Project

Distributed Denial of Service Attacks in MYSQL DATABASE PROJECT

Prepared by:

KSHITIJ KUMAR

Guided by:

ZAKIR HUSSAIN

TABLE OF CONTENTS

INDEX

CHAPTER 1
DDOS ATTACK DESCRIPTION

Distributed Denial-of-Service (DDoS) attack is a type of cyberattack where an attacker attempts to make a computer or network resource unavailable by overwhelming it with traffic from multiple sources. This is typically done by using a network of compromised devices (bots) to flood the targeted system with traffic, causing it to become overwhelmed and unable to handle legitimate requests.

How a ddos attack works:

A DDoS attack works by overwhelming a target system, such as a website or server, with an excessive amount of traffic.

A DDoS attack typically involves the following steps:

Botnet Formation: The attacker assembles a network of compromised devices called a botnet. These devices, which can include computers, servers, IoT devices, or smartphones infected with malware, are under the attacker's control and used to launch the attack.

Traffic generation: The attacker instructs the bots to send traffic to the targeted system, such as a website or network.

Traffic Redirection: To amplify the attack, the attacker may use techniques like IP spoofing or reflection/amplification attacks. IP spoofing involves forging the source IP address of the attacking traffic to make it appear as if it's coming from legitimate sources.

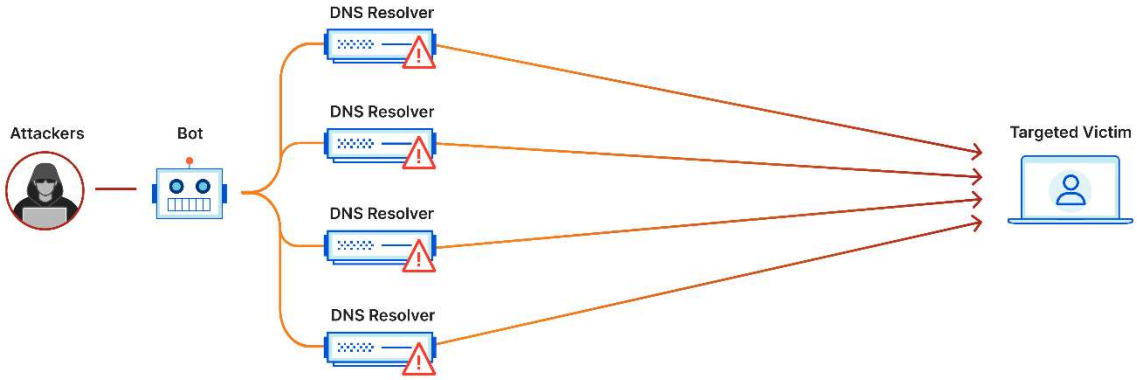
Traffic flood: The bots flood the targeted system with traffic, overwhelming its capacity to handle requests.

Types of DDoS attacks:

1. Volume-based attacks:

Targeted websites are flooded with voluminous malicious requests using amplification and other techniques to create massive traffic and deplete the bandwidth and other resources.

Amplification example:



DNS Amplification

2. Protocol attacks:

Exploiting weaknesses in network protocols to consume system resources.

The goal of the attack:

Protocol attacks, also known as a state-exhaustion attacks, cause a service disruption by over-consuming server resources and/or the resources of network equipment like firewalls and load balancers.

Protocol attacks utilize weaknesses in layer 3 and layer 4 of the protocol stack to render the target inaccessible.

Protocol attack example:

SYN flood

A SYN Flood is analogous to a worker in a supply room receiving requests from the front of the store.

The worker receives a request, goes and gets the package, and waits for confirmation before bringing the package out front. The worker then gets many more package requests without confirmation until they can't carry any more packages, become overwhelmed, and requests start going unanswered.

