

Report: Dynamic DNS Sinkhole System

Project Title: Dynamic DNS Sinkhole System

Submitted To: Selin Tor

Date: May 26,2025

1. Introduction

The Domain Name System (DNS) is a foundational component of network communication, translating human-readable domain names into IP addresses. However, DNS is often exploited by cybercriminals for malicious activities, including data exfiltration, malware distribution, and command-and-control (C2) operations. These threats often evade traditional defenses due to dynamic domain generation and the lack of context-aware filtering.

This project addresses these challenges by implementing a **behavior-based DNS sinkhole system** that enforces **Zero Trust** principles. The system continuously monitors DNS queries and evaluates domains using behavioural indicators rather than static blacklists.

Key Drivers for This Project:

- **DNS Abuse Prevention:** Mitigate threats originating from malicious or algorithmically generated domains.
- **Zero Trust Enforcement:** Validate every domain query, regardless of origin or trust level.
- **Real-Time Response:** Automatically redirect suspicious domains to a local sinkhole (127.0.0.1) without manual intervention.
- **Open-Source Ecosystem:** Built using Suricata IDS, Unbound DNS resolver, and a custom Python classifier.

This solution provides an autonomous, intelligent DNS defense aligned with modern cybersecurity principles.

2. Objective:

In the modern digital landscape, domain name system (DNS) traffic is frequently exploited by malicious actors to initiate command-and-control (C2) communications, data exfiltration, or

the distribution of malware through dynamically generated domains. Traditional security mechanisms often fail to detect or mitigate such threats due to their reliance on static blacklists and signature-based models. This project aims to design and implement a dynamic DNS sinkhole system that enforces Zero Trust principles by identifying and blocking suspicious domains based on real-time behavioural analysis.

The core objective is to build an automated solution using open-source tools—Suricata IDS for DNS traffic monitoring, Unbound as the DNS resolver, and a custom Python daemon that classifies domains based on entropy, query frequency, and TTL anomalies. Domains deemed suspicious are redirected to a local sinkhole (127.0.0.1) to prevent further communication. This system provides an adaptive, autonomous DNS defense mechanism aligned with zero-trust security architecture.

2. System Components

2.1 Suricata IDS

- Monitors DNS queries on the system.
- Logs DNS traffic to `eve.json` in structured JSON format.

2.2 Unbound DNS Resolver

- Acts as the local DNS server.
- Configured to redirect malicious domains to 127.0.0.1.
- Uses a dynamic configuration file `sinkhole.conf`.

2.3 Python Daemon (`dns_classifier.py`)

- Parses Suricata logs.
- Analyzes domain behavior based on:
 - Frequency of queries
 - Time-to-Live (TTL) values
 - Entropy score
- Updates `sinkhole.conf` dynamically.
- Reloads Unbound to apply new sinkhole rules.

4. Methodology

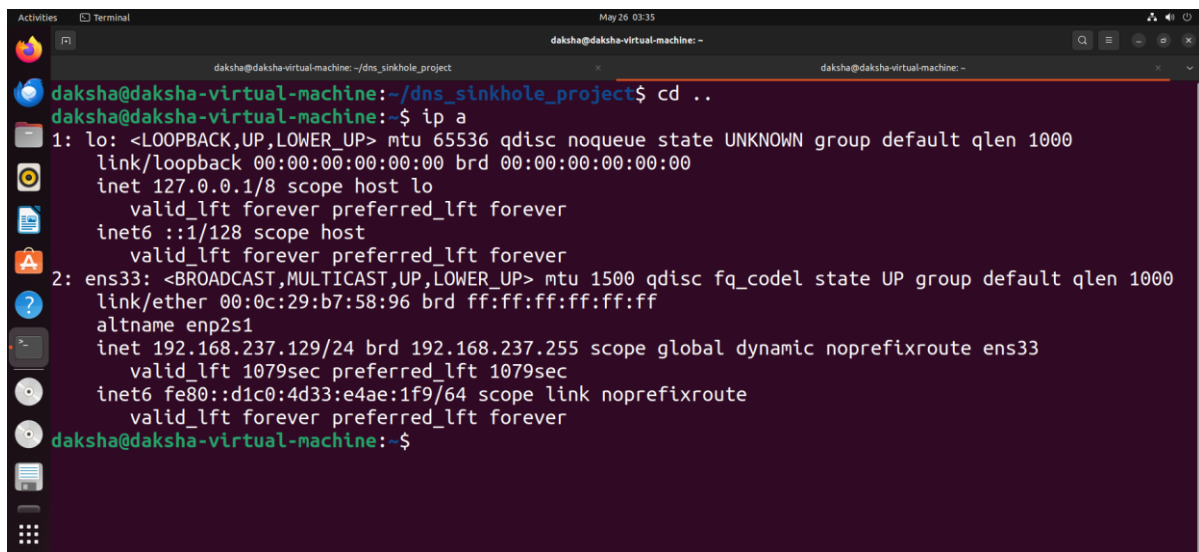
This section outlines each step followed to implement and validate the DNS sinkhole system. The process was executed on an Ubuntu VM with Suricata, Unbound, and Python.

