DNS Inspector and CDN DNS Server Documentation

Table of Contents

1. **Introduction**

2. DNS Inspector
    - **Overview**
    - **Features**
    - **Installation**
    - **Usage**
    - **Command-Line Arguments**
    - **Simulated Scenarios**
    - **Report Generation**

3. **CDN (content delivery network) DNS server**
    - **Overview**
    - **Features**
    - **How It Works**
    - **Installation**
    - **Usage**
    - **Server Configuration**

4. **Importance and Usefulness**

5. **Future Enhancements**

6. **Conclusion**

Introduction

This documentation covers two Python projects: the DNS Inspector and the CDN DNS Server. The DNS Inspector is a tool for capturing and analyzing DNS traffic, simulating scenarios like server failure and DNS poisoning, and generating reports. The CDN DNS Server is a simulation of a DNS server for Content Delivery Networks (CDNs), demonstrating load-balancing strategies across multiple servers.

DNS Inspector

Overview

The DNS Inspector is a Python script that captures and analyzes DNS traffic on a specified network interface. It can simulate scenarios like server failure and DNS poisoning attacks, and generate a report summarizing the captured DNS traffic.

Features

- Traffic Capture: Capture DNS traffic on a specified network interface using the Scapy library.
- DNS Record Analysis: Analyze the captured DNS traffic and extract DNS records like A and CNAME records.
- Scenario Simulation: Simulate server failure and DNS poisoning attack scenarios.
- Report Generation: Generate a text-based report summarizing the captured DNS traffic.

Installation

To use the DNS Inspector, you need to have Python and the Scapy library installed on your system.

1. Install Python (if not already installed) from the official Python website: https://www.python.org/downloads/
2. Install the Scapy library using pip:
- pip install scapy
3. Clone or download the project repository from GitHub.

Usage

1. Open a terminal or command prompt and navigate to the project directory.
2. Run the Inspector.py script with the appropriate command-line arguments (see below).

Command-Line Arguments

The DNS Inspector supports the following command-line arguments:

- -i or --interface: Specify the network interface to capture traffic on.
- -s or --simulate: Simulate a specific scenario (server_failure or dns_poisoning).
- -r or --report: Generate a report summarizing the captured DNS traffic.

Example usage:

- python Inspector.py -i eth0 -s dns_poisoning -r

This command captures DNS traffic on the eth0 interface, simulates a DNS poisoning attack scenario, and generates a report.

Simulated Scenarios

The DNS Inspector can simulate the following scenarios:

- Server Failure: Simulates a scenario where a server fails to respond to DNS queries.
- DNS Poisoning: Simulates a DNS poisoning attack scenario, where malicious DNS responses are injected into the network.

Report Generation

The DNS Inspector can generate a text-based report summarizing the captured DNS traffic. The report includes information about the domains queried and the corresponding DNS records (A and CNAME records) received from the DNS server.

The report is saved as Inspector_report.txt in the project directory.

CDN DNS Server

Overview

The CDN DNS Server is a Python script that simulates a DNS server for Content Delivery Networks (CDNs). It provides a mechanism to map domain names to a list of IP addresses representing the CDN servers responsible for serving that domain's content. When a DNS query is received for a supported domain, the server responds with a randomly selected IP address from the corresponding list, effectively load-balancing the traffic across multiple CDN servers.

Features

- DNS Query Handling: Process incoming DNS queries for supported domains.
- Load Balancing: Randomly select an IP address from the configured list of CDN servers for a domain, simulating load-balancing across multiple servers.
- DNS Response Generation: Construct and send appropriate DNS responses containing the selected IP address for the requested domain.

How It Works

The CDN DNS Server works as follows:

1. Server Configuration: The server maintains a dictionary (servers) that maps domain names to lists of IP addresses representing the CDN servers for each domain.
2. DNS Query Handling: The handle_query function is responsible for processing incoming DNS queries. It parses the query to extract the requested domain name and query type.
3. Domain Mapping: If the requested domain name is found in the servers dictionary, the function selects a random IP address from the corresponding list of CDN server IP addresses.
4. Response Construction: The handle_query function constructs a DNS response packet containing the randomly selected IP address for the requested domain. If the requested domain is not found in the configuration, the function returns None, indicating that no response should be sent.
5. Server Execution: The main function creates a UDP socket bound to port 10053 and listens for incoming DNS queries. When a query is received, it calls the handle_query function to process the query and generate a response. The response is then sent back to the client via the UDP socket.

