Report: DNS Sinkhole for Domain-Based Threat Hunting

1. Introduction

Cyber adversaries commonly exploit the Domain Name System (DNS) to enable various stages of an attack lifecycle, including malware distribution, command-and-control (C2) communication, and data exfiltration. Due to the fundamental and trusted nature of DNS, such malicious activities often remain undetected by traditional perimeter defenses.

A **DNS sinkhole** is a proactive cybersecurity mechanism used to intercept DNS queries for known malicious domains and redirect them to a non-routable address (e.g., localhost). This approach allows organizations to detect, log, and block potentially harmful domain resolutions, thereby impeding attacker operations and identifying infected hosts within a network.

This task involved the design and deployment of a cloud-hosted DNS sinkhole server using BIND9 on an Ubuntu-based AWS EC2 instance. The solution includes query logging, real-time domain pattern detection using Python, automated blocking of domains, and security alerting through AWS SNS. The implementation aligns with industry best practices and provides an operational foundation for DNS-based threat detection and response.

2. Objective

The primary goals of this task were as follows:

- Deploy and configure a DNS server (BIND9) to act as a sinkhole.
- Enable real-time DNS logging and monitoring of outbound domain queries.
- Detect suspicious domain patterns (e.g., phishing, malware C2) using log analysis.
- Dynamically update the DNS blacklist to block malicious domains.

- Send automated alerts to the security team using AWS Simple Notification Service (SNS).
- Schedule the domain detection and blocking mechanism for continuous protection.

This solution aims to enhance organizational threat visibility, support incident response, and disrupt domain-based attack vectors.