Step 1: Network Interface Identification

Command:

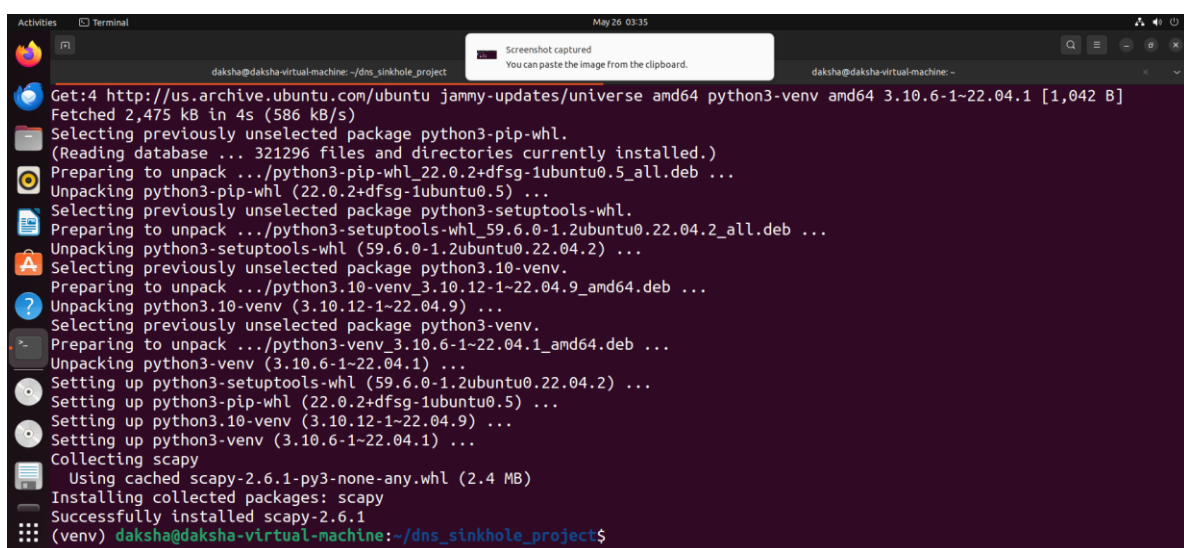
`ip a`

Purpose: To find the active interface (ens33) and loopback (lo) for DNS traffic monitoring.



```
daksha@daksha-virtual-machine:~/dns_sinkhole_project$ cd ..
daksha@daksha-virtual-machine:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:b7:58:96 brd ff:ff:ff:ff:ff:ff
    altname enp2s1
    inet 192.168.237.129/24 brd 192.168.237.255 scope global dynamic noprefixroute ens33
        valid_lft 1079sec preferred_lft 1079sec
    inet6 fe80::d1c0:4d33:e4ae:1f9/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
daksha@daksha-virtual-machine:~$
```

