • Integrated support modules with the bot.py • VM Setup • Developed the Web Based C2 Server • Scope of Work • Github Repo Management • Project Presentation • Target Website • Testing
Akarsh Kumar Ojha	• Spyware Module • Keylogger Module • Target Website • VM Setup • Testing • Project Completion Report
Nishith Challa	• Brute Force Module • Presentation Draft
Dhoni	• DoS Module • VM Setup • Presentation Draft
Madhuri Sadgir	• Keylogger Module • Presentation Draft

🔗 E. GitHub Repository

D. Tools and Libraries Used

- Python3
  - Flask, requests, subprocess, threading, etc
- Wireshark (packet analysis)
- VirtualBox, VMWare
  OpenVPN (server + clients)
- Linux (Ubuntu, Kali, MXLinux, Debian), Windows 10 VMs