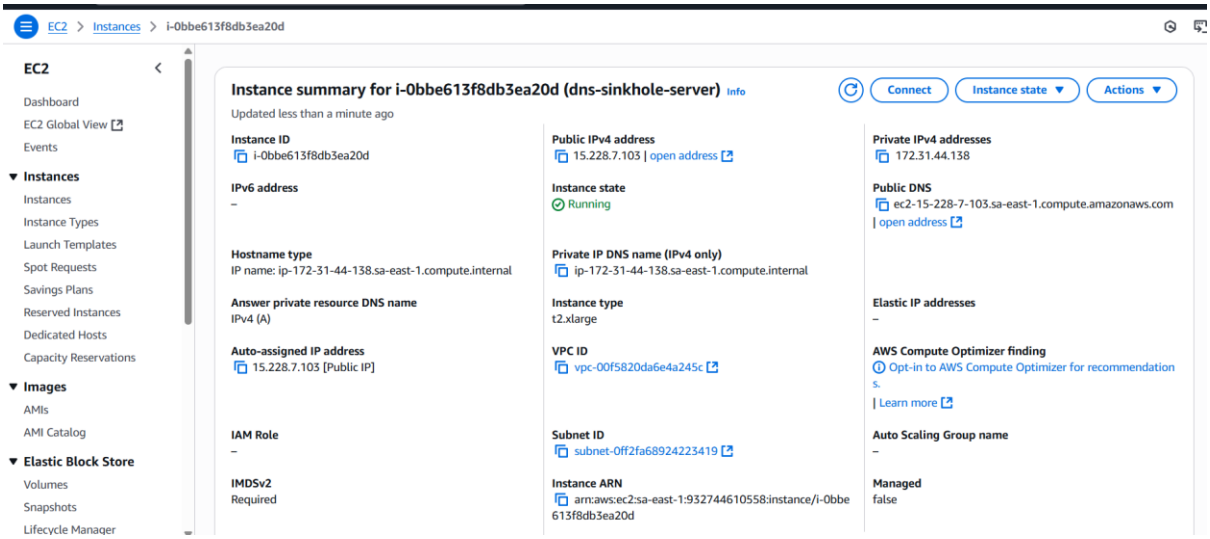
3. Tools and Technologies Used

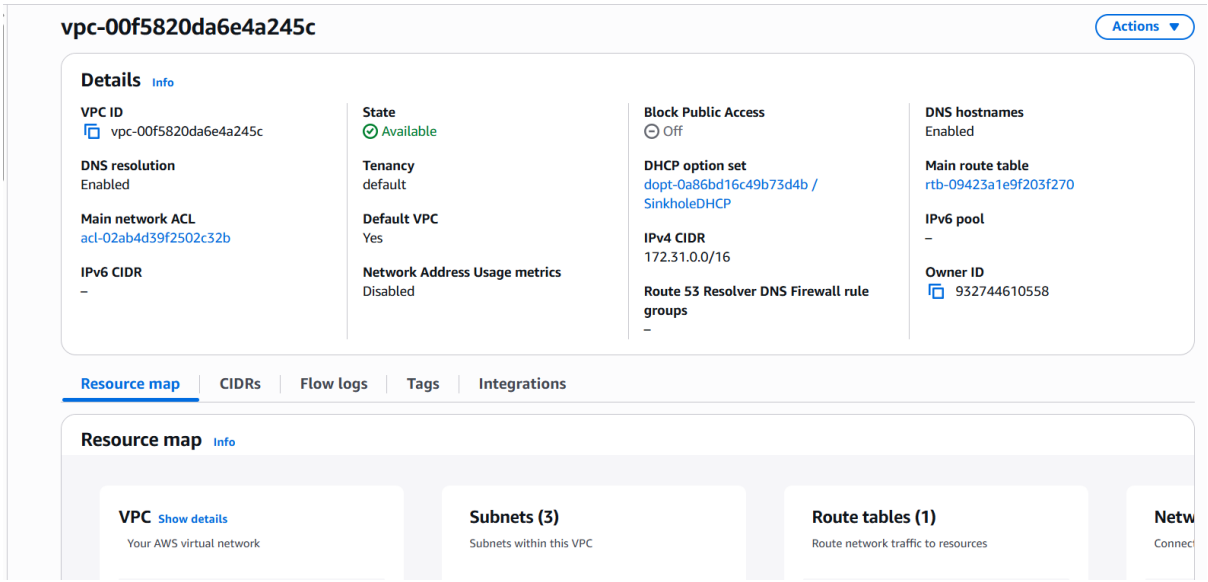
Tool / Technology	Purpose
Ubuntu 22.04 (EC2)	Host operating system for the DNS sinkhole server.
BIND9	DNS server software used to implement the sinkhole and custom zones.
Python 3	Used to write detection and automation scripts.
Crontab	Used to schedule regular execution of detection scripts.
Boto3 (AWS SDK)	Enables communication with AWS SNS from Python.
AWS SNS	Sends alerts to subscribers (email/SMS) when suspicious domains are found.
Regular Expressions (Regex)	Used to extract queried domain names from BIND9 logs.

4. Methodology

4.1. Environment Setup

- Launched an Ubuntu 22.04 EC2 instance in AWS, ensuring port 53 (UDP/TCP) was open to internal VPC systems.





(Vcp dashboard)

- Installed BIND9 and its utilities:

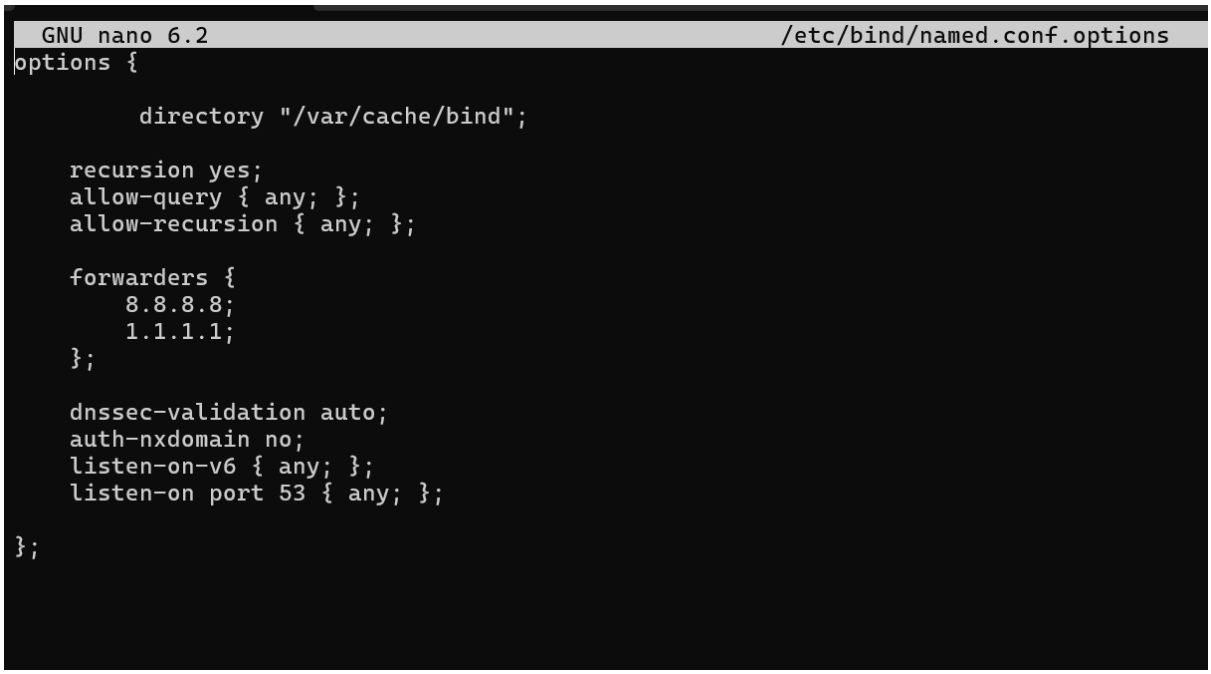
sudo apt update

sudo apt install bind9 bind9utils bind9-doc -y

4.2. BIND9 Configuration

Recursive Resolver Setup

- Edited /etc/bind/named.conf.options:

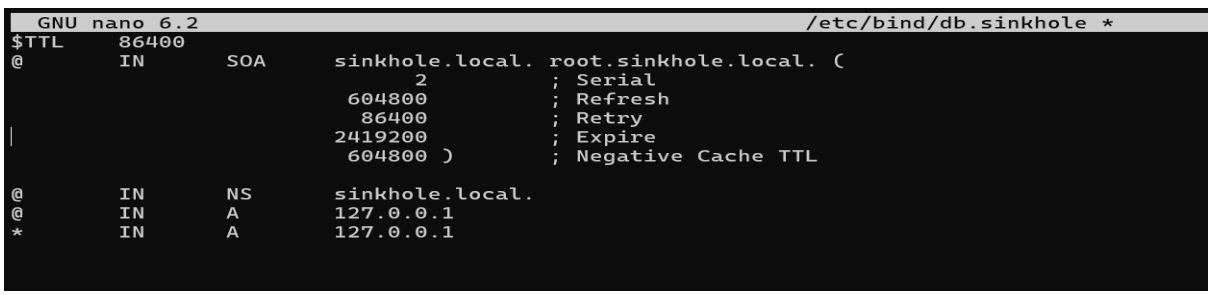


Query Logging Configuration

- Enabled DNS query logging:
- Created the log file and set permissions:

4.3. Sinkhole Zone Setup

Default Sinkhole Zone File



Named Zone Inclusion

```
 GNU nano 6.2                                    /etc/bind/named.conf.local *
zone "badexample.com" {
    type master;
    file "/etc/bind/db.sinkhole";
};

zone "maliciousdomain.net" {
    type master;
    file "/etc/bind/db.sinkhole";
};
```

4.4. Python-Based Detection Script

- Created detect_block.py to:
    - Read /var/log/bind9/query.log.
    - Extract domain names using regex.
    - Match against suspicious keywords (bad, malicious, phish).
    - Add new malicious domains to /etc/bind/named.conf.local.

```
 GNU nano 6.2                                    /etc/bind/named.conf.options *
options {
    directory "/var/cache/bind";

    recursion yes;
    allow-query { any; };
    allow-recursion { any; };

    forwarders {
        8.8.8.8;
        1.1.1.1;
    };

    dnssec-validation auto;
    auth-nxdomain no;
    listen-on-v6 { any; };
    listen-on port 53 { any; };

    logging {
        channel query_logging {
            file "/var/log/bind9/query.log";
            severity info;
        };
        category queries {
            query_logging;
        };
    };

};
```