Installation

To use the CDN DNS Server, follow these steps:

1. Make sure you have Python installed on your system.
2. Clone or download the project repository from GitHub.

Usage

1. Open the CDN.py file in a text editor.
2. Modify the server's dictionary according to your requirements. This dictionary maps domain names to lists of IP addresses representing the CDN servers for each domain.
3. Save the changes to the CDN.py file.
4. Open a terminal or command prompt and navigate to the project directory.
5. Run the CDN.py script:

- python CDN.py

6. The script will start the CDN DNS server on port 10053 and print a message indicating that it has started.
7. You can now send DNS queries to the server on port 10053. The server will respond with a randomly selected IP address from the configured list for the requested domain.

Server Configuration

The servers dictionary in the CDN.py file allows you to configure the domain-to-IP mappings for your CDN. The dictionary should have the following structure:

python

```
servers = {
    "example.com": ["192.168.1.1", "192.168.1.2", "192.168.1.3"],
    "another.com": ["10.0.0.1", "10.0.0.2"],
    # Add more domains and IP addresses as needed
}
```

In this example, the domain example.com is mapped to three IP addresses (192.168.1.1, 192.168.1.2, and 192.168.1.3), and the domain another.com is mapped to two IP addresses (10.0.0.1 and 10.0.0.2).

Importance and Usefulness

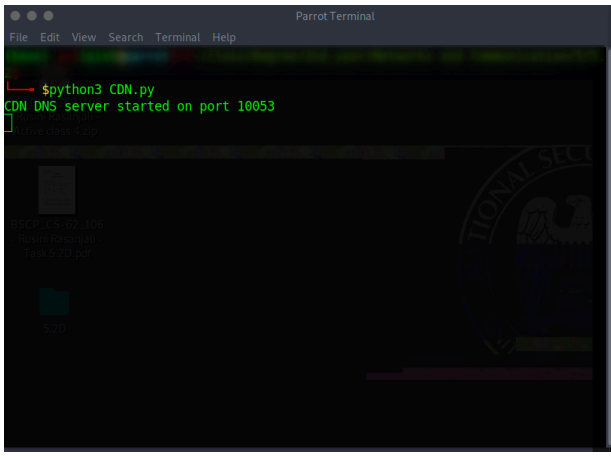
These projects are important and useful for the following reasons:

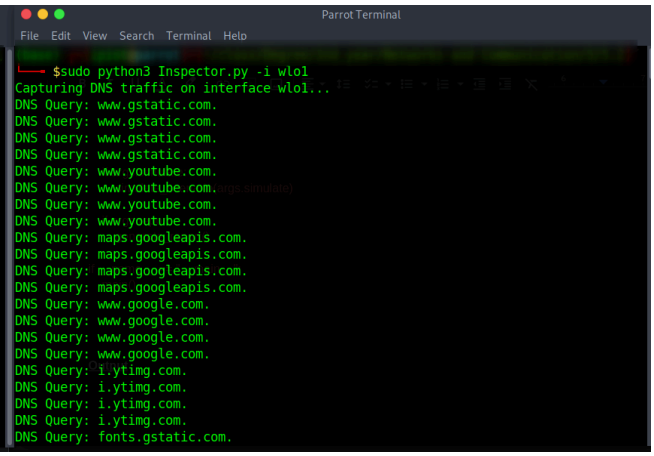
- DNS Traffic Analysis: The DNS Inspector allows you to capture and analyze DNS traffic, providing insights into the behavior of DNS servers and potential vulnerabilities or issues.
- Scenario Simulation: The DNS Inspector's ability to simulate scenarios like server failure and DNS poisoning attacks can be valuable for testing and understanding the impact of such scenarios on networks and applications.
- CDN Simulation: The CDN DNS Server provides a simulation environment for understanding how CDNs work in terms of DNS resolution and load-balancing across multiple servers. This can be valuable for educational purposes and learning about CDN concepts and strategies.
- Load Balancing: By randomly selecting an IP address from the configured list for a domain, the CDN DNS Server demonstrates a basic load-balancing mechanism used by CDNs to distribute traffic across multiple servers, improving performance and reducing the load on individual servers.
- DNS Response Generation: Both projects demonstrate how DNS responses are constructed and sent back to clients, providing insight into the DNS protocol and packet structure.
- Network Programming: These projects showcase the use of Python's socket programming modules, offering an opportunity to learn about network programming concepts and techniques.