(Finding out network interfaces)



```
Get:4 http://us.archive.ubuntu.com/ubuntu jammy-updates/universe amd64 python3-venv amd64 3.10.6-1~22.04.1 [1,042 B]
Fetched 2,475 kB in 4s (586 kB/s)
Selecting previously unselected package python3-pip-whl.
(Reading database ... 321296 files and directories currently installed.)
Preparing to unpack .../python3-pip-whl_22.0.2+dfsg-1ubuntu0.5_all.deb ...
Unpacking python3-pip-whl (22.0.2+dfsg-1ubuntu0.5) ...
Selecting previously unselected package python3-setuptools-whl.
Preparing to unpack .../python3-setuptools-whl_59.6.0-1.2ubuntu0.22.04.2_all.deb ...
Unpacking python3-setuptools-whl (59.6.0-1.2ubuntu0.22.04.2) ...
Selecting previously unselected package python3.10-venv.
Preparing to unpack .../python3.10-venv_3.10.12-1~22.04.9_amd64.deb ...
Unpacking python3.10-venv (3.10.12-1~22.04.9) ...
Selecting previously unselected package python3-venv.
Preparing to unpack .../python3-venv_3.10.6-1~22.04.1_amd64.deb ...
Unpacking python3-venv (3.10.6-1~22.04.1) ...
Setting up python3-setuptools-whl (59.6.0-1.2ubuntu0.22.04.2) ...
Setting up python3-pip-whl (22.0.2+dfsg-1ubuntu0.5) ...
Setting up python3.10-venv (3.10.12-1~22.04.9) ...
Setting up python3-venv (3.10.6-1~22.04.1) ...
Collecting scapy
  Using cached scapy-2.6.1-py3-none-any.whl (2.4 MB)
Installing collected packages: scapy
Successfully installed scapy-2.6.1
(venv) daksha@daksha-virtual-machine:~/dns_sinkhole_project$
```

(Installing scapy)

Step 2: Installing and Configuring Unbound DNS

- Installed via APT.
- Configured to listen on 127.0.0.1 and accept local queries.
- Sinkhole zones placed in /etc/unbound/unbound.conf.d/sinkhole.conf

```
(venv) daksha@daksha-virtual-machine:~/dns_sinkhole_project$ sudo systemctl status unbound
● unbound.service - Unbound DNS server
   Loaded: loaded (/lib/systemd/system/unbound.service; enabled; vendor preset: enabled)
   Active: active (running) since Mon 2025-05-26 03:43:34 EDT; 1min 3s ago
     Docs: man:unbound(8)
   Process: 5916 ExecStartPre=/usr/lib/unbound/package-helper chroot_setup (code=exited, status=0/SUCCESS)
   Process: 5919 ExecStartPre=/usr/lib/unbound/package-helper root_trust_anchor_update (code=exited, status=0/SUCCESS)
  Main PID: 5922 (unbound)
    Tasks: 1 (limit: 2211)
   Memory: 7.3M
      CPU: 817ms
   CGroup: /system.slice/unbound.service
           └─5922 /usr/sbin/unbound -d -p

May 26 03:43:33 daksha-virtual-machine systemd[1]: Starting Unbound DNS server...
May 26 03:43:34 daksha-virtual-machine unbound[5922]: [5922:0] notice: init module 0: subnet
May 26 03:43:34 daksha-virtual-machine unbound[5922]: [5922:0] notice: init module 1: validator
May 26 03:43:34 daksha-virtual-machine unbound[5922]: [5922:0] notice: init module 2: iterator
May 26 03:43:34 daksha-virtual-machine systemd[1]: Started Unbound DNS server.
May 26 03:43:34 daksha-virtual-machine unbound[5922]: [5922:0] info: start of service (unbound 1.13.1).
(venv) daksha@daksha-virtual-machine:~/dns_sinkhole_project$
```

(Unbound Active Running Status)

Step 3: Testing Unbound Resolver

Command:

`dig bad-domain.test @127.0.0.1`

Expected: Redirected to 127.0.0.1 if sinkholed

```
(venv) daksha@daksha-virtual-machine:~/dns_sinkhole_project$ dig bad-domain.test @127.0.0.1

; <<>> DiG 9.18.30-0ubuntu0.22.04.2-Ubuntu <<>> bad-domain.test @127.0.0.1
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 46289
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
;; QUESTION SECTION:
;bad-domain.test.                IN      A

;; ANSWER SECTION:
bad-domain.test.                3600    IN      A      127.0.0.1

;; Query time: 13 msec
;; SERVER: 127.0.0.1#53(127.0.0.1) (UDP)
;; WHEN: Mon May 26 03:43:44 EDT 2025
;; MSG SIZE rcvd: 60
```

(Testing Unbound Resolver)

Step 4: Installing and Configuring Suricata

Edited /etc/suricata/suricata.yaml to enable DNS logging.