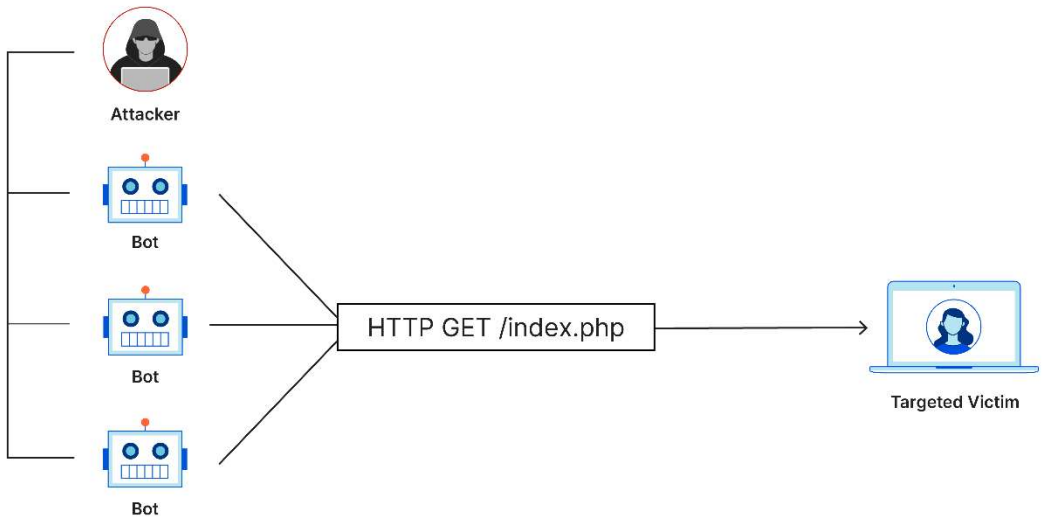
This attack exploits the TCP handshake — the sequence of communications by which two computers initiate a network connection — by sending a target a large number of TCP "Initial Connection Request" SYN packets with spoofed source IP addresses.

The target machine responds to each connection request and then waits for the final step in the handshake, which never occurs, exhausting the target's resources in the process.

3. Application attacks:

The goal of these attacks is to exhaust the target's resources to create a denial-of-service. The attacks target the layer where web pages are generated on the server and delivered in response to HTTP requests. A single HTTP request is computationally cheap to execute on the client side, but it can be expensive for the target server to respond to, as the server often loads multiple files and runs database queries in order to create a web page.

Layer 7 attacks are difficult to defend against, since it can be hard to differentiate malicious traffic from legitimate traffic.



Application layer attack example

HTTP flood:

This attack is similar to pressing refresh in a web browser over and over on many different computers at once – large numbers of HTTP requests flood the server, resulting in denial-of-service.

This type of attack ranges from simple to complex.

Simpler implementations may access one URL with the same range of attacking IP addresses, referrers and user agents. Complex versions may use a large number of attacking IP addresses, and target random urls using random referrers and user agents.

DDoS attacks can be launched using various techniques, including:

1. Botnets: Networks of compromised devices used to launch attacks.
2. Malware: Software designed to harm or exploit systems.
3. Scripting: Using scripts to automate attack processes.
4. Amplification attacks: Using third-party services to amplify traffic.

DDoS prevention methods:

- **Attack surface reduction:** Limiting attack surface exposure can help minimize the effect of a DDoS attack. Several methods for reducing this exposure include restricting traffic to specific locations, implementing a load balancer, and blocking communication from outdated or unused ports, protocols, and applications.
- **Anycast network diffusion:** An Anycast network helps increase the surface area of an organization's network, so that it can more easily absorb volumetric traffic spikes (and prevent outages) by dispersing traffic across multiple distributed servers.
- **Real-time, adaptive threat monitoring:** Log monitoring can help pinpoint potential threats by analysing network traffic patterns, monitoring traffic spikes or other unusual activity, and adapting to defend against anomalous or malicious requests, protocols, and IP blocks.
- **Caching:** A cache stores copies of requested content so that fewer requests are serviced by origin servers. Using a content delivery network (CDN) to cache resources can reduce the strain on an organization's servers and make it more difficult for them to become overloaded by both legitimate and malicious requests.
- **Rate limiting:** Rate limiting restricts the volume of network traffic over a specific time period, essentially preventing web servers from getting overwhelmed by requests from specific IP addresses. Rate limiting can be used to prevent DDoS attacks that use botnets to spam an endpoint with an abnormal amount of requests at once.

DDoS prevention tools

Web application firewall (WAF): A WAF helps block attacks by using customizable policies to filter, inspect, and block malicious HTTP traffic between web applications and the Internet. With a WAF, organizations can enforce a positive and negative security model that controls incoming traffic from specific locations and IP addresses.

Always-on DDoS mitigation: A DDoS mitigation provider can help prevent DDoS attacks by continuously analysing network traffic, implementing policy changes in response to emerging attack patterns, and providing an expansive and reliable network of data centres. When evaluating cloud-based DDoS mitigation services, look for a provider that offers adaptive, scalable, and always-on threat protection against sophisticated and volumetric attacks.

CHAPTER 2

DATABASES USED IN THIS PROJECT

In this DDoS Attack Analysis project, multiple databases are used to organize and store data related to attack detection, network traffic, system logs, botnet activities, and mitigation strategies. Each database focuses on a specific aspect of the project, ensuring a clear structure for storing and analyzing data to help identify and prevent DDoS attacks effectively.