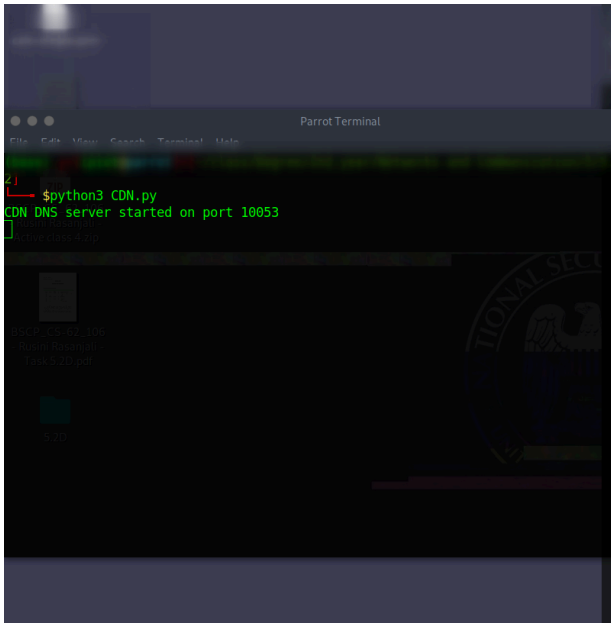
Future Enhancements

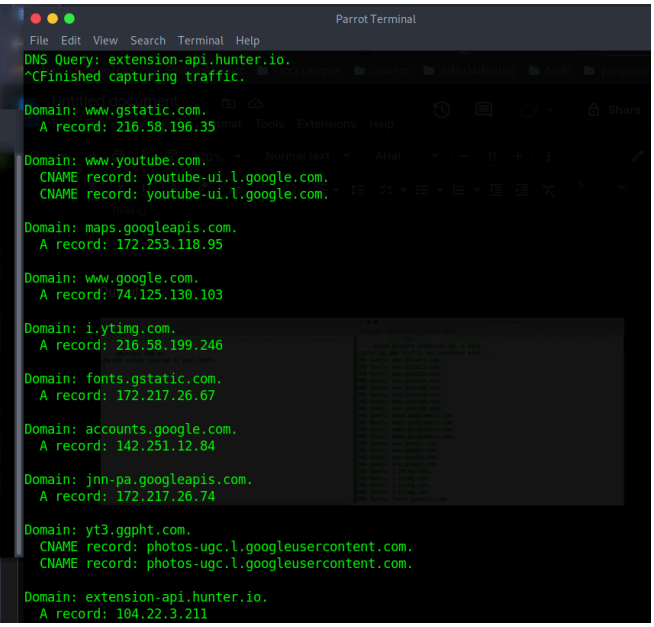
● Extend Scenario Simulation: To offer a more thorough testing and learning environment, incorporate more complex scenario simulations, such as DNS spoofing, cache poisoning, or distributed denial-of-service (DDoS) assaults.
● Add real-time traffic monitoring features to the script so that it may continuously record and show DNS traffic as it happens, enabling real-time analysis and alerting systems.
● Graphical User Interface (GUI): Create an intuitive GUI to improve the DNS Inspector's accessibility and usefulness. This will make it simpler to alter capture parameters, view collected data, and engage with the simulated situations.
● advanced Traffic Filtering: Use sophisticated filtering features to record and examine particular kinds of DNS traffic according to parameters like IP addresses, domain names, or record types.
● Integration with Other Tools: Look into the possibilities of combining the DNS Inspector with other security platforms or network analysis tools to facilitate easy data sharing and cooperative analysis.