Enabled:

dns:

enabled: yes

version: 2

tcp: yes

udp: yes

```
GNU nano 6.2 /etc/suricata/suricata.yaml *
- http:
  extended: yes      # enable this for extended logging information
  # custom allows additional HTTP fields to be included in eve-log.
  # the example below adds three additional fields when uncommented
  #custom: [Accept-Encoding, Accept-Language, Authorization]
  # set this value to one and only one from {both, request, response}
  # to dump all HTTP headers for every HTTP request and/or response
  # dump-all-headers: none
- dns:
  # This configuration uses the new DNS logging format,
  # the old configuration is still available:
  # https://suricata.readthedocs.io/en/latest/output/eve/eve-json-output.html#dns-v1-format

  # As of Suricata 5.0, version 2 of the eve dns output
  # format is the default.
  version: 2
  enabled: yes
  tcp: yes
  udp: yes
```

(Enabling DNS logging)

```
GNU nano 6.2 /etc/suricata/suricata.yaml *
#filetype: regular # 'regular', 'unix_stream' or 'unix_dgram'

# Extensible Event Format (nicknamed EVE) event log in JSON format
- eve-log:
  enabled: yes
  filetype: regular
  filename: /var/log/suricata/eve.json
  types:
    - dns
    - alert
    - flow
    - http
  # Enable for multi-threaded eve.json output; output files are amended with
  # with an identifier, e.g., eve.9.json
  #threaded: false
  #prefix: "@cee: " # prefix to prepend to each log entry
  # the following are valid when type: syslog above
  #identity: "suricata"
  #facility: local5
  #level: Info ## possible levels: Emergency, Alert, Critical,
```

Step 5: Verifying Suricata Logs

Command:

tail -f /var/log/suricata/eve.json | grep dns

```
(venv) daksha@daksha-virtual-machine:~/dns_sinkhole_project$ sudo suricata -T -c /etc/suricata/suricata.yaml -v
26/5/2025 -- 04:09:55 - <Info> - Running suricata under test mode
26/5/2025 -- 04:09:55 - <Info> - Configuration node 'types' redefined.
26/5/2025 -- 04:09:55 - <Notice> - This is Suricata version 6.0.4 RELEASE running in SYSTEM mode
26/5/2025 -- 04:09:55 - <Info> - CPUs/cores online: 2
26/5/2025 -- 04:09:56 - <Info> - fast output device (regular) initialized: fast.log
26/5/2025 -- 04:09:56 - <Info> - eve-log output device (regular) initialized: /var/log/suricata/eve.json
26/5/2025 -- 04:09:56 - <Info> - stats output device (regular) initialized: stats.log
```

(Testing suricata for event logging)