1. Database: Attack_Detection

The Attack_Detection database stores information related to identified attacks, their types, sources, detection rules, and generated alerts. It enables efficient tracking and management of attacks in real time.

Tables in Attack_Detection:

1. Attacks: Stores details of each detected attack, including the attack type, date, and source IP.
2. Attack_Types: Contains a list of attack types with descriptions.
3. Sources: Stores IP addresses of attack sources and their country of origin.
4. Detection_Rules: Contains rules used to detect specific attack types.
5. Alerts: Records alerts generated when an attack is detected, along with alert levels and attack details.

2. Database: Network_Traffic

The Network_Traffic database manages all network traffic data, including IP addresses, protocols, traffic statistics, and devices. Monitoring this data helps in detecting unusual traffic spikes that could indicate a DDoS attack.

Tables in Network_Traffic:

1. Traffic: Logs network traffic details such as timestamps, source/destination IPs, and protocols.
2. Protocols: Stores protocol details like TCP, UDP, and ICMP.

3. IP_Addresses: Contains IP addresses involved in network communications.
4. Network_Devices: Stores information about routers, switches, and other devices.
5. Traffic_Stats: Logs traffic volume to detect potential traffic anomalies.

3. Database: System_Logging

The System_Logging database stores system event logs, user actions, and performance metrics. It helps monitor system behavior and troubleshoot potential issues during or after an attack.

Tables in System_Logging:

1. Logs: Stores system logs with details like message and log level.
2. Log_Types: Contains types of logs, such as error, warning, or info.
3. System_Events: Logs events such as service restarts and crashes.
4. User_Actions: Records user activities like login, logout, and file uploads.
5. System_Metrics: Logs system performance data such as CPU and memory usage.

4. Database: Botnet_Information

The Botnet_Information database tracks botnets involved in DDoS attacks, including infected devices, botnet commands, and activities. It helps in analyzing and responding to botnet-driven attacks.

Tables in Botnet_Information:

1. Botnets: Stores information on known botnets and control server IPs.
2. Infected_Devices: Records devices infected by botnets and their IP addresses.
3. Command_Control: Logs commands sent by botnet controllers to infected devices.
4. Botnet_Activity: Tracks botnet activities such as attacks launched.

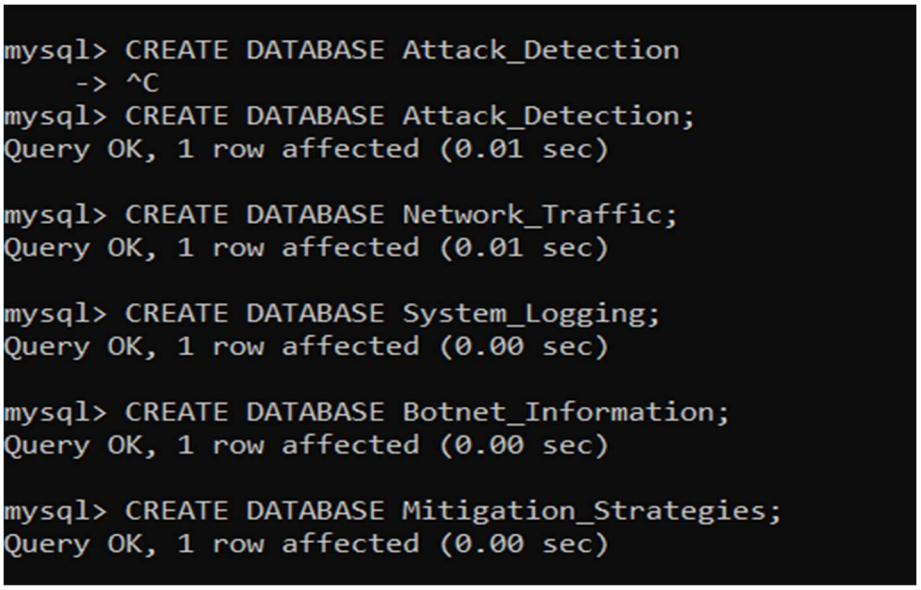
5. Botnet_Membership: Logs when devices join or leave a botnet.

5. Database: Mitigation_Strategies

The Mitigation_Strategies database stores information about the defense strategies applied to mitigate attacks and their effectiveness. It tracks the actions taken during incident responses and evaluates the success of mitigation measures.

Tables in Mitigation_Strategies:

1. Strategies: Contains various mitigation strategies such as rate limiting and IP blacklisting.
2. Active_Strategies: Logs strategies that are currently in use and their application date.
3. Strategy_Effectiveness: Measures the effectiveness of strategies in stopping attacks.
4. Incident_Response: Stores records of responses to specific incidents, including actions taken.
5. Mitigation_Logs: Tracks all mitigation actions for audit and analysis purposes.



CHAPTER 3

TABLES USED IN EACH DATABASES

This chapter outlines the tables used within each database in the project. Each table is structured to store specific types of data related to DDoS attack detection, network traffic, system logging, botnet activity, and mitigation strategies. These tables ensure that the data is organized in a way that supports efficient querying and analysis for both real-time and post-incident review.

1. Database: Attack_Detection

The Attack_Detection database is essential for recording information about attacks, the types of attacks, their sources, and alerts generated by the detection system. The following tables are used:

Tables in Attack_Detection:

1. Attacks:
   - Fields: id, attack_type, attack_date, source_ip
   - Purpose: Stores details of all detected attacks, including the attack type and source.
2. Attack_Types:
   - Fields: id, type_name, description
   - Purpose: Contains the various types of attacks (e.g., SYN flood, UDP flood), along with descriptions.
3. Sources:
   - Fields: id, source_ip, source_country
   - Purpose: Stores the IP addresses of the attack sources and their corresponding countries.
4. Detection_Rules:
   - Fields: id, rule_name, rule_description

- Purpose: Contains the rules used to detect attacks (e.g., threshold-based, signature-based).

5. Alerts:
   - Fields: id, attack_id, alert_date, alert_level
   - Purpose: Records alerts generated when an attack is detected, specifying the level of urgency.

```
mysql> USE Attack_Detection;
Database changed
mysql> CREATE TABLE attacks (id INT, attack_type INT, attack_date VARCHAR(250), source_ip VARCHAR(250));
Query OK, 0 rows affected (0.03 sec)

mysql> CREATE TABLE Attack_types (id INT, type_name VARCHAR(250), description TEXT);
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE TABLE Sources (id INT, source_ip VARCHAR(250), source_country VARCHAR(250));
Query OK, 0 rows affected (0.02 sec)

mysql> CREATE TABLE Detection_rules (id INT, rule_name VARCHAR(250), rule_description TEXT);
Query OK, 0 rows affected (0.02 sec)

mysql> CREATE TABLE Alerts (id INT, attack_id INT,alert_date VARCHAR(250), alert_level VARCHAR(250));
Query OK, 0 rows affected (0.02 sec)
```

EXAMPLE OF INSERTING VALUES IN TABLE:

```
mysql> INSERT INTO alerts VALUES (1, 1, "2022-01-01 12:00:00", "192.168.1.100"),
(2, 2, "2022-01-02 13:00:00", "192.168.1.101"), (3, 3, "2022-01-03 14:00:00", "19
2.168.1.102"), (4, 1, "2022-01-04 15:00:00", "192.168.1.103"), (5, 2, "2022-01-05
 16:00:00", "192.168.1.104");
Query OK, 5 rows affected (0.01 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM alerts;
+------+-----------+---------------------+---------------+
| id   | attack_id | alert_date          | alert_level   |
+------+-----------+---------------------+---------------+
|    1 |         1 | 2022-01-01 12:00:00 | 192.168.1.100 |
|    2 |         2 | 2022-01-02 13:00:00 | 192.168.1.101 |
|    3 |         3 | 2022-01-03 14:00:00 | 192.168.1.102 |
|    4 |         1 | 2022-01-04 15:00:00 | 192.168.1.103 |
|    5 |         2 | 2022-01-05 16:00:00 | 192.168.1.104 |
+------+-----------+---------------------+---------------+
5 rows in set (0.00 sec)
```

2. Database: Network_Traffic

The Network_Traffic database focuses on managing all network-related data, including IP addresses, protocols, traffic patterns, and device information. These tables are crucial for monitoring the flow of network traffic and identifying abnormalities.

Tables in Network_Traffic:

1. **Traffic:**
   - Fields: id, timestamp, source_ip, destination_ip, protocol
   - Purpose: Records all network traffic, with timestamps and the source/destination of the data packets.
2. **Protocols:**
   - Fields: id, protocol_name, description
   - Purpose: Stores details of various network protocols used in traffic (e.g., TCP, UDP, ICMP).
3. **IP_Addresses:**
   - Fields: id, ip_address, ip_type
   - Purpose: Contains both private and public IP addresses involved in network communication.
4. **Network_Devices:**
   - Fields: id, device_name, device_type
   - Purpose: Stores information about network devices (e.g., routers, switches) participating in traffic.
5. **Traffic_Stats:**
   - Fields: id, timestamp, traffic_volume
   - Purpose: Logs traffic volumes to monitor spikes in activity, which may indicate DDoS attacks.

```
mysql> CREATE TABLE traffic (id INT, timestamp VARCHAR(250), source_ip VARCHAR(250), destination_ip VARC
HAR(250), protocol VARCHAR(100));
Query OK, 0 rows affected (0.02 sec)

mysql> DESC TABLE TRAFFIC;
+----+-------------+---------+------------+------+---------------+------+---------+------+------+-------
---+-------+
| id | select_type | table   | partitions | type | possible_keys | key  | key_len | ref  | rows | filter
ed | Extra |
+----+-------------+---------+------------+------+---------------+------+---------+------+------+-------
---+-------+
|  1 | SIMPLE      | TRAFFIC | NULL       | ALL  | NULL          | NULL | NULL    | NULL |    1 |   100.
00 | NULL  |
+----+-------------+---------+------------+------+---------------+------+---------+------+------+-------
---+-------+
1 row in set, 1 warning (0.01 sec)

mysql> CREATE TABLE protocols (id INT, protocol_name VARCHAR(100), description TEXT);
Query OK, 0 rows affected (0.02 sec)

mysql> CREATE TABLE ip_addresses (id INT, ip_address VARCHAR(250), ip_type VARCHAR(100), source_country
VARCHAR(250), Entity_Type VARCHAR(100));
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE TABLE network_devices (id INT, device_name VARCHAR(250), device_type VARCHAR(250));
Query OK, 0 rows affected (0.02 sec)

mysql> CREATE TABLE traffic_stats (id INT, timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP, traffic_volume
 INT);
Query OK, 0 rows affected (0.02 sec)
```

EXAMPLE OF INSERTING VALUES IN TABLE:

```
mysql> INSERT INTO traffic VALUES (1, "2022-01-01 12:00:00", "192.168.1.100", "192.168.1.1", "TCP"),
    -> (2, "2022-01-02 13:00:00", "192.168.1.101", "192.168.1.2", "UDP"), (3, "2022-01-03 14:00:00", "19
2.168.1.102", "192.168.1.3", "HTTP"), (4, "2022-01-04 15:00:00", "192.168.1.103", "192.168.1.4", "FTP"),
 (5, "2022-01-05 16:00:00", "192.168.1.104", "192.168.1.5", "SSH");
Query OK, 5 rows affected (0.01 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM TRAFFIC;
+------+---------------------+--------------+----------------+----------+
| id   | timestamp           | source_ip    | destination_ip | protocol |
+------+---------------------+--------------+----------------+----------+
|    1 | 2022-01-01 12:00:00 | 192.168.1.100 | 192.168.1.1    | TCP      |
|    2 | 2022-01-02 13:00:00 | 192.168.1.101 | 192.168.1.2    | UDP      |
|    3 | 2022-01-03 14:00:00 | 192.168.1.102 | 192.168.1.3    | HTTP     |
|    4 | 2022-01-04 15:00:00 | 192.168.1.103 | 192.168.1.4    | FTP      |
|    5 | 2022-01-05 16:00:00 | 192.168.1.104 | 192.168.1.5    | SSH      |
+------+---------------------+--------------+----------------+----------+
5 rows in set (0.00 sec)
```

3. Database: System_Logging

The System_Logging database stores system logs and events that are generated during normal operations and during attacks. These logs are essential for post-incident analysis and monitoring the health of the system.

Tables in System_Logging:

1. **Logs**:
   - Fields: id, log_message, log_level, timestamp
   - Purpose: Stores system logs, categorized by severity levels (e.g., info, warning, error).
2. **Log_Types:**
   - Fields: id, log_type_name, description
   - Purpose: Contains different types of logs, such as system errors, warnings, or informational logs.
3. **System_Events:**
   - Fields: id, event_description, event_date
   - Purpose: Records key system events, like service restarts, system crashes, and maintenance actions.
4. **User_Actions:**
   - Fields: id, user_id, action_description, action_date
   - Purpose: Logs actions taken by users, such as logins, file uploads, and changes to system settings.
5. **System_Metrics:**
   - Fields: id, cpu_usage, memory_usage, disk_usage, timestamp
   - Purpose: Logs system performance metrics, useful for monitoring system load and health.

```
mysql> USE System_Logging;
Database changed
mysql> CREATE TABLE logs (id INT, log_timestamp VARCHAR(200), message TEXT, log_level INT);
Query OK, 0 rows affected (0.02 sec)

mysql> CREATE TABLE log_levels (id INT, level_name VARCHAR(200), description TEXT);
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE TABLE system_events (id INT, event_name VARCHAR(200), event_description TEXT);
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE TABLE error_logs (id INT, log_timestamp VARCHAR(200), error_code VARCHAR(100), error_message TEXT);
Query OK, 0 rows affected (0.02 sec)

mysql> CREATE TABLE auth_logs (id INT, log_timestamp VARCHAR(200), user_id INT, action VARCHAR(200));
Query OK, 0 rows affected (0.02 sec)
```

EXAMPLE OF INSERTING VALUES IN TABLE:

```
mysql> INSERT INTO logs (id, log_timestamp, message, log_level) VALUES(1, '2024-0
9-16 09:00:00', 'System booted', 1),(2, '2024-09-16 09:05:00', 'User login succes
sful', 2),(3, '2024-09-16 09:10:00', 'Firewall rule updated', 3),(4, '2024-09-16
09:15:00', 'Network traffic high', 4),(5, '2024-09-16 09:20:00', 'DDoS attack det
ected', 5);
Query OK, 5 rows affected (0.01 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM LOGS;
+------+---------------------+----------------------+-----------+
| id   | log_timestamp       | message              | log_level |
+------+---------------------+----------------------+-----------+
|    1 | 2024-09-16 09:00:00 | System booted        |         1 |
|    2 | 2024-09-16 09:05:00 | User login successful |         2 |
|    3 | 2024-09-16 09:10:00 | Firewall rule updated |         3 |
|    4 | 2024-09-16 09:15:00 | Network traffic high  |         4 |
|    5 | 2024-09-16 09:20:00 | DDoS attack detected  |         5 |
+------+---------------------+----------------------+-----------+
5 rows in set (0.00 sec)
```

4. Database: Botnet_Information

The Botnet_Information database is used to store data related to botnets and the devices they infect. It also tracks commands sent by botnet controllers and monitors their activities.

Tables in Botnet_Information:

1. **Botnets:**
   - Fields: id, botnet_name, control_server_ip
   - Purpose: Stores details about known botnets and their control servers.
2. **Infected_Devices:**
   - Fields: id, device_ip, botnet_id
   - Purpose: Contains information about devices infected by botnets.
3. **Command_Control:**
   - Fields: id, command_description, issued_by, issued_date
   - Purpose: Stores the commands sent by botnet controllers to their infected devices.
4. **Botnet_Activity:**
   - Fields: id, activity_description, activity_date
   - Purpose: Tracks botnet activities, including attacks initiated or other malicious behaviors.
5. **Botnet_Membership:**
   - Fields: id, device_ip, membership_date
   - Purpose: Logs when devices join or leave a botnet, aiding in tracking botnet growth.

```
mysql> USE Botnet_Information;
Database changed
mysql> CREATE TABLE botnet_nodes (id INT, ip_address VARCHAR(100), hostname VARCHAR(200), location VARCHAR(200));
Query OK, 0 rows affected (0.02 sec)

mysql> CREATE TABLE botnet_commands (id INT, command_name VARCHAR(200), command_type VARCHAR(200));
Query OK, 0 rows affected (0.02 sec)

mysql> CREATE TABLE infected_devices (id INT, device_name VARCHAR(200), infection_date VARCHAR(200));
Query OK, 0 rows affected (0.02 sec)

mysql> CREATE TABLE malware_types (id INT, type_name VARCHAR(200), description TEXT);
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE TABLE control_servers (id INT, ip_address VARCHAR(200), country VARCHAR(200));
Query OK, 0 rows affected (0.02 sec)
```

EXAMPLE OF INSERTING VALUES IN TABLE:

```
mysql> INSERT INTO botnet_nodes (id, ip_address, hostname, location) VALUES(1, '192.168.2.1', 'BotNode_1', 'USA'),(2, '19
2.168.2.2', 'BotNode_2', 'China'),(3, '192.168.2.3', 'BotNode_3', 'Russia'),(4, '192.168.2.4', 'BotNode_4', 'India'),(5,
'192.168.2.5', 'BotNode_5', 'Brazil');
Query OK, 5 rows affected (0.01 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM botnet_nodes;
+------+------------+-----------+----------+
| id   | ip_address | hostname  | location |
+------+------------+-----------+----------+
|    1 | 192.168.2.1 | BotNode_1 | USA      |
|    2 | 192.168.2.2 | BotNode_2 | China    |
|    3 | 192.168.2.3 | BotNode_3 | Russia   |
|    4 | 192.168.2.4 | BotNode_4 | India    |
|    5 | 192.168.2.5 | BotNode_5 | Brazil   |
+------+------------+-----------+----------+
5 rows in set (0.00 sec)
```

5. Database: Mitigation_Strategies

The Mitigation_Strategies database stores data related to the various strategies and actions taken to mitigate DDoS attacks. It tracks the strategies used, their effectiveness, and incident responses.

Tables in Mitigation_Strategies:

1. **Strategies:**
   - Fields: id, strategy_name, description
   - Purpose: Stores different mitigation strategies, such as IP blocking, traffic shaping, and rate limiting.
2. **Active_Strategies:**
   - Fields: id, strategy_id, start_date, end_date
   - Purpose: Logs active strategies being applied, including the start and end dates.
3. **Strategy_Effectiveness:**
   - Fields: id, strategy_id, effectiveness_rating, evaluation_date
   - Purpose: Measures the effectiveness of each strategy over time, allowing for analysis and optimization.
4. **Incident_Response:**
   - Fields: id, incident_id, response_description, response_date
   - Purpose: Stores records of responses taken during specific incidents, documenting actions taken.
5. **Mitigation_Logs:**

- Fields: id, log_description, log_date
- Purpose: Logs all actions and decisions related to mitigation efforts for future reference and auditing.

```
mysql> USE Mitigation_Strategies;
Database changed
mysql> CREATE TABLE strategies (id INT, strategy_name VARCHAR(200), description TEXT);
Query OK, 0 rows affected (0.02 sec)

mysql> CREATE TABLE ddos_mitigation (id INT, strategy_id INT, effectiveness VARCHAR(200));
Query OK, 0 rows affected (0.02 sec)

mysql> CREATE TABLE firewall_rules (id INT, rule_name VARCHAR(200), action VARCHAR(100));
Query OK, 0 rows affected (0.06 sec)

mysql> CREATE TABLE rate_limiting (id INT, ip_address VARCHAR(200), limit INT);
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to
mysql> CREATE TABLE rate_limiting (id INT, ip_address VARCHAR(200), limit INT);
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to
mysql> CREATE TABLE rate_limiting (id INT, ip_address VARCHAR(200), limits INT);
Query OK, 0 rows affected (0.02 sec)

mysql> CREATE TABLE load_balancing (id INT, server_name VARCHAR(200), status VARCHAR(200));
Query OK, 0 rows affected (0.02 sec)
```

EXAMPLE OF INSERTING VALUES IN TABLE:

```
mysql> INSERT INTO strategies (id, strategy_name, description) VALUES(1, 'Rate Limiting', 'Limit the number of requests p
er IP'),(2, 'Geo-blocking', 'Block traffic from specific countries'),(3, 'Bot Detection', 'Identify and block bot traffic
'),(4, 'Load Balancing', 'Distribute traffic across servers'),(5, 'IP Whitelisting', 'Allow traffic from trusted IPs only
');
Query OK, 5 rows affected (0.01 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM STRATEGIES;
+------+-----------------+--------------------------------------+
| id   | strategy_name   | description                          |
+------+-----------------+--------------------------------------+
|    1 | Rate Limiting   | Limit the number of requests per IP  |
|    2 | Geo-blocking    | Block traffic from specific countries |
|    3 | Bot Detection   | Identify and block bot traffic       |
|    4 | Load Balancing  | Distribute traffic across servers    |
|    5 | IP Whitelisting | Allow traffic from trusted IPs only  |
+------+-----------------+--------------------------------------+
5 rows in set (0.00 sec)
```

CHAPTER 4

QUERIES IDENTIFIED BY THE NETWORK INFRA SECURITY TEAM

This chapter highlights key queries that the network security team uses to monitor and analyze the data collected across the various databases. These queries help identify potential DDoS attacks, track traffic patterns, analyze system logs, and evaluate the effectiveness of mitigation strategies. Below are some example queries with explanations and sample results to demonstrate their utility in analyzing the network's security.

1. Query: Retrieve All Detected Attacks with Attack Type and Source Information

This query fetches all detected attacks, their corresponding attack types, and the originating source information (IP address and country). It is used to analyze detected attacks and assess their severity.

```
mysql> SELECT Attacks.id, Attacks.attack_type, Attacks.source_ip, Attack_types.type_name, Attack_types.description FROM Attacks INNER JOIN attack_types ON attacks.id = attack_types.id;
+------+-------------+--------------+---------------------+-----------------------------------+
| id   | attack_type | source_ip    | type_name           | description                       |
+------+-------------+--------------+---------------------+-----------------------------------+
|    1 |           1 | 192.168.1.100 | DDoS                | Distributed Denial of Service     |
|    2 |           2 | 192.168.1.101 | SQL Injection       | Structured Query Language Injection |
|    3 |           3 | 192.168.1.102 | Cross-Site Scripting | XSS                               |
|    4 |           1 | 192.168.1.103 | Brute Force         | Password Guessing                 |
|    5 |           2 | 192.168.1.104 | Phishing            | Social Engineering                |
+------+-------------+--------------+---------------------+-----------------------------------+
5 rows in set (0.00 sec)
```

2. Query: List All Traffic from a Specific Source IP

This query helps to track the network traffic generated by a particular source IP. This can be useful in identifying malicious sources involved in DDoS attacks.

```
mysql> USE Network_Traffic;
Database changed
mysql> SELECT traffic.timestamp, traffic.source_ip, traffic.destination_ip, trafic_stats.trafic_volume FROM traffic INNER JOIN trafic_stats ON traffic.id = trafic_stats.id
WHERE traffic.source_ip = '192.168.1.100';
ERROR 1146 (42S02): Table 'network_traffic.trafic_stats' doesn't exist
mysql> SELECT traffic.timestamp, traffic.source_ip, traffic.destination_ip, traffic_stats.traffic_volume FROM traffic INNER JOIN traffic_stats ON traffic.id = traffic_stats
.id WHERE traffic.source_ip = '192.168.1.100';
+---------------------+--------------+----------------+----------------+
| timestamp           | source_ip    | destination_ip | traffic_volume |
+---------------------+--------------+----------------+----------------+
| 2022-01-01 12:00:00 | 192.168.1.100 | 192.168.1.1    |            500 |
+---------------------+--------------+----------------+----------------+
1 row in set (0.00 sec)
```

3. Query: Identify Active Mitigation Strategies

This query retrieves all mitigation strategies that are currently active. The results provide insight into what strategies are being used in real-time to prevent or mitigate DDoS attacks.

```
mysql> SELECT strategies.id AS strategies_id, strategies.strategy_name, firewall_rules.rule_name, firewall_rules.action FROM strategies INNER JOIN firewall_rules ON strategies.id = firewall_rules.id;
+--------------+----------------+-------------+--------+
| strategies_id | strategy_name  | rule_name   | action |
+--------------+----------------+-------------+--------+
|            1 | Rate Limiting  | Allow HTTPS | Allow  |
|            2 | Geo-blocking   | Block Telnet | Deny  |
|            3 | Bot Detection  | Allow SSH   | Allow  |
|            4 | Load Balancing | Block FTP   | Deny   |
|            5 | IP Whitelisting | Allow HTTP | Allow  |
+--------------+----------------+-------------+--------+
5 rows in set (0.00 sec)
```

4. Query: Top 5 Source IPs Generating the Most Traffic

This query identifies the top 5 source IP addresses generating the highest volume of network traffic. It helps in detecting suspicious IPs that might be participating in a DDoS attack.

```
mysql> USE Network_Traffic;
Database changed
mysql> SELECT ip_addresses.ip_address, SUM(traffic_stats.traffic_volume) AS Total_Volume FROM ip_addresses LEFT JOIN traffic_stats ON ip_addresses.id = traffic_stats.id GROUP BY
    -> ip_addresses.ip_address
    -> ORDER BY Total_Volume DESC
    -> LIMIT 5;
+---------------+--------------+
| ip_address    | Total_Volume |
+---------------+--------------+
| 8.8.8.8       |         1500 |
| 10.0.0.1      |         1200 |
| 172.16.254.1  |          800 |
| 192.168.1.101 |          750 |
| 192.168.1.100 |          500 |
+---------------+--------------+
5 rows in set (0.01 sec)
```

CHAPTER 5

FINAL GOAL OF THIS PROJECT

The primary goal of this project is to create a robust system that can detect, analyze, and mitigate Distributed Denial of Service (DDoS) attacks in real-time. By leveraging an efficient database and query system, the project aims to protect network infrastructure from malicious threats and ensure uninterrupted services.

Key Objectives

1. **DDoS Attack Detection**
    - Detect and flag suspicious traffic patterns to identify DDoS attacks early, minimizing service disruption.
2. **Real-Time Traffic Monitoring**
    - Continuously monitor network traffic to detect anomalies and identify abnormal spikes typical of DDoS attacks.
3. **Incident Logging**
    - Maintain detailed logs for post-attack analysis, helping teams understand the source and nature of attacks.
4. **Botnet and Infected Device Detection**
    - Identify and track devices infected by botnets, a major source of DDoS attacks, for better mitigation.
5. **Mitigation Strategies**
    - Implement automated or manual mitigation techniques, such as traffic filtering and blacklisting, to reduce attack impact.

Long-Term Vision

The project aims for scalable, adaptable protection against evolving DDoS threats, with a focus on proactive threat management and enhancing organizational security.