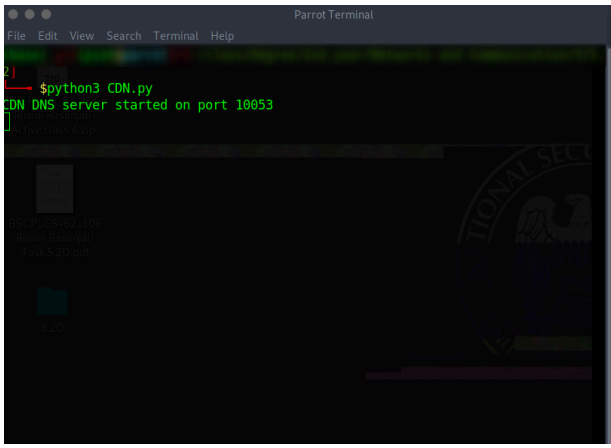
Output

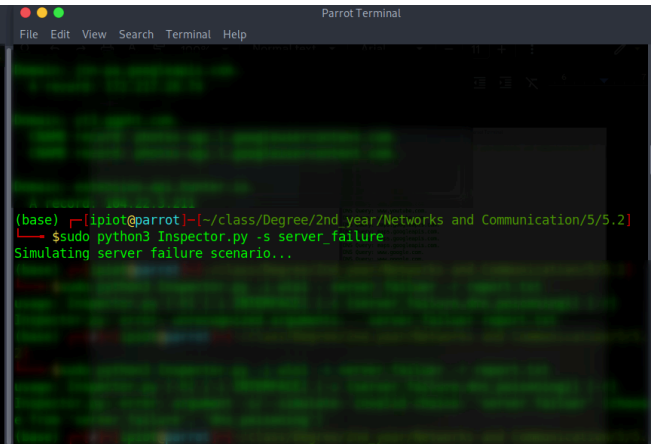

Parrot Terminal
File Edit View Search Terminal Help
$python3 CDN.py
CDN DNS server started on port 10053