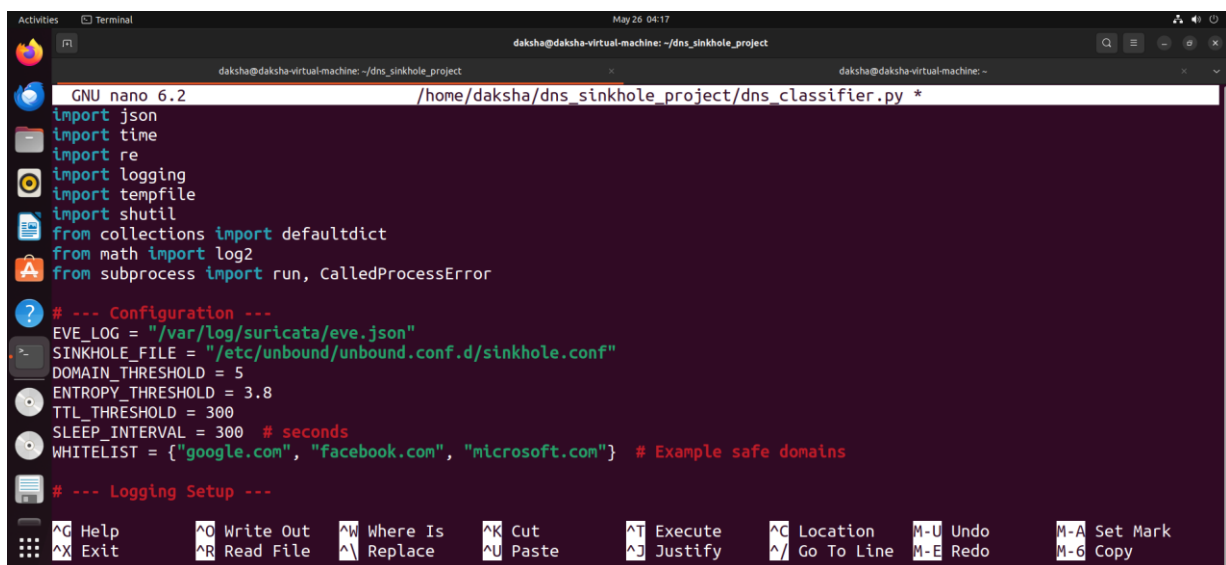
```
daksha@daksha-virtual-machine:~/dns_sinkhole_project$ tail -f dns_classifier.log
2025-05-26 04:38:15,991 [INFO] Suspicious domain detected: com (Count=12, TTL=0.00, Entropy=1.58)
2025-05-26 04:38:15,992 [INFO] Suspicious domain detected: ubuntu.com (Count=8, TTL=0.00, Entropy=2.85)
2025-05-26 04:38:16,013 [INFO] Sinkhole file updated with 7 domains.
2025-05-26 04:38:17,416 [INFO] Unbound DNS reloaded successfully.
2025-05-26 04:38:23,551 [INFO] Daemon interrupted and stopped.
2025-05-26 04:40:27,229 [INFO] Started behavioral DNS sinkhole daemon.
2025-05-26 04:40:27,298 [INFO] Suspicious domain detected: connectivity-check.ubuntu.com (Count=9, TTL=0.00, Entropy=3.69)
2025-05-26 04:40:27,298 [INFO] Suspicious domain detected: com (Count=6, TTL=0.00, Entropy=1.58)
2025-05-26 04:40:27,299 [INFO] Sinkhole file updated with 7 domains.
2025-05-26 04:40:27,517 [INFO] Unbound DNS reloaded successfully.
```

(Verifying Suricata Logs)

Step 6: Building the Python Daemon

Created dns_classifier.py to:

- Parse eve.json
- Score domains
- Update sinkhole config
- Reload Unbound



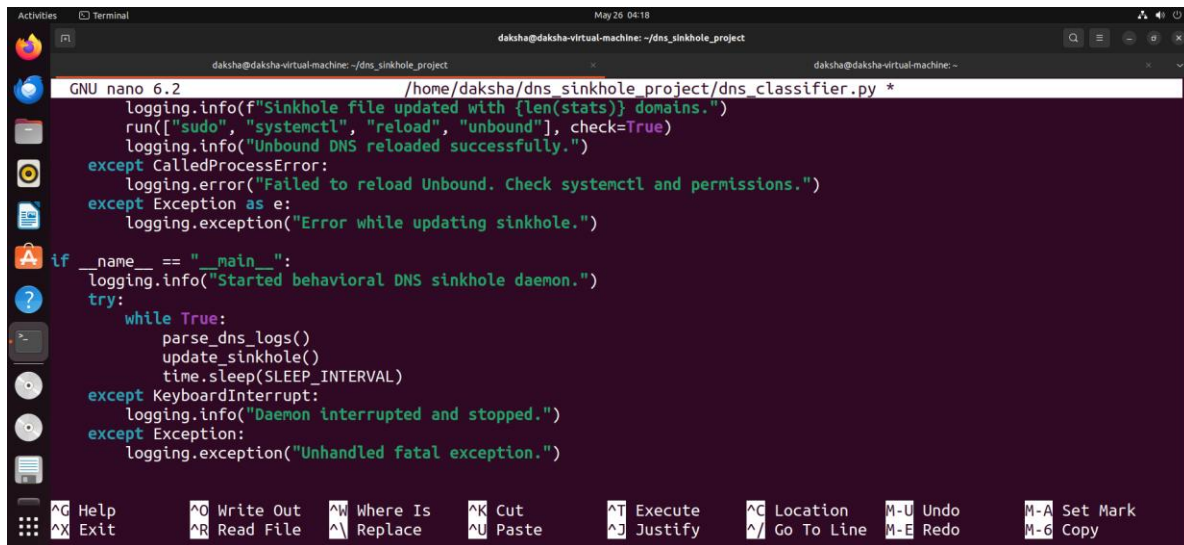
```
GNU nano 6.2 /home/daksha/dns_sinkhole_project/dns_classifier.py *
import json
import time
import re
import logging
import tempfile
import shutil
from collections import defaultdict
from math import log2
from subprocess import run, CalledProcessError

# --- Configuration ---
EVE_LOG = "/var/log/suricata/eve.json"
SINKHOLE_FILE = "/etc/unbound/unbound.conf.d/sinkhole.conf"
DOMAIN_THRESHOLD = 5
ENTROPY_THRESHOLD = 3.8
TTL_THRESHOLD = 300
SLEEP_INTERVAL = 300 # seconds
WHITELIST = {"google.com", "facebook.com", "microsoft.com"} # Example safe domains

# --- Logging Setup ---

Help Write Out Where Is Cut Execute Location M-U Undo M-A Set Mark
Exit Read File Replace NU Paste J Justify / Go To Line M-E Redo M-6 Copy
```

(Building the Python Daemon)



```
GNU nano 6.2 /home/daksha/dns_sinkhole_project/dns_classifier.py *
logging.info(f"Sinkhole file updated with {len(stats)} domains.")
run(["sudo", "systemctl", "reload", "unbound"], check=True)
logging.info("Unbound DNS reloaded successfully.")
except CalledProcessError:
    logging.error("Failed to reload Unbound. Check systemctl and permissions.")
except Exception as e:
    logging.exception("Error while updating sinkhole.")

if __name__ == "__main__":
    logging.info("Started behavioral DNS sinkhole daemon.")
    try:
        while True:
            parse_dns_logs()
            update_sinkhole()
            time.sleep(SLEEP_INTERVAL)
    except KeyboardInterrupt:
        logging.info("Daemon interrupted and stopped.")
    except Exception:
        logging.exception("Unhandled fatal exception.")

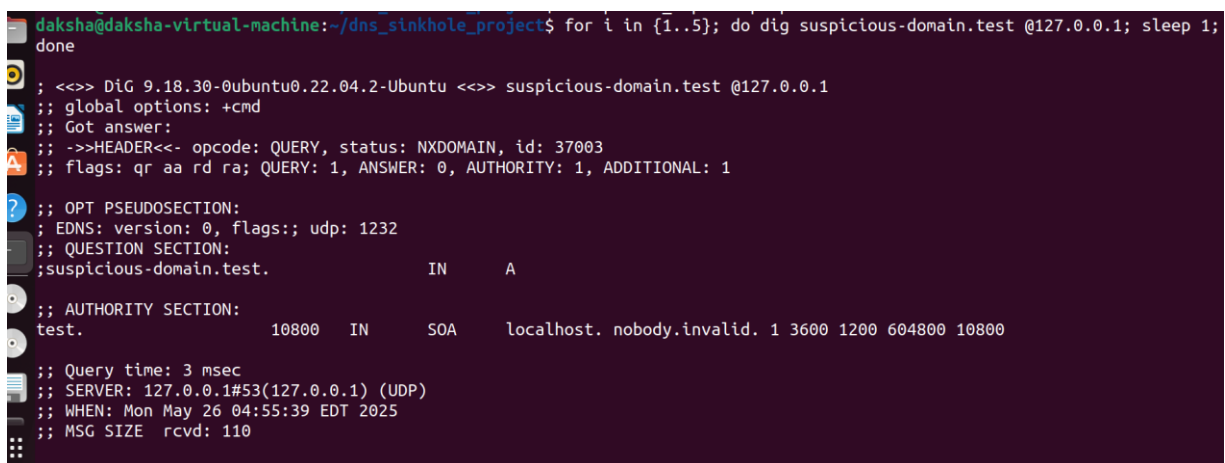
^G Help      ^O Write Out ^W Where Is  ^K Cut       ^T Execute   ^C Location  M-U Undo     M-A Set Mark
^X Exit      ^R Read File ^M Replace   ^U Paste     ^J Justify   ^/_ Go To Line M-E Redo     M-6 Copy
```