    - Restart BIND9 to apply blocks.

```
root@ip-172-31-44-138:/home/ubuntu# sudo systemctl enable named
Synchronizing state of named.service with SysV service script with /lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable named
root@ip-172-31-44-138:/home/ubuntu# sudo systemctl start named
root@ip-172-31-44-138:/home/ubuntu# sudo systemctl status named
● named.service - BIND Domain Name Server
     Loaded: loaded (/lib/systemd/system/named.service; enabled; vendor preset: enabled)
     Active: active (running) since Thu 2025-05-22 10:41:40 UTC; 1min 5s ago
       Docs: man:named(8)
   Main PID: 15471 (named)
      Tasks: 14 (limit: 19168)
     Memory: 7.5M
        CPU: 30ms
     CGroup: /system.slice/named.service
             └─15471 /usr/sbin/named -u bind

May 22 10:41:40 ip-172-31-44-138 named[15471]: network unreachable resolving './DNSKEY/IN': 2001:500:9f::42#53
May 22 10:41:40 ip-172-31-44-138 named[15471]: network unreachable resolving './NS/IN': 2001:500:9f::42#53
May 22 10:41:40 ip-172-31-44-138 named[15471]: network unreachable resolving './DNSKEY/IN': 2001:500:2::c#53
May 22 10:41:40 ip-172-31-44-138 named[15471]: network unreachable resolving './NS/IN': 2001:500:2::c#53
May 22 10:41:40 ip-172-31-44-138 named[15471]: running
May 22 10:41:40 ip-172-31-44-138 systemd[1]: Started BIND Domain Name Server.
May 22 10:41:41 ip-172-31-44-138 named[15471]: network unreachable resolving './DNSKEY/IN': 2001:dc3::35#53
May 22 10:41:41 ip-172-31-44-138 named[15471]: network unreachable resolving './DNSKEY/IN': 2001:500:2f::f#53
May 22 10:41:41 ip-172-31-44-138 named[15471]: managed-keys-zone: Initializing automatic trust anchor management for zone '.'; DNSKEY>
May 22 10:41:41 ip-172-31-44-138 named[15471]: managed-keys-zone: Initializing automatic trust anchor management for zone '.'; DNSKEY>
```

- Send an alert via SNS with the domain list.

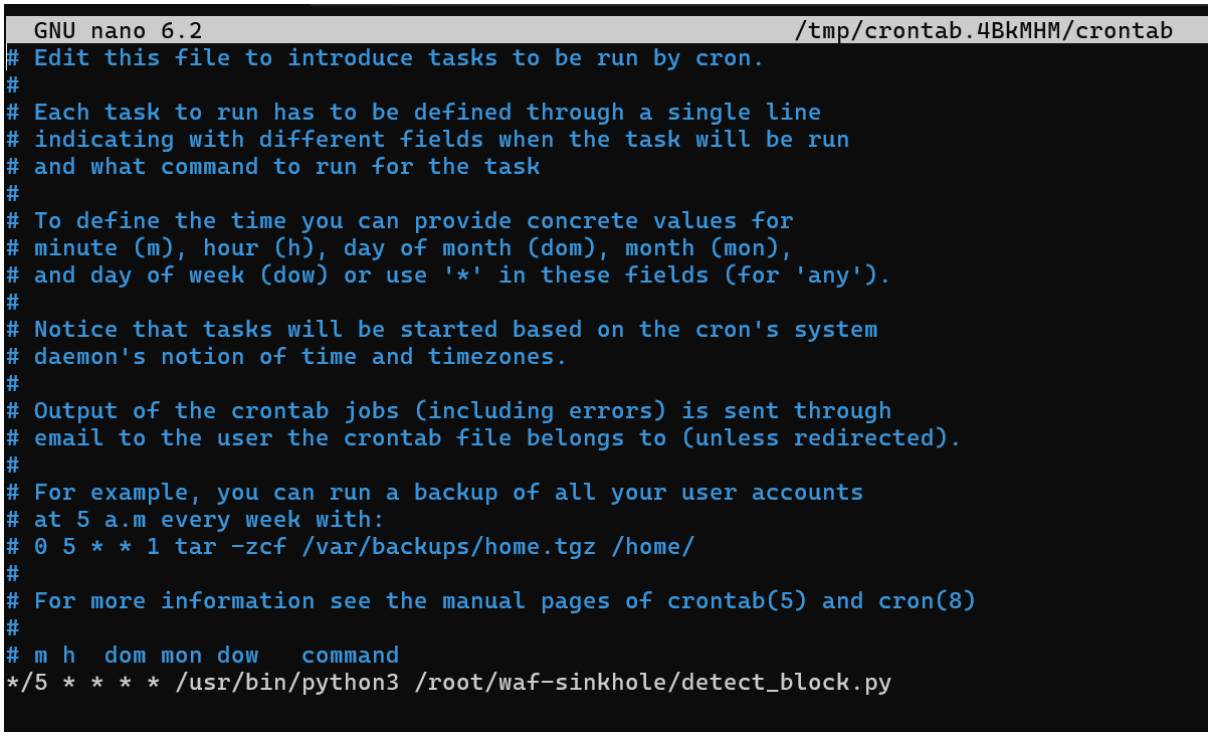
SNS Integration

- Used boto3 with the appropriate region and topic:

```
 GNU nano 6.2                                                detect_block.py *
import re
import os
import boto3


LOG_FILE = "/var/log/bind9/query.log"
BLACKLIST_FILE = "/etc/bind/named.conf.local"
SINKHOLE_ZONE = "/etc/bind/db.sinkhole"

AWS_REGION = 'us-east-1'
SNS_TOPIC_ARN = 'arn:aws:sns:sa-east-1:932744610558:dns-sinkhole-alerts:4c7d8f98-448c-4dd1-ba8e-ee18aaae9a1a'

def send_alert(domain_list):
    try:
        sns = boto3.client('sns', region_name=AWS_REGION)
        message = f"🚨 Suspicious domains detected and blocked:\n\n" + "\n".join(domain_list)
        sns.publish(
            TopicArn=SNS_TOPIC_ARN,
            Message=message,
            Subject='[DNS Sinkhole Alert] Malicious Domains Blocked'
        )
        print(f"SNS Alert sent for: {domain_list}")
    except Exception as e:
        print(f"Error sending SNS alert: {e}")


def extract_domains(log_file):
    if not os.path.exists(log_file):
        print(f"Log file not found: {log_file}")
        return []
    with open(log_file, "r") as f:
```

4.5. Automation Using Cron

- Scheduled the script to run every 5 minutes using crontab:



5. Results and Validation

Checkpoint	Result
DNS Sinkhole Functionality	Successfully resolved malicious domains to 127.0.0.1.
Query Logging	Active logging observed at /var/log/bind9/query.log.
Python Detection Script	Accurately extracted suspicious domains and appended sinkhole zones.
BIND Restart Automation	Zones activated in real-time post-detection.
SNS Alerting	Timely alerts received with domain details via email.
Cron Execution Logs	Logs available at /var/log/detect_block.log for auditability.

Sample Alert Message Received:

Subject: DNS Sinkhole Alert

Message: Suspicious domains detected: bad-phish.com, malicious-host.org

6. Recommendations for Improvement

Area	Recommendation
Threat Intelligence Feeds	Integrate with real-time feeds (e.g., AbuseIPDB, PhishTank, OTX).
Whitelist Capability	Add logic to exclude known safe domains to prevent false positives.
Centralized Logging	Forward logs to ELK/Graylog for SIEM correlation and visualization.
HA Configuration	Set up a secondary BIND9 server for redundancy and failover.
Advanced Pattern Matching	Use machine learning or YARA rules for better domain anomaly detection.
Alert Metadata	Include source IP, timestamp, and EC2 instance ID in alert messages.

7. Conclusion

This task demonstrated a successful implementation of a DNS sinkhole server using BIND9 on AWS, reinforced with automated detection and alerting. The infrastructure enables real-time monitoring of DNS queries, automatic blocking of suspicious domains, and prompt notification to the security team. It forms a vital component of a defense-in-depth strategy and significantly improves an organization’s visibility into domain-based threats.

With strategic enhancements like threat feed integration, centralized logging, and scalability improvements, this DNS sinkhole solution can evolve into a comprehensive enterprise-grade threat intelligence system.