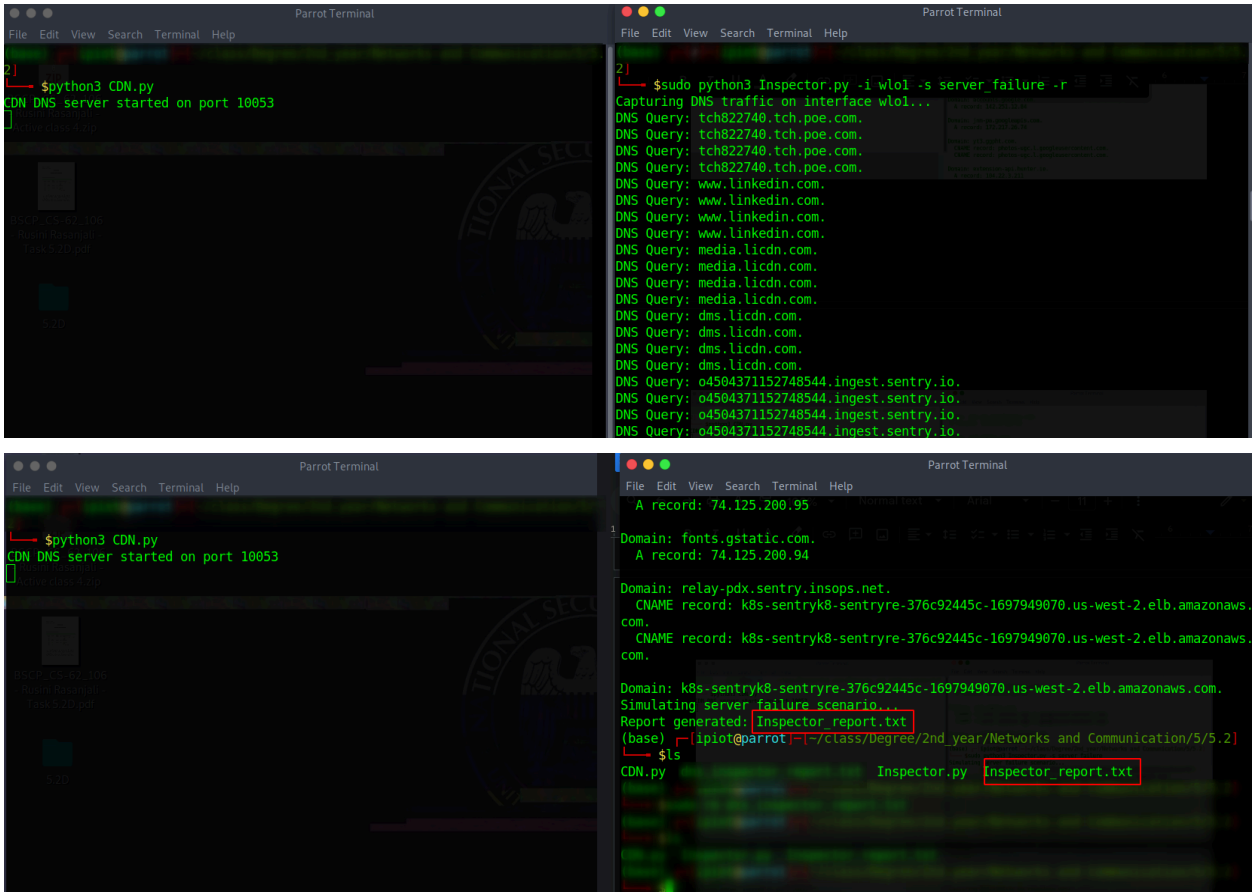

Parrot Terminal
File Edit View Search Terminal Help
$sudo python3 Inspector.py -i wlo1
Capturing DNS traffic on interface wlo1...
DNS Query: www.gstatic.com.
DNS Query: www.gstatic.com.
DNS Query: www.gstatic.com.
DNS Query: www.gstatic.com.
DNS Query: www.youtube.com.
DNS Query: www.youtube.com.
DNS Query: www.youtube.com.
DNS Query: www.youtube.com.
DNS Query: maps.googleapis.com.
DNS Query: maps.googleapis.com.
DNS Query: maps.googleapis.com.
DNS Query: maps.googleapis.com.
DNS Query: www.google.com.
DNS Query: www.google.com.
DNS Query: www.google.com.
DNS Query: www.google.com.
DNS Query: i.ytimg.com.
DNS Query: i.ytimg.com.
DNS Query: i.ytimg.com.
DNS Query: i.ytimg.com.
DNS Query: fonts.gstatic.com.


Parrot Terminal
$python3 CDN.py
CDN DNS server started on port 10053


Parrot Terminal
File Edit View Search Terminal Help
DNS Query: extension-api.hunter.io.
^CFinished capturing traffic.
Domain: www.gstatic.com.
A record: 216.58.196.35
Domain: www.youtube.com.
CNAME record: youtube-ui.l.google.com.
CNAME record: youtube-ui.l.google.com.
Domain: maps.googleapis.com.
A record: 172.253.118.95
Domain: www.google.com.
A record: 74.125.130.103
Domain: i.ytimg.com.
A record: 216.58.199.246
Domain: fonts.gstatic.com.
A record: 172.217.26.67
Domain: accounts.google.com.
A record: 142.251.12.84
Domain: jnn-pa.googleapis.com.
A record: 172.217.26.74
Domain: yt3.ggpht.com.
CNAME record: photos-ugc.l.googleusercontent.com.
CNAME record: photos-ugc.l.googleusercontent.com.
Domain: extension-api.hunter.io.
A record: 104.22.3.211


Parrot Terminal
File Edit View Search Terminal Help
$python3 CDN.py
CDN DNS server started on port 10053


Parrot Terminal
File Edit View Search Terminal Help
(base) [ipiot@parrot]-[~/class/Degree/2nd_year/Networks and Communication/5/5.2]
$sudo python3 Inspector.py -s server_failure
Simulating server failure scenario...



In the above screenshots i have demonstrated all of the possibilities i have implemented in my Inspector project, initially i have showed that initializing a simple interface scan as “sudo python3 Inspector.py -i wlo1” this will scan the wlo1 interface i initialize it because I am using a wifi connection. When I canceled the scan I saw the domain A records and CNAME records. Afterwards I ran a scan with this command “sudo python3 Inspector.py -s server_failure” to check if there was a server failure issue. And then I ran a scan with this command “sudo python3 Inspector.py -i wlo1 -s server_failure -r ” this will complete the above two mentioned tasks as well as after completion it will generate a report.