(Building the Python Daemon)

Step 7: Generating Suspicious DNS Activity

Command:

for i in {1..10}; do dig suspicious-domain.test @127.0.0.1; sleep 1; done



```
daksha@daksha-virtual-machine:~/dns_sinkhole_project$ for i in {1..5}; do dig suspicious-domain.test @127.0.0.1; sleep 1; done
; <<>> DiG 9.18.30-0ubuntu0.22.04.2-Ubuntu <<>> suspicious-domain.test @127.0.0.1
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 37003
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 1, ADDITIONAL: 1
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
;; QUESTION SECTION:
;suspicious-domain.test.                IN      A
;; AUTHORITY SECTION:
test. 10800 IN SOA localhost. nobody.invalid. 1 3600 1200 604800 10800
;; Query time: 3 msec
;; SERVER: 127.0.0.1#53(127.0.0.1) (UDP)
;; WHEN: Mon May 26 04:55:39 EDT 2025
;; MSG SIZE rcvd: 110
```

(Generating Suspicious DNS Activity)

Step 8: Confirming Detection

- Monitored dns_classifier.log
- Verified that suspicious domains were detected and sinkholed

```
daksha@daksha-virtual-machine:~/dns_sinkhole_project$ tail -f dns_classifier.log
2025-05-26 04:38:15,991 [INFO] Suspicious domain detected: com (Count=12, TTL=0.00, Entropy=1.58)
2025-05-26 04:38:15,992 [INFO] Suspicious domain detected: ubuntu.com (Count=8, TTL=0.00, Entropy=2.85)
2025-05-26 04:38:16,013 [INFO] Sinkhole file updated with 7 domains.
2025-05-26 04:38:17,416 [INFO] Unbound DNS reloaded successfully.
2025-05-26 04:38:23,551 [INFO] Daemon interrupted and stopped.
2025-05-26 04:40:27,229 [INFO] Started behavioral DNS sinkhole daemon.
2025-05-26 04:40:27,298 [INFO] Suspicious domain detected: connectivity-check.ubuntu.com (Count=9, TTL=0.00, Entropy=3.69)
2025-05-26 04:40:27,298 [INFO] Suspicious domain detected: com (Count=6, TTL=0.00, Entropy=1.58)
2025-05-26 04:40:27,299 [INFO] Sinkhole file updated with 7 domains.
2025-05-26 04:40:27,517 [INFO] Unbound DNS reloaded successfully.
```

(Confirming Detection)

Step 9: Capturing Traffic with tcpdump

Command:

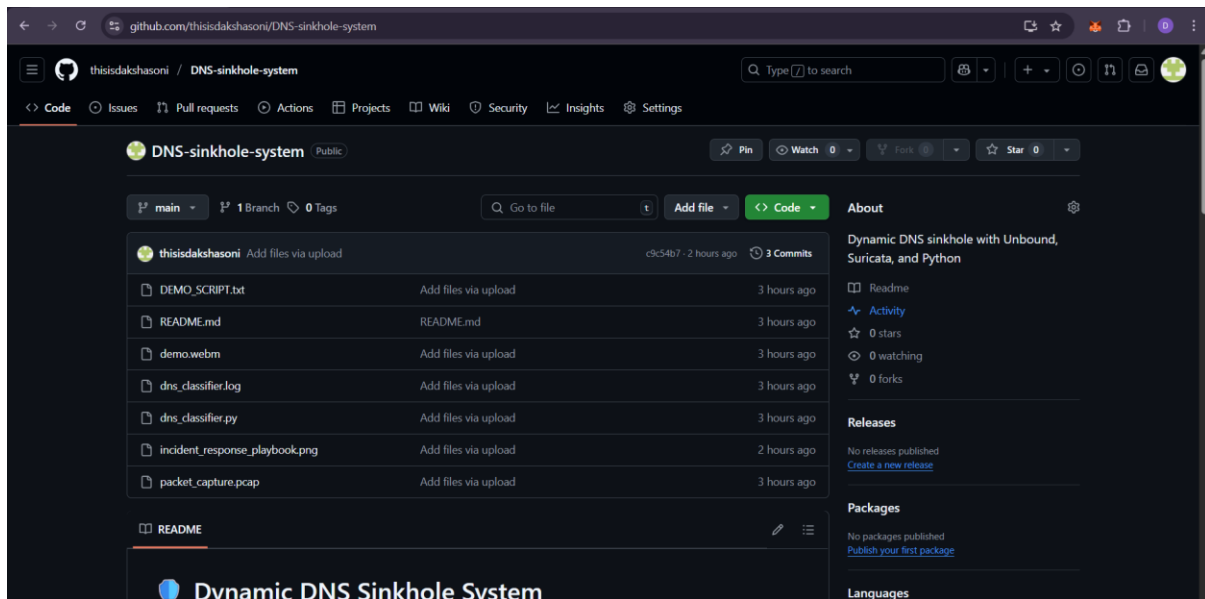
```
sudo tcpdump -i lo port 53 -w packet_capture.pcap
```

```
0 packets dropped by kernel
daksha@daksha-virtual-machine:~/dns_sinkhole_project$ sudo tcpdump -i lo port 53 -w packet_capture.pcap
tcpdump: listening on lo, link-type EN10MB (Ethernet), snapshot length 262144 bytes
^C14 packets captured
28 packets received by filter
0 packets dropped by kernel
daksha@daksha-virtual-machine:~/dns_sinkhole_project$ =
```

(Capturing Traffic with tcpdump)

Step 10: Packaging Deliverables

- Created ZIP and GitHub repo
- Included:
 - Source code
 - PCAP
 - Video
 - Logs
 - Demo script
 - Incident Response diagram



5. Recommendations

- Automate Suricata startup using a systemd service or cron job to ensure continuous monitoring on reboot.
- Implement a dashboard or CLI report summarizing domain detections, thresholds crossed, and frequency trends for easier analysis.
- Whitelist critical domains (e.g., system update servers) to avoid blocking legitimate traffic like connectivity-check.ubuntu.com.
- Enhance domain classification by integrating external threat intelligence feeds (e.g., abuse.ch, Cisco Umbrella, etc.).
- Integrate with SIEM platforms like Splunk or ELK to centralize alerting and enable historical pattern correlation.
- Persist historical domain scores in a local database (e.g., SQLite) to track domain evolution over time.
- Export sinkhole logs to a CSV or JSON format for compliance auditing or threat reporting.
- Use machine learning to improve anomaly detection (e.g., random forest or isolation forest for DGA detection).

- Add a manual override system to force-block or force-allow domains based on analyst review.
- Harden Unbound configuration further with DNSSEC, rate limiting, and upstream DNS resolver authentication.

6. Conclusion

The Dynamic DNS Sinkhole System demonstrates a practical implementation of Zero Trust security principles at the DNS level. By combining open-source tools like Suricata for real-time traffic monitoring and Unbound for DNS resolution, the system provides autonomous detection and redirection of suspicious domain activity. The Python-based classifier adds intelligence through behavioral analysis using entropy, TTL, and frequency metrics—enabling proactive mitigation of potential threats such as botnet command-and-control domains or DGA-generated traffic. The modular design ensures adaptability to varied environments and future integrations with threat intelligence feeds or SIEM platforms. Testing confirmed the system's ability to dynamically update DNS blocklists and respond to anomalous behavior within seconds. This project not only strengthens network defense posture but also emphasizes the critical role DNS-layer controls can play in modern cybersecurity. With additional enhancements, it could scale into a lightweight yet powerful network defense component for enterprise and cloud-native